# DMI COLLEGE OF ENGINEERING

**(An Autonomous Institution)**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CS1202- DATA STRUCTURES**

**PART A (2 Marks) Questions & Short Answers**

## UNIT I - LISTS

1. **Define Data Structure. List some common data structures.**
   A data structure is a way to store and organize data. It helps in accessing and modifying data efficiently. Common types include Array, Linked List, Stack, Queue, Tree. Each has its own use depending on the problem.

2. **Define ADT. Give any two examples.**
   ADT stands for Abstract Data Type. It defines what operations can be done, not how. Examples: Stack and Queue. They hide internal details and show only behavior.

3. **State the uses of Abstract Data type.**
   ADT helps in writing clean and reusable code. It improves modularity and hides implementation. Used in software design and problem solving. Makes programs easier to understand and maintain.

4. **Distinguish between linear and non-linear data structures.**

| Feature | Linear Data Structure | Non-Linear Data Structure |
|---|---|---|
| Data Arrangement | In a straight line | In a hierarchical format |
| Examples | Array, Stack, Queue | Tree, Graph |
| Traversal | One-by-one from start to end | Multiple paths possible |
| Complexity | Easier to implement | More complex and powerful |

5. **What are the advantages of linked lists over arrays.**
   Linked lists use dynamic memory allocation. Insertion and deletion are easier than arrays. No need to fix size in advance. Memory is used efficiently without wastage.

6. **What is the use of Header node in a linked list.**
   Header node stores starting info of the list. It helps in easy traversal and management. Often used to simplify operations. It may also store metadata like size.

7. **Define multilist.**
   Multilist connects nodes in multiple directions. Each node may belong to more than one list. Used in complex data like schedules. Helps in managing grouped or shared data.

8. **Infer the usage of Multilists.**
   Multilists help in managing grouped data. Used in databases and matrix storage. They allow flexible access paths. Useful in applications with multiple relationships.

9. **What is circular linked list.**
   Last node points back to the first node. It forms a loop with no NULL at end. Used in round-robin scheduling. Traversal continues without stopping.

10. **Define List ADT.**
    List ADT stores ordered elements. Supports insert, delete, and search. Can be array-based or linked. Used in many applications like memory management.

11. **What are the ways of implementing linked list.**
    Linked lists can be singly, doubly, or circular. Each uses pointers to connect nodes. Implemented using dynamic memory. Choice depends on the application need.

12. **What is sentinel node in the Linked list? Mention its use.**
    Sentinel node is a dummy node. It simplifies edge cases in operations. Used to avoid NULL checks. Helps in easier insertion and deletion.

13. **How the singly linked lists can be represented.**
    Each node has data and next pointer. Last node points to NULL. Traversal is one-way only. Used for simple linear operations.

14. **How the doubly linked list can be represented.**
    Each node has data, next and previous pointers. Allows movement in both directions. First node's previous is NULL. Used for flexible navigation.

15. **List the advantages of doubly Linked List.**
    Supports two-way traversal. Easy deletion without full scan. More flexible than singly list. Can go forward and backward easily.

16. **List down the applications of List.**
    Used in memory management and compilers. Helps in task scheduling and symbol tables. Also used in polynomial operations. Supports undo/redo features in editors.

17. **What are the operations performed in list.**
    Insert, delete, search, update, traverse. Operations depend on list type. Used in many algorithms. Helps in managing dynamic data.

18. **What are the merits and demerits of array implementation of lists.**
    Merits: Fast access, simple structure. Demerits: Fixed size, costly insert/delete. Less flexible than linked list. Wastes memory if size is overestimated.

19. **What is a Singly linked list.**
    Each node points to the next node. Last node points to NULL. Traversal is forward only. Used for simple linear data storage.

20. **How circular linked list is represented from singly Linked List.**
    Last node's next points to first node. No NULL at end. Forms a loop for continuous access. Used in circular buffers and scheduling.

**21. How circular linked list is represented from doubly linked list.**

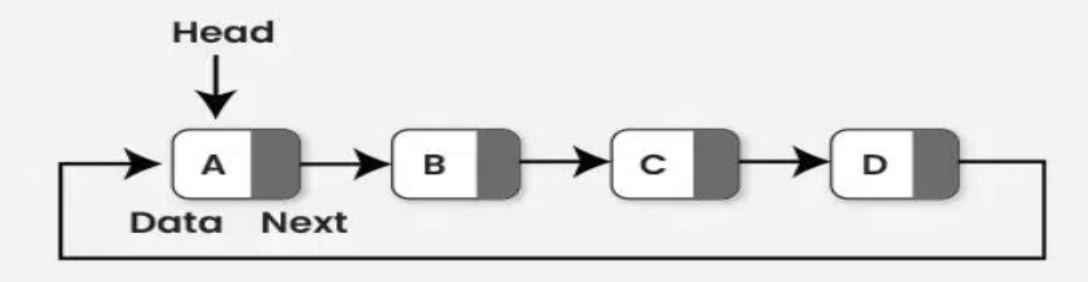
Last node's next points to first. First node's previous points to last. Supports looping in both directions. Used in advanced scheduling systems.

**22. Differentiate single linked list with circular linked list.**

| Feature | Singly Linked List | Circular Linked List |
|---|---|---|
| End Node | Points to `NULL` | Points back to the **first node** |
| Traversal | Stops at `NULL` | Loops continuously until stopped |
| Direction | One-way only | One-way or two-way (if doubly circular) |
| Use Case | Simple linear operations | Round-robin scheduling, circular buffers |

**23. Outline a Circular linked list with a diagram.**

Circular list connects last node to first. Traversal continues without NULL. Used in circular queues and buffers. See circular linked list diagram for visual.




**24. How can a polynomial be represented using a singly linked list.**

Each node stores coefficient and exponent. Nodes are linked in decreasing exponent order. Used for polynomial addition and multiplication. Helps in dynamic polynomial operations.

**25. List three examples that use linked list.**

- Task Scheduling in Operating Systems
- Polynomial Operations in Mathematics
- Undo/Redo Feature in Text Editors

## UNIT II - STACKS AND QUEUES

**1. Define Stack**

A stack is a linear data structure that follows Last In, First Out (LIFO). The last item added is the first removed. You can only insert or remove from the top. Like a plate stack — top plate goes out first.

**2. What are the operations of the stack?**

Push adds an item to the top, Pop removes the item from the top, Peek shows the top without removing, IsEmpty checks if stack is empty.

**3. Write the routine to push an element into a stack**

```
#define MAX_SIZE 100
int stack[MAX_SIZE];
int top = -1;

void push(int element) {
    if (top < MAX_SIZE - 1) {
        top++;
        stack[top] = element;
    }
}
```

**4. How are the operations performed on linked list implementation of a stack?**
Push creates a new node and links it to the old top; then updates top. Pop removes the top node and moves top to next. No fixed size. Uses dynamic memory.

**5. What are the applications of stack?**
Used to evaluate expressions, undo actions in editors, backtracking algorithms, syntax checking like brackets, and call stack management in programs.

**6. What are the methods to implement stack in C?**
Using arrays for fixed-size stacks or linked lists for dynamic ones. Both support push, pop, and peek operations. Linked lists avoid overflow issues.

**7. How is the stack implemented by linked list?**
Each node stores data and a pointer. The top points to the latest node. Push adds node at front, Pop removes from front. Structure is dynamic.

**8. Write the routine to pop an element from a stack**

```
int pop() {
    if (top >= 0) {
        int value = stack[top];
        top--;
        return value;
    }
}
```

**9. Define queue**
Queue is a linear data structure that works as First In First Out (FIFO). Items are added at rear and removed from front. Like a line at a counter.

**10. What is the disadvantage of linear queue? How to overcome it?**
After deleting items, space at front is wasted. When rear hits the end, no more insertions. Circular queue solves this by reusing space at front.

**11. Write the routine to insert an element onto a queue**

```
#define MAX_SIZE 100
int queue[MAX_SIZE];
int front = 0;
int rear = -1;

  void enqueue(int element) {
     if (rear < MAX_SIZE - 1) {
        rear++;
        queue[rear] = element;
     }
  }
```

**12. What are the types of queue?**
Simple Queue uses FIFO order. Circular Queue connects end to front. Deque allows insertion/deletion at both ends. Priority Queue works by priority.

**13. Define double-ended queue**
Deque allows insertion and deletion from both front and rear. It offers flexibility over normal queue. It can behave like stack or queue as needed.

**14. What are the methods to implement queue in C?**
Using arrays for fixed size or linked lists for dynamic structure. Arrays are simple, linked lists grow easily. Both support enqueue and dequeue.

**15. How is the queue implemented by linked list?**
Each node has data and pointer to next. Front points to first node, rear to last. Enqueue adds at rear, Dequeue removes from front.

**16. Write the routine to delete an element from a queue**

```
int dequeue() {
   if (front <= rear) {
      int value = queue[front];
      front++;
      return value;
   }
}
```

**17. What are the applications of queue?**
Used for CPU scheduling, disk scheduling, BFS traversal in graphs, print job handling, and managing requests in systems like servers and OS.

**18. Define circular queue**
Circular queue connects last position to first to reuse space. It avoids waste when front has gaps. Helps utilize full array space efficiently.

**19. Define Priority Queue**

Elements are removed based on priority instead of order. Higher priority items go first. Useful in operating systems and simulations.

**20. Distinguish between stack and queue**

| Aspect | Stack | Queue |
|---|---|---|
| Principle | LIFO (Last In First Out) | FIFO (First In First Out) |
| Insertion | Push at top | Enqueue at rear |
| Deletion | Pop from top | Dequeue from front |

**21. Convert the infix (a+b)(c+d)/f into postfix & prefix**

Infix: (a+b)(c+d)/f

Postfix: ab+cd+f/

Prefix: /*+ab+cdf Use operator precedence and stack conversion rules.

**22. Write postfix form of –A+B-C+D**

Assuming left-to-right operations: A- B + C - D + Operators handled based on order and precedence.

**23. Evaluate postfix expression 2 3 1 * + 9 -**

1. Push **2**
2. Push **3**
3. Push **1**
4. *****: Pop **1** and **3**, multiply → **3 * 1 = 3**, push **3**
5. **+**: Pop **3** and **2**, add → **2 + 3 = 5**, push **5**
6. Push **9**
7. **-**: Pop **9** and **5**, subtract → **5 - 9 = -4**

**Result: -4**

**24.Check the symbols are balanced [(2+3)*{5/1)+2}] using Stack**

1. **Read [**: push [
2. **Read (**: push (
3. **Read ]**: top is [, matches ]? Yes → pop [
4. **Read *{**: push {
5. **Read (**: push (
6. **Read }**: top is (, but }`` matches {`? No → mismatch!

Since we found a mismatch with }, the symbols are **not balanced**.

Thus, **the symbols are not balanced**.

**25.How do you test for an empty queue?**

To test for an empty queue, check if the front pointer (or index) is equal to the rear pointer (or index) in a queue implementation. If both are equal, the queue is empty. In some implementations, a separate boolean or count variable indicates emptiness. Always verify the specific condition based on the queue type used (array or linked list).

# UNIT III – TREES

1. **Construct a binary tree for the expression (a+b) * (c - d).**
   To construct the binary tree, start with the operator * as the root.Create two subtrees: one for (a+b) and another for (c-d).Connect + to its children a and b, and - to c and d.Finally, connect both subtrees to the root *.

2. **What is the difference between B tree and B+ tree?**
   A B tree stores data in both internal and leaf nodes, allowing efficient searches.In a B+ tree, all data is stored only in the leaf nodes, linked for easy traversal.Internal nodes in B+ trees only contain keys for guiding search paths.B+ trees are optimized for range queries and are mainly used in databases.

3. **What are the steps to convert a general tree into binary tree?**
   Choose the first child of each node and make it the left child in the binary tree.
   Link all other children of the node as right siblings of the first child.
   Apply the same process recursively to each child and sibling in the tree.
   This method preserves the structure of the general tree within the binary tree format.

5. What are the applications of binary trees?
   - Used in **Binary Search Trees (BST)** to search data quickly
   - Help in **sorting data** using tree traversal methods
   - Used in **expression trees** for evaluating math expressions in compilers
   - Employed for **hierarchical storage** in file systems and **indexing** in databases

4. **Distinguish between binary tree & binary search tree.**

| Feature | Binary Tree | Binary Search Tree (BST) |
|---|---|---|
| **Node Arrangement** | No specific order; nodes can be arranged arbitrarily | Ordered: left child < parent < right child |
| **Search Efficiency** | Slower search operations | Faster search due to sorted structure |
| **Use Cases** | Used in general hierarchical structures, e.g., file systems | Used for efficient data retrieval and searching |
| **Operations (Insert/Delete)** | May require full traversal | Optimized with minimal traversal steps |

6. **The depth of a full binary tree is 8. Compute the number of leaf nodes.**
   Identify the depth of the full binary tree, which is 8 in this case.
   Use the formula: Number of leaves = 2 raised to the power of the depth.
   Calculate: 2^8 = 256.
   Answer: The tree has 256 leaf nodes at depth 8.

7. **What are the two methods of binary tree implementation?**
   The first method uses an array to represent the binary tree, suitable when the tree is complete.
   The second method uses linked nodes, where each node points to its child nodes.
   Array implementation offers easy access but can waste space in skewed trees.
   Linked list implementation is flexible and dynamic but uses more memory.

**8. What are the properties of AVL trees?**

AVL trees are self-balancing binary search trees.
They maintain a balance factor for each node: height of left subtree minus height of right subtree.
The balance factor must be -1, 0, or 1 for all nodes.
Rotations are used to restore balance after insertions or deletions.

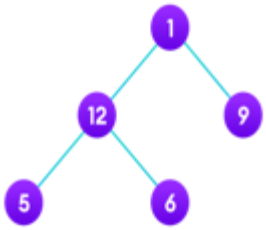
**9. Mention the types of rotations performed on AVL tree.**

Left rotation is used when the right subtree becomes heavy.
Right rotation is used when the left subtree becomes heavy.
Left-right rotation is a combination to balance a left-right heavy subtree.
Right-left rotation balances a right-left heavy subtree.

**10. Define expression trees.**

An expression tree is a binary tree where leaves are operands, and internal nodes are operators.
It represents the structure of a mathematical expression.Evaluation of the expression is done by traversing the tree in postorder.It simplifies parsing and calculating expressions in compilers and calculators.

**11. Perform Inorder, Preorder and Postorder Traversal in the given graph**




Inorder Traversal (Left, Root, Right) 5, 12, 6, 1, 9
Preorder Traversal (Root, Left, Right) 1, 12, 5, 6, 9
Postorder Traversal (Left, Right, Root) 5, 6, 12, 9, 1

**12. Compare full binary tree and skewed binary tree.**

| Feature | Full Binary Tree | Skewed Binary Tree |
|---|---|---|
| Structure | Each node has 0 or 2 children | Each node has only one child |
| Balance | Perfectly balanced | Unbalanced; linear in shape |
| Performance Efficiency | Efficient for search and traversal | Poor performance due to increased height |
| Visual Representation | Branches symmetrically | Looks like a linked list (left or right skew) |

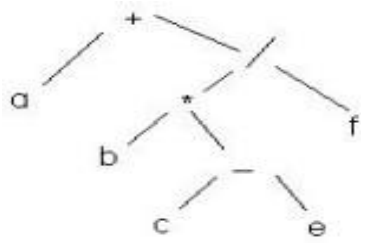
**13. What are the different ways of representing a Binary Tree?**

Binary trees can be represented using pointers (linked list).
They can also be stored in arrays for complete binary trees.
Pointer-based representation allows flexible and dynamic structure.
Array-based is simple for complete trees but inefficient for skewed trees.

**14. Distinguish between binary search tree and AVL tree.**

| Feature | Binary Search Tree (BST) | AVL Tree |
|---|---|---|
| **Balance Maintenance** | May become unbalanced over time | Maintains strict balance using rotations |
| **Operation Efficiency** | Slower if unbalanced | Faster due to height balancing |
| **Implementation Complexity** | Easier to implement | More complex due to balancing logic |
| **Insert/Delete Behavior** | Simple insert/delete, no balance enforcement | Adjusts structure during insert/delete |

**15. Give the prefix & postfix form of the expression: (a + (b * c) + d).**




**Prefix Notation (Preorder Traversal) :**+ a / * b - c e f

**Postfix Notation (Postorder Traversal):**a b c e - * f / +

**16.What is a heap? Outline properties of a heap.**

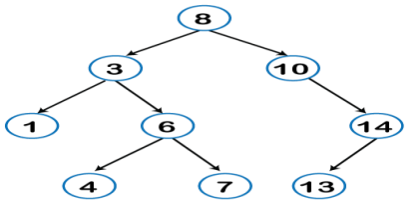
A heap is a complete binary tree satisfying the heap property.
In max-heap, each parent node is greater than or equal to its children.
In min-heap, each parent node is less than or equal to its children.
Heaps are used in priority queues and heap sort algorithms.

**17. Why Binary Search Tree is Binary Tree but not Binary Search Tree?**

- A Binary Search Tree (BST) is a special Binary Tree where left < root < right — it follows specific ordering rules.
- A Binary Tree simply allows each node to have up to two children, with no value constraints.
- Every BST is a Binary Tree because it satisfies the structural rule.
- Not every Binary Tree is a BST — it may violate the ordering condition.

**18. For the tree given below**

(i) List the leaf nodes
(ii) List the Siblings of 6
(iii) Compute the height




(i) Leaf nodes: 4, 7, 13, 1 (ii) Siblings of 6: 1 (iii) Height: 3

**19. What is a complete binary tree?**

A complete binary tree has all levels filled except possibly the last.
In the last level, nodes are filled from left to right.
It is efficient for array-based storage, such as heap implementation.
Ensures minimal height and optimal operations.

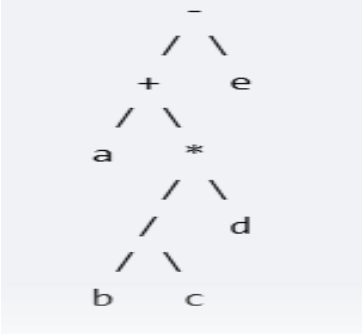
**20. What is a Heap Tree?**

A heap tree is a complete binary tree used to implement priority queues.
It satisfies the heap property: parent ≥ children (max-heap) or ≤ children (min-heap).
It allows efficient insertion and deletion with O(log n) complexity.
Used in algorithms like heapsort and priority scheduling.

**21.Why AVL tree is binary Search tree but Binary Search Tree is not AVL tree?**

- An AVL tree is a binary search tree that has extra rules to stay balanced.
- AVL trees fix themselves after insert or delete to keep their shape even.
- A normal binary search tree can become uneven and slow because it doesn't balance itself.
- **All AVL trees follow BST rules,** but **not all BSTs are AVL**—only the ones that keep their
- Balance.

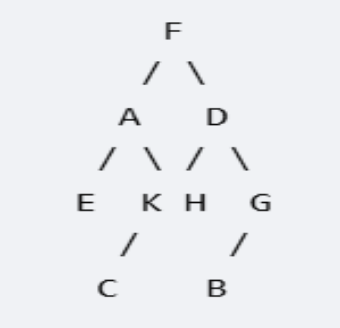
**22. Create an expression tree for the expression. ((a + ((b/c)*d))- e)**




**23. How do you calculate the balance factor for an AVL tree?**

Calculate the height of the left subtree and the height of the right subtree.
Subtract the height of the right subtree from the left.
Balance factor = height(left) - height(right).
If the balance factor is outside -1, 0, 1, rotations are needed to rebalance.

**24.A binary tree T has 9 nodes . The inorder and postorder traversals of T yield the following:**
**Inorder Traversal (I) : E A C K F H D B G,Post Traversal (Po): E C K A H B G D F**
**Draw the binary tree**



**25.How do you test for an empty queue?**

- Check if the front and rear pointers are equal or both null
- Verify if a size or count variable equals zero
- If either condition is true, the queue is considered empty
- Helps prevent invalid enqueue/dequeue operations and maintains queue integrity

## UNIT IV – GRAPHS

**1. What is meant by bi-connected graph?**

A bi-connected graph stays connected even if any one vertex is removed. It has no articulation points (critical nodes). There are two disjoint paths between every pair of vertices. It helps in designing fault-tolerant networks. Diagram and explanation

**2. Define Hamiltonian circuit**

A Hamiltonian circuit visits every vertex exactly once and returns to the starting point. It forms a closed loop in the graph. Not all graphs have Hamiltonian circuits. Used in problems like the Traveling Salesman Problem. Visual example

**3. Define Graph?**

A graph is a collection of nodes (called vertices) and connections (called edges). It shows relationships between objects. Graphs can be directed or undirected. Used in networks, maps, and algorithms. Definition

**4. What is connected graph?**

A connected graph has a path between every pair of vertices. No vertex is isolated. If any two vertices are reachable, the graph is connected. Used in communication and transport networks. Explanation

**5. How many edges in a complete graph that contain n nodes**

A complete graph connects every pair of vertices. It has $n(n-1)/2$ edges. Each node connects to all others. Used in dense network modeling. Formula proof

**6. What are Euler circuits?**

An Euler circuit uses every edge exactly once and returns to the start. All vertices must have even degree. It's a closed trail covering all edges. Used in route planning and puzzles. Example

**7. Define adjacency list**

An adjacency list stores each vertex and its connected neighbors. It uses less space than a matrix. Efficient for sparse graphs. Used in BFS and DFS algorithms. Explanation

**8. Write the difference between tree and graph**

| Feature | Tree | Graph |
|---|---|---|
| Structure | Hierarchical | Network-like |
| Cycles | No cycles | May have cycles |
| Connectivity | Always connected | May be disconnected |
| Edge count | n−1 edges for n nodes | Varies |

**9. What is digraph? Give an example**
A digraph is a directed graph where edges have direction. Each edge goes from one vertex to another. Used in traffic flow and task scheduling. Example: A → B → C Visual

**10. How floyd's algorithm works?**
Floyd's algorithm finds shortest paths between all pairs of vertices. It uses a distance matrix and updates paths iteratively. Works for both directed and weighted graphs. Time complexity is $O(n^3)$. Working steps

**11. What is the use of Dijkstra's algorithm?**
Dijkstra's algorithm finds the shortest path from a source to all other nodes. It works with non-negative edge weights. Used in GPS and network routing. Efficient for sparse graphs. Use case

**12. What is a complete Graph.**
A complete graph connects every pair of vertices. Each node has an edge to all others. It has $n(n-1)/2$ edges for n nodes. Used in dense network modeling. Definition

**13. Define minimum cost spanning tree?**
A minimum cost spanning tree connects all vertices with least total edge weight. It has no cycles. Used in network design and wiring. Algorithms: Kruskal's and Prim's. Explanation

**14. Define Adjacency Matrix**
An adjacency matrix is a 2D array showing edge connections. If there's an edge between i and j, matrix[i][j] = 1. Used for dense graphs. Easy to implement but uses more space. Details

**15. Define topological sort?**
Topological sort orders vertices in a directed acyclic graph (DAG). It ensures that for every edge u → v, u comes before v. Used in task scheduling. Algorithms: DFS or Kahn's. Explanation

**16. What is a cycle or a circuit?**
A cycle is a path that starts and ends at the same vertex. It visits other vertices only once. A circuit is a closed trail. Used in detecting loops in graphs. Definition

**17. What is an acyclic graph?**
An acyclic graph has no cycles. In directed graphs, it's called a DAG. Used in dependency resolution. Example: Task scheduling. Explanation

**18. Define Basic Operations of Graph**
Basic operations include adding/removing vertices and edges. Traversal: BFS and DFS. Searching paths and checking connectivity. Used in algorithms and network analysis. Operations

**19. What is meant by strongly connected in a graph?**
In a directed graph, it's strongly connected if every vertex is reachable from every other. There's a path from u to v and v to u. Used in web crawling and network design. Definition

**20. Define shortest path problem** It finds the minimum distance between two vertices. Used in maps, routing, and logistics. Algorithms: Dijkstra, Bellman-Ford, Floyd-Warshall. Can be single-source or all-pairs. Explanation

**21. Outline a directed graph with an example**

A directed graph has edges with direction. Example: A → B, B → C, C → A
Used in flowcharts and dependency graphs. Visual

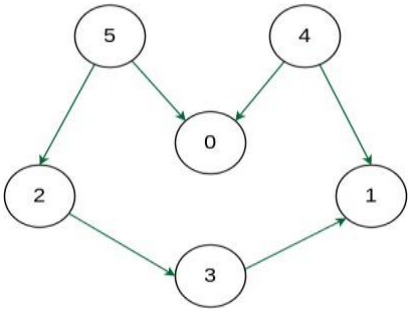
**22. Why tree is a graph & graph is not a tree prove with an example**

| Feature | Tree | Graph |
|---|---|---|
| Definition | A connected acyclic graph | May be cyclic or disconnected |
| Example | A → B → C | A → B, B → C, C → A (cycle) |
| Edge count | n−1 edges | Varies |
| Cycles | No | May have |

**23. Express how a graph differs from a spanning tree with an example**

| Feature | Graph | Spanning Tree |
|---|---|---|
| Cycles | May have cycles | No cycles |
| Edge count | Can be more than n−1 | Exactly n−1 edges |
| Connectivity | May be disconnected | Always connected |
| Example | Graph with extra edges | Tree covering all vertices |

**24.Give the topological ordering for the DAG of given graph**




**Topological Order**: A → B → C → E → D → F

**25. How BFS differ from DFS**

| Feature | BFS | DFS |
|---|---|---|
| Traversal | Level by level | Depth first |
| Data Structure Used | Queue (FIFO) | Stack (LIFO) |
| Path Discovery | Finds shortest path in unweighted graphs | May not find shortest path |
| Memory Usage | Requires more memory for wide graphs | Uses less memory for deep graphs |

## UNIT V -SEARCHING, SORTING AND HASHING TECHNIQUES

**1.What are the various factors to be considered in deciding a sorting algorithm**

- Type and size of the data to be sorted
- Amount of memory available during execution
- Time complexity and how quickly the algorithm runs
- Whether the algorithm maintains the original order of equal elements (stability)

**2. Distinguish between linear and binary search technique**

| Feature | Linear Search | Binary Search |
|---|---|---|
| Data Type | Any order | Must be sorted |
| Speed | Slower for large data | Faster with sorting |
| Complexity | O(n) | O(log n) |
| Comparison Method | Checks each item | Checks middle & splits |

**3. Write the routine for Insertion sort**

```
void insertionSort(int a[], int n) {
    for (int i = 1; i < n; i++) {
        int key = a[i], j = i - 1;
        while (j >= 0 && a[j] > key) a[j + 1] = a[j--];
        a[j + 1] = key;
    }
}
```

**4.Define sorting**

Sorting means arranging items in order, either increasing or decreasing. Helps in faster searching and easy analysis. Used in databases, reports, and algorithms. Sorting improves performance of many applications.

**6. Distinguish between internal and external sorting?**

| Feature | Internal Sorting | External Sorting |
|---|---|---|
| Data Size | Small, fits in RAM | Large, stored externally |
| Speed | Fast | Slow due to disk I/O |
| Usage Area | In-memory operations | Big data applications |
| Examples | Quick, bubble sort | Merge, polyphase sort |

**7. Define Double hashing.**

Double hashing uses two formulas to find storage index. If first index is full, it tries second using hash2. Formula: (h1(key) + i*h2(key)) % table_size. Helps avoid clustering and ensures better distribution.

**8. Define bubble sort**

Bubble sort compares adjacent items and swaps them if needed. Repeats multiple passes until all items are sorted. Simple and easy to understand but slow for big data. Time complexity is $O(n^2)$, best case O(n) if already sorted.

**8. What are the steps for selection sort?**
Find smallest element in unsorted part of array. Swap it with first unsorted position. Repeat for next position till array is sorted. Simple but has time complexity of $O(n^2)$.

**9. What is meant by shell sort?**
Shell sort compares elements far apart in the beginning. Gap between compared elements reduces gradually. It's like improving insertion sort using gap method. Faster than bubble or insertion for medium data.

**10. What are the steps in Radix sort?**
Find max digit length in data. Sort numbers by each digit starting from least. Use counting sort as subroutine for each digit. Repeat till all digits are processed, list becomes sorted.

**11. What is open addressing?**
Open addressing handles collision by searching another empty spot. It probes table until empty index is found. Methods include linear, quadratic, and double hashing. All data is stored directly in hash table.

**12. What are the collision resolution methods?**
Linear probing: check next index. Quadratic probing: use squared gaps. Double hashing: use second hash function. Separate chaining: store items in a linked list.

**13. Define separate chaining**
Separate chaining stores items at same index using a linked list. Each bucket holds multiple values if collisions occur. Easy to implement and avoids long probing. Used when collisions are frequent.

**14. What are the use of hash table?**
Hash tables provide fast access to data using keys. Used in databases, language interpreters, and compilers. Support constant time insertion and searching. Efficient when number of items is large.

**15. Define searching**
Searching means locating a specific item from a data structure. Can be done using linear or binary methods. Used to find records in databases or lists. Improves efficiency in data handling.

**16. What are the problems in hashing?**
Collisions may occur when two keys point to same index. Poor hash functions create uneven distribution. Cluster formation reduces performance. Requires rehashing when table becomes too full.

**17. Differentiate between Insertion sort and Bubble sort?**

| Feature | Insertion Sort | Bubble Sort |
|---|---|---|
| Method | Inserts at correct spot | Swaps adjacent values |
| Efficiency | Good for nearly sorted | Slower for large sets |
| Best Case | $O(n)$ | $O(n)$ |
| Worst Case | $O(n^2)$ | $O(n^2)$ |

**18. What is closed addressing?**
Closed addressing is another term for separate chaining. Collisions are resolved by storing multiple items in a list. Each index may have several entries using linked lists. Helps reduce probing and manage collisions smoothly.

**19. What is Rehashing?**
Rehashing expands the size of the hash table when needed. All elements are reinserted using new hash function. Improves performance by reducing collisions. Used when load factor becomes high.

**20. What is Extendible Hashing?**
Extendible hashing adjusts directory size dynamically. It splits buckets when too many values are stored. Used in databases to manage large data. Avoids full-table rebuilding during expansion.

**21. Outline divide and conquer algorithm design paradigm**
Divide problem into smaller sub-problems. Conquer by solving each part recursively. Combine the results to get final solution. Examples: quick sort, merge sort, binary search.

**22. What is a hash function?**
Hash function converts data (key) into index values. Should distribute keys evenly across table. Simple example: key % table size. Used in hashing to speed up access time.

**23. Outline perfect minimal hashing function**
Perfect minimal hash assigns unique index with no collision. Used for fixed set of keys known in advance. Helps create compact and collision-free tables. Ideal for static datasets and compiler design.

**24. Identify the principal behind the external sorting algorithms**
Split big data into manageable chunks. Sort chunks using internal sorting method. Merge sorted chunks to get final output. Used when data exceeds main memory capacity.

**25. What are the different hash functions?**

- **Division method:** key % table_size
- **Multiplication method:** multiply key by a constant, use fractional part
- **Folding method:** split key into parts and add them
- **Universal hashing:** choose a hash function randomly from a family

**********