5.6.DATA STRUCTURES USED IN OPERATING SYSTEMS (OS) AND DBMS

Data structures are fundamental to both **Operating Systems (OS)** and **Database Management Systems (DBMS)** because they enable efficient storage, retrieval, scheduling, indexing, and resource management.

1. Arrays

Definition

An array stores elements of the same type in contiguous memory locations.

int arr[5] = {10,20,30,40,50};

Time Complexity:

Operation	Complexity
Access	O(1)
Search	O(n)
Insert	O(n)
Delete	O(n)

Applications in OS

Process Table

The OS maintains information about processes:

Process ID
Process State
Priority
Memory Address

Stored as:

Process Table[ ]

Page Tables

Virtual memory uses arrays for page mapping.

Page Number → Frame Number

Applications in DBMS

Buffer Pool

Pages loaded into memory are stored in array-like structures.

Record Storage

Fixed-size records are often stored in arrays within pages.

---

2. Linked List

Definition

Nodes connected using pointers.

10 → 20 → 30 → NULL

Node:

```
struct Node
{
   int data;
   struct Node *next;
};
```

Applications in OS

Ready Queue

Processes waiting for CPU:

P1 → P2 → P3

Free Memory List

Available memory blocks:

Block1 → Block2 → Block3

File Allocation Table

Linked allocation of disk blocks.

Applications in DBMS

Overflow Records

When a page becomes full:

Page1 → Overflow Page

Chained Hashing

Collision handling in hash indexes.

---

3. Doubly Linked List

NULL ← A ↔ B ↔ C → NULL

Each node stores:

prev
data
next

Applications in OS

LRU Page Replacement

Most recently used pages are moved to the front.

Head ↔ ... ↔ Tail

Fast deletion and insertion.

Process Scheduling

Bidirectional traversal of process queues.

Applications in DBMS

Buffer Management

Maintains recently accessed pages.

Transaction Lists

Track active transactions efficiently.

---

4. Stack

Principle

LIFO
Last In First Out

Operations:

Push
Pop
Peek

Applications in OS

Function Calls

Each call creates a stack frame.

```
main()
 |
 +-- fun1()
      |
      +-- fun2()
```

Interrupt Handling

Stores execution context.

Applications in DBMS

Query Parsing

Expression evaluation:

(A+B)*C

Uses stacks.

Recursive Queries

Execution tracking.

---

5. Queue

Principle

FIFO
First In First Out

Operations:

Enqueue
Dequeue

Applications in OS

CPU Scheduling

Ready Queue

Example:

P1 → P2 → P3

Printer Queue

Print jobs processed in order.

I/O Requests

Disk requests waiting for service.

Applications in DBMS

Transaction Scheduling

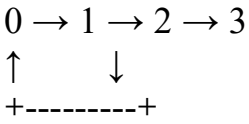
Queries waiting for execution.

Request Handling

Multiple user requests.

---

6. Circular Queue



Applications in OS

Round Robin Scheduling

P1 → P2 → P3 → P1

Each process gets fixed time slice.

Applications in DBMS

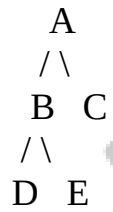
Resource Pools

Managing reusable connections.

---

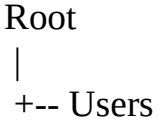
7. Tree

Hierarchical data structure.



Applications in OS

Directory Structure



```
    |
    +-- Documents
```

Process Tree

```
init
 |
 +-- Process1
 |
 +-- Process2
```

Applications in DBMS

Query Execution Trees

Represent SQL execution plans.

Hierarchical Databases

Parent-child relationships.

---

8. Binary Search Tree (BST)

```
    50
   /  \
  30   70
```

Properties:

Left < Root < Right

Applications in DBMS

Ordered Data Retrieval

Fast searching.

Complexity:

Operation	Average
Search	O(log n)
Insert	O(log n)

Operation	Average
Delete	O(log n)

---

9. B-Tree

Most important DBMS data structure.

Structure

```
        [30]
       /    \
  [10 20]  [40 50]
```

Why B-Trees?

Disk access is expensive.

B-Tree minimizes:

Disk Reads
Disk Writes

Applications in DBMS

Indexing

```
CREATE INDEX idx ON Student(ID);
```

Uses B-Tree internally.

Advantages:

- Balanced
- Efficient range queries
- Few disk accesses

Complexity:

Search = O(log n)
Insert = O(log n)
Delete = O(log n)

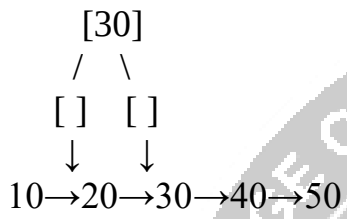
---

10. B+ Tree

Most widely used DBMS indexing structure.

Internal Nodes → Keys
Leaf Nodes → Actual Records

Example:



Applications

Database Indexes

Used in:

- MySQL
- PostgreSQL
- Oracle
- SQL Server

Advantages:

- Faster range search
- Sequential access
- Efficient disk utilization

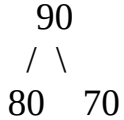
---

11. Heap

Definition

Complete binary tree.

Max Heap



Applications in OS

Priority Scheduling

Highest priority process selected first.

Event Scheduling

Kernel task management.

Applications in DBMS

Top-K Queries

SELECT TOP 10 salary

Query Optimization

Priority-based execution.

---

12. Hash Table

Stores:

Key → Value

Example:

1001 → Student Record

Applications in OS

Symbol Tables

Variable lookups.

Page Lookup

Memory management.

Applications in DBMS

Hash Indexing

WHERE id = 100

Average complexity:

O(1)

---

13. Graph

Consists of:

Vertices
Edges

Example:

A ---- B
| |
C ---- D

Applications in OS

Deadlock Detection

Resource Allocation Graph.

Process → Resource

Network Routing

Path determination.

Applications in DBMS

Graph Databases

Examples:

- Neo4j
- TigerGraph

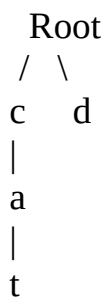
Used for:

- Social networks
- Recommendation systems
- Fraud detection

---

14. Trie

Stores strings efficiently.



Applications in OS
Command Auto-completion

Shell suggestions.

Dictionary Search

Fast word matching.

Applications in DBMS
Text Search

Prefix matching:

LIKE 'abc%'

15. Disjoint Set (Union-Find)

Supports:

Find()
Union()

Applications in OS

Network Connectivity

Checking connected components.

Resource Grouping

Managing clusters.

Applications in DBMS

Clustering Algorithms

Grouping related records.

Distributed Databases

Node connectivity tracking.

Important Interview Table

Data Structure	OS Use	DBMS Use
Array	Process Table, Page Table	Buffer Pool
Linked List	Ready Queue, Free List	Overflow Pages
Doubly Linked List	LRU Cache	Buffer Manager
Stack	Function Calls	Query Parsing
Queue	CPU Scheduling	Transaction Queue
Circular Queue	Round Robin	Connection Pool
Tree	Directory Structure	Query Tree
BST	Searching	Ordered Records
B-Tree	File Systems	Indexing
B+ Tree	File Systems	Database Indexes

Data Structure	OS Use	DBMS Use
Heap	Priority Scheduling	Top-K Queries
Hash Table	Page Lookup	Hash Index
Graph	Deadlock Detection	Graph Databases
Trie	Auto-complete	Text Search
Disjoint Set	Connectivity	Clustering

Page Replacement Index (OS &DBMS)

What is Page Replacement?

In a virtual memory system, when a process needs a page that is not in RAM, a **page fault** occurs.

If RAM is full, the OS must:

1. Select a page currently in memory.
2. Remove (replace) it.
3. Load the required page.

This process is called **Page Replacement**.

Page Fault
↓
Memory Full?
↓
Yes
↓
Select Victim Page
↓
Replace Page

Why Page Replacement?

Example:

Frames Available = 3

Reference String:
1 2 3 4

Memory:

1
1 2
1 2 3

Now page 4 arrives.

Memory is full.

One page must be removed.

Goals

- Reduce page faults
- Improve CPU utilization
- Improve system performance

---

2. FIFO (First In First Out)

Principle

The oldest page is removed first.

Example

Reference String:

1 2 3 4

Frames = 3

1
1 2
1 2 3

When page 4 arrives:

Remove 1
Load 4

2 3 4

Advantages

- Simple
- Easy implementation

Disadvantages

- May remove frequently used pages
- Suffers from Belady's Anomaly

Belady's Anomaly

Increasing frames can sometimes increase page faults.

Example:

Frames = 3
Page Faults = 9

Frames = 4
Page Faults = 10

Unexpected behavior.

Occurs in FIFO.

---

3. LRU (Least Recently Used)

Principle

Remove the page that has not been used for the longest time.

Example:

Pages:
1 2 3

Recent Usage:
3 used recently
2 used recently
1 not used for long time

Replace:

1

Implementation

Uses:

- Stack
- Doubly Linked List
- Hash Table

Advantages

- Better than FIFO
- Practical

Disadvantages

- Tracking usage is costly

Example

Reference:

1 2 3 1 4

Frames = 3

1
1 2
1 2 3

Page 1 used again.

Now page 4 arrives.

Least recently used:

2

Replace 2.

Result:

1 3 4

---

4. Optimal Page Replacement (OPT)

Principle

Replace page that will not be used for the longest future period.

Example:

Reference:
1 2 3 4 1 2 5

Look ahead.

Remove page whose next use is farthest.

Advantages

- Minimum page faults
- Theoretical best

Disadvantages

- Future knowledge impossible

Used mainly for comparison.

---

5. LFU (Least Frequently Used)

Replace page with lowest usage count.

Example:

Page 1 → used 10 times
Page 2 → used 2 times
Page 3 → used 7 times

Replace:

Page 2

Advantage

Keeps popular pages.

Disadvantage

Old frequently-used pages may stay forever.

---

6. Clock (Second Chance) Algorithm

Improved FIFO.

Each page has:

Reference Bit

Page structure:

Page	Reference Bit
P1	1
P2	0
P3	1

Circular pointer scans pages.

Rules:

Bit=0 → Replace
Bit=1 → Set 0 and skip

Advantage

- Efficient
- Used in real OS implementations

Comparison of Page Replacement Algorithms

Algorithm	Idea	Performance
FIFO	Oldest page removed	Poor
LRU	Least recently used	Good
OPT	Future prediction	Best
LFU	Least frequently used	Moderate
Clock	Second chance FIFO	Very Good

Data Structures Used in Page Replacement

Algorithm	Data Structure
FIFO	Queue
LRU	Doubly Linked List + Hash Table
Clock	Circular Queue
LFU	Heap + Hash Table
OPT	Array/List

---

7. Indexing in DBMS

What is an Index?

An index is a data structure that speeds up data retrieval.

Without index:

```
SELECT * FROM Student
WHERE ID = 100;
```

DBMS scans entire table.

Complexity:

O(n)

With index:

O(log n)

Real-Life Example

Book:

Without Index:
Read all pages

With Index:
Jump directly to topic page

Same idea in DBMS.

Primary Index

Created on primary key.

Example:

```
Student(
   RollNo PRIMARY KEY,
   Name
)
```

Index:

RollNo → Record Address

Secondary Index

Created on non-primary attributes.

Example:

```
Student(
   RollNo,
```

```
    Name,
    City
)
```

Index on:

City

Allows quick search:

WHERE City='Chennai'

Clustered Index

Records physically stored in index order.

Example:

1
2
3
4
5

Stored physically in same order.

Characteristics

- Only one clustered index per table.
- Faster range queries.

Non-Clustered Index

Separate structure.

Index
↓
Pointer
↓
Actual Record

Many non-clustered indexes possible.

B-Tree Index

Most common indexing structure.

Example:

```
    50
   /  \
  20   80
```

Properties:

- Balanced
- Sorted
- Efficient disk access

Complexity:

Search = O(log n)
Insert = O(log n)
Delete = O(log n)

B+ Tree Index

Most widely used index in modern DBMS.

Structure:

Internal Nodes
↓
Search Keys

Leaf Nodes
↓
Actual Records

Example:

```
      [30]
     /    \
   [ ]    [ ]
    ↓      ↓
```

10 → 20 → 30 → 40 → 50

Advantages

- Faster range search
- Sequential traversal
- Fewer disk accesses

Used by many relational databases.

Hash Index

Uses hash function.

Hash(Key)
↓
Bucket

Example:

ID=100
Hash(100)=Bucket 5

Complexity

Average:

O(1)

Good For

WHERE ID = 100

Poor For

WHERE ID > 100

Range queries are inefficient.

Index Data Structures

Index Type	Data Structure
Primary Index	B+ Tree

Index Type	Data Structure
Secondary Index	B+ Tree
Hash Index	Hash Table
Clustered Index	B+ Tree
Non-Clustered Index	B+ Tree
Full Text Index	Trie / Inverted Index

.

Which page replacement algorithm is best?

- Theoretical: OPT
- Practical: LRU / Clock

What is Belady's Anomaly?

More frames causing more page faults in FIFO.

Why is LRU better than FIFO?

Uses recent access history.

What is indexing?

Technique to speed up data retrieval.

Which index is most used in DBMS?

B+ Tree Index

Difference between B-Tree and B+ Tree?

B-Tree	B+ Tree
Records in all nodes	Records only in leaf nodes
Range search slower	Range search faster

B-Tree	B+ Tree
Less common	Most widely used

Hash Index vs B+ Tree?

Hash Index	B+ Tree
O(1) exact lookup	O(log n) lookup
Poor range queries	Excellent range queries
Equality search	Equality + Range search