Simulation of a File System Using a Directed Acyclic Graph (DAG) Structure:
i. Simulate a file system using a Directed Acyclic Graph (DAG) structure.
ii. Write a program to demonstrate file sharing between directories.

File System Directory Structures

🧭 Outline

1. Single-Level Directory
2. Two-Level Directory
3. Tree-Structured Directory
4. Directed Acyclic Graph (DAG) Structure
5. Comparison Summary

1️⃣ Single-Level Directory

📌 Definition

A **single-level directory** system contains **only one directory** for all users and all files.

📊 Diagram

```
     Directory
     /   |   \
  File1 File2 File3
```

🔍 Characteristics

- All files are stored in one directory
- File names must be **unique**
- No subdirectories

✅ Advantages

- Simple to implement
- Easy file searching for small systems

❌ Disadvantages

- Name conflicts
- No grouping of files
- Not suitable for multi-user systems

📝 Use Case

- Early operating systems
- Very small or embedded systems

2️⃣ Two-Level Directory

📌 Definition

A **two-level directory** system has:

- One **master directory**
- Separate **user directories** under it

📊 Diagram

```
          Master Directory
           /             \
       User1              User2
      /    \                |
  FileA   FileB           FileC
```

🔍 Characteristics

- Each user has a separate directory
- File names need to be unique **only within a user directory**

✅ Advantages

- Avoids file name conflicts
- Provides user file isolation

❌ Disadvantages

- No subdirectories within user directories
- Limited organization

📝 Use Case

- Multi-user systems with basic organization

3️⃣ Tree-Structured Directory

📌 Definition

A **tree-structured directory** allows directories to contain **subdirectories**, forming a hierarchical structure.

📊 Diagram

```
          Root
           |
          Home
         /    \
     User1    User2
       |
     Docs
       |
    File1
```

🔍 Characteristics

- Hierarchical organization
- Each file has **only one parent**
- No file sharing

✅ Advantages

- Logical grouping of files
- Easy navigation and management

❌ **Disadvantages**

- File sharing is not possible
- Duplication of files may occur

📝 **Use Case**

- Most modern operating systems (basic form)

4️⃣ Directed Acyclic Graph (DAG) Structure

📌 **Definition**

A **Directed Acyclic Graph (DAG)** structure allows **file and directory sharing** while preventing cycles.

📊 **Diagram**

```
      Root
      /    \
   DirA    DirB
      \    /
      File1
```

🔍 **Characteristics**

- Files can have **multiple parent directories**
- Uses **links** to share files
- No cycles allowed

✅ **Advantages**

- Efficient file sharing
- Saves storage space
- Maintains consistency

❌ **Disadvantages**

- Deletion is complex
- Requires reference counting
- Cycle prevention needed

📝 **Use Case**

- UNIX systems (hard links)

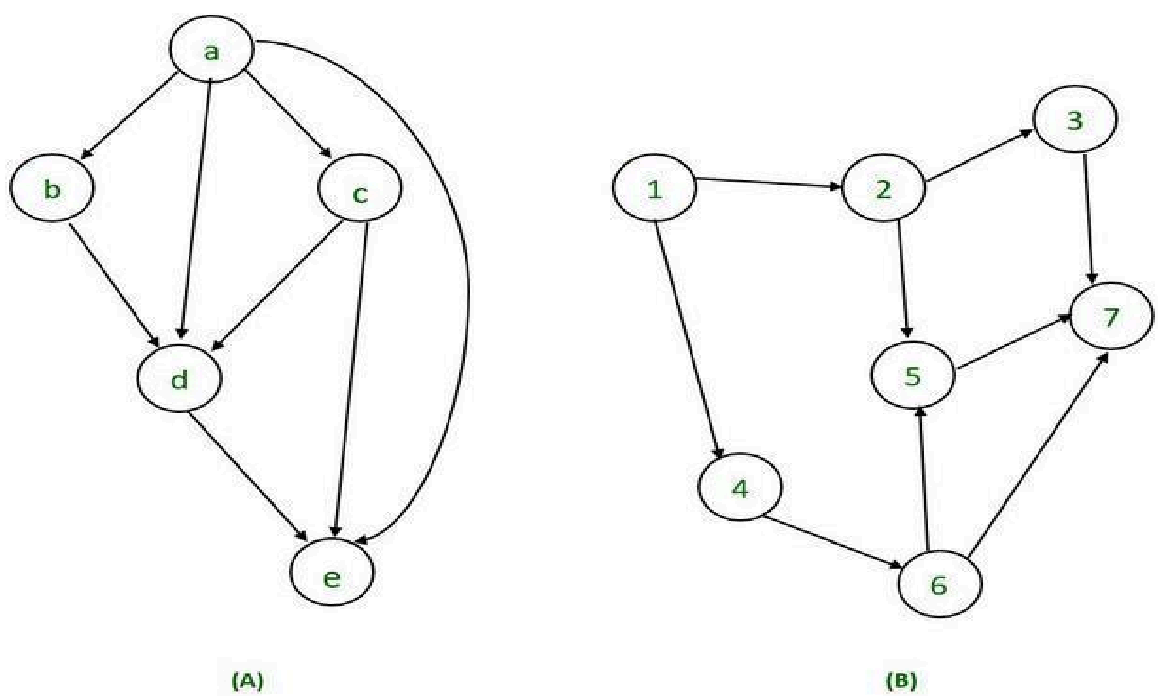
5️⃣ Comparison Summary (Very Exam-Friendly)

Structure	File Sharing	Hierarchy	Complexity
Single-Level	❌ No	❌ No	Very Low
Two-Level	❌ No	Partial	Low
Tree	❌ No	✅ Yes	Medium
DAG	✅ Yes	✅ Yes	High

<u>What is a Directed Acyclic Graph?</u>

A Directed Acyclic Graph (DAG) is a directed graph that does not contain any cycles.

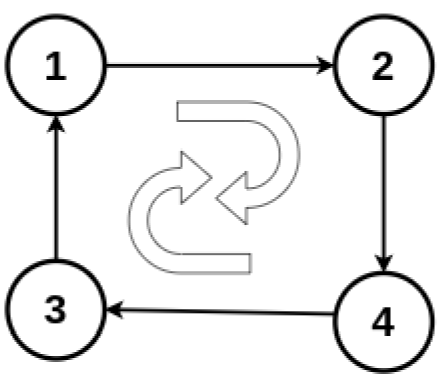
Direct Acyclic Graph



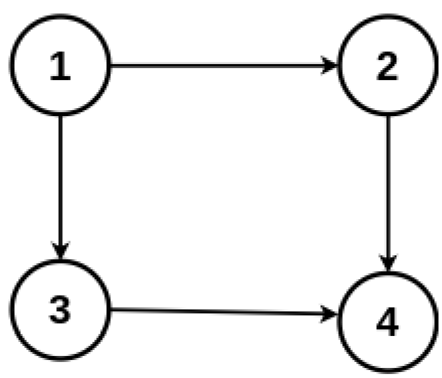
<u>Meaning of Directed Acyclic Graph:</u>

Directed Acyclic Graph has two important features:

- **<u>Directed Edges:</u> In Directed Acyclic Graph, each edge has a direction, meaning it goes from one vertex (node) to another. This direction signifies a one-way relationship or dependency between nodes.**
- **<u>Acyclic:</u> The term "acyclic" indicates that there are no cycles or closed loops within the graph. In other words, you cannot traverse a sequence of directed edges and return to the same node, following the edge directions. Formation of cycles is prohibited in DAG. Hence this characteristic is essential.**



Direct Cyclic Graph



Direct Acyclic Graph

Directed Acyclic Graph

Practical Applications of DAG:

- **Data flow Analysis: In compiler design and optimization, DAGs are used to represent data flow within a program. This aids in optimizing code by identifying redundant calculations and dead code. DAGs are also used to represent the structure of [basic blocks](#) in Compiler Design.**
- **Task Scheduling: DAGs are used in project management and job scheduling. Each task or job is represented as a node in the**

DAG, with directed edges indicating dependencies. The acyclic nature of the DAG ensures tasks are scheduled in a logical order, preventing circular dependencies.

i . Simulation of a File System Using a DAG Structure

In a DAG-based file system model, the structure can be visualized as follows:

Nodes: Represent files and directories.

Directed Edges: Represent the "contains" or "parent-child" relationship. An edge from directory A to file B means B is contained within A.

Acyclic Property: Prevents loops (e.g., directory A containing directory B, which in turn contains directory A), ensuring a file or directory cannot be its own ancestor. This allows for efficient navigation and management.

File Sharing: Implemented by allowing multiple directory nodes to have outgoing edges pointing to the same file node. These are essentially hard links, which point to the same underlying physical data/inode.

This structure is a flexible alternative to the traditional tree structure, in which each file can have only a single parent directory.

ii. **Write a program to demonstrate file sharing between directories.**

Pseudocode:

🧭 Outline

1. **Define data structures**
2. **Create a file**
3. **Create directories**
4. **Share the file between directories**
5. **Maintain reference count**
6. **Handle deletion safely**

🧩 Pseudocode

```
START

DEFINE STRUCT File
    name
    reference_count
END STRUCT

DEFINE STRUCT Directory
    name
```

```
    pointer_to_file
END STRUCT

// Step 1: Create a file
CREATE File F
SET F.name = "shared_file"
SET F.reference_count = 0

// Step 2: Create directories
CREATE Directory D1
SET D1.name = "DirA"

CREATE Directory D2
SET D2.name = "DirB"

// Step 3: Share file between directories
SET D1.pointer_to_file = F
INCREMENT F.reference_count

SET D2.pointer_to_file = F
INCREMENT F.reference_count

// Step 4: Display status
PRINT F.name
PRINT F.reference_count
PRINT D1.name contains F.name
PRINT D2.name contains F.name

// Step 5: Remove file from one directory
SET D1.pointer_to_file = NULL
```

```
DECREMENT F.reference_count

PRINT updated reference_count

// Step 6: Check before deletion
IF F.reference_count == 0 THEN
    DELETE F
    PRINT "File deleted from system"
ELSE
    PRINT "File still shared"
END IF

END
```