for stack

definition

A stack is a fundamental linear data structure that operates based on the Last In, First Out (LIFO) principle. Imagine a stack of plates in a cafeteria. The first plate placed on the table (the bottom plate) is the last one to be removed. In computer science terms, a stack is an abstract data type (ADT) that allows two core operations: Push: This operation adds a new element to the collection. The new element is placed on the "top" of the stack. Pop: This operation removes the most recently added element from the collection, which is always the element at the "top" of the stack. Think of the top as the open end of the stack where elements are added and removed. - LIFO Order: Elements are accessed and manipulated in strict LIFO order. This is in contrast to a queue, which follows a First In, First Out (FIFO) order. - Limited Access: Only the top element of the stack is directly accessible. Retrieving elements in the middle or bottom requires popping elements off the stack until the desired element is reached. - Dynamic Size (Optional): Stacks can be implemented with a fixed size or can grow and shrink dynamically as elements are added and removed. Stacks are versatile data structures with numerous applications due to their efficient LIFO behavior and simple memory management.

applications

- Expression Evaluation
- Backtracking
- Delimiter Checking
- Reversing Data
- Processing Function Calls
- Undo/Redo Functionality
- Browser History
- Syntax Parsing
- Data Structure Conversion
- Recursion
- Graph Algorithms
- Tower of Hanoi
- Assembly Language
- Calling Convention
- Compilers and Interpreters

Advantages

- Efficiency
- LIFO (Last In, First Out)
- Simple Memory Management
- Limited Memory Usage

Disadvantages

- Limited Access
- Potential for Overflow
- Not Suitable for Random Access
- Limited Capacity

queue

Definition:

A queue is a linear data structure that adheres to the First In, First Out (FIFO) principle. Imagine a line of people waiting for a service. The first person in line is the first one to be served. In computer science, a queue is an abstract data type (ADT) that supports two basic operations: - Enqueue: This operation adds a new element to the back (rear) of the queue. - Dequeue: This operation removes the element that has been waiting at the front of the queue the longest. In summary, queues offer efficient FIFO order management and are relatively simple to implement. However, they have limitations in terms of access and removal from specific positions. Choosing between stacks and queues depends on the specific needs of your application.

Applications:

- Task Scheduling
- Data Processing
- Simulation Modeling
- Breadth-First Search (BFS)
- Message Passing
- Multitasking
- Device Buffering
- Priority Queues
- Level Order Tree Traversal
- Job Scheduling
- Network Packet Processing
- Audio/Video Playback
- Event Processing
- Asynchronous Communication
- Algorithm Design

Advantages:

- Efficient FIFO Order
- Simple Implementation
- Dynamic Size (Optional)

Disadvantages:

- Limited Access
- Inefficient Removal from Specific Position
- Potential for Overflow

Linked List

Definition

A linked list is a linear data structure where elements, called **nodes**, are not stored in contiguous memory locations. Instead, each node contains data and a reference (or pointer) to the next node in the sequence. This pointer linkage creates a chain-like structure, allowing the linked list to grow or shrink dynamically as needed. - Structure of a Node A typical node in a linked list has two components: - Data This field stores the actual information the linked list holds. The data type can vary depending on the application, such as integers, strings, or even complex objects. - Next Pointer This pointer references the next node in the list. The last node's pointer typically points to `null` to signify the end of the list.

Types of Linked Lists

- Singly Linked List
- Doubly Linked List
- Circular Linked List

Applications

Linked lists have a diverse range of applications due to their dynamic nature and flexibility - Dynamic Data Structures - Efficient Insertions and Deletions - Sparse Data Representation - Implementing Other Data Structures - Symbol Tables - Polymorphism Handling - Sparse Matrices - Undo/Redo Functionality in Text Editors - Music Playlists - File Allocation - Free Memory Management - Implementing Graphs (Adjacency List) - Implementing Hash Tables with Chaining - Large Number Arithmetic

Advantages

- Dynamic Size
- Efficient Insertions/Deletions
- No Memory Wastage
- Implementation of Complex Data Structures

Disadvantages

- Slower Random Access

- More Memory Overhead
- Cache Inefficiency

Binary Search Tree (BST)

Definition:

A BST is a tree data structure where each node has at most two child nodes:

- Left Child: The value of the left child node is less than the value of the parent node.
- Right Child: The value of the right child node is greater than the value of the parent node.

This parent-child relationship ensures that as you traverse the tree:

- Moving left leads to progressively smaller values.
- Moving right leads to progressively larger values.

Advantages:

- Efficient Searching
- Ordered Traversal
- Dynamic Data Structure

Disadvantages:

- Not Self-Balancing
- Limited Functionality
- Memory Overhead

Applications:

- Symbol Tables
- File Indexing
- Priority Queues
- Data Validation
- Spell Checkers
- In-Memory Databases
- Machine Learning
- Game Development
- Network Routing
- Data Compression
- Image Processing
- Cryptography
- Algorithmic Puzzles
- Syntax Highlighting
- Static Code Analysis

B-Tree

Definition:

- A B-Tree is a self-balancing tree data structure designed for efficient storage and retrieval of large datasets, particularly when dealing with disk-based storage. It addresses limitations of binary search trees (BSTs) by allowing each node to contain multiple keys and child pointers.
- A B-Tree is a tree data structure with the following properties:

Each node can hold a minimum of m keys (where m is a predefined order) and m+1 child pointers. Keys within a node are stored in sorted order. All leaves (nodes without child pointers) are at the same level. Search, insertion, and deletion operations maintain specific balance conditions to ensure efficient performance.

Advantages:

- Efficient Search
- Scalability
- Self-Balancing

Disadvantages:

- Increased Complexity
- Memory Overhead
- Search Overhead

Applications:

- Database Indexing
- File Systems
- Search Engines
- Geospatial Indexing
- Network Routing
- Cache Management
- Machine Learning
- Network Security
- Time Series Databases
- Computational Geometry
- Cryptography
- Natural Language Processing (NLP)
- Data Compression Tools
- Virtual Memory Management
- In-Memory Databases

hashing shit

types

- fold and add
- middle square method
- radix transformation

prediction

- binomial formula
- poission formula

collision resolution

- linear progressive overflow
- quadratic progressive overflow
- buckets
- double hashing
- chained progressive