UNIT II: Linear Data Structures

A Comprehensive Study Guide with Real-Time Examples

Arrays

Arrays are collections of elements stored in contiguous memory locations, accessed by index. Each element can be retrieved in O(1) constant time.

Characteristics:

Fixed Size: Memory allocated at compile time

Homogeneous: All elements must be of the same data type

Random Access: Direct access to any element using index

Contiguous Memory: Elements stored in sequential memory locations

Real-Time Example:

Student Grade Array: A classroom has 30 students with grades stored in an array. To retrieve the 5th student's grade, you directly access array[4] in O(1) time. This is efficient for scenarios where you need frequent random access without insertion/deletion.

Advantages and Disadvantages:

Advantages: Fast random access (O(1)), memory efficient, cache-friendly

Disadvantages: Fixed size, expensive insertion/deletion (O(n)), memory wastage if not fully used

Linked Lists

Linked lists are linear data structures where elements (nodes) are connected via pointers/references. Unlike arrays, they are not stored in contiguous memory.

Singly Linked List

Each node contains data and a pointer to the next node only. The last node points to NULL.

Structure:

Node Structure: [Data | Next Pointer] → [Data | Next Pointer] → [Data | NULL]

Key Operations:

Insertion: O(n) in general, O(1) at head

Deletion: O(n) in general, O(1) at head

Traversal: O(n)

Search: O(n)

Real-Time Example:

Browser History: When you navigate through web pages, each page visit is a node containing the URL and a pointer to the next visited page. Going back through history is simple as you move through the linked list backwards.

Advantages and Disadvantages:

Advantages: Dynamic size, efficient insertion/deletion at known positions (O(1)), no memory wastage

Disadvantages: No random access (O(n)), extra memory for pointers, cache-unfriendly

Doubly Linked List

Each node contains data, a pointer to the next node, and a pointer to the previous node. This allows bidirectional traversal.

Structure:

Node: [Prev | Data | Next] ← → [Prev | Data | Next] ← → [Prev | Data | Next]

Key Advantages:

Bidirectional Traversal: Can move forward and backward

Efficient Deletion: No need to traverse to find previous node

Flexible: Better for algorithms requiring backward traversal

Real-Time Example:

Music Playlist: A music player with previous/next buttons implements a doubly linked list. Each song node points to the next and previous tracks, allowing you to navigate both forward and backward through your playlist efficiently.

Circular Linked List

A variant where the last node points back to the first node instead of NULL, forming a circle.

Structure:

Circular: [Data | Next] → [Data | Next] → [Data | Next] ↻ (back to first)

Key Characteristics:

No NULL termination: Last node points to first node

Continuous Traversal: Can traverse indefinitely

Memory Efficient: Single pointer to any node accesses entire list

Real-Time Example:

Round-Robin Scheduling: Operating systems use circular linked lists for CPU scheduling. Each process gets a time slice, and after its turn, the pointer moves to the next process. The circular structure ensures fair rotation among all processes waiting for CPU time.

Another Real-Time Example:

Traffic Light Controller: Traffic lights cycle through Red → Yellow → Green → Red continuously. A circular linked list perfectly represents this where each color node points to the next, and the last (Green) points back to the first (Red).

Stack

A Last-In-First-Out (LIFO) data structure where insertions and deletions occur at the same end called the top.

Key Operations:

Push: Add element to top - O(1)

Pop: Remove element from top - O(1)

Peek: View top element - O(1)

isEmpty: Check if stack is empty - O(1)

Implementation:

Array-based: Fixed size, but efficient for known maximum size

Linked List-based: Dynamic size, suitable for variable stack requirements

Real-Time Examples:

1. Function Call Stack: When functions call each other in a program, each function call is pushed onto the call stack. When a function returns, it's popped off. If Function A calls Function B which calls Function C, the stack looks like: [A | B | C (top)]. C executes first and returns, then B, then A.

2. Undo/Redo Mechanism: Text editors use stacks for undo functionality. Each action (typing, formatting) is pushed onto an undo stack. When you press Ctrl+Z, the most recent action is popped and reversed. Pressing Ctrl+Y pushes it to a redo stack.

3. Browser Back Button: Your browsing history uses a stack. Each page visited is pushed onto the stack. Clicking back pops the current page, revealing the previous one.

4. Expression Evaluation: Converting infix expressions (A+B) to postfix (AB+) and evaluating them uses stacks. This is fundamental in compiler design.

Queue

A First-In-First-Out (FIFO) data structure where insertions occur at the rear and deletions at the front.

Key Operations:

Enqueue: Add element to rear - O(1)

Dequeue: Remove element from front - O(1)

Peek: View front element - O(1)

isEmpty: Check if queue is empty - O(1)

Variants:

Circular Queue: Reuses space, more efficient memory utilization

Priority Queue: Elements dequeued based on priority, not just insertion order

Double-Ended Queue (Deque): Insertion and deletion from both ends

Real-Time Examples:

1. ATM Queue Management: Customers arriving at an ATM are added to a queue (enqueue). The customer at the front uses the ATM, and when done, is removed (dequeue). New customers join the rear. This ensures fair FIFO service.

2. CPU Task Scheduling: Operating systems queue processes waiting for CPU execution. Processes are added to the ready queue and scheduled in FIFO order, ensuring no process starves indefinitely.

3. Printer Queue: Documents sent to a printer are queued. The printer processes documents in the order they were sent, not randomly. First document sent is printed first.

4. Customer Service: Call centers use queues. Incoming calls are queued, and agents handle them in FIFO order. When an agent becomes free, the next customer in the queue is served.

5. Message Broadcasting: Social media platforms use queues for processing user messages and notifications, ensuring messages are delivered in order.

Applications of Linear Data Structures

Arrays:

Database Records: Store rows of data for fast row access

Image Processing: 2D arrays represent pixel grids

Sparse Matrices: Represent mathematical matrices

Linked Lists:

Dynamic Memory Allocation: Memory allocated as needed

Polynomial Representation: Each term as a node

Symbol Tables: In compilers, store variable information

Stacks:

Compiler Design: Syntax analysis, expression evaluation

Memory Management: Function call stack, local variables

Backtracking Algorithms: Maze solving, N-Queens problem

Queues:

Operating System: Process scheduling, I/O buffer management

Network Protocols: Packet routing, bandwidth management

Simulation: Model real-world systems like hospitals, banks