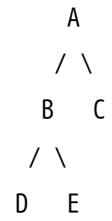
# IMPORTANT QUESTIONS:-

## 7-*Marks questions for RGPV EXA*M

## 1. Define Tree Terminology

**Tree** ek non-linear data structure hai jisme data hierarchical form me store hota hai.

Example:




### Important Terms

**1. Node**

Tree ka har element node kehlata hai.

Example: A, B, C, D, E

**2. Root Node**

Tree ka topmost node root hota hai.

Example: A

**3. Parent Node**

Jo node kisi dusre node ko connect karta hai.

Example: A parent hai B aur C ka.

**4. Child Node**

Jo node kisi parent se connected hota hai.

Example: B aur C child hain A ke.

**5. Leaf Node**

Jiske koi child nahi hote.

Example: C, D, E

**6. Edge**

Do nodes ko connect karne wali line edge hoti hai.

**7. Degree of Node**

Kisi node ke total children ki number.

Example: B ke 2 children hain, so degree of B = 2.

**8. Degree of Tree**

Tree me kisi bhi node ka maximum degree.

**9. Level**

Root ka level 0 hota hai, uske children level 1 par hote hain.

**10. Height of Tree**

Root se deepest leaf tak longest path ki length.

**11. Depth of Node**

Root se kisi particular node tak distance.

**Conclusion:**

Tree terminology samajhna important hai because BST, AVL, Heap, B Tree sab isi base par depend karte hain.

---

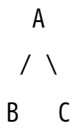
# 2. Explain Binary Tree Types

**Binary Tree** ek tree hai jisme har node ke maximum 2 children ho sakte hain.

## Types of Binary Tree

### 1. Full Binary Tree

Har node ke ya to 0 child hote hain ya 2 children.




### 2. Complete Binary Tree

Sab levels completely filled hote hain except last level, aur last level left side se filled hota hai.

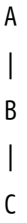
### 3. Perfect Binary Tree

Har internal node ke 2 children hote hain aur sab leaf nodes same level par hote hain.

### 4. Skewed Binary Tree

Tree ek side me grow karta hai.

Left skewed:




Right skewed:

```
\
 C
```

**5. Balanced Binary Tree**

Left aur right subtree ki height me zyada difference nahi hota.

**Conclusion:**

Binary tree ke different types searching, sorting aur memory management applications me use hote hain.

---

# 3. Explain BST with Example

**BST ka full form hai Binary Search Tree.**

BST ek binary tree hai jisme:

```
Left subtree < Root < Right subtree
```

## Example

```
      50
     /  \
   30    70
  / \    / \
20 40 60 80
```

Yaha:

- 30, 20, 40 are less than 50
- 70, 60, 80 are greater than 50

## BST Operations

### 1. Searching

Example: Search 60

```
60 > 50 → right
60 < 70 → left
60 found
```

### 2. Insertion

New value ko root se compare karke correct position par insert karte hain.

Example: Insert 25

```
25 < 50 → left
25 < 30 → left
25 > 20 → right
```

### 3. Deletion

Deletion ke 3 cases hote hain:

- Leaf node deletion
- Node with one child
- Node with two children

## Advantages

- Searching fast hoti hai
- Sorted data mil sakta hai using inorder traversal
- Dynamic structure hai

## Disadvantages

- Agar tree skewed ho jaye to searching slow ho sakti hai

**Conclusion:**

BST searching aur sorting ke liye useful tree data structure hai.

# 14_Marks Questions For RGPV EXAM

# 1. Explain BST Operations and Traversals

# Introduction

BST ka full form hai:

```
Binary Search Tree
```

BST ek binary tree hai jisme:

```
Left subtree < Root < Right subtree
```

---

# Example of BST

```
      50
     /  \
   30    70
```

```
  / \    / \
 20 40 60 80
```

---

# BST Operations

---

# 1. Insertion

New node ko correct position par insert kiya jata hai.

---

# Algorithm

```
If value < root
    insert left

Else
    insert right
```

---

# Example

Insert:

```
25
```

Steps:

```
25 < 50 → left
25 < 30 → left
25 > 20 → right
```

---

# 2. Searching

Target value ko compare karke search karte hain.

---

# Example

Search:

```
60
```

Steps:

```
60 > 50 → right
60 < 70 → left
Found
```

---

# 3. Deletion

Deletion ke 3 cases:

---

## Case 1: Leaf Node

Simply delete.

---

## Case 2: One Child

Child ko parent se connect karo.

---

## Case 3: Two Children

Inorder successor replace karo.

---

# Tree Traversals

Traversal means nodes ko visit karna.

---

# 1. Preorder Traversal

```
Root → Left → Right
```

Example:

```
50 30 20 40 70 60 80
```

---

# 2. Inorder Traversal

```
Left → Root → Right
```

Example:

```
20 30 40 50 60 70 80
```

---

# 3. Postorder Traversal

```
Left → Right → Root
```

Example:

```
20 40 30 60 80 70 50
```

---

# Advantages of BST

- Fast searching
- Sorted traversal
- Dynamic structure

---

# Disadvantages

- Skewed tree may slow searching

---

# Applications

- Database searching
- File systems
- Dictionaries

---

# Conclusion

BST searching and sorting ke liye efficient tree structure hai.

---

# 2. Explain AVL Tree with Rotations

# Introduction

AVL tree ek self-balancing BST hai.

Har node ka balance factor maintain hota hai.

---

# Balance Factor

```
Height(left subtree)
-
Height(right subtree)
```

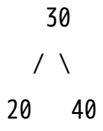
---

# AVL Condition

```
-1 ≤ Balance Factor ≤ 1
```

---

# Example




Balanced AVL tree.

---

# Need of AVL Tree

Normal BST skewed ho sakta hai.

AVL tree automatically balance maintain karta hai.

---

# Rotations in AVL Tree

---

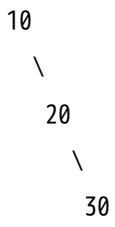
# 1. Left Rotation

Right-heavy tree ko balance karta hai.

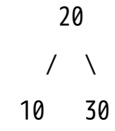
---

# Example

Before:




After rotation:




---

# 2. Right Rotation

Left-heavy tree ko balance karta hai.

---

# Example

Before:

```
      30
     /
    20
   /
 10
```

After:

```
    20
   /  \
 10   30
```

---

# 3. Left-Right Rotation

Combination of left and right rotation.

---

# 4. Right-Left Rotation

Combination of right and left rotation.

---

# Advantages

- Fast searching
- Balanced structure
- Height minimized

---

# Disadvantages

- Complex implementation
- More rotations required

---

# Applications

- Databases
- Searching systems
- Memory management

---

# Conclusion

AVL tree self-balancing BST hai jo rotations ke through balance maintain karta hai.

---

# 3. Explain Heap Tree and Applications

# Introduction

Heap ek complete binary tree hai.

Heap mainly two types ka hota hai:

1. Max Heap
2. Min Heap

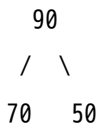
---

# Max Heap

Parent node children se greater hota hai.

---

# Example



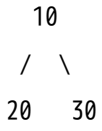

---

# Min Heap

Parent node children se smaller hota hai.

---

# Example




---

# Heap Operations

---

# 1. Insertion

New node insert karke heapify perform karte hain.

---

# 2. Deletion

Root delete karke heap property restore karte hain.

---

# Heapify

Heap property maintain karne ki process.

---

# Heap Sort

Heap tree sorting ke liye use hota hai.

---

# Steps

```
1. Build max heap
2. Swap root with last
3. Heapify remaining elements
4. Repeat
```

---

# Advantages

- Efficient sorting
- Fast priority handling

---

# Disadvantages

- Complex implementation

---

# Applications of Heap

- Priority queue
- CPU scheduling
- Heap sort
- Graph algorithms

---

# Time Complexity

```
Insertion = O(log n)
Deletion = O(log n)
```

---

# Conclusion

Heap tree complete binary tree hai jo priority-based applications me widely use hota hai.