1) Write an algorithm for depth first traversal of a graph

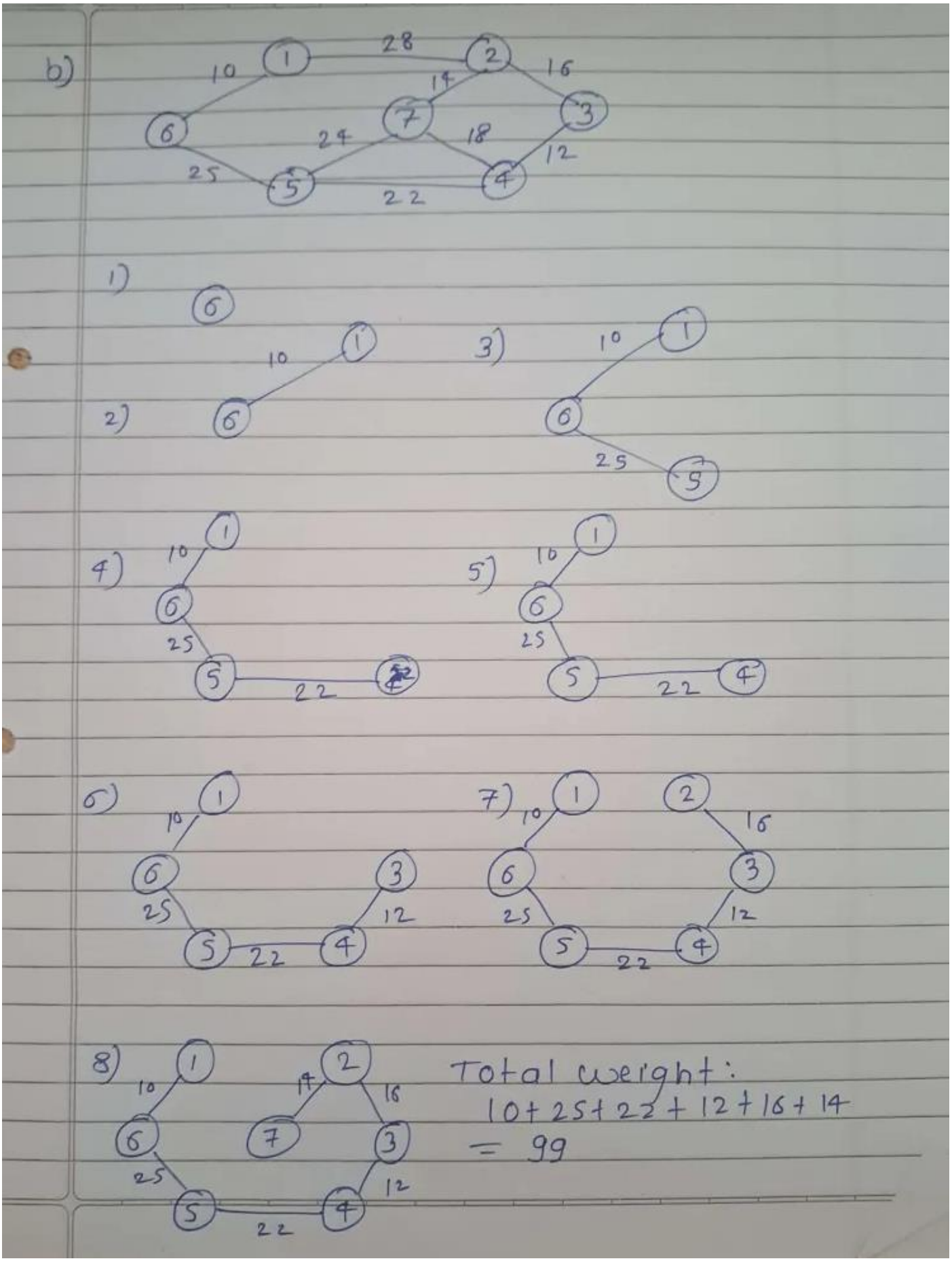
DFS - Algorithm

- **Step 1** - Define a Stack of size number of vertices in the graph.
- **Step 2** - Select any vertex as **starting point** for traversal. Visit that vertex and push it on to the Stack.
- **Step 3** - Visit any one of the non-visited **adjacent** vertices of a vertex which is at the top of stack and push it on to the stack.
- **Step 4** - Repeat step 3 until there is no new vertex to be visited from the vertex which is at the top of the stack.
- **Step 5** - When there is no new vertex to visit then use **back tracking** and pop one vertex from the stack.
- **Step 6** - Repeat steps 3, 4 and 5 until stack becomes Empty.

28

- **Step 7** - When stack becomes Empty, then produce final

2) Construct the minimum spanning tree (MST) for the given graph using Prim's Algorithm staring from vertex 6.



3) What is topological sorting? Find topological sorting of given graph.

Topological sorting for Directed Acyclic Graph (DAG) is a linear ordering of vertices such that for every directed edge u v, vertex u comes before v in the ordering.

Topological Sorting for a graph is not possible if the graph is not a DAG.

Example 3.10.1 *Sort the digraph for topological sort using source removal algorithm.*

SPPU : May-14, Marks 3

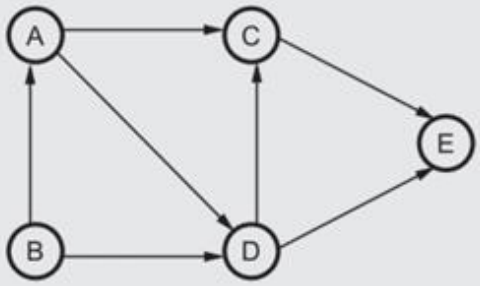


Fig. 3.10.1

Solution : We will follow following steps to obtain topologically sorted list.

Choose vertex B, because it has no incoming edge, delete it along with its adjacent edges.

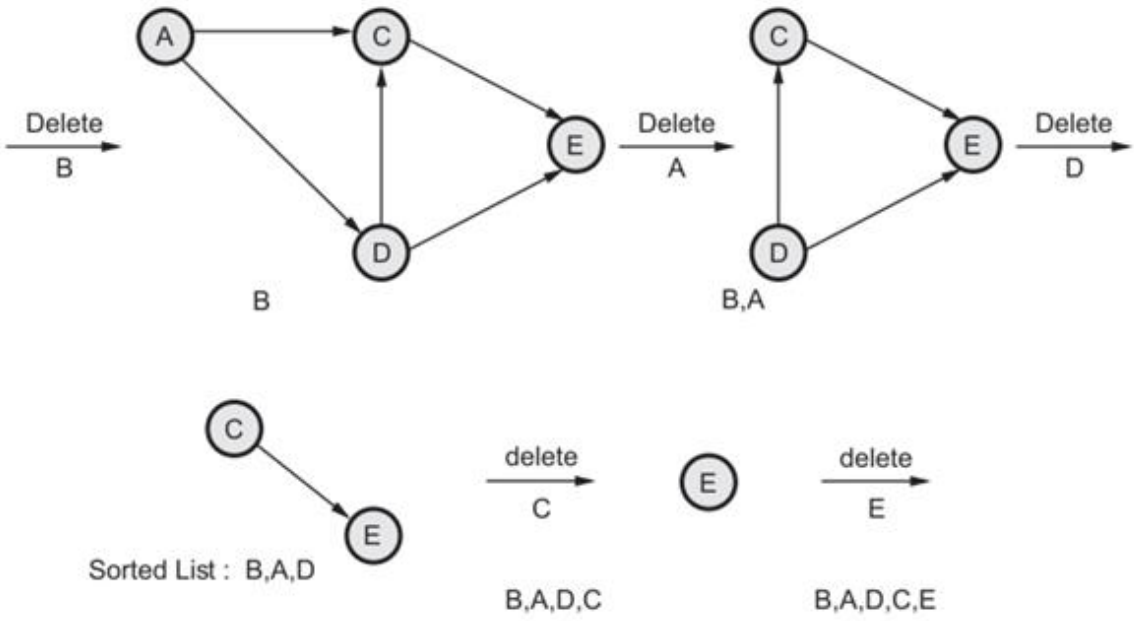


Fig. 3.10.2

Hence the list after topological sorting will be B, A, D, C, E.

4) Write an algorithm for breadth first traversal of a graph.

Step 1 - Define a **Queue** of size total number of vertices in the graph.

Step 2 - Select any vertex as **starting point** for traversal. Visit that vertex and insert it into the Queue.

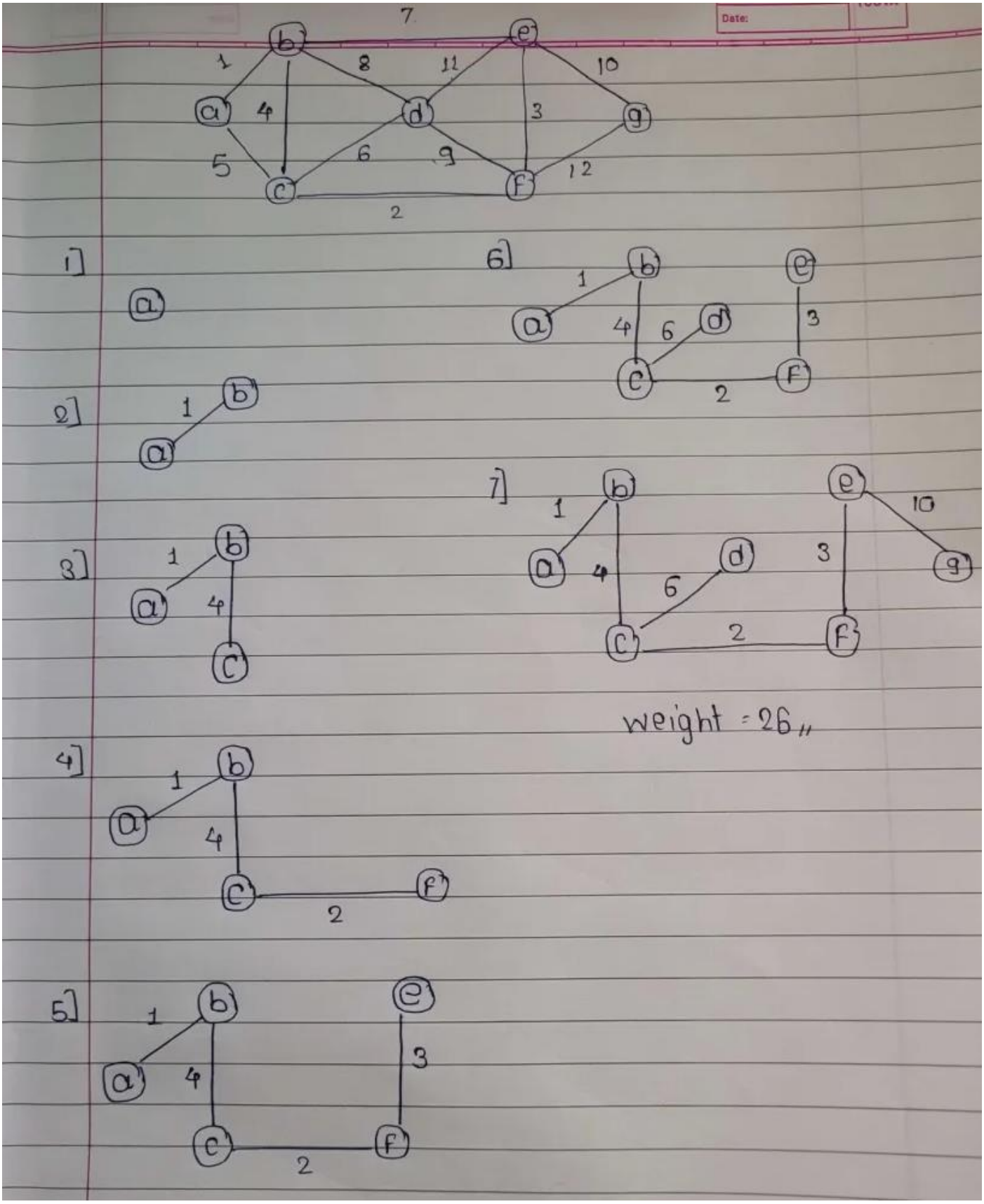
Step 3 - Visit all the **non-visited adjacent vertices** of the vertex which is at front of the Queue and insert them into the Queue.

Step 4 - When there is no new vertex to be visited from the vertex which is at front of the Queue then **delete that vertex**.

Step 5 - Repeat steps 3 and 4 until queue becomes empty.

Step 6 - When queue becomes empty, then produce final spanning tree by removing unused edges from the graph

5) Using Prim's Algorithm, find the cost of minimum spanning tree (MST) of the given graph starting from vertex 'a'



6) Define the following terms : i) Complete Graph ii) Connected Graph iii) Subgraph

3.1.4 A Complete Graph

An undirected graph, in which every vertex has an edge to all other vertices is called a complete graph.

A complete graph with N vertices has $\frac{N(N-1)}{2}$ edges.

Example of a complete graph is shown in Fig. 3.1.4.

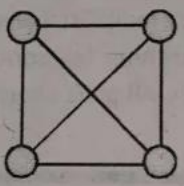


Fig. 3.1.4 : A complete graph

In a complete graph, these is an edge between every pair of vertices.

Number of edges (a pair of vertices) in a graph with n vertices is equal to combination of n elements, taking 2 at a time :

3.1.9 Connected Graph

A graph is said to be connected if there exists a path between every pair of vertices V_i and V_j.

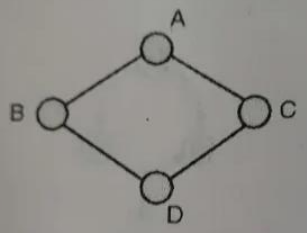
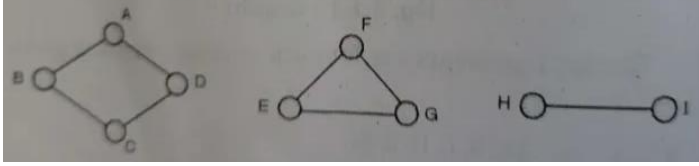
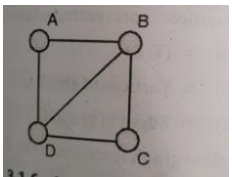


Fig. 3.1.7 : A connected graph





3.1.6 : A graph with cycles

3.1.10 Subgraph

A subgraph of G is a graph G_1 such that $V(G_1)$ is a subset of $V(G)$ and $E(G_1)$ is a subset of $E(G)$.

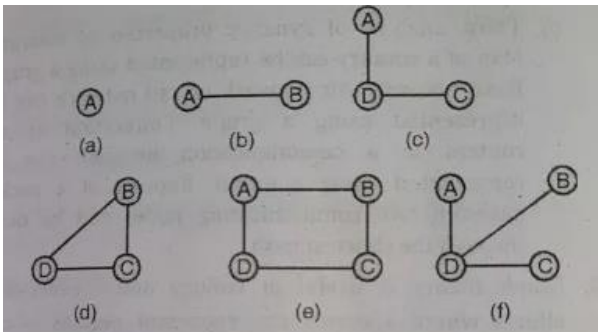


Fig. 3.1.9 : Some subgraphs of the graph of Fig. 3.1.6

7) Write Floyd Warshall Algorithm

1. Create a matrix A^0 of dimension n*n where n is the number of vertices. The row and the column are indexed as i and j respectively. i and j are the vertices of the graph.

Each cell A[i][j] is filled with the distance from the i^{th} vertex to the j^{th} vertex. If there is no path from i^{th} vertex to j^{th} vertex, the cell is left as infinity.

2. Now, create a matrix A^1 using matrix A^0. The elements in the first column and the first row are left as they are. The remaining cells are filled in the following way.

Let k be the intermediate vertex in the shortest path from source to destination. In this step, k is the first vertex. A[i][j] is filled with (A[i][k] + A[k][j]) if (A[i][j] > A[i][k] + A[k][j]).

That is, if the direct distance from the source to the destination is greater than the path through the vertex k, then the cell is filled with A[i][k] + A[k][j].

In this step, k is vertex 1. We calculate the distance from source vertex to destination vertex through this vertex k.

3. Similarly, A^2 is created using A^1. The elements in the second column and the second row are left as they are.

In this step, k is the second vertex (i.e. vertex 2). The remaining steps are the same as in

Calculate the distance from the source vertex to destination vertex through this vertex 2

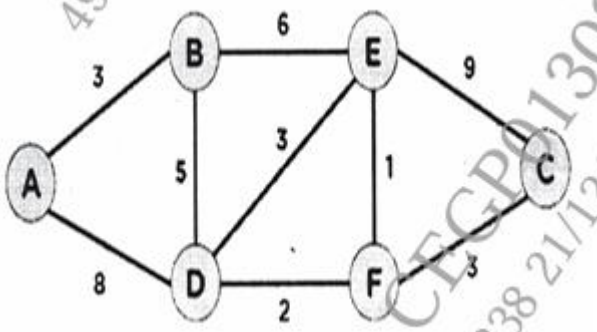
4. Similarly, A^3 and A^4 is also created.

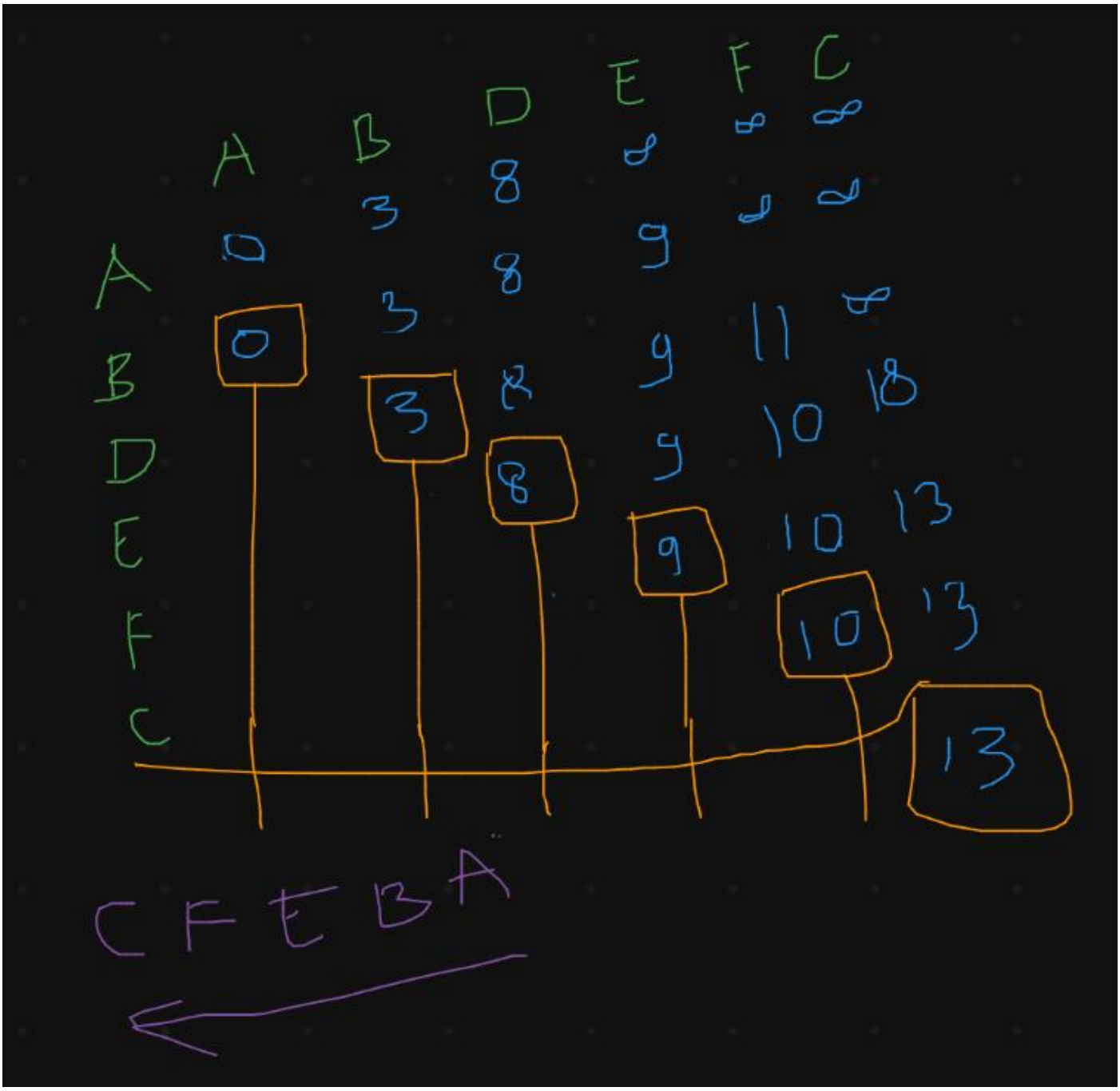
Calculate the distance from the source vertex to destination vertex through this vertex 3

Calculate the distance from the source vertex to destination vertex through this vertex 4

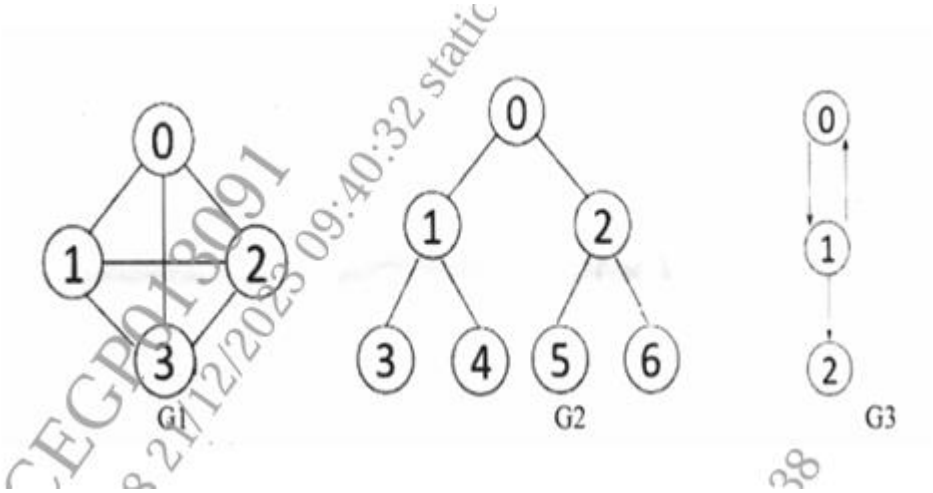
A4 gives the shortest path between each pair of vertices.

8) Apply Dijkstra's Algorithm for the graph given below, and find the shortest path from node A to node C.

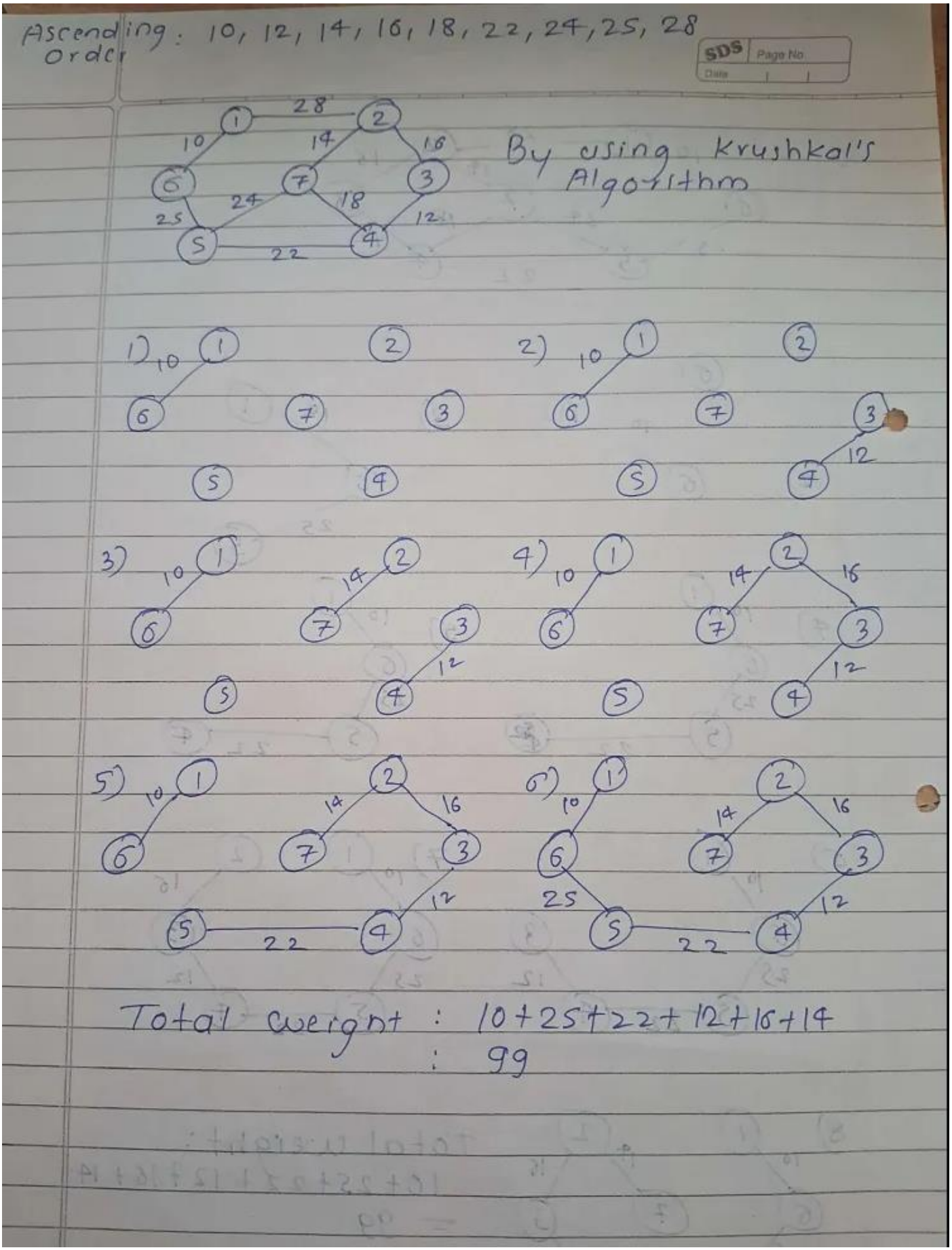




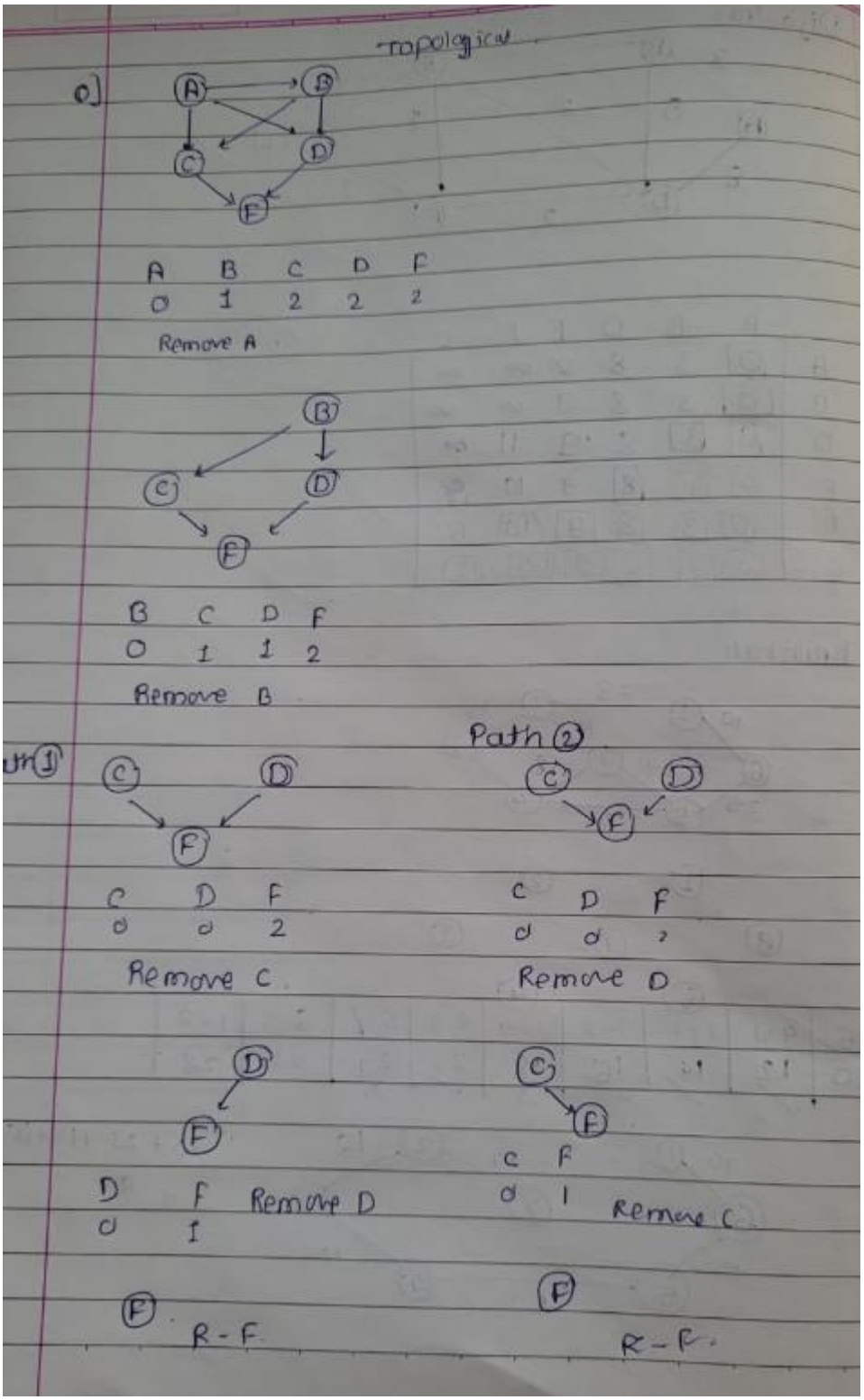
9) Define indegree & outdegree of a directed graph. Write degree for G1 & G2. Write indegree & outdegree of each vertex for G3 graph.



10) Construct the minimum spanning tree (MST) for the given graph using Kruskal's Algorithm.



11) Find the number of different topological orderings possible for the given graph



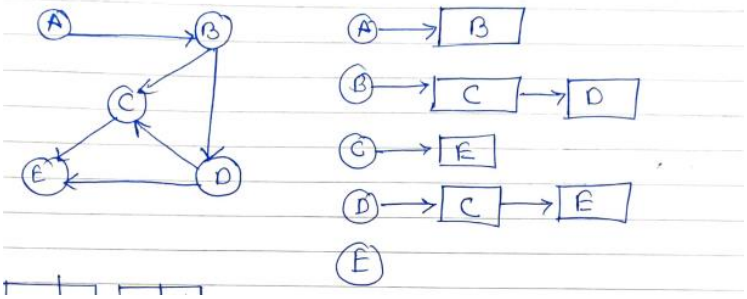
12) Elaborate following terminologies : i) Graph ii) Adjacency List iii) Adjacency Matrix

- A Graph is a non-linear data structure consisting of **nodes** and **edges**.
- The nodes are sometimes also referred to as **vertices** and the edges are lines or arcs that connect any two nodes in the graph.
- More formally a Graph can be defined as,

A graph is a pair of set **<V, E>**, where **V** is the set of vertices and **E** is the set of edges, connecting the pairs of vertices.

Adjacency List.

- In this representation, every vertex of graph contains list of its adjacency vertices.
- A linked representation is an adjacency list.
- You keep a list of neighbours for each vertex in the graph in this representation. It means that each vertex in the graph has a list of its neighbouring vertices.



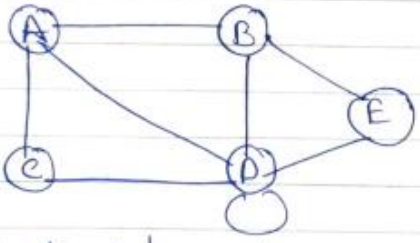
1 Adjacency Matrix :- In this representation, graph can be represented using a matrix of size total no. of vertices by total no. of vertices;

means if a graph with 4 vertices can be represented using a matrix of 4 × 4 size.

In this matrix, rows and columns both represent vertices.

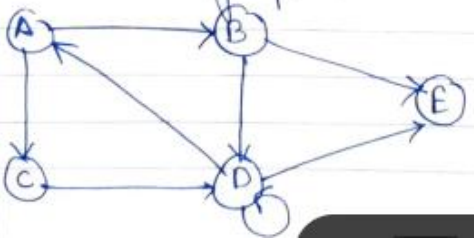
This matrix is filled with either 1 or 0. Here, 1 represent there is an edge from row vertex to column vertex and 0 represent there is no edge from row vertex to column vertex.

For undirected graph



	A	B	C	D	E
A	0	1	1	1	0
B	1	0	0	1	1
C	1	0	0	1	0
D	1	1	1	1	1
E	0	1	0	1	0

For directed graph



	A	B	C	D	E
A	0	1	1	0	0
B	0	0	0	1	1
C	0	0	0	1	0
D	1	0	0	1	1
E	0	0	0	0	0

13) Differentiate between tree and graph

Feature	Tree	Graph
Definition	A connected acyclic graph.	A collection of nodes (vertices) and edges (links).
Cycles	No cycles allowed (acyclic).	Cycles may or may not be present.
Connectivity	Always connected (one path between any two nodes).	May be connected or disconnected.
Edges	Has exactly `n-1` edges for `n` nodes.	Can have any number of edges (0 to `n(n-1)/2`).
Hierarchy	Hierarchical structure (parent-child).	No strict hierarchy; more general connections.
Direction	Usually directed (in rooted trees).	Can be directed or undirected.
Traversal	DFS, BFS used; simpler due to structure.	DFS, BFS used; can be more complex due to cycles.
Example	Family tree, file system hierarchy.	Social network, road map, web links.

14) Write pseudo code for Floyd-Warshall algorithm.

```
Input:
    A 2D array dist[][] representing the weight of the edge from vertex i to vertex j.
    - If there is no edge, set dist[i][j] = ∞
    - Set dist[i][i] = 0 for all i

Algorithm:

for k from 0 to V-1:                    // k is the intermediate vertex
    for i from 0 to V-1:                // i is the starting vertex
        for j from 0 to V-1:            // j is the ending vertex
            if dist[i][k] + dist[k][j] < dist[i][j]:
                dist[i][j] = dist[i][k] + dist[k][j]

Output:
    dist[][] will now contain the shortest distances between all pairs of vertices
```

Notes:

- `V` is the number of vertices in the graph.
- The triple loop checks and updates the shortest path from every vertex `i` to `j` using `k` as an intermediate step.

15) Write Prim's algorithm to find minimum spanning tree

Step 1: Keep a track of all the vertices that have been visited and added to the spanning tree.

Step 2: Initially remove all the parallel edges and loops.

Step 3: Choose a random vertex, and add it to the spanning tree. This becomes the root node.

Step 4: Add a new vertex, say x, such that

x is not in the already built spanning tree.

x is connected to the built spanning tree using minimum weight edge. (Thus, x can be adjacent to any of the nodes that have already been added in the spanning tree).

Adding x to the spanning tree should not form cycles.

Step 5: Repeat the Step 4, till all the vertices of the graph are added to the spanning tree.

Step 6: Print the total cost of the spanning tree.

16) Write the applications of : i) Graph ii) BFS iii) DFS

1) Applications of Graphs:

Graphs are widely used in various fields. Some key applications include:

Application Area	Description
Social Networks	Nodes represent people, edges represent relationships/friendships.
Computer Networks	Routers and devices as nodes, connections as edges.
Google Maps / GPS	Locations as nodes, roads as edges for route finding.
Web Crawling	Pages are nodes, hyperlinks are edges.
Dependency Graphs	Used in compilers, task scheduling (e.g., package installations).
Recommendation Systems	Products/users are nodes; preferences form edges.
AI / Game Development	Pathfinding algorithms on maps or grids (e.g., A*, Dijkstra).

2) Applications of BFS (Breadth-First Search):

BFS explores level by level and is useful where shortest path or minimal steps are needed.

Application	Description
Shortest Path in Unweighted Graphs	BFS finds shortest path by exploring all neighbors first.
Web Crawlers	Visit web pages level-wise from a source link.
Social Networking	Finding people within 'k' connections (friend suggestions).
Broadcasting in Networks	Data spread evenly like BFS across nodes.
Cycle Detection in Undirected Graphs	Can be used to detect loops/cycles.
AI / Puzzle Solving	State-space search like solving Rubik's cube or chess moves.

3) Applications of DFS (Depth-First Search):

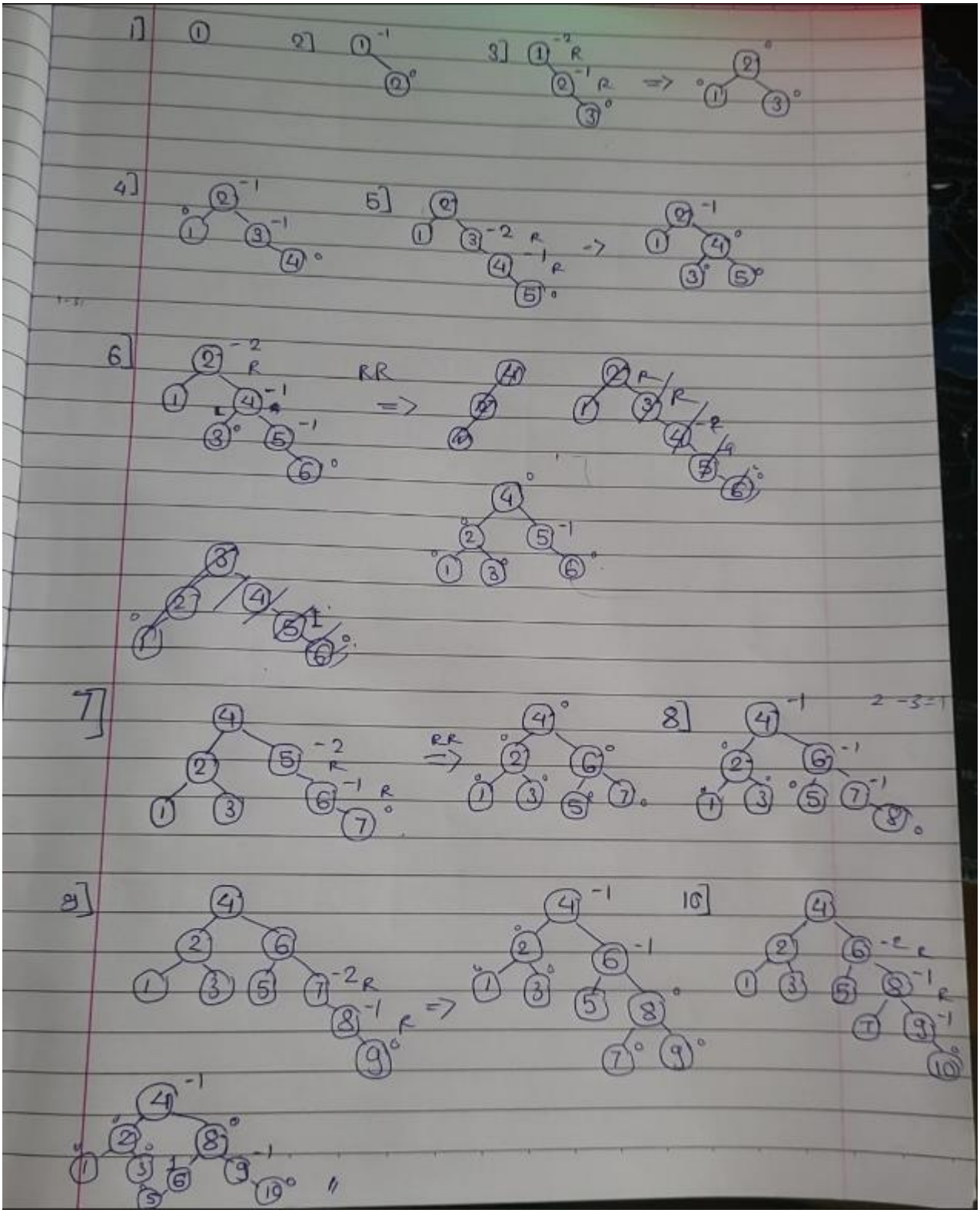
DFS goes deep into one path before backtracking, useful for complex traversal or backtracking.

Application	Description
Topological Sorting	For scheduling tasks with dependencies (like course prerequisites).
Cycle Detection in Directed Graphs	DFS is ideal for detecting cycles.
Maze Solving / Path Finding	DFS explores one full path before backtracking, useful in backtracking problems.
Connected Components	Identify all connected subgraphs in a graph.
Solving Puzzles / Games	Like Sudoku, N-Queens using recursive DFS.
Artificial Intelligence (Game Tree)	Simulate different game move paths.

DSA

UNIT 4

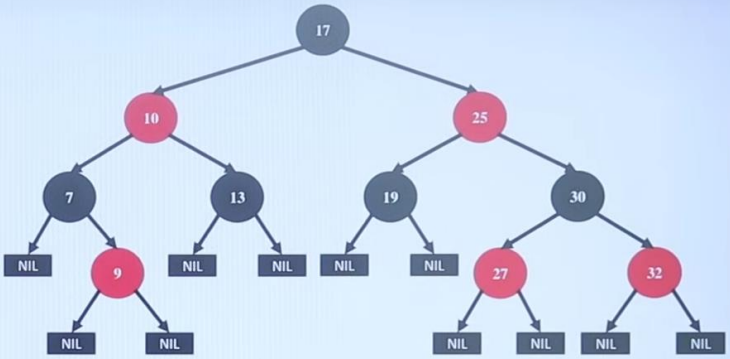
17) Construct an AVL Tree by inserting numbers from 1 to 8



18) Define Red Black tree. List its properties. Give example of it.

- A Red-Black Tree is a type of self-balancing binary search tree with specific properties that ensure it remains approximately balanced, providing efficient insertion, deletion, and lookup operations. It was introduced by Rudolf Bayer in 1972.

Properties of a Red-Black Tree



1. **Node Color**:
   Each node is either red or black.
2. **Root Property**:
   The root is always black.
3. **Red Node Property**:
   Red nodes cannot have red children (i.e., no two red nodes can be adjacent).
4. **Black Depth Property**:
   Every path from a node to its descendant leaves must have the same number of black nodes.
5. **Leaf Nodes**:
   All leaf nodes (NIL nodes) are black and do not store any data. They are placeholders used to maintain the tree's structure.

19) Write functions for RR and RL rotation with respect to AVL tree.

🔁 RR Rotation (Right-Right Case) – *Single Left Rotation*

When it happens:

- A new node is inserted into the **right subtree** of the **right child** of an unbalanced node.
- This causes the balance factor of the node to become **-2**, and the balance factor of its right child is **≤ 0**.

Logic:

1. Identify the unbalanced node (let's call it `A`).
2. Let `B` be `A`'s right child.
3. Perform a **single left rotation**:
   - `A` becomes the **left child** of `B`.
   - The **left subtree of** `B` (if any) becomes the **right subtree of** `A`.
4. Update the heights of the affected nodes.

Goal: Shift the heavier subtree (right-right chain) up to restore balance.

🔁 RL Rotation (Right-Left Case) – *Double Rotation: Right then Left*

When it happens:

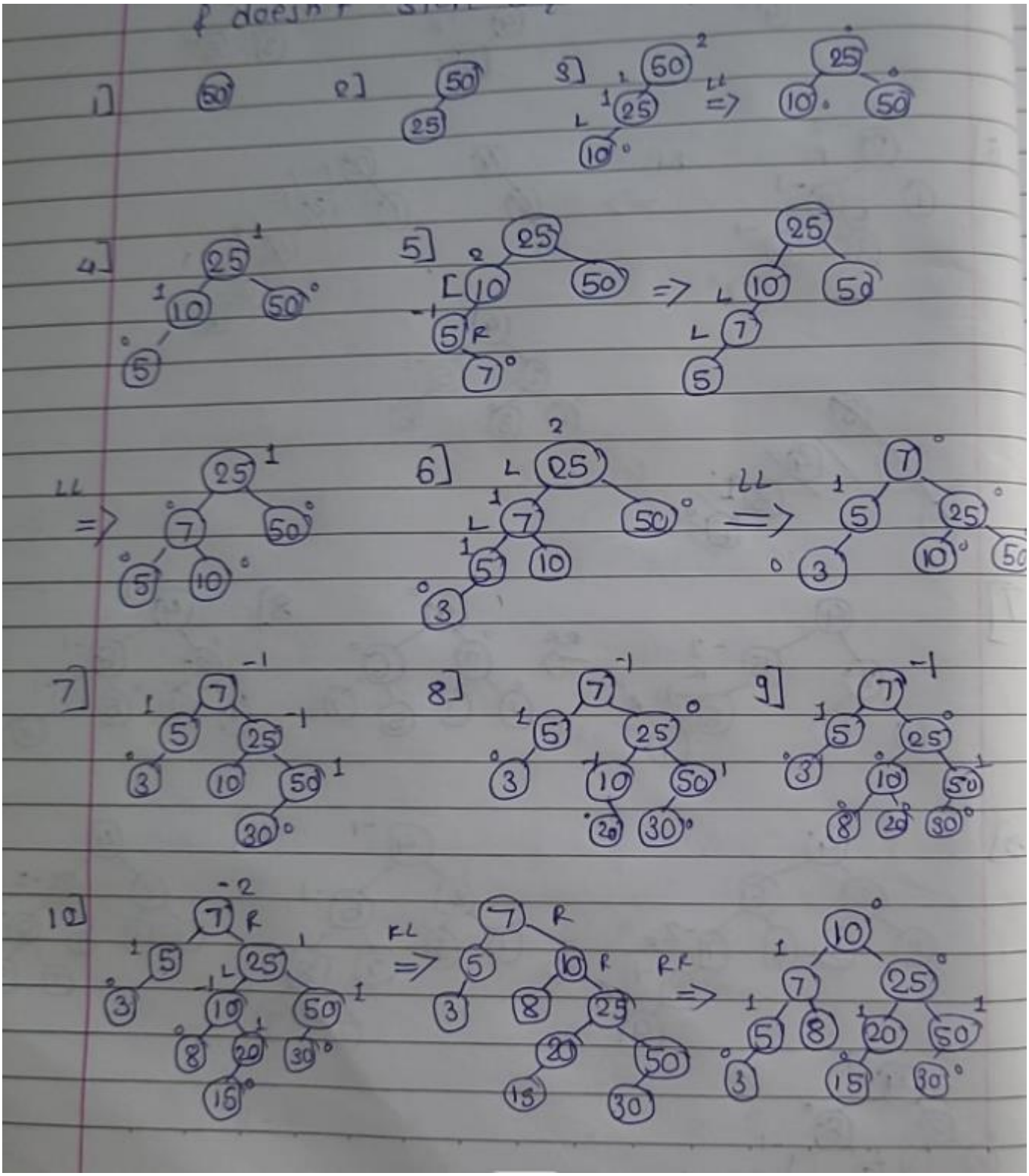
- A new node is inserted into the **left subtree** of the **right child** of an unbalanced node.
- This causes the balance factor of the node to be **-2**, and the balance factor of its right child is **> 0**.

Logic:

1. Identify the unbalanced node (`A`).
2. Let `B` be `A`'s right child, and `C` be `B`'s left child.
3. First, perform a **right rotation** on `B`:
   - `C` becomes the new right child of `A`.
4. Then perform a **left rotation** on `A`:
   - `C` becomes the new root of this subtree.
   - `A` becomes `C`'s left child, and `B` becomes `C`'s right child.
5. Update the heights of the affected nodes.

Goal: Break the zig-zag pattern and rebalance by lifting the middle node `C` up.

20) Construct an AVL Tree for following data : 50, 25, 10, 5, 7, 3, 30, 20, 8, 15

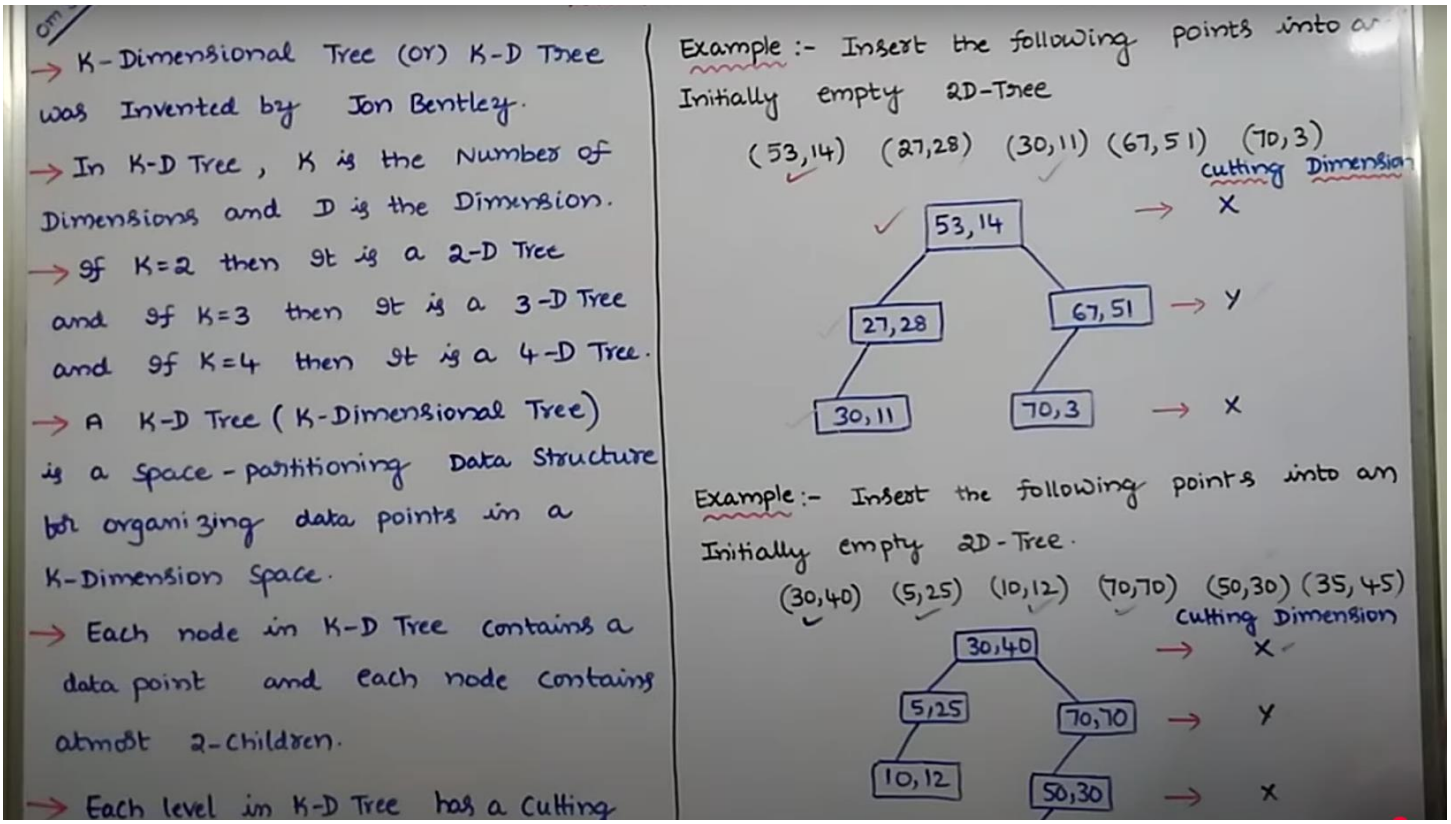


21) Explain with example K dimensional tree.

A **K-dimensional (K-d) tree** is a **binary search tree** used to organize points in a **K-dimensional space**. It is commonly used for multidimensional searches like range queries and nearest neighbor searches.

✅ Core Idea

- Each node represents a point in K-dimensional space.
- The tree cycles through the dimensions to split the space:
  - At depth `d`, split on dimension `d % K`.
- Left subtree contains points **less than** the node on that dimension.
- Right subtree contains points **greater than or equal to** the node on that dimension.



22) Explain static and dynamic tree tables with suitable example.

Representation of Symbol Table

Symbol table can be represented by

* Static Tree table
* Dynamic Tree table

Static Symbol table are in general used to store fixed amount of information whereas the dynamic Symbol tables are used for storing the information in dynamic form.

* Static tree tables :- when Symbol are known in advance and no insertion and deletion is allowed, it is called Static tree table
* Example of this type of table is a reserved word table in a compiler.
* Dynamic tree table is used when Symbols are not known in advance but are inserted as they come and deleted if not required.

Example of dynamic Symbol table : An AVL tree is Possible to implement with the help of Static Symbol table.

Example of Static Symbol table : Optimal Binary Search tree (OBST), Huffman coding is Possible to implement with the help of Static Symbol table.

This can be represented as a **static tree table** because:

- The hierarchy is known beforehand.
- No frequent updates.
- Used mostly for display purposes.

📄 Static Table Data:

ID	Name	ParentID
1	CEO	NULL
2	CTO	1
3	Dev Manager	2
4	QA Manager	2
5	CFO	1
6	Accountant	5

🌳 What Are Tree Tables?

Tree tables are hierarchical data structures used to represent data in a tree-like format—commonly in databases, UI tables, and memory structures. They allow parent-child relationships between rows (nodes).

📌 Static Tree Table

🔷 Definition:

A **static tree table** has a **fixed structure** defined at compile-time or load-time. The relationships and data are **not meant to change** during runtime. These are typically **read-only** structures.

✅ Example:

Suppose you have a **company org chart** that doesn't change frequently:

objectivec Copy Edit

```
CEO
├── CTO
│   ├── Dev Manager
│   └── QA Manager
└── CFO
    └── Accountant
```

🔄 Dynamic Tree Table

🔷 Definition:

A **dynamic tree table** supports **runtime modifications** such as insertion, deletion, and restructuring of nodes. It's used where hierarchy **frequently changes**, like in **file systems** or **project task managers**.

✅ Example:

A file system directory:

arduino Copy Edit

```
/home
├── user1
│   └── docs
└── user2
    └── photos
```

Here, users can:

- Add/delete folders.
- Move items between directories.
- Dynamically expand/collapse subtrees.

📄 Dynamic Table Data (updatable in real-time):

ID	Name	ParentID
1	home	NULL
2	user1	1
3	docs	2
4	user2	1
5	photos	4

If a new folder `music` is added under `user2`, it updates dynamically:

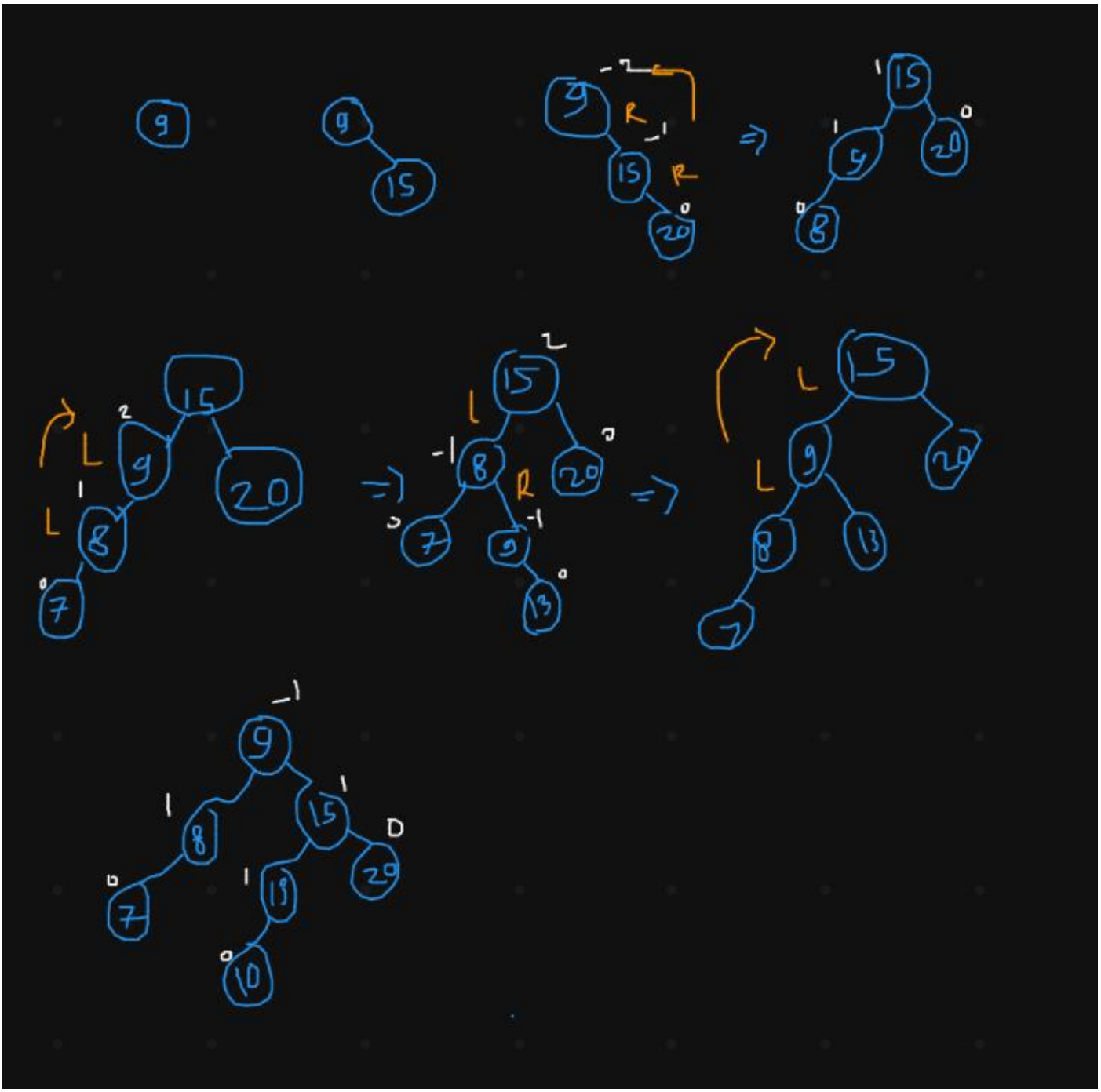
sql Copy Edit

```sql
INSERT INTO tree_table VALUES (6, 'music', 4);
```

🔍 Key Differences

Feature	Static Tree Table	Dynamic Tree Table
Structure	Fixed	Modifiable at runtime
Use case	Org charts, category trees	File explorers, task managers
Performance	Faster (no updates)	Slower (due to updates)

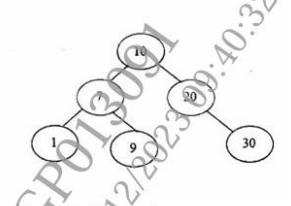
23) Construct AVL tree for insertion of following data 9, 15, 20, 8, 7, 13, 10.



24) Draw splay tree after

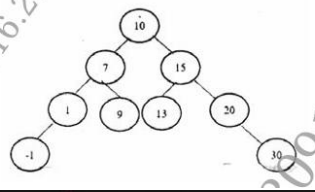
i) Zig rotation

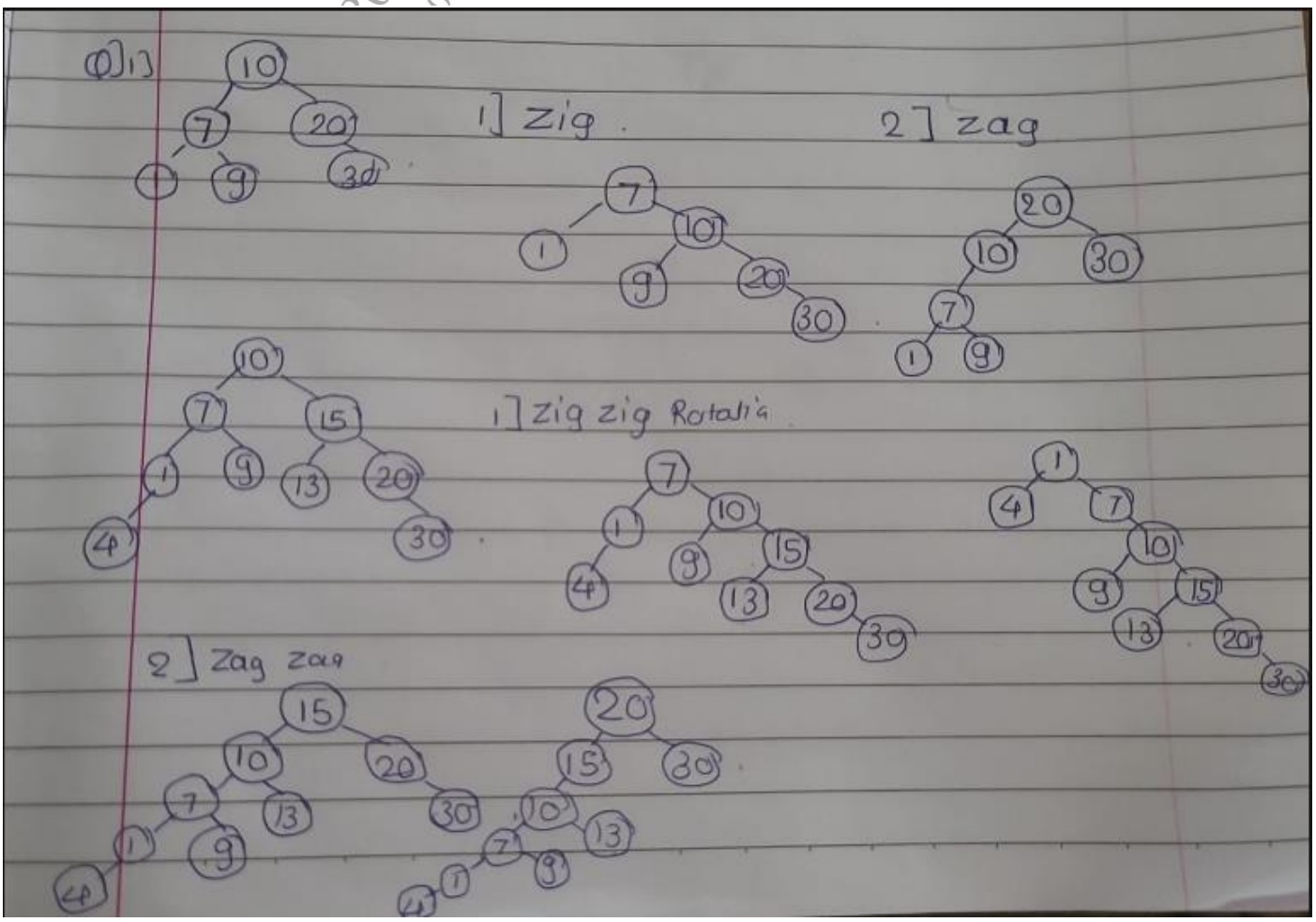
ii) Zag rotation for follwoing tree-



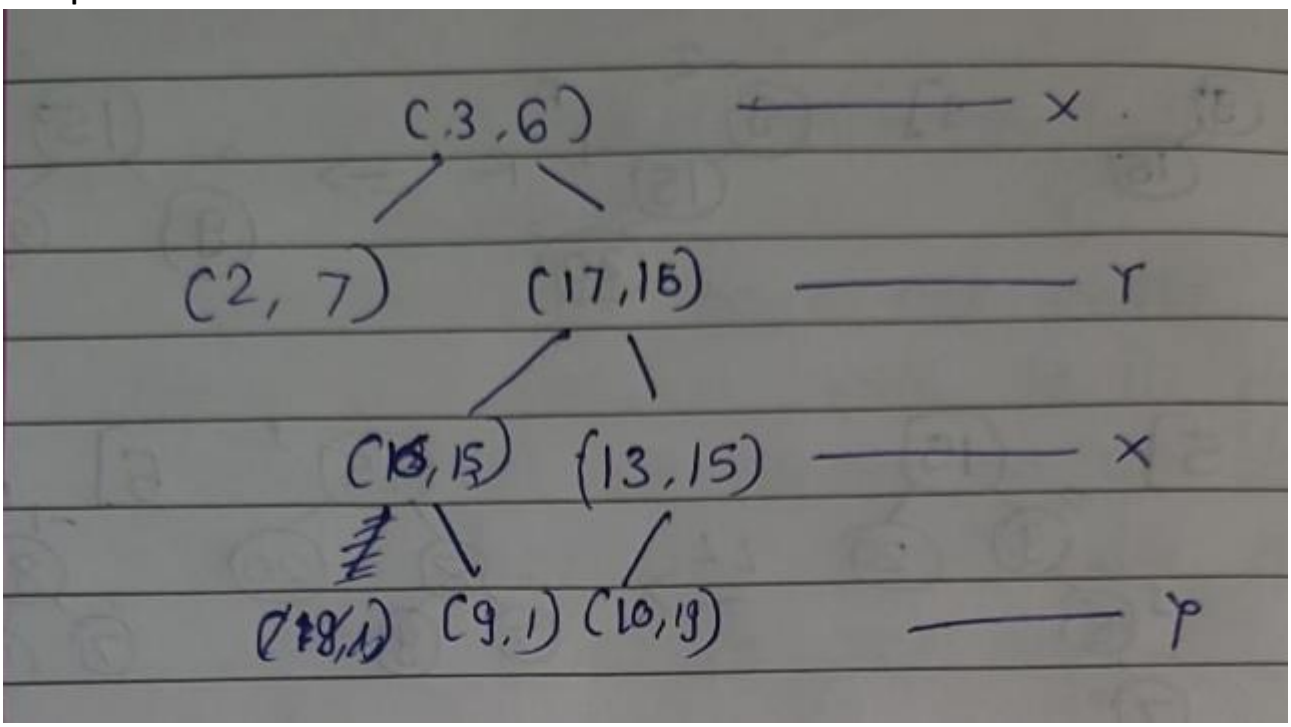
iii) Zig Zig Rotation

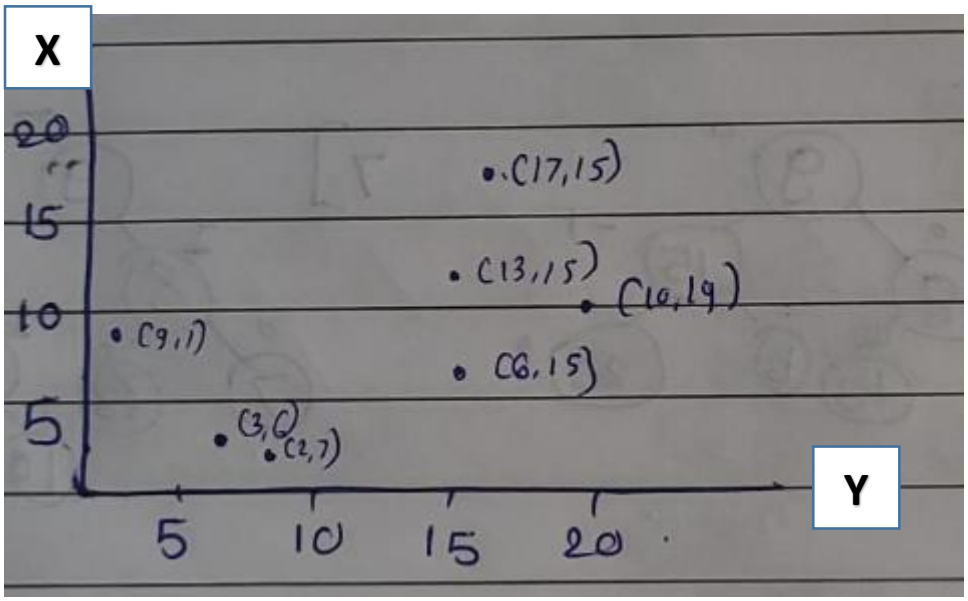
iv) Zag Zag Rotations for following tree-



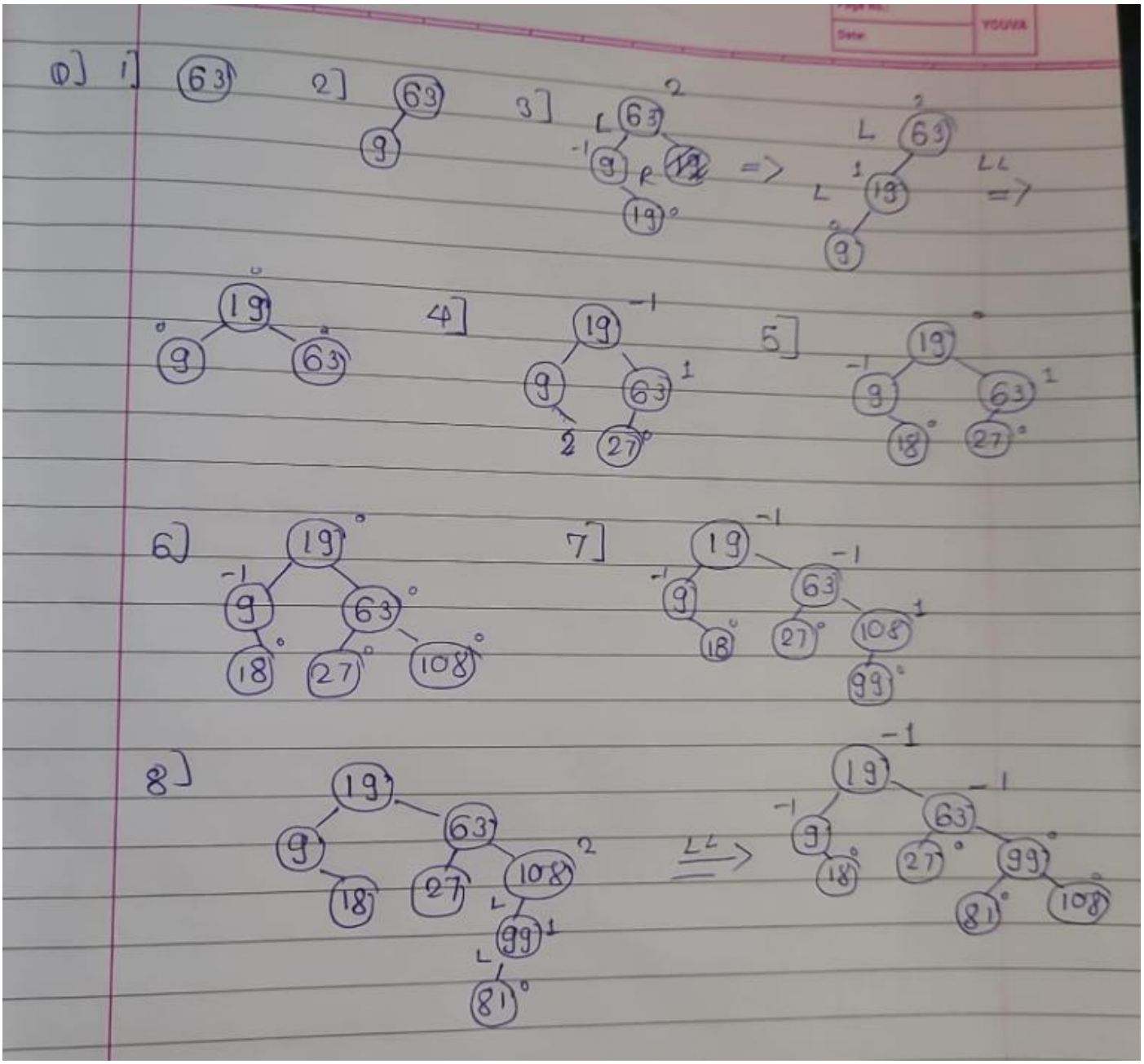


25) Create 2D tree for following data: (3, 6), (17, 15), (13, 15), (6, 12), (9, 1) (2,7), (10, 19). Also plot all the points in XY plane.





26) Construct AVL tree for insertion of following data: 63, 9, 19, 27, 18, 108, 99, 81



27) Write the functions for split & skew operations in AA tree.

🔁 `Skew()` – Logic

Goal:
Remove a **left horizontal link**, which violates the AA tree invariant that **left children must be at a lower level than their parent**.

When to apply:
If a node's **left child exists** and its **level is equal** to the node's level.

What it does:

- Perform a **right rotation** between the node and its left child.
- This fixes the issue by making the former left child the new parent and decreasing the depth of the left link.

Effect:
Balances the tree by ensuring all horizontal (same-level) links go **to the right** only.

🔁 `Split()` – Logic

Goal:
Break **two consecutive right horizontal links**, which would make the tree unbalanced and deeper on the right side.

When to apply:
If a node has a **right child**, and that right child also has a **right child**, and both are at the **same level** as the node.

What it does:

- Perform a **left rotation** between the node and its right child.
- Increase the level of the new parent (the rotated node) by 1.

Effect:
Prevents the tree from degenerating into a right-leaning linked list. Maintains logarithmic depth.

28) Explain following terms w.r.t. symbol table : i) Insert & lookup operations ii) Advantages iii) Disadvantages

Symbol Table – Overview

A **symbol table** is a data structure used by **compilers or interpreters** to store **information about identifiers** (like variable names, function names, objects, etc.) appearing in the source code.

---

i) Insert Operation

Purpose:
To **add a new identifier** (symbol) to the symbol table with relevant attributes like type, scope, memory location, etc.

When it's used:
During **declaration** of variables, functions, classes, etc.

Example:
If the compiler encounters `int x;`
→ It inserts an entry: `{ name: "x", type: "int", scope: "local/global", memory_location: ... }`

ii) Lookup Operation

Purpose:
To **retrieve information** about a symbol when it's **referenced** in code.

When it's used:
During **type checking, code generation, or semantic analysis**, the compiler looks up identifiers to verify their declarations.

Example:
If code has `x = x + 1;`
→ Lookup checks whether `x` was declared and retrieves its type or scope to ensure correctness.

---

iii) Advantages of Symbol Tables

1. ✅ **Efficient identifier management** – Keeps track of variables, functions, scopes.
2. ✅ **Enables semantic checks** – Ensures variables are declared before use.
3. ✅ **Supports nested scopes** – Helps in managing local/global variables.
4. ✅ **Optimizes memory usage** – Manages memory locations for variables.
5. ✅ **Crucial for code generation** – Helps map identifiers to actual memory addresses.

iv) Disadvantages of Symbol Tables

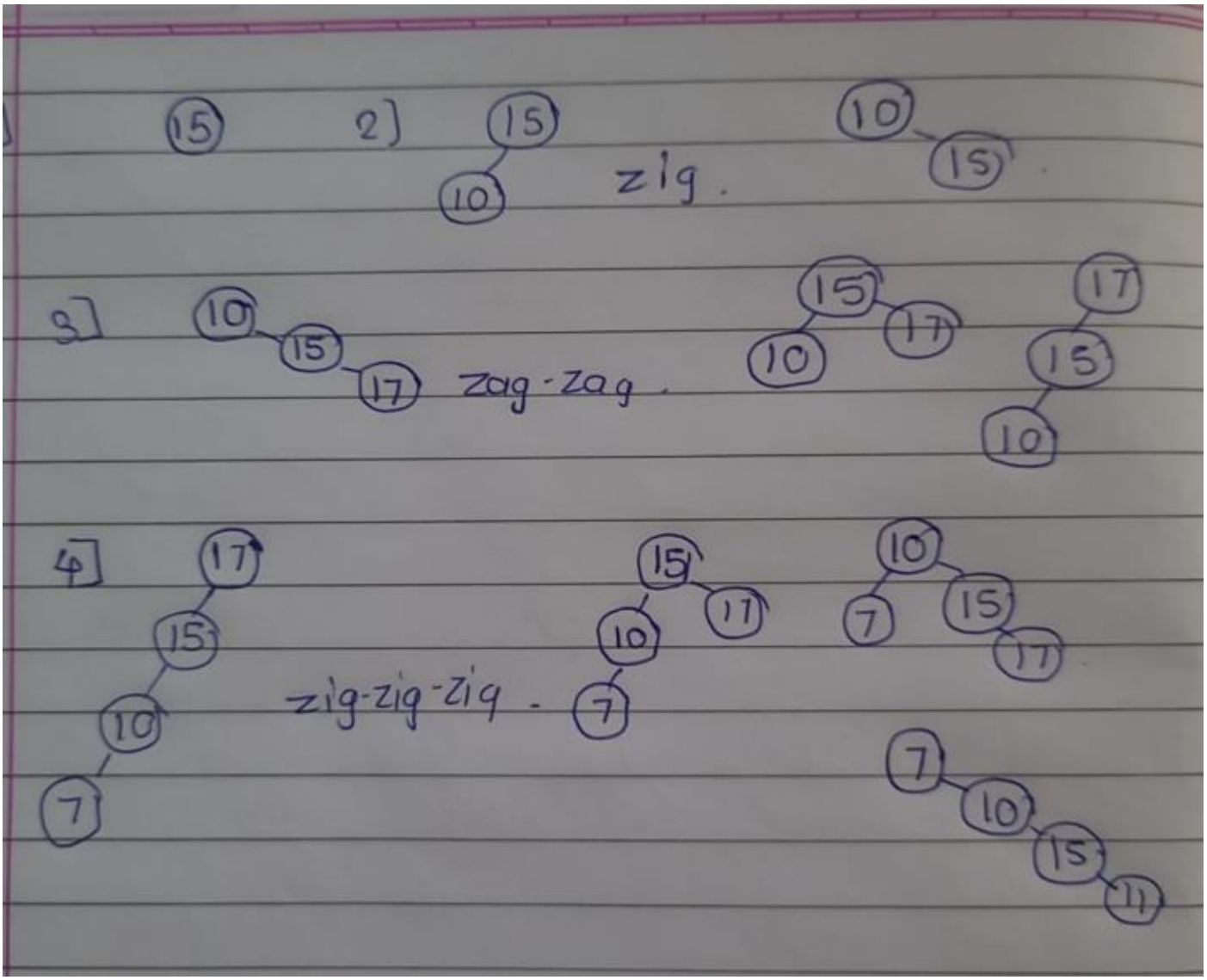
1. ❌ **Performance overhead** – Especially for large programs or with poor data structure choices.
2. ❌ **Complexity in scope handling** – Managing nested scopes can require extra logic (e.g., stack of tables).
3. ❌ **Memory consumption** – Symbol tables can grow large, consuming memory if not managed properly.
4. ❌ **Implementation difficulty** – Requires careful planning of structure (e.g., using hash tables, trees, etc.).

📌 Common Data Structures Used

- **Hash Tables** – Fast lookup and insertion (most common).
- **Binary Search Trees (BSTs)** – Useful for sorted order.
- **Tries** – Efficient for storing keywords and identifiers.
- **Stacks of Tables** – For handling scopes in block-structured languages.

29) Construct an AVL tree having the following elements : H, I, J, B, A, E, C, F, D, G

30) Insert 15, 10, 17, 7 in splay tree.



31) What is the need of AA tree? List the five invariants that AA tree must satisfy.

✅ What is the Need for an AA Tree?

The **AA Tree** is a type of self-balancing binary search tree designed to **simplify the implementation** of balancing logic while maintaining good performance. It is primarily used to:

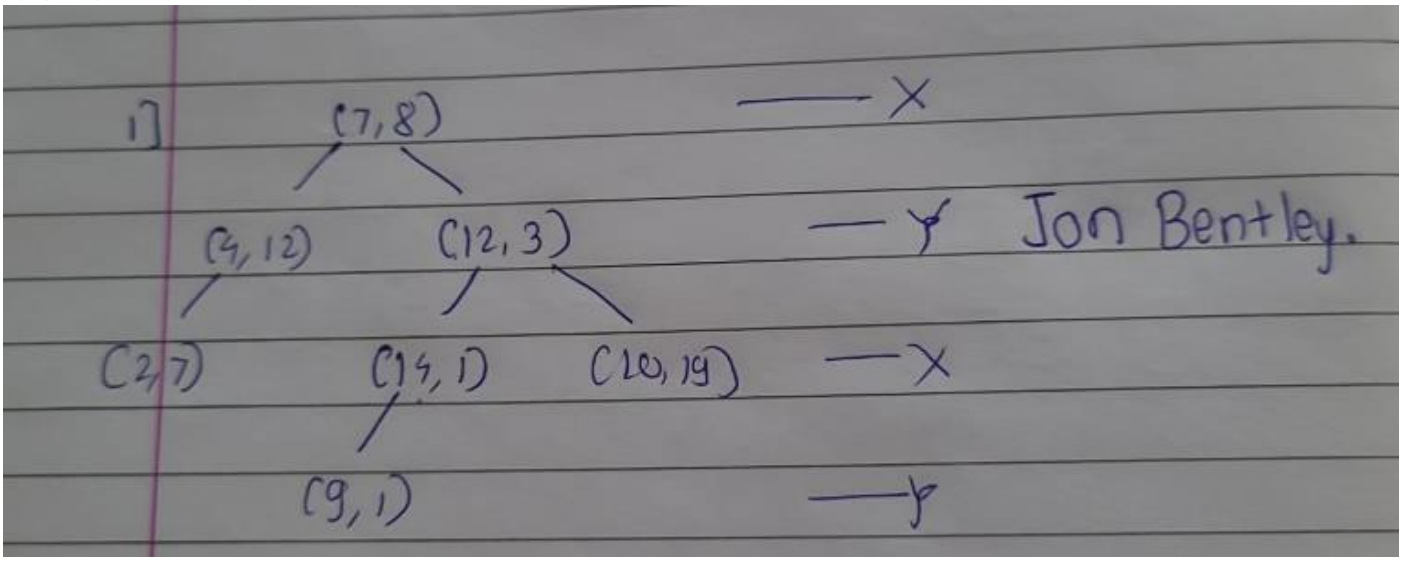
1. ✅ **Maintain a balanced BST** with simpler logic than red-black trees.
2. ✅ Ensure **logarithmic time** for insertions, deletions, and lookups.
3. ✅ Provide a data structure that is **easier to code, test, and debug** compared to AVL or red-black trees.
4. ✅ Offer a good **balance between simplicity and performance** in practice.

📏 Five Invariants of an AA Tree

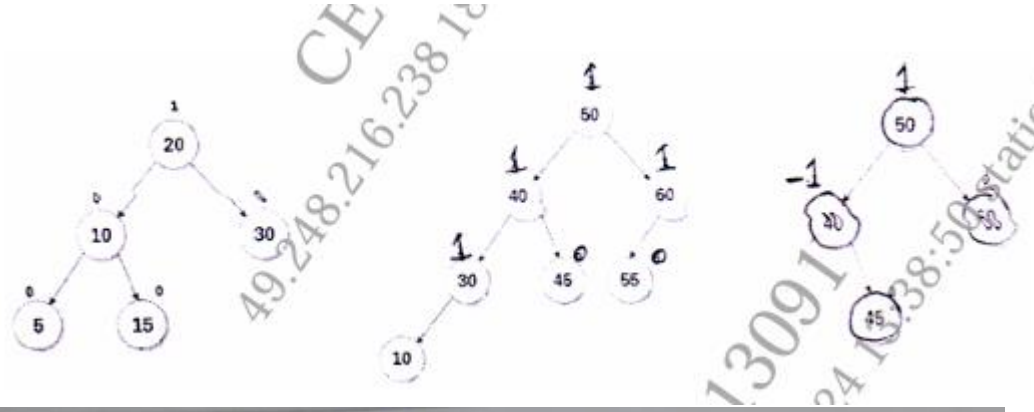
To preserve its structure and ensure balance, an AA tree must satisfy these **five invariants**:

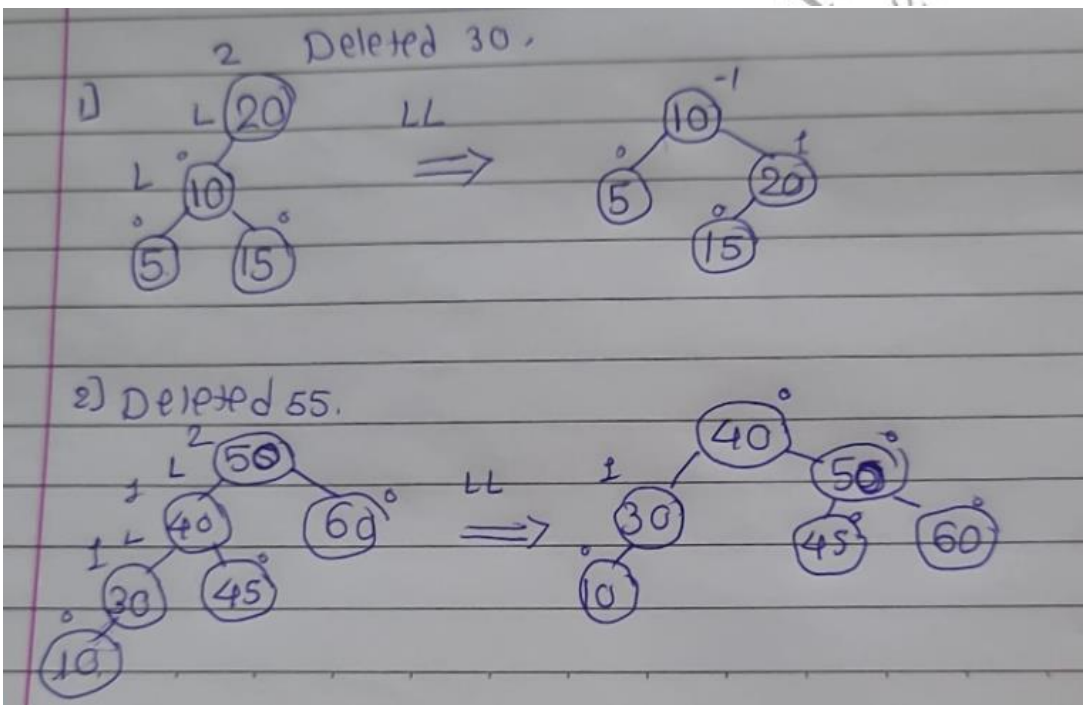
1. **Invariant 1 – Left Children Rule**:
   The **left child of a node must have a level strictly less** than its parent.
   (Prevents left horizontal links.)
2. **Invariant 2 – Right Children Rule**:
   The **right child of a node may have the same level** or one level less than the parent.
   (Allows right horizontal links, like red links in red-black trees.)
3. **Invariant 3 – No Right-Right Chain at Same Level**:
   A node **cannot have a right child and right grandchild both at the same level**.
   (Prevents long right chains, preserving balance.)
4. **Invariant 4 – Level of Leaf Nodes**:
   All **leaf nodes must have level 1**.
5. **Invariant 5 – Node with Level > 1 Must Have Two Children**:
   Any node with **level greater than 1 must have both left and right children** (non-null).

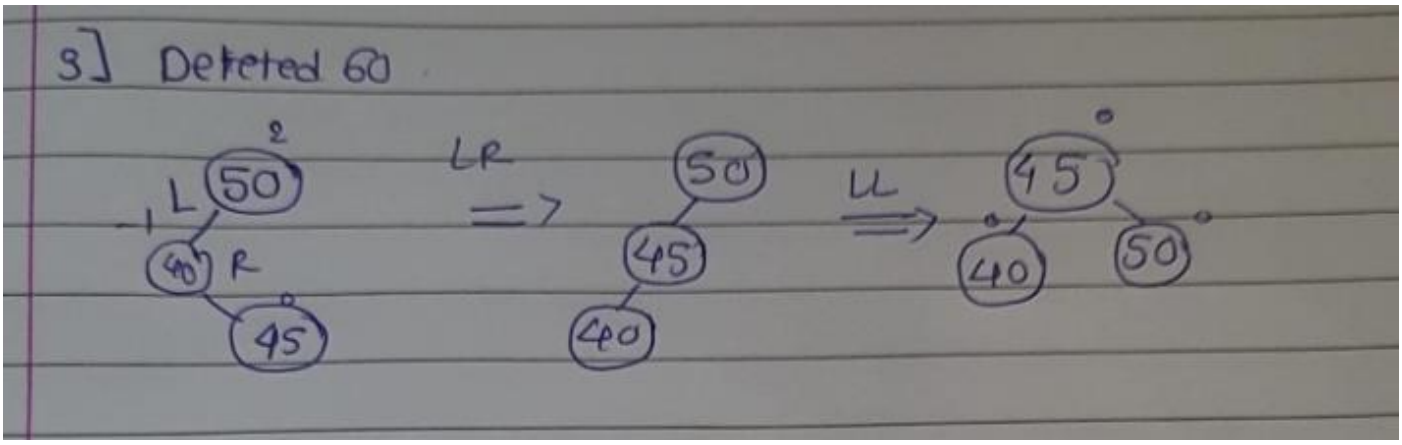
32) Who developed K-D tree? What is the purpose of K-O tree? Insert step by step (7, 8), (12, 3), (14, 1), (4, 12), (9, 1), (2, 7) and (10, 19) into K-D tree.



33) Show the balanced AVL tree after deletion of mentioned node : i) Delete 30 ii) Delete 55 iii) Delete 60



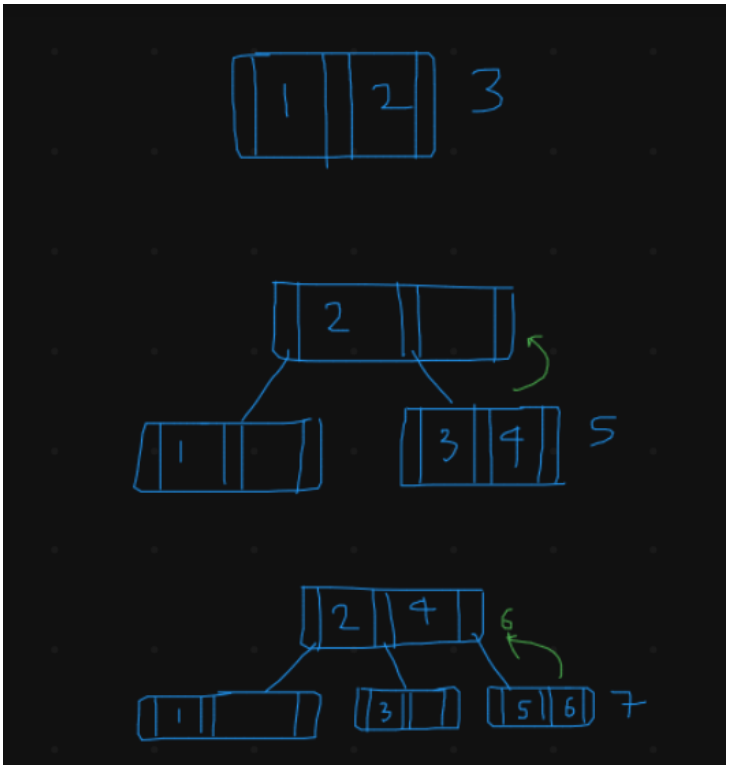


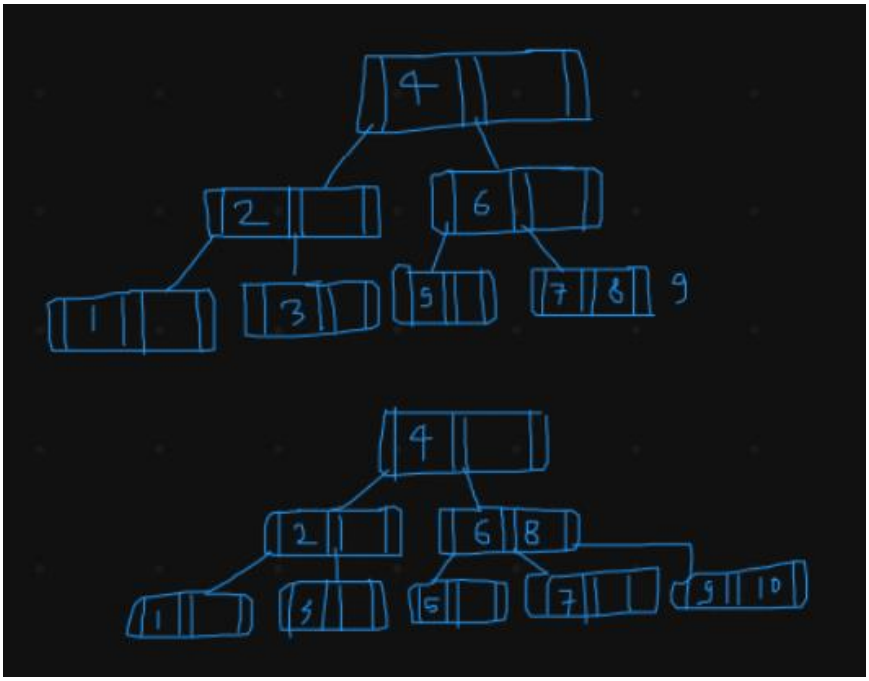


DSA

UNIT 5

34) Construct a B-Tree of order 3 by inserting numbers from 1 to 10.





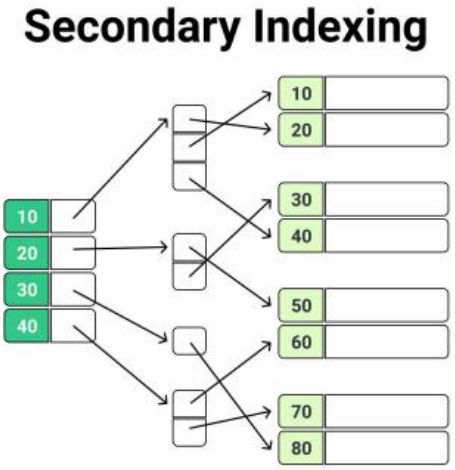
35) Explain following primary index, Secondary index, Sparse index and Dense index with example.

Primary Indexing

- Primary Index is an **ordered file which has fixed length** size with two fields.
- The first field is the same a **primary key** and second field is pointed to that **specific data block**.
- In the primary index, there is always one to one relationship between the entries in the index table.
- Primary Indexing is further divided into two types.
  - Dense Index
  - Sparse Index

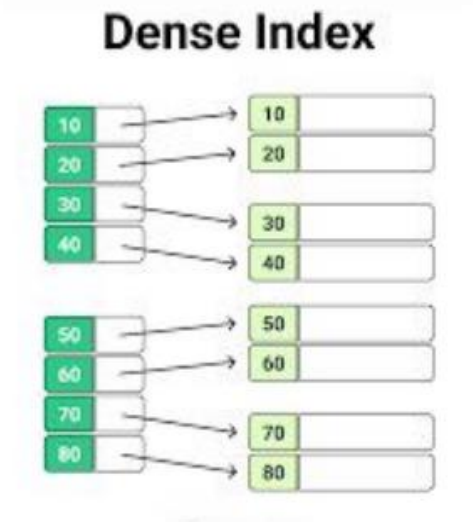
Secondary Indexing

- The secondary index can be generated by ***a field which has a unique value for each record***.
- It is also known as a ***non-clustering index.***
- This two-level database indexing technique is ***used to reduce the mapping size of the first level.***
- For the first level, a large range of numbers is selected, because of this mapping size always remains small.



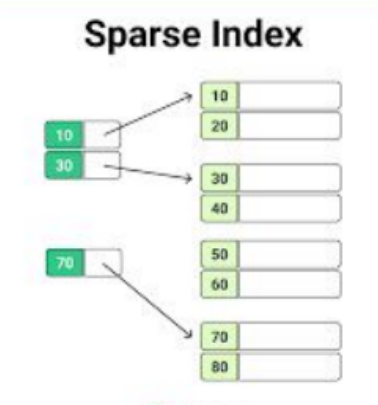
Dense Index

- In a dense index, a ***record is created for every search key*** valued in the database.
- Dense indexing helps you to ***search faster but needs more space*** to store index records.
- In dense indexing, records contain search key value and ***points to the real record on the disk***.



Sparse Index

- The sparse index is an index record that ***appears for only some of the values*** in the file.
- Sparse Index helps you to ***resolve the issues of dense indexing.***
- In sparse indexing technique, ***a range of index columns stores the same data block address***, and when data needs to be retrieved, this block address will be fetched.
- Sparse indexing method ***stores index records for only some search key values***.
- It needs ***less space, less maintenance overhead for insertion, and deletions*** but it is ***slower compared to the dense index for locating records.***



36) Construct a B Tree of order 5 with the following data : D H Z K B P Q E A S W T C L N Y M

37) What is trie tree? Explain insert and search operation on it.

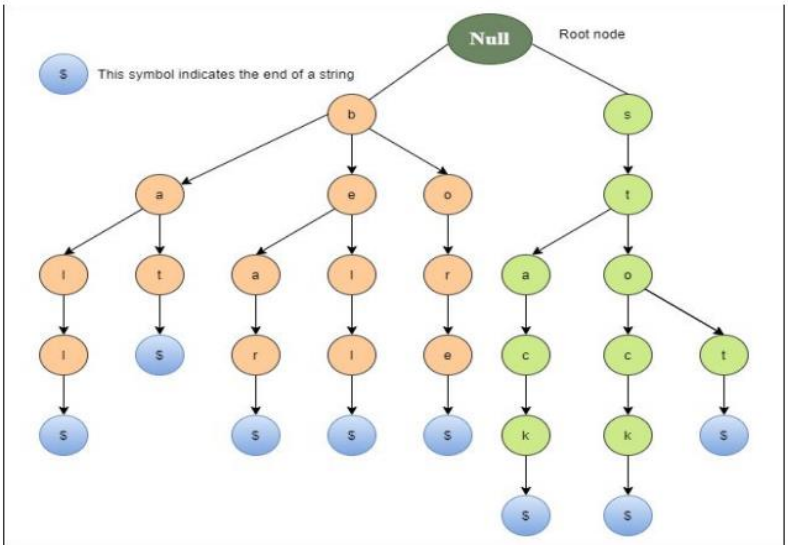
Trie Tree

- The word "**Trie**" is taken from the word "**retrieval**"
- Trie is a sorted tree-based data-structure that stores the set of strings.
- It has the number of pointers equal to the number of characters of the alphabet in each node.
- Trie is also known as the digital tree or prefix tree.

- Properties of the Trie for a set of the string:

1. The root node of the trie always represents the null node.
2. Each child of nodes is sorted alphabetically.
3. Each node can have a maximum of **26** children (A to Z).
4. Each node (except the root) can store one letter of the alphabet.

bell
bear
bore
bat
ball
stop
stock
stack



- Advantages of Trie

1. Faster insertion and search for the string than hash tables and binary search trees.
2. It provides an alphabetical filter of entries.

- Disadvantages of Trie

1. It requires more memory to store the strings.
2. It is slower than the hash table.

🔁 Insert Operation (Logic):

1. Start at the root.
2. For each character in the word:
    - If the character **doesn't exist** as a child of the current node, **create a new node**.
    - Move to that child node.
3. After inserting the last character, mark the node with `isEndOfWord = true`.

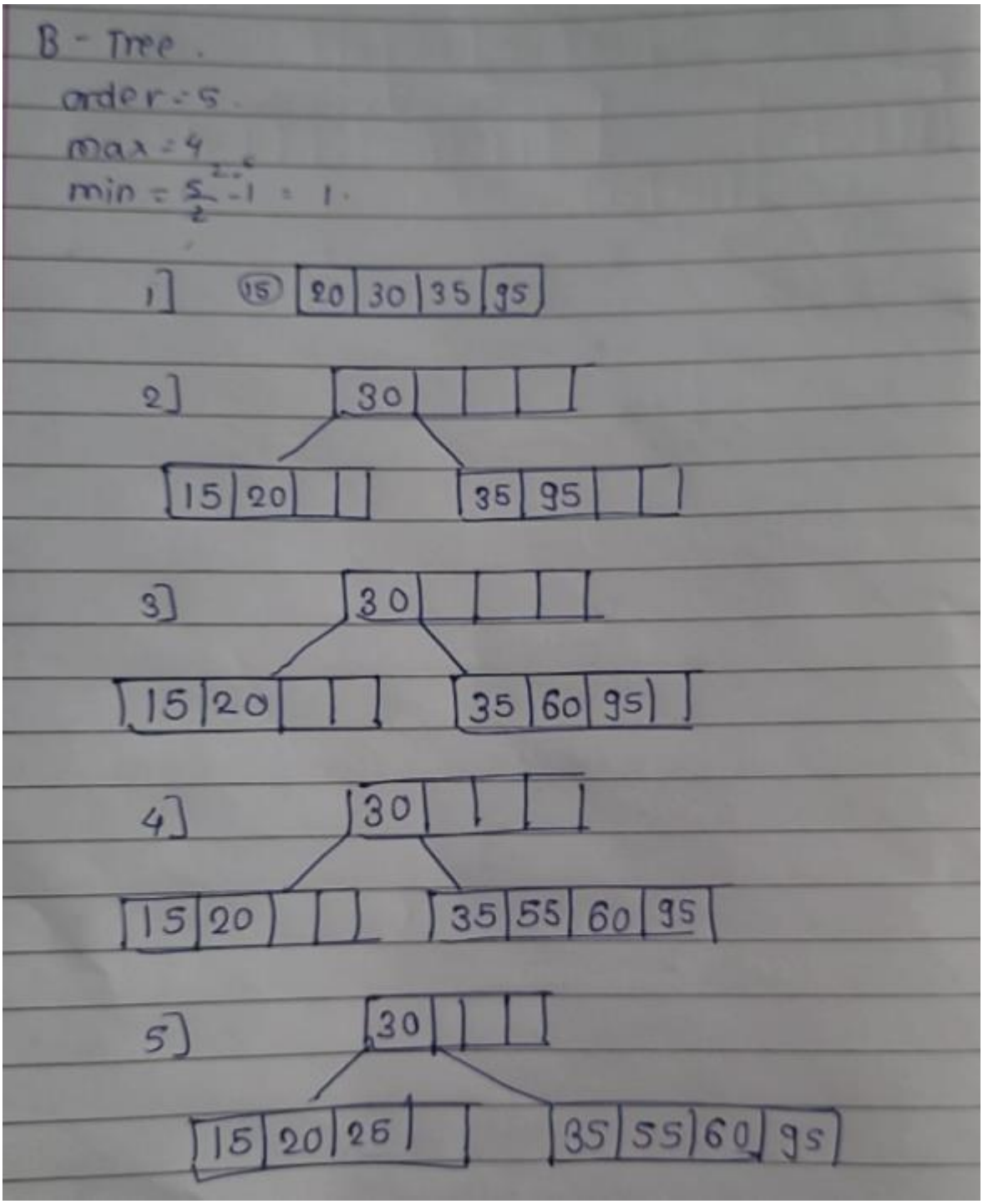
> Example: Insert `"top"`

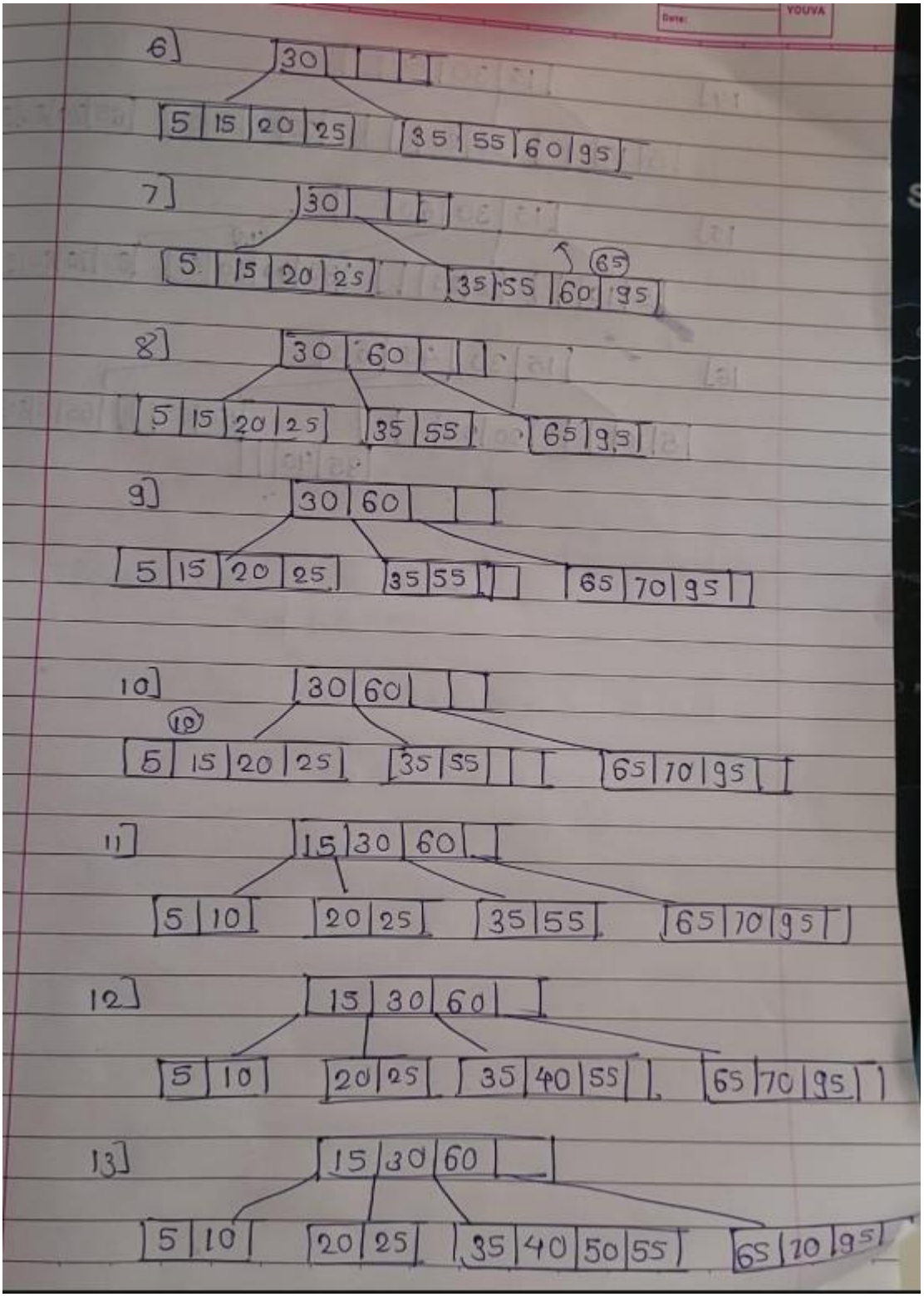
- 't' → new node
- 'o' → new node
- 'p' → new node + mark as end of word

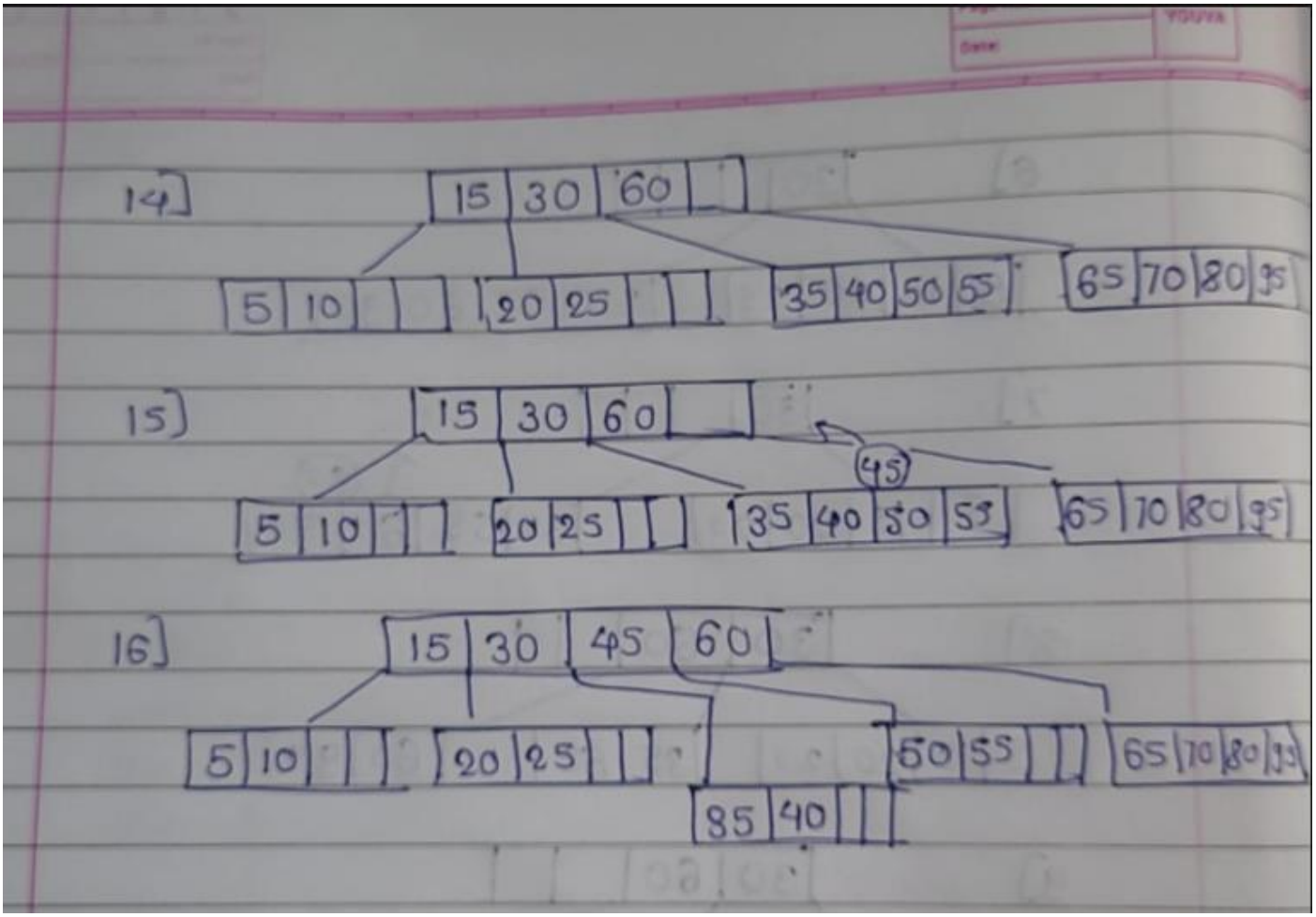
🔍 Search Operation (Logic):

1. Start at the root.
2. For each character in the word:
    - If the character exists as a child, move to it.
    - If not, return `False` (word not found).
3. After the last character, check `isEndOfWord`.
    - If true → word exists.
    - If false → only prefix exists, not full word.

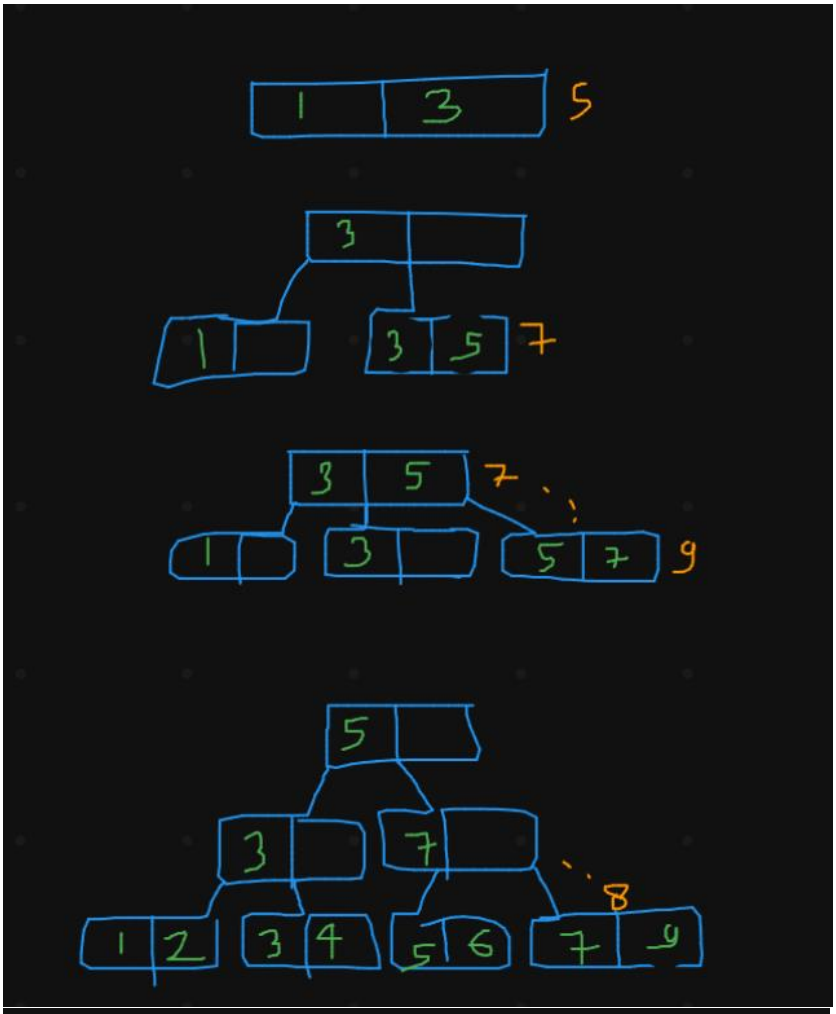
38) Create a B- Tree of order 5 from the following list of data items: 30, 20, 35, 95, 15, 60, 55, 25, 5, 65, 70, 10, 40, 50, 80, 45

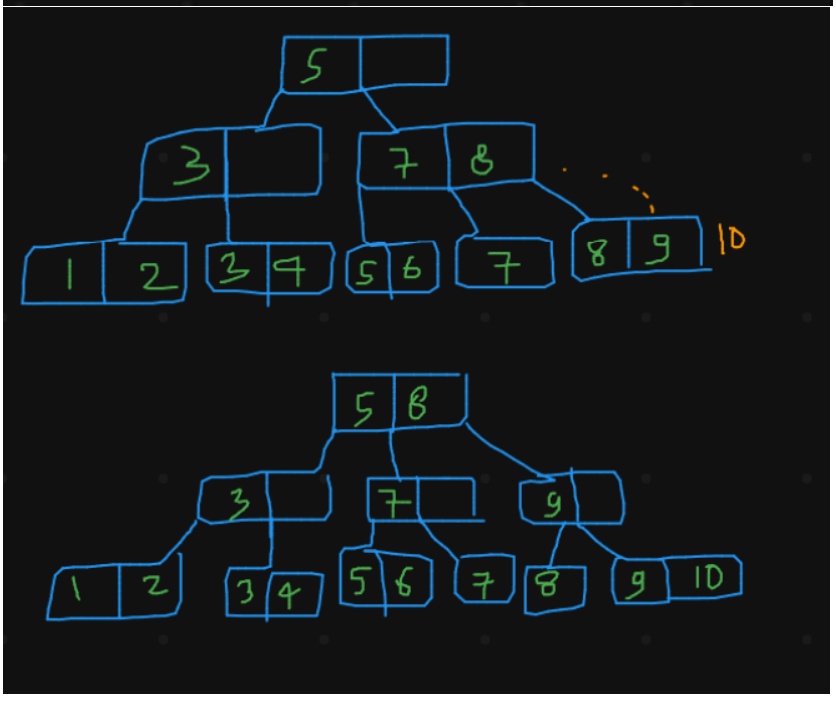



6]
30
5 15 20 25
35 55 60 95
7]
30
5 15 20 25
65
35 55 60 95
8]
30 60
5 15 20 25
35 55
65 95
9]
30 60
5 15 20 25
35 55
65 70 95
10]
30 60
10
5 15 20 25
35 55
65 70 95
11]
15 30 60
5 10
20 25
35 55
65 70 95
12]
15 30 60
5 10
20 25
35 40 55
65 70 95
13]
15 30 60
5 10
20 25
35 40 50 55
65 70 95


14]
15 30 60
5 10
20 25
35 40 50 55
65 70 80
15]
15 30 60
45
5 10
20 25
35 40 50
65 70 80
16]
15 30 45 60
5 10
20 25
50 55
65 70 80
85 40

39) Create a B+Tree of order 3 from the following list of data items: 1, 3, 5, 7, 9, 2, 4, 6, 8, 10





40) Define trie tree. Compare trie tree with hash table. Draw trie tree for following data: bear, sell, bell, bid, stock, bull, buy, stop.

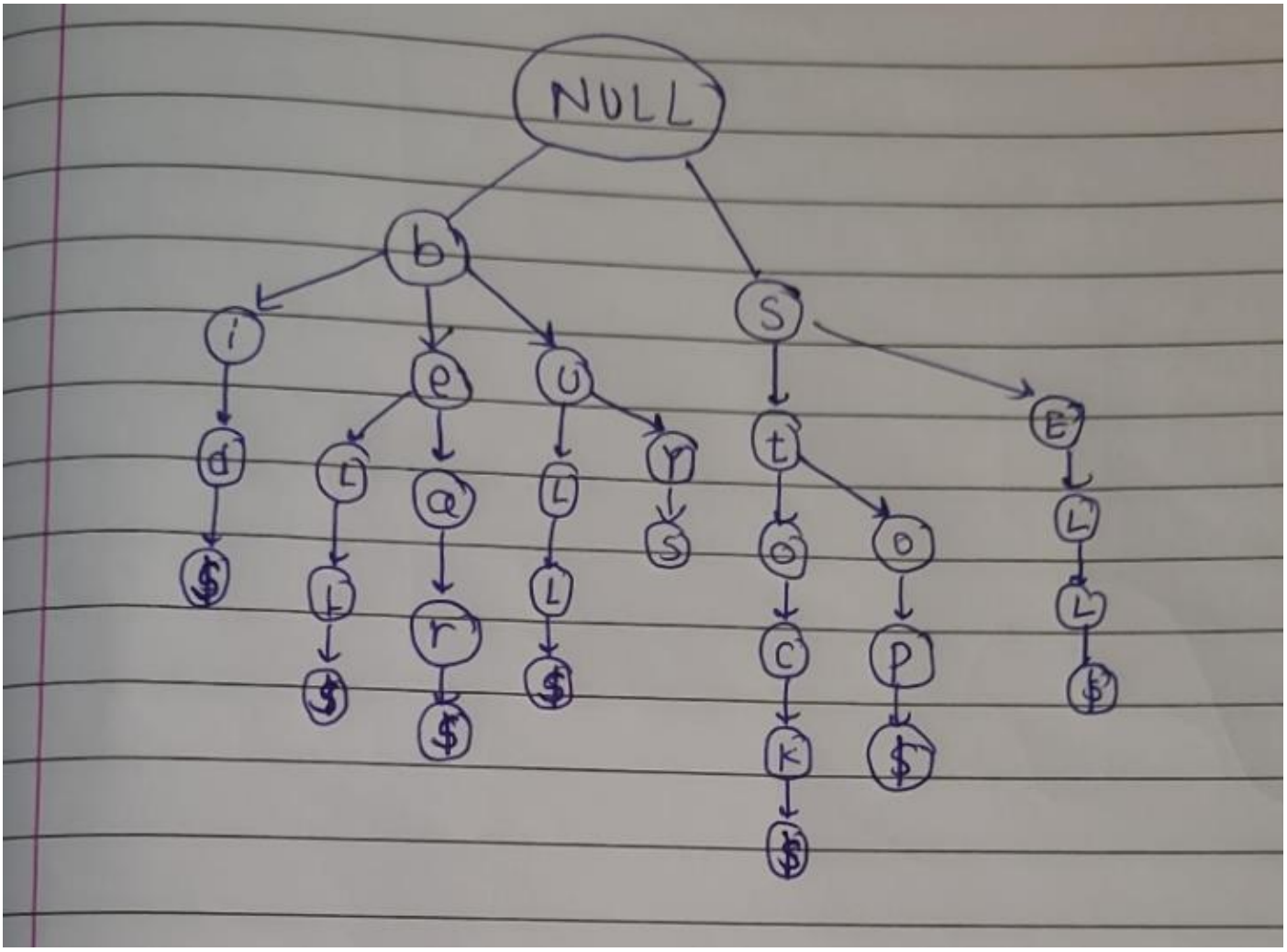
✅ Definition: Trie Tree

A **Trie** (short for *retrieval* and pronounced "try") is a **tree-based data structure** used to efficiently store and retrieve keys in a dataset of strings, especially when there are many common prefixes.

- Each node represents a character of a word.
- Paths from the root to the nodes represent words or prefixes.
- The final node of a word is marked (usually with a boolean flag like `isEndOfWord = true`).

🆚 Comparison: Trie Tree vs Hash Table

Feature	Trie Tree	Hash Table
Key Type	Generally strings (char by char)	Can be any hashable type
Search Time	O(L), where L = length of key	Average O(1), Worst O(n) due to collisions
Prefix Search	✅ Very efficient (built-in via structure)	❌ Not supported
Memory Usage	🔺 Higher (due to node pointers per character)	🔻 Lower, unless dealing with large collisions
Ordering	✅ Maintains lexicographic order	❌ No inherent order
Collision Handling	❌ Not needed	✅ Required (chaining, open addressing, etc.)
Use Cases	Autocomplete, dictionary, spell-checkers	Key-value pairs, caching, fast lookups
Scalability	Good for shared prefixes and large vocabularies	Good for arbitrary keys with minimal collisions


NULL
b
S
i
e
U
t
E
d
L
a
L
Y
o
L
$
L
r
L
S
C
P
K
$

41) What is indexing? What are the advantages of indexing? Discuss clustering index with example.

- To reduce record access time.
- Indexing is a data structure technique that helps to speed up data retrieval.
- As we can quickly locate and access the data in the database, it is a must-know data structure that will be needed for database optimizing.
- Indexing minimizes the number of disk accesses required when a query is processed.
- Indexes are created as a combination of the two columns.

Index structure

Single index

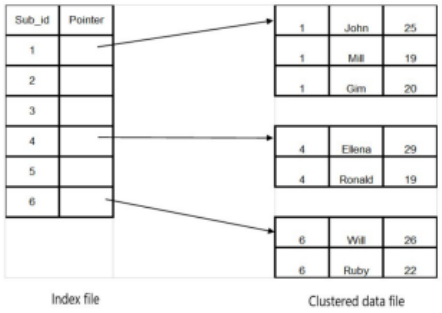
- **First column** is the **Search key**. It contains a copy of the primary key or candidate key of the table. The values of this column may be sorted or not. But if the values are sorted, the corresponding data can be accessed easily.
- **Second column** is the **Data reference** or **Pointer**. It contains the address of the disk block where we can find the corresponding key value.

✅ Advantages of Indexing

Advantage	Explanation
🔍 Faster Query Processing	Speeds up SELECT queries using WHERE, JOIN, or ORDER BY
⏱️ Reduces Disk I/O	Avoids full table scans by accessing only relevant rows
📈 Improves Sorting	Efficient ORDER BY and GROUP BY operations
🔗 Supports Uniqueness	Unique indexes enforce column uniqueness (e.g., on `ID`)
🚀 Speeds Up Joins	Join conditions are resolved quickly using indexes

Cluster Indexing

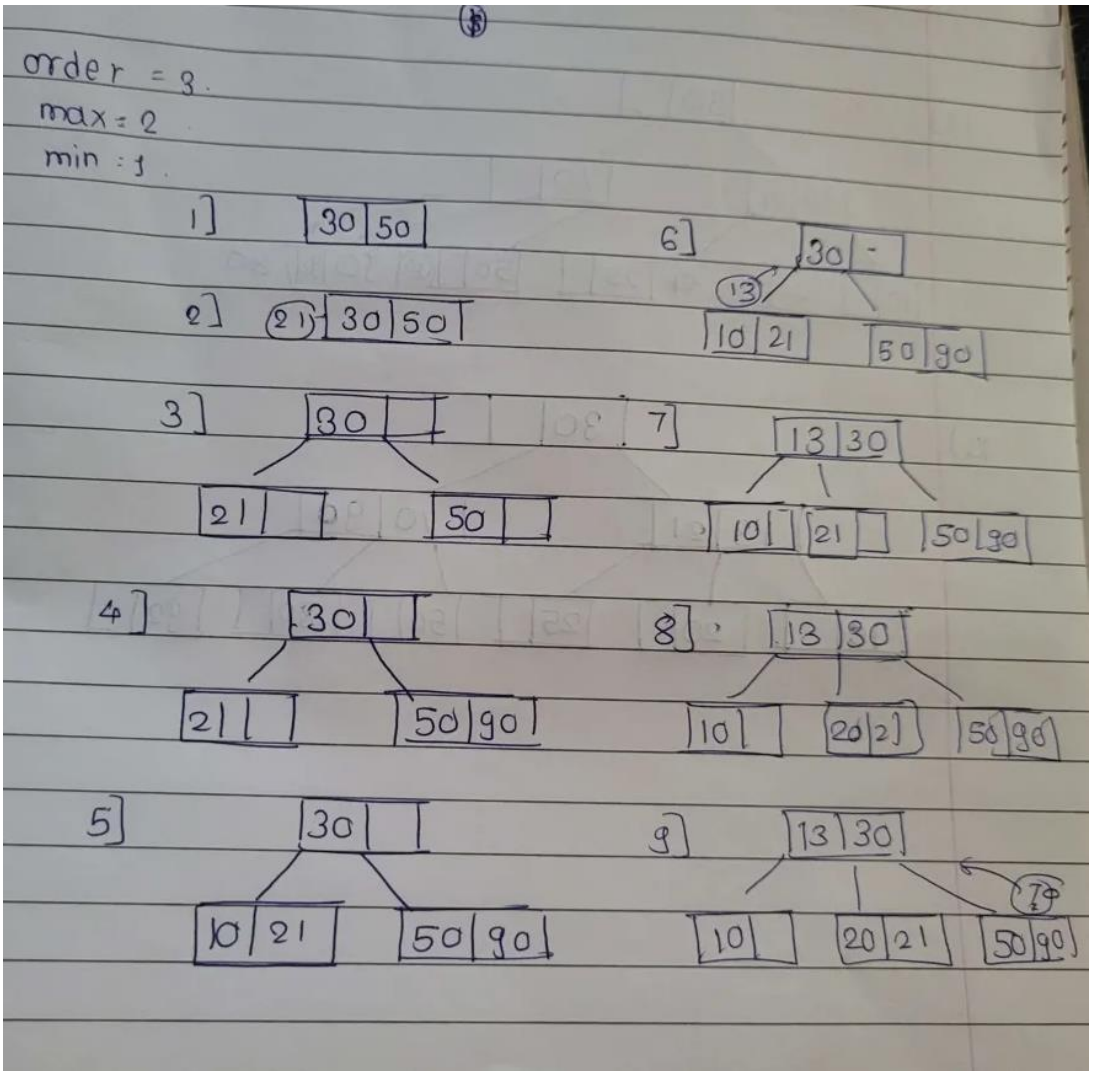
- Sometimes the ***index is created on non-primary key*** columns which might not be unique for each record.
- In such a situation, you ***can group two or more columns to get the unique values and create an index*** which is called clustered Index.
- This also ***helps you to identify the record faster***.

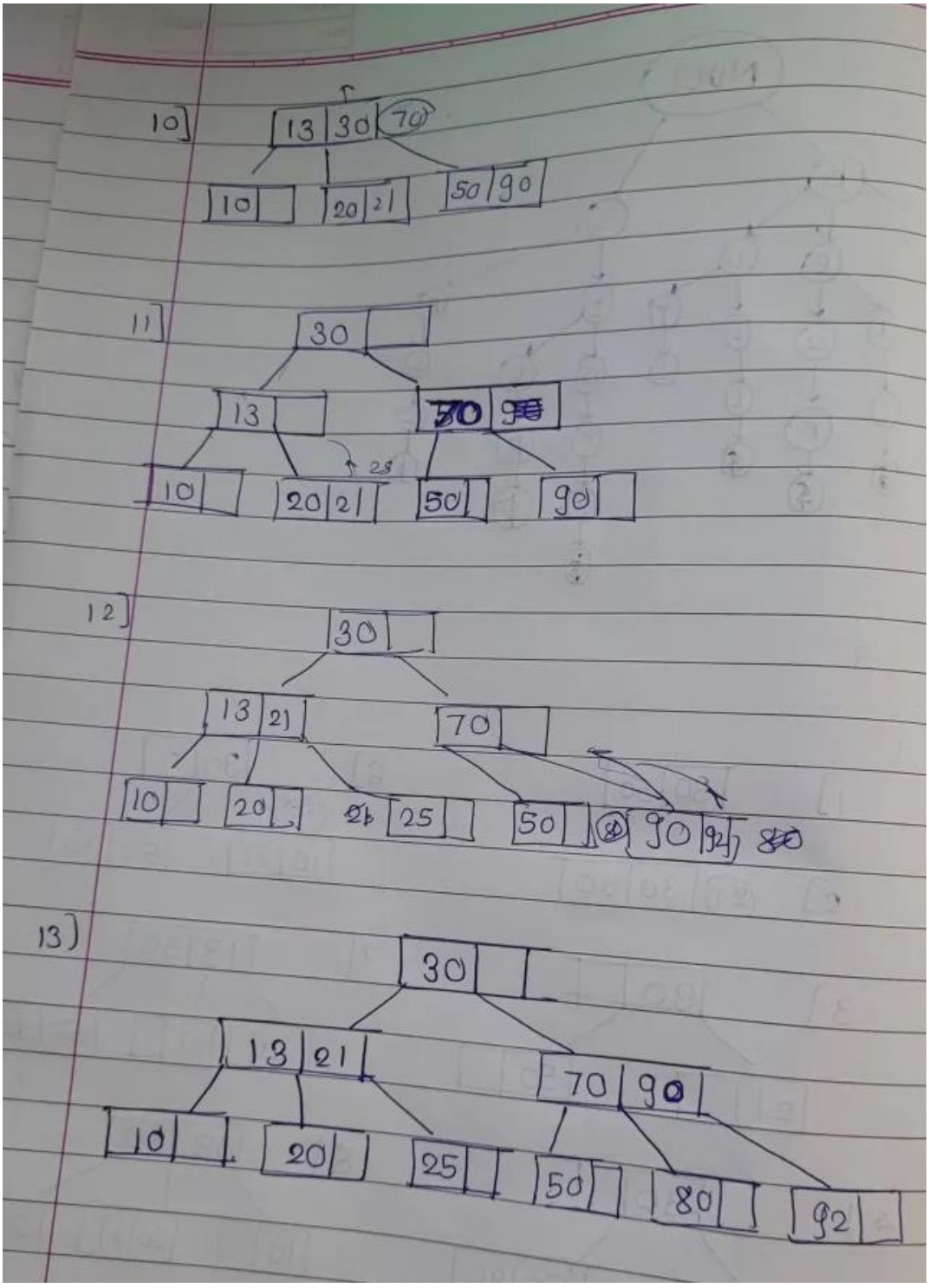


Example

- Consider a company recruited many employees in various departments. In this case, ***clustering index should be created for all employees who belong to the same dept.***
- In a single cluster it is considered that an index points to the cluster as a whole.

42) Construct a B-Tree of order 3 for following data : 50, 30, 21, 90, 10, 13, 20, 70, 25, 92, 80.




10]
13 30 70
10
20 21
50 90
11]
30
13
70 90
10
20 21
50
90
25
12]
30
13 21
70
10
20
25
50
90 92 80
13]
30
13 21
70 90
10
20
25
50
80
92

43) Why B+ tree? List its properties and advantages.

🌳 Why B+ Tree?

A **B+ Tree** is an advanced form of a B-Tree used widely in **database indexing** and **file systems**. It's specifically designed to handle **large volumes of data** efficiently, allowing **fast search, insert, delete**, and **range queries**.

✅ **Motivation for B+ Tree:**

- All actual data entries are stored **only at the leaf level**.
- Internal nodes **only guide the search** and do not store full records.
- It supports **range-based queries** very efficiently (e.g., `WHERE age BETWEEN 20 AND 30`).
- Better disk block utilization compared to B-Trees.

📚 Properties of B+ Tree:

1. **Balanced Tree**:
   - Always height-balanced.
   - All leaves are at the **same level**.
2. **Multiple Children**:
   - Each node (except the root) has **between ⌈m/2⌉ and m children** (for order m).
3. **Internal Nodes Store Only Keys**:
   - Internal (non-leaf) nodes store **keys only**, not actual records.
4. **Leaf Nodes Store Full Data**:
   - Actual records or pointers to records are stored in the **leaf nodes**.
5. **Linked Leaf Nodes**:
   - All leaf nodes are **linked as a linked list**, enabling fast sequential access.
6. **Sorted Keys**:
   - All keys are kept in **sorted order**, facilitating efficient binary search.

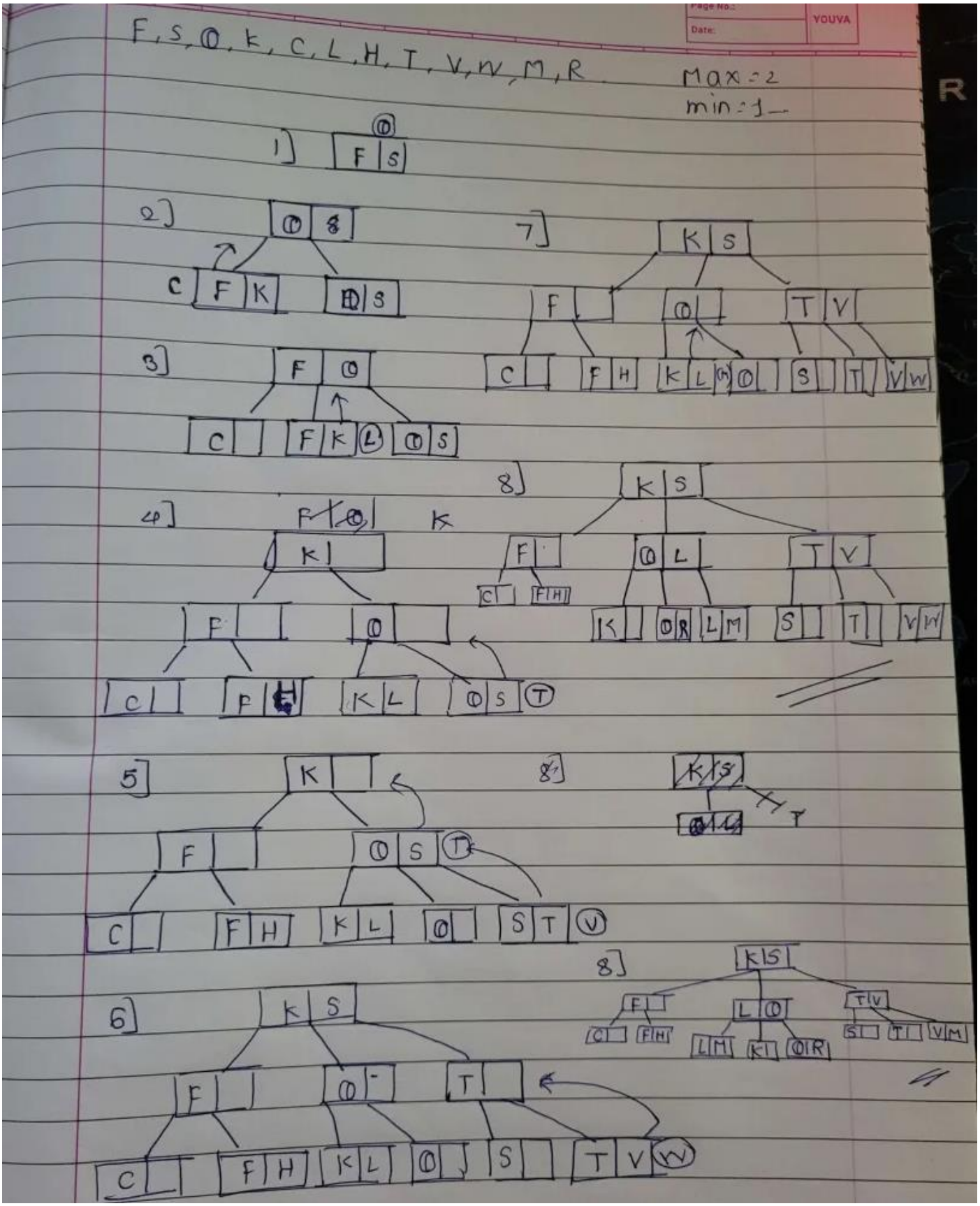
✅ Advantages of B+ Tree:

Advantage	Explanation
🔍 **Efficient Searching**	Faster than B-Trees for disk-based lookups; depth is smaller
📈 **Efficient Range Queries**	Sequential access through linked leaves (no backtracking needed)
💾 **Better Disk Read/Writes**	Internal nodes fit well into memory blocks; fewer disk I/Os
➕ **Supports Insert/Delete**	Handles dynamic growth and shrinkage of data efficiently
🧠 **Memory Efficient Indexing**	Internal nodes store less data (only keys), so more keys per block
🔁 **Full Table Scan**	Easier due to linked leaves (e.g., for ORDER BY queries)

🆚 B+ Tree vs B-Tree

Feature	B+ Tree	B-Tree
Data in Internal	❌ Only keys (no actual data)	✅ Can contain data
Data in Leaves	✅ All data is in leaf nodes	❌ Data may be scattered across levels
Sequential Access	✅ Very efficient (linked leaves)	❌ Less efficient
Space Efficiency	✅ Higher (due to denser internal nodes)	❌ Lower (data duplication)
Range Queries	✅ Fast and simple	❌ Slower, may require backtracking

44) Build B+ tree of order 3 for the following : F, S, Q, K, C, L, H, T, V, W, M, R



45) Explain multilist files & coral rings.

1 Multilist Files

A **multilist file** is a data structure that allows a record to participate in **multiple linked lists** simultaneously — each for a **different key or relationship**.

🔹 **Key Idea:**

- Each record contains **multiple pointers**, one for each list.
- Used when you need to **traverse records in different logical orders** (e.g., by department, by project).

✅ **Example:**

Imagine a file of **employees**. You want to access them by:

- Department
- Project

Each employee record will have:

- A **Department pointer** → to the next employee in the same department.
- A **Project pointer** → to the next employee in the same project.

This lets you efficiently traverse by either key **without duplicating records**.

2 Coral Rings

Coral Rings are a type of **circular linked structure** used in databases for:

- **Bi-directional traversal** (forward and backward),
- Maintaining multiple **logical access paths**.

🔹 **Structure:**

- Each node (record) in a **coral ring** contains pointers to the **next** and **previous** node.
- The last node points back to the **first**, forming a **closed ring** (circular list).

✅ **Example:**

Used for managing:

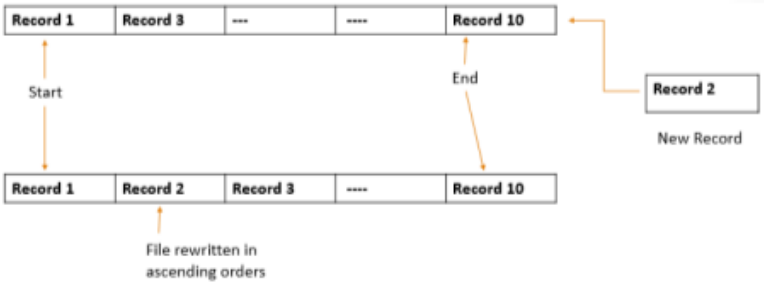
- **Access control lists**
- **Directory traversal**
- **Round-robin scheduling**

🔁 **Benefits:**

- Can start from **any point** and still access all records.
- Efficient for **circular iteration** (like in ring buffers).

46) What is Sequential and index sequential file organization? State its advantages and disadvantages

- **Sequential file**
  - Records are stored and accessed in a particular/sequential order sorted using a key field.
  - Retrieval requires searching sequentially through the entire file start to end of the file.
  - As the record in a file are sorted in a sorted order, the binary search technique can be used to reduce the searching time



- **Advantages of sequential file organization**
  - The sorting makes it easy to access records.
  - The binary technique can be used to reduce record search time.
- **Disadvantages of sequential file organization**
  - The sorting does not remove the need to access other records as the search looks for particular records.
  - Sequential records cannot support modern technologies that require fast access to stored records.
  - Random searching is not possible.

Indexed sequential access file organization

Indexed sequential access file is a combination of both sequential file and direct access file organization.

In this records are stored randomly on a direct access device such as magnetic disk by a primary key.

This file have multiple keys, An index consists of keys and addresses.

The data can be access either sequentially or randomly using the index. The index is stored in a file and read into memory when the file is opened.

Advantages of Indexed sequential access file organization

Flexibility provided as sequential file and random file access is possible.

It can accesses the records very fast if the index table is properly organized.

The records can be inserted at any position in the file.

It reduces the degree of the sequential search.

Disadvantages of Indexed sequential access file organization

Indexed sequential access file requires unique keys

It is expensive because it requires another software.

It is less efficient in the use of storage space as compared to other file organizations.

VS Quick Comparison Table

Feature	Sequential File	Index Sequential File
Access Type	Sequential only	Sequential + Random
Search Speed	Slow (linear or binary)	Fast (via index)
Insertion/Deletion	Expensive	Easier (but may need reindexing)
Maintenance	Simple	More complex
Storage Overhead	Low	Higher (due to index storage)

47) Explain inverted file & cellular partitions.

📘 1. Inverted File

An **inverted file** is a type of **indexing technique**, especially used in **information retrieval systems** (like search engines and databases).

✅ Key Idea:

Instead of mapping **records to fields**, it maps **field values (or words)** to the **records that contain them**.

🗂️ Example:

Imagine a library system where you want to search by subject:

You have 3 books:

Book ID	Title	Keywords
B1	Data Structures	Trees, Hashing
B2	Algorithms	Graphs, Sorting
B3	Advanced Data Structures	Trees, Graphs

An **inverted file** will look like:

Keyword	Book IDs
Trees	B1, B3
Graphs	B2, B3
Sorting	B2
Hashing	B1

So if someone searches "Trees," the system immediately shows B1 and B3 without scanning all books.

🟢 Advantages:

- ✅ Fast keyword-based search
- ✅ Supports multiple indexes (e.g., on author, year, etc.)
- ✅ Ideal for full-text search (used in Google, Lucene, etc.)

🔴 Disadvantages:

- ❌ Needs storage space for index
- ❌ Index must be updated when data changes

📗 2. Cellular Partition

A **cellular partition** is a **geometric file partitioning technique** used in spatial databases, often when dealing with **2D or 3D data** like maps or geographic information.

✅ **Key Idea:**

The data space is **divided into cells or grids**, and each **cell contains records** that fall within its boundaries.

📍 **Example:**

Imagine you're storing **locations of delivery orders** in a city.

- Divide the city into **10x10 grid cells**
- Each grid (cell) will store orders in that **zone**
- When querying "Show me all orders near Zone (3,4)," the system checks that **cell and neighboring ones**, not the whole dataset

🟢 **Advantages:**

- ✅ Efficient for **range queries and spatial searches**
- ✅ Easy to implement with known bounds
- ✅ Reduces the search space significantly

🔴 **Disadvantages:**

- ❌ Poor distribution can lead to **overloaded cells**
- ❌ May not adapt well to **non-uniform data** (e.g., clustered points)

48) Explain direct access file organization. State its advantages and disadvantages

Direct or random access file

Direct access file is also known as random access or relative file organization.

Files that have been designed to make direct record retrieval as easy and efficiently.

Useful for immediate access to large amounts of information

The records are stored at known address.

Address is calculated by applying a mathematical function to the key field.

In direct access file, all records are stored in direct access storage device.

The records does not need to be in sequence as they are updated directly and rewritten back in the same location.

It is also called as hashing.

Advantages of direct access file organization

Direct access file helps in online transaction processing system (OLTP)

In direct access file, sorting of the records are not required.

It accesses the desired records immediately, hence it updates several files very quickly.

It has better control over record allocation and retrieval .

Disadvantages of direct access file organization

Direct access file does not provide back up facility.

It is expensive than other methods.

It has less storage space as compared to sequential file.

49) What is linked organization? Explain inverted file and coral rings with respect to linked organization.

📦 What is Linked Organization?

Linked Organization is a method of storing records in a file where each record contains a **pointer (or link)** to the **next related record**. This forms a **linked list** or network of records.

Instead of placing records sequentially in memory or on disk, linked organization allows records to be **scattered across locations**, with each pointing to the next. It's widely used when:

- Insertion and deletion need to be flexible
- Relationships between records are complex (e.g., many-to-many)

🔗 Linked Organization in Detail

✅ Key Characteristics:

- Records are **not physically adjacent**
- Each record contains **data + pointer(s)**
- Enables **efficient traversal**, insertion, and deletion

50) List & explain two possible ways of representing records

51) Differentiate between indexed sequential file and direct access file.

Feature	Indexed Sequential File	Direct Access File
Storage Organization	Records are stored **sequentially** with an **index**	Records are stored based on a **calculated address**
Access Type	Supports **sequential** and **indexed (partially random)** access	Supports **random (direct)** access only
Use of Index	Requires an **index table** to locate records	No index is used; uses a **hash function**
Access Speed	Slower than direct access (due to indexing)	Fastest access (direct address computation)
Flexibility	Flexible — allows both sequential and indexed access	Less flexible — only random access is efficient
Example Use Case	Bank transaction records, payroll systems	Real-time systems like airline reservation, databases
Insertion/Deletion	More complex due to index maintenance	Can be easier if space is managed well
Complexity	More complex due to index + data file management	Depends on hashing logic and collision handling

All The Best !!

- **Karan Salunkhe ☺**
- **Anish Joshi ☺**