

**BHARATIVIDYAPEETH’S**

**INSTITUTEOFCOMPUTERAPPLICATIONS&MANAGEMENT (BVICAM)**

(AffiliatedtoGuruGobindSinghIndraprasthaUniversity,ApprovedbyAICTE,NewDelhi)A-4,PaschimVihar,RohtakRoad,NewDelhi-110063,Visitusat:<http://www.bvicam.in/>

**Course Code: MCA-109 Course Name: Data And File Structures**

Practice Questions (Theory)

|  |  |
| --- | --- |
| UNIT I | |
|  | In C, how do you declare and initialize a one-dimensional array? |
|  | What is the time complexity for accessing an element in an array by index? |
|  | Explain the concept of a singly linked list. How is it different from a doubly linked list? |
|  | How do you implement a stack using an array in C? Provide push and pop operations. |
|  | Define a queue in C. What are the basic operations that can be performed on a queue? |
|  | Describe the process of traversing an array in C using a loop. |
|  | Explain the logic behind traversing a linked list in C and accessing its elements. |
|  | How does a stack work in C? Provide an example scenario where a stack can be used. |
|  | Differentiate between a linear queue and a circular queue. Explain their advantages and disadvantages. |
|  | Discuss the concept of a doubly linked list in C and its applications. |
|  | Write a C program to find the sum of all elements in an array. |
|  | Implement a function in C to insert a node at the beginning of a linked list. |
|  | Develop a C program to reverse a stack using an auxiliary stack. |
|  | Create a circular queue in C and perform enqueue and dequeue operations. |
|  | Design a C program to merge two sorted arrays into a single sorted array. |
|  | Compare the time complexity of linear search and binary search algorithms for arrays in C. |
|  | Analyze the time complexity of inserting an element at the beginning and end of a linked list. |
|  | Investigate the memory usage of a stack implemented using arrays versus linked lists in C. |
|  | Examine the performance of a priority queue versus a regular queue in C. |
|  | Evaluate the efficiency of implementing a queue using a linked list compared to using arrays in C. |
|  | Assess the suitability of using arrays to implement a sparse matrix in C. |
|  | Critique the efficiency of various sorting algorithms (e.g., bubble sort, selection sort) for arrays in C. |
|  | Evaluate the effectiveness of using a linked list to implement a dynamic data structure in C. |
|  | Appraise the advantages and disadvantages of using stacks in recursion in C programming. |
|  | Judge the reliability of a queue data structure for handling concurrent operations in a multi-threaded C program. |
|  | In C, how do you declare and initialize a one-dimensional array? |
|  | What is the time complexity for accessing an element in an array by index? |
|  | Explain the concept of a singly linked list. How is it different from a doubly linked list? |
|  | How do you implement a stack using an array in C? Provide push and pop operations. |
|  | Define a queue in C. What are the basic operations that can be performed on a queue? |
|  | **UNIT II** |
|  | In C, how do you declare and initialize a binary tree node structure? |
|  | What is the height of a binary tree? How is it calculated? |
|  | Explain the properties of a binary heap. How is it different from a binary search tree? |
|  | Define the concept of a balanced tree. How does it relate to AVL trees and B-trees? |
|  | Describe the process of traversing a binary tree in C using depth-first search (DFS). |
|  | How does a max-heap differ from a min-heap? Provide an example of each. |
|  | Explain the process of balancing an AVL tree after insertion or deletion operations. |
|  | Discuss the rules for inserting and deleting nodes in a B-tree in C. |
|  | Write a C program to implement a binary search tree (BST) and perform insertion and deletion operations. |
|  | Implement a function in C to insert a node into a binary heap and maintain the heap property. |
|  | Develop a C program to balance an AVL tree after performing a series of insertions and deletions. |
|  | Create a B-tree in C and perform a search operation to find a specific key. |
|  | . Compare and contrast the time complexity of various tree traversal algorithms (e.g., in-order, pre-order, post-order). |
|  | Analyze the time complexity of heapify operation in a binary heap and its impact on heap construction. |
|  | Investigate the time complexity of rotation operations in an AVL tree and their effect on tree balancing. |
|  | Examine the performance of a B-tree in terms of search, insertion, and deletion operations compared to other tree data structures. |
|  | . Assess the suitability of using a binary search tree to store and retrieve sorted data efficiently in C. |
|  | Critique the efficiency of various heap operations (e.g., insert, delete, extract-min/max) in C. |
|  | Evaluate the effectiveness of AVL trees in maintaining balance and optimizing search performance in C. |
|  | Judge the reliability of B-trees for handling large datasets and supporting efficient range queries in C. |
|  | UNIT III |
|  | In C, how do you represent a graph using an adjacency matrix? Provide an example. |
|  | Define the concept of an edge in a graph. How is it different from a vertex? |
|  | Explain the difference between a directed graph and an undirected graph in C. Provide examples of each. |
|  | Describe the process of traversing a graph using depth-first search (DFS) in C. |
|  | Write a C program to implement a breadth-first search (BFS) algorithm for graph traversal. |
|  | Develop a C program to find the shortest path between two vertices in a graph using Dijkstra's algorithm. |
|  | Analyze the time complexity of performing depth-first search (DFS) on a graph with V vertices and E edges in C. |
|  | Examine the space complexity of representing a graph using an adjacency list versus an adjacency matrix in C. |
|  | . Evaluate the effectiveness of using breadth-first search (BFS) versus depth-first search (DFS) for finding connected components in a graph in C. |
|  | Judge the reliability of using Dijkstra's algorithm for finding shortest paths in a weighted graph with negative edge weights in C. |
|  |  |
|  | In C, how do you implement a hash table using chaining? Provide an example. |
|  | Define the term "collision" in the context of hashing. How is it handled in separate chaining? |
|  | . Explain the process of hashing a key to find its corresponding index in a hash table in C. |
|  | Describe the concept of open addressing in hashing. How does it differ from chaining? |
|  | Write a C program to implement a hash table using separate chaining for collision resolution. |
|  | Develop a C program to perform linear probing for collision resolution in a hash table. |
|  | Analyze the time complexity of searching, inserting, and deleting elements in a hash table implemented with separate chaining. |
|  | Examine the impact of load factor on the performance of a hash table in C. |
|  | Evaluate the efficiency of various collision resolution techniques in hashing, such as separate chaining and linear probing, in C. |
|  | Judge the reliability of using hashing for implementing data structures like sets and maps in C programming. |

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*Wish you luck!\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***