(Affiliated to University of Mumbai)

Branch: CSE(IoT&CSIBCT)

Year/ Semester: SE/III

Time: 03 hours

Note: 1. All questions are compulsory.
2. Figures to right indicate full marks.
3. Assume suitable data wherever necessary.

		Marks
Q. 1	Attempt any FOUR. (All questions carry equal marks)	**20**
A.	Differentiate between linear and nonlinear data structure.	05

Point	**Linear DS**	**Non-linear DS**
Structure	Sequential	Hierarchical/Network
Traversal	Single-level	Multi-level
Examples	Array, Stack, Queue, Linked List	Tree, Graph
Memory	Contiguous/linked	Non-contiguous
Implementation	Easy	Complex

B.	Construct B tree of order 5 using following elements 7,11,1,20,5,15,18,9,12,37,34,8,42,24 [7] / \ [1, 5] [11, 15, 20] [7, 15] / | \ [1, 5] [9, 11] [18, 20] [7, 15, 34] / | | \ [1, 5] [8,9,11,12] [18,20,24] [37,42]	Rules of insertion explained – 1 mark Correct splits – 2 marks Final correct tree structure – 2 marks
C.	Explain adjacency list and adjacency matrix representation of graph with example.	• Definition of adjacency matrix – 1 mark • Example matrix – 1.5 marks • Definition of adjacency list – 1 mark • Example list – 1.5 marks
D.	Explain binary search algorithm with example.	• Binary Search principle – 1 mark • Algorithm/Pseudocode

<table>
<tr><td></td><td></td><td>– 2 marks
• Example – 2 marks</td></tr>
<tr><td>E.</td><td>Apply fractional knapsack algorithm problem for the following. n = 6, p= (18, 5, 9, 10, 12, 7) w = (7, 2, 3, 5, 3, 2), Max sack capacity M = 13.
<table>
<tr><td>Item</td><td>1</td><td>2</td><td>3</td><td>4</td><td>5</td><td>6</td></tr>
<tr><td>profit</td><td>18</td><td>5</td><td>9</td><td>10</td><td>12</td><td>7</td></tr>
<tr><td>weight</td><td>7</td><td>2</td><td>3</td><td>5</td><td>3</td><td>2</td></tr>
<tr><td>P/W</td><td>2.57</td><td>2.5</td><td>3</td><td>2</td><td>4</td><td>3.5</td></tr>
</table>
2. Sort in descending order
<table>
<tr><td>Item</td><td>5</td><td>6</td><td>3</td><td>1</td><td>2</td><td>4</td></tr>
<tr><td>profit</td><td>12</td><td>7</td><td>9</td><td>18</td><td>5</td><td>10</td></tr>
<tr><td>weight</td><td>3</td><td>2</td><td>3</td><td>7</td><td>2</td><td>5</td></tr>
<tr><td>P/W</td><td>4</td><td>3.5</td><td>3</td><td>2.57</td><td>2.5</td><td>2</td></tr>
</table>
Step 3: Fill the Knapsack Greedily
• Item 5: weight = 3 → remaining = 10 → profit = 12
• Item 6: weight = 2 → remaining = 8 → profit = 12 + 7 = 19
• Item 3: weight = 3 → remaining = 5 → profit = 19 + 9 = 28
• Item 1: weight = 7 → only 5 units fit → take 5/7 fraction
→ profit = 28 + (18 × 5/7) = 28 + 12.86 ≈ 40.86
◆ Maximum Profit ≈ 40.86</td><td>• Compute p/w ratio – 2 marks
• Sort items – 1 mark
• Pick items fractionally – 1 mark
• Final max profit – 1 mark</td></tr>
<tr><td>F.</td><td>Solve the sum of subsets problem using backtracking approach.
Set = {1, 9, 7, 5, 18, 12, 20, 15}, n=8 and sum value = 35

Using backtracking, the valid subsets are:
{1, 5, 9, 20}
{1, 7, 12, 15}
{5, 12, 18}
{7, 9, 18}</td><td>• Backtracking approach described – 2 marks
• Recursive tree / tracking – 2 marks
• One valid subset(s) result – 1 mark</td></tr>
<tr><td>Q.2</td><td>Attempt any FOUR. (All questions carry equal marks)</td><td></td></tr>
<tr><td>A.</td><td>Write C program to implement Queue using singly linked list.
#include <stdio.h>
#include <stdlib.h>

// Structure of a node
struct Node {
int data;
struct Node* next;
};

struct Node* front = NULL;
struct Node* rear = NULL;

// Function to insert element into the queue (Enqueue)
void enqueue(int value) {
struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));</td><td>• Node structure – 2 marks
• enqueue() – 3 marks
• dequeue() – 3 marks
• display() – 2 marks</td></tr>
</table>

```
    newNode->data = value;
    newNode->next = NULL;

    if (front == NULL && rear == NULL) {
        front = rear = newNode;
    } else {
        rear->next = newNode;
        rear = newNode;
    }
    printf("%d inserted into the queue.\n", value);
}

// Function to remove element from the queue (Dequeue)
void dequeue() {
    if (front == NULL) {
        printf("Queue is empty! Cannot dequeue.\n");
        return;
    }
    struct Node* temp = front;
    printf("%d removed from the queue.\n", front->data);
    front = front->next;

    if (front == NULL) {
        rear = NULL;
    }
    free(temp);
}

// Function to display the queue
void display() {
    if (front == NULL) {
        printf("Queue is empty!\n");
        return;
    }
    struct Node* temp = front;
    printf("Queue elements: ");
    while (temp != NULL) {
        printf("%d -> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}

// Main function
int main() {
    int choice, value;

    while (1) {
        printf("\n--- Queue using Singly Linked List ---\n");
```

	printf("1. Enqueue\n");
	printf("2. Dequeue\n");
	printf("3. Display\n");
	printf("4. Exit\n");
	printf("Enter your choice: ");
	scanf("%d", &choice);
	
	switch (choice) {
	case 1:
	printf("Enter value to insert: ");
	scanf("%d", &value);
	enqueue(value);
	break;
	case 2:
	dequeue();
	break;
	case 3:
	display();
	break;
	case 4:
	exit(0);
	default:
	printf("Invalid choice! Please try again.\n");
	}
	}
	return 0;
	}	

B.	Explain B+ tree and construct B+ tree of order 5 using following elements. 6,10,1,21,5,15,19,9,12,38,35,7,43,25 **Node capacity**: • Internal nodes: Max **M − 1 = 4** keys. • Leaf nodes: Max **M − 1 = 4** keys. • Min keys per node: **⌈M/2⌉ − 1 = 2** (except root). Internal Nodes: [15, 25] / | \ [1,5,6,10] [9,12,15,19,21] [25,35,38,43] Leaf Nodes (linked list): [1,5,6,10] → [9,12,15,19,21] → [25,35,38,43]	• Explanation – 2 marks • Insertions – 4 marks • Final tree – 4 marks
C.	Explain doubly linked list node structure and write c code for doubly linked list for inserting node at the beginning, end of the list and display the list. #include <stdio.h> #include <stdlib.h> struct Node {	• Node structure – 2 marks • Insert at beginning – 2 marks • Insert at end – 2 marks • Display – 2 marks • Explanation/commentin

```
    int data;
    struct Node* prev;
    struct Node* next;
};

struct Node* head = NULL;
void insertAtBeginning(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->prev = NULL;
    newNode->next = head;

    if (head != NULL) {
        head->prev = newNode;
    }
    head = newNode;

    printf("%d inserted at the beginning.\n", value);
}
void insertAtEnd(int value) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = value;
    newNode->next = NULL;

    if (head == NULL) {
        newNode->prev = NULL;
        head = newNode;
        printf("%d inserted at the end.\n", value);
        return;
    }

    struct Node* temp = head;
    while (temp->next != NULL) {
        temp = temp->next;
    }

    temp->next = newNode;
    newNode->prev = temp;

    printf("%d inserted at the end.\n", value);
}
void displayList() {
    if (head == NULL) {
        printf("List is empty.\n");
        return;
    }

    struct Node* temp = head;
    printf("Doubly Linked List: ");
```

g – 2 marks

```
    while (temp != NULL) {
        printf("%d <-> ", temp->data);
        temp = temp->next;
    }
    printf("NULL\n");
}
```

D. Describe AVL tree with all rotations.

- Definition – 2 marks
- Left rotation – 2 marks
- Right rotation – 2 marks
- LR rotation – 2 marks
- RL rotation – 2 marks

1. Right Rotation (LL Rotation)

- Used when a node is inserted in the **left subtree of the left child**
- Example:

```
  30
  /
 20
/
10
```

Rotation at 30:

```
   20
  /  \
 10   30
```

2. Left Rotation (RR Rotation)

- Used when a node is inserted in the **right subtree of the right child**
- Example:

```
10
  \
   20
     \
      30
```

Rotation at 10:

```
   20
  /  \
 10   30
```

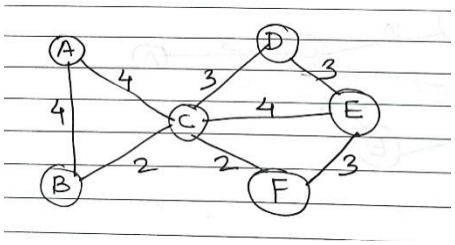
3. Left-Right Rotation (LR Rotation)

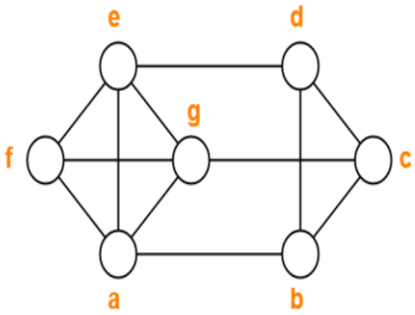
- Used when a node is inserted in the **right subtree of the left child**
- Example:

```
  30
  /
 10
   \
    20
```

Step 1: Left Rotation at 10
Step 2: Right Rotation at 30
Final tree:

```
   20
```

	/ \ 10 30	
	4. Right-Left Rotation (RL Rotation) • Used when a node is inserted in the **left subtree of the right child** • Example: 10 \ 30 / 20 Step 1: Right Rotation at 30 Step 2: Left Rotation at 10 Final tree: 20 / \ 10 30	
E.	Construct Minimum cost spanning tree (MST) using Kruskal's and Prim's method  **Kruskal algoritms:** 1. Sort edges by weight (ascending): 2. Now pick edges in order, avoiding cycles: • Pick **B–C (2)** → connect B and C. • Pick **C–F (2)** → connect F into component {B,C,F}. • Pick **C–D (3)** → connect D → component {B,C,F,D}. • Pick **D–E (3)** → connect E → component {B,C,F,D,E}. • Next candidate F–E (3) would form a cycle among {B,C,F,D,E} → **skip**. • Next pick **A–B (4)** → connects A to the component. 3. Selected edges **(Kruskal): B–C (2), C–F (2), C–D (3), D–E (3), A–B (4)** Total weight = 2 + 2 + 3 + 3 + 4 = **14** **Prims algorithms:** Start at C. At each step pick the smallest weight edge crossing the current tree boundary. Initial tree: {C} Edges out of C: B(2), F(2), D(3), A(4), E(4) 1. Pick **B–C (2)** → tree {C,B} New available edges (from tree): C–F(2), C–D(3), A–B(4), A–	• Graph/example – 2 marks • Kruskal steps – 3 marks • Prim steps – 3 marks • Final MST cost – 2 marks

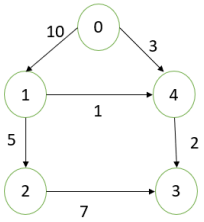
	C(4), C–E(4) 2. Pick **C–F (2)** → tree {C,B,F} New available edges: C–D(3), D–E(3), F–E(3), A–B(4), A–C(4), C–E(4) 3. Pick **C–D (3)** → tree {C,B,F,D} New edges: D–E(3), F–E(3), A–B(4), A–C(4), C–E(4) 4. Pick **D–E (3)** → tree {C,B,F,D,E} (F–E would create cycle — skip) 5. Finally pick **A–B (4)** (or A–C(4)) → tree includes A Selected edges (Prim): **C–B (2), C–F (2), C–D (3), D–E (3), A–B (4)** Total weight = **14** (same as Kruskal)	
F.	What is graph coloring problem? Apply graph coloring technique and find chromatic no. of a graph.  Step-by-Step Coloring of Graph Your graph has 8 nodes: **a, b, c, d, e, f, g, h**: 1. Start with node **g** (most connected: degree 5) • Assign **Color 1** to **g** 2. Nodes adjacent to g: **e, f, a, b, c** • Assign: • **Color 2** to **e** • **Color 3** to **f** • **Color 2** to **a** (not adjacent to e) • **Color 3** to **b** (not adjacent to f) • **Color 4** to **c** (adjacent to b and g) 3.	• Definition – 2 marks • Steps of coloring – 5 marks • Chromatic number – 3 marks

Q.3	Attempt any FOUR	
A.	Explain BFS technique and write an algorithm of BFS in detail. **BFS (Breadth First Search)** • BFS is a graph traversal technique. • It visits all the vertices **level by level**. • It uses a **queue (FIFO)** data structure. • Works for **connected and disconnected graphs**. • Useful for finding **shortest path in unweighted graphs**, cycle detection, etc. **BFS Algorithm (Adjacency List)** BFS(G, start): 1. Create a boolean array visited[V] initialized to false 2. Create an empty queue Q 3. Mark visited[start] = true and enqueue start into Q 4. While Q is not empty: a. u = Dequeue Q b. Print u (or process it) c. For every neighbor v of u: if visited[v] == false: visited[v] = true Enqueue v into Q **Time Complexity:** O(V + E)	• Definition & Concept of BFS 1 mark • Explanation of working/technique 1.5 marks • BFS algorithm 2 marks • Example 0.5 mark
B.	Explain graph traversal techniques in detail. Graph traversal = visiting all vertices systematically. Two main techniques: **a) BFS (Breadth First Search)** • Uses Queue • Level-wise traversal • Best for shortest path (unweighted)	• Definition of hashing 1 mark • Explanation of hashing need/use 0.5 mark • Static hashing 1 mark • Collision resolution techniques 1.5 marks

Vertex	Color
g	1
e	2
f	3
a	2
b	3
c	4
d	3

Chromatic Number = 4

	Example Order: A → B → C → D → E **b) DFS (Depth First Search)** • Uses Stack (or Recursion) • Explores as deep as possible before backtracking	• Example 1 mark
C.	Apply Linear Probing, Quadratic Probing on following data:89,18,49,58,69. The hash table size m = 10 and the hash function h(k) = k mod 10. • Linear Probing Formula: (h(k) + i) mod m **Key h(k) Position** 89 9 9 18 8 8 49 9 9 (occupied) → 0 58 8 8 (occupied) → 9 (occupied) → 0 (occupied) → 1 69 9 9,0,1,2 → 2 • **Quadratic Probing** Formula: (h(k) + i²) mod m **Key h(k) Position** 89 9 9 18 8 8 49 9 9 → (9 + 1²)%10 = 0 58 8 8 → (8 + 1²)=9 → (8+2²)=12%10=2 69 9 9 → 0 → (9+2²)=13%10=3	•
D.	Define hashing and explain different types of hashing techniques with example. **Hashing** A technique to **store and retrieve data in constant time (O(1))** using a hash function to compute an index. **Hash Function** h(k) = k mod m **Types of Hashing** 1. **Division Method** h(k) = k mod m Example: key=19, m=10 → 19 mod 10 = 9 2. **Multiplication Method** h(k) = floor(m * (k * A mod 1)) Example: A=0.618, m=10 3. **Mid-square Method** Square the key and take middle digits. 4. **Folding Method** Break key and sum parts. **Collision Resolution** • Linear Probing • Quadratic Probing	• Definition of hashing 1 mark • Explanation of hashing need/use 0.5 mark • Static hashing 1 mark • Collision resolution techniques (LP/QP/Chaining) 1.5 marks • Example/application 1 mark

	• Double Hashing • Separate Chaining	
E.	Find Dijkstra's shortest path from vertex 0 for following graph .  Iteration Steps 1. Start at 0: • 0 → 1 = 10 → update distance of 1 to 10 • 0 → 4 = 3 → update distance of 4 to 3 2. Visit 4 (min distance = 3): • 4 → 3 = 2 → update distance of 3 to 5 • 4 → 1 = 1 → 3 + 1 = 4 → update distance of 1 to 4 3. Visit 1 (min distance = 4): • 1 → 2 = 5 → 4 + 5 = 9 → update distance of 2 to 9 4. Visit 3 (min distance = 5): • No outgoing edges 5. Visit 2 (min distance = 9): • 2 → 3 = 7 → already visited, no update 6. Vertex 5 remains unreachable **Vertex** | **Distance** | **Path** 0 | 0 | 0 1 | 4 | 0 → 4 → 1 2 | 9 | 0 → 4 → 1 → 2 3 | 5 | 0 → 4 → 3 4 | 3 | 0 → 4 5 | ∞ | —	• Initial table setup (distance & visited array) 1 mark • Step-by-step relaxation process 2 marks • Final shortest path values 1.5 marks • Diagram/justification 0.5 mark
F.	Explain 4*4 queen problem using backtracking. Place 4 queens on a 4×4 chessboard such that no two queens attack each other. **Constraints:** • No two queens in the same row, column, or diagonal. **Backtracking Idea:** 1. Place queen in row 1. 2. Go to next row. 3. Try safe column.	• Problem definition 0.5 mark • Backtracking concept 1 mark • Steps/algorithm explanation 2 marks • Valid solution/placement diagram 1.5 marks

	4. If no column is safe → backtrack. 5. Continue until 4 queens placed. **Solution Matrix** **. Q . .** **. . . Q** **Q . . .** **. . Q .** **Another solution** . . Q . Q . . . . . . Q . Q . .	