| **Course Code: AI233AI** | **Date:** 22**-**Feb-2024 |
| --- | --- |
| **Sem:** III | **Duration:** 90 Minutes |
| **CIE-II**  **Fundamentals of Data Structures and Data Analysis (DSDA**)  **Answer all the Questions** | |

| **SL. No** | | **Questions** | **M** | **BT** | **CO** |
| --- | --- | --- | --- | --- | --- |
| 1 | a) | Write an algorithm to create an expression tree for a valid postfix expression. Show the construction of an expression tree by considering **abc+\*d/**  Repeat while not the end of the expression P  2. Read the postfix expression one symbol S at a time  3. If S is an operand then  i) Create a node for the operand  ii) Push the pointer to the created node onto a stack  4. If S is a binary operator then  i) Create a node for the operator  ii) T1 = Pop from the stack a pointer to an operand  iii) T2 = Pop from the stack a pointer to an operand  iv) Make T2 the left subtree and T1 the right subtree of the  operator node  v) Push the pointer to the operator node onto the stack  [End of while]  5. T = Pop from the stack a pointer to expression tree  6. Return  **Construction** | 1.5M  1.5M  1M  2M | 02 | 01 |
|  | b) | Prove if n is the total number of nodes in a complete binary tree of height h, then h =|log 2n | + 1  **Proof:** From the definition of a complete binary tree of height h, it is filled up to height h – 1 and in the  last level, it may have partially filled with nodes. Hence, we can write:  2h-1 -1 < n ≤ 2h – 1  Since the maximum number of nodes at height h-1 is 2h-1 -1 and at height, h is 2h – 1.  or we can write 2h-1 ≤ n < 2h (iii)  Taking the logarithm of (iii) we get  h - 1 log2 n < h  Therefore, the value of log2 n lies between h and h-1. Now if we take floor value of log2 n then it  will be h-1.  Hence, h = ⌊log2 n ⌋ + 1 or h ≤ ⌊log2 n +1 ⌋ | 2M  2M | 03 | 01 |
| 2 | a) | Write a C program to demonstrate the construction of a Binary Search Tree(BST) of runs scored by a player in various matches. The structure of the tree node is given below. After creating the BST, print the runs scored by the player along with the match number in ascending order.  struct treenode  {  int matchno, runs;  struct treenode \*lchild, \*rchild;  }  #include <stdio.h>  #include <stdlib.h>  struct treeNode {  int matchNumber;  int runsScored;  struct treeNode \*left;  struct treeNode \*right;  };  struct treeNode\* createNode(int matchNumber, int runsScored) {  struct treeNode\* newNode = (struct treeNode\*)malloc(sizeof(struct treeNode));  newNode->matchNumber = matchNumber;  newNode->runsScored = runsScored;  newNode->left = NULL;  newNode->right = NULL;  return newNode;  }  struct treeNode\* insertNode(struct treeNode\* root, int matchNumber, int runsScored) {  if (root == NULL) {  return createNode(matchNumber, runsScored);  }    if (matchNumber < root->matchNumber) {  root->left = insertNode(root->left, matchNumber, runsScored);  } else if (matchNumber > root->matchNumber) {  root->right = insertNode(root->right, matchNumber, runsScored);  }    return root;  }  void printInOrder(struct treeNode\* root) {  if (root != NULL) {  printInOrder(root->left);  printf("Match %d: Runs scored: %d\n", root->matchNumber, root->runsScored);  printInOrder(root->right);  }  }  int main() {  struct treeNode\* root = NULL;  int matches[] = {3, 1, 5, 2, 4};  int runs[] = {50, 30, 80, 20, 60};  int n = sizeof(matches) / sizeof(matches[0]);  for (int i = 0; i < n; i++) {  root = insertNode(root, matches[i], runs[i]);  }  printf("Runs scored by the player in various matches:\n");  printInOrder(root);  return 0;  }  Initializing to variables  Function to create a new node and to insert into BST  Function to print in ascending order | 2M  2M  2M | 03 | 03 |
| b) | Discuss the threaded binary tree data structure by considering an example.  Threaded Binary Tree T may be stored in computer memory by using a linked representation. Here, the  thread can be represented by a negative value of the location and ordinary pointer can be represented  by the positive value of the location.  1 2 3 4 5 6 7 8 9 10 11 12 13 14 15  Data E A G C B F D H  LCHILD −3 8 −6 1 12 5 0 −10  RCHILD −6 6 −10 10 −3 15 −8 0  **OR**  **Binary Tree Representation with Linked List**  Binary Tree can be also representation with Linked List. In this representation, each node of a binary  tree consists three fields such that   * The first field contains the pointer field, which points to the left child. * The second field contains the data. * The third field contains the pointer field, which points to the right child | 2M  2M | 02 | 01 |
| 3 | a) | Explain the concept of Graph Isomorphism by considering an example of Social Networks.  Network A, where individuals are represented as vertices, and edges represent friendships between individuals.  Network B, another social network with a potentially different set of individuals and friendships, represented similarly as a graph.  Now, let's say we want to determine if these two social networks are isomorphic. We would examine the structure of both graphs to see if there's a way to match up the individuals (vertices) and friendships (edges) in one graph with those in the other while preserving the relationships between them. | 2M  2M | 02 | 03 |
|  | b) | Write Dijkstra’s algorithm to find the single source shortest path distances to all the nodes in a Graph. Give the output of Dijkstra’s algorithm for the graph in Figure 3b, and assume the source vertex as 0.  Figure 3b  1 function Dijkstra(Graph, source):  2  3 for each vertex v in Graph.Vertices:  4 dist[v] ← INFINITY  5 prev[v] ← UNDEFINED  6 add v to Q  7 dist[source] ← 0  8  9 while Q is not empty:  10 u ← vertex in Q with min dist[u]  11 remove u from Q  12  13 for each neighbor v of u still in Q:  14 alt ← dist[u] + Graph.Edges(u, v)  15 if alt < dist[v]:  16 dist[v] ← alt  17 prev[v] ← u  18  19 return dist[], prev[]    1-15  2-13  3-2  4-5  5-9  6-10  7-13 | 1M  1M  1M  3M | 03 | 01 |
| 4 | a) | Write an algorithm for the DFS of a Graph. Give the tracing of the algorithm for the graph in Figure 4a with the starting vertex as 0.  Figure 4a  **Algorithm: DFS (G, s)**  *[G is a given graph of N vertices, s is the source node]*  1. Set VISITED [s] = 1  2. Visit s  3. Repeat for each vertex v adjacent to s do  4. If VISITED [v] = 0 then  5. Call DFS (v)  [End of if]  [End of For loop]  6. Return  0-3-5-7-6-2-1-4 | 3M  3M | 03 | 01 |
| b) | Breadth-First Search is one of the main algorithms used for indexing web pages, justify this statement.  Crawling the Web  Discovering new pages  Robustness to changes  Respecting policies  Identification  Detailed explanation for all | 2M  2M | 03 | 03 |
| 5 | a) | Hashing is producing the location's address to store the given key value. Discuss how hashing is used in password verification.  Hash function explanation  Password storage explanation  Authentication explnantion  Hash Function: A hash function is a mathematical algorithm that takes an input (in this case, a password) and produces a fixed-size string of characters, which is typically a hash value or digest.  Password Storage: When a user creates an account or updates their password, the plaintext password is never stored directly in the database. Instead, the password is passed through a hash function, and the resulting hash value is stored. This way, even if the database is compromised, an attacker cannot directly retrieve the user's passwords.  Authentication: When a user attempts to log in, they provide their plaintext password. The system hashes this password using the same hash function and compares the resulting hash value with the stored hash value associated with that user's account. | 1M  2M  1M | 02 | 03 |
| b) | i. The keys 12, 18, 13, 2, 3, 23, 5 and 15 are inserted into an initially empty hash table of length 10 using open addressing with hash function h(k) = k mod 10 and **linear probing**. What is the resultant hash table?  0  1  2 12  3 13  4 3  5 23  6 15  7  8 18  9 5  ii. Given the input {4371, 1323, 6173, 4199, 4344, 9679, 1989}, a fixed table size of 10, and a hash function H(X) = X mod 10 and **quadratic probing**. What is the resultant hash table?    0 9679  1 4371  2 1989  3 1323  4 4344  5  6 6173  7  8  9 4199 | 3M  3M | 03 | 01 |

| **Course Outcome** | |
| --- | --- |
| **CO1** | Apply the knowledge of data structures in providing solutions to some software development requirements. |
| **CO2** | Perform data analysis of some real-world scientific/business use cases and present the analysis results. |
| **CO3** | Investigate appropriate data structures and understand requirements in solving some problems of industry and society. |
| **CO4** | Use data analysis tools to illustrate the principles of data interpretation, statistical analysis, and graphical visualizations of the datasets. |
| **CO5** | Appraise data structures and analysis knowledge to build a successful career as an AIML engineer, work in teams, and communicate their ideas effectively. |

**M-Marks, BT-Blooms Taxonomy Levels, CO-Course Outcomes**

| **Marks Distribution** | **Particulars** | **CO1** | **CO2** | **CO3** | **CO4** | **CO5** | **L1** | **L2** | **L3** | **L4** | **L5** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Max Marks** | 32 | -- | 18 | -- | -- | -- | 18 | 32 | -- | -- |