USN | | | | | | | | | | |

Scheme & Solution
Internal Assessment Test 2 –May 2025

Sub:	ANALYSIS AND DESIGN OF ALGORITHMS					Sub Code:	BCS401	Branch:	ISE	
Date:	26/05/2025	Duration:	90 min's	Max Marks:	50	Sem/Sec:	IV A, B & C		OBE	

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1.	Explain AVL Tree and its rotations, Construct an AVL tree for the following numbers 5,6,8,3,2,4,7 **Definition- [4 Marks]** **Solution step by step: - [6 Marks]** **Answer** An AVL Tree is a self-balancing binary search tree where the difference between heights of left and right subtrees cannot be more than one for all nodes. It is named after its inventors Adelson-Velsky and Landis. **Balance Factor** Balance Factor = Height of Left Subtree - Height of Right Subtree Allowed values for a balanced node: -1, 0, +1 **Types of Rotations in AVL Tree** 1. Right Rotation (RR): Used when the left subtree of the left child is unbalanced. 2. Left Rotation (LL): Used when the right subtree of the right child is unbalanced. 3. Left-Right Rotation (LR): Used when the right subtree of the left child is unbalanced. 4. Right-Left Rotation (RL): Used when the left subtree of the right child is unbalanced. **Constructing AVL Tree for: 5, 6, 8, 3, 2, 4, 7** Step 1: Insert 5 Tree: 5 Step 2: Insert 6 Tree: 5	**10**	CO3	L3

```
  \
   6
```

Step 3: Insert 8

Tree:

```
 5
  \
   6
    \
     8
```

Unbalanced at 5 (BF=-2), apply Left Rotation.
New Tree:

```
   6
  / \
 5   8
```

Step 4: Insert 3

Tree:

```
   6
  / \
 5   8
/
3
```

Step 5: Insert 2

Tree:

```
   6
  / \
 5   8
/
3
/
2
```

Unbalanced at 5 (BF=2), apply Right Rotation.
New Tree:

```
   6
  / \
 3   8
/ \
2   5
```

Step 6: Insert 4

Tree:

```
   6
  / \
 3   8
```

	/ \ 2 5 / 4 Unbalanced at 3 (BF=-2), apply Right-Left Rotation. New Tree: 6 / \ 4 8 / \ 3 5 / 2 Step 7: Insert 7 Tree: 6 / \ 4 8 / \ 3 5 / 2 \ 7 Balanced Tree. **Final AVL Tree Structure** 6 / \ 4 8 / \ / 3 5 7 / 2			
2.	Write Huffman's algorithm, Construct Huffman tree and resulting code word for the following. Encode the text DAD_CBE Character | A | B | C | D | E | ----- Probability | 0.5 | 0.35 | 0.5 | 0.1 | 0.4 | 0.2 **Definition & Algorithm- [5 Marks]**	**10**	CO4	L3

Solution step by step: - [6 Marks]

Answer

Huffman's Algorithm

1. Create a leaf node for each character and add it to a priority queue based on its probability.
2. While there is more than one node in the queue:
   a. Remove the two nodes with the lowest probabilities.
   b. Create a new internal node with these two nodes as children and the sum of their probabilities as the new probability.
   c. Add the new node back into the queue.
3. The remaining node is the root of the Huffman Tree.
4. Assign codes to characters by traversing the tree: left edge = 0, right edge = 1.

Given Data

Characters: A, B, C, D, E, _
Probabilities: A=0.5, B=0.35, C=0.5, D=0.1, E=0.4, _=0.2

Constructing the Huffman Tree

Step 1: Create nodes for each character with their probabilities.
Step 2: Combine the two smallest nodes (D=0.1 and _=0.2) -> New node1 (0.3)
Step 3: Combine node1 (0.3) and B (0.35) -> New node2 (0.65)
Step 4: Combine E (0.4) and node2 (0.65) -> New node3 (1.05)
Step 5: Combine A (0.5) and C (0.5) -> New node4 (1.0)
Step 6: Combine node4 (1.0) and node3 (1.05) -> Root node (2.05)

Codewords from Huffman Tree

The codewords (assuming left = 0, right = 1 from the constructed tree) might be:
A = 00
C = 01
E = 10
B = 111
_ = 1101
D = 1100

Encoding the Text: DAD_CBE

Text: D A D _ C B E
Encoding:
D = 1100
A = 00
D = 1100
_ = 1101
C = 01
B = 111
E = 10
Encoded Text: 1100 00 1100 1101 01 111 10

3.	Explain the concept of Backtracking, Construct state space tree to solve 4 queens' problem **Definition & Algorithm- [5 Marks]** **Solution step by step: - [5 Marks]** **Answer** Backtracking is a general algorithmic technique for solving problems recursively by trying to build a solution incrementally. It removes those solutions that fail to satisfy the constraints of the problem at any point of time (prunes the search space). If a solution path leads to a conflict or dead-end, the algorithm backtracks and tries another path. Backtracking is commonly used for problems such as: - N-Queens Problem - Sudoku Solver - Crossword puzzles - Combinatorial problems (e.g., subset generation, permutations) **4 Queens Problem** The goal is to place 4 queens on a 4x4 chessboard such that no two queens threaten each other. This means no two queens can be in the same row, column, or diagonal. **State Space Tree for 4 Queens** Each level of the tree represents a row of the chessboard. Each node at a level represents a possible column position for the queen in that row. If a queen can be safely placed, we proceed to the next row; otherwise, we backtrack. Partial State Space Tree: Level 0: Place queen at (0,0) ├── (1,2) │ ├── (2,1) │ │ ├── (3,3) → Solution 1: [(0,0), (1,2), (2,1), (3,3)] │ │ └── (3,x) [Others lead to conflict] │ └── (2,x) [Other positions conflict] ├── (1,x) [Other columns conflict] └── Try (0,1), (0,2), (0,3) similarly and build further branches The process continues recursively and prunes invalid branches. There are two distinct solutions for the 4 queens problem.	**10**	CO5	L3

4.	Implement a branch-and-bound algorithm for 0/1 Knapsack with profits = [60, 100, 120], weights = [10, 20, 30], capacity = 50.
	
	Definition & Algorithm- [5 Marks]
	
	Solution step by step: - [5 Marks]
	
	Answer
	
	Branch and Bound Algorithm (for 0/1 Knapsack)
	This algorithm builds a solution tree, where:
	• Each **node** represents a subset of items.
	• We **branch** into two possibilities at each level:
	  o Include the current item.
	  o Exclude the current item.
	• We compute an upper **bound** for each node to estimate the best possible profit from that node onward.
	• We use a **priority queue** (max-heap) to always explore the node with the best bound next.
	
	• bound = current profit + (remaining capacity) × (next item's profit/weight)
	
	☑ **Step 1: Sort items by profit/weight ratio**
	Item Profit Weight Ratio (P/W)
	1 60 10 6.0
	2 100 20 5.0
	3 120 30 4.0
	Order remains the same.
		10	CO5	L3

	☑ **Step 2: Initialize root node (level = -1)** • level = -1 (before any item is considered) • profit = 0, weight = 0 • Compute **bound** using greedy method: **Bound Calculation:** We try to add as many full items as possible: • Add item 1 → total weight = 10, profit = 60 • Add item 2 → total weight = 30, profit = 160 • Add item 3 partially: remaining capacity = 20 → add 20/30 of 120 = 80 • **Total bound = 160 + 80 = 240** So: **Root node = (level = -1, profit = 0, weight = 0, bound = 240)**			
	☑ **Step 3: Explore Node A (Include item 1)** • level = 0 • Include item 1 (Profit = 60, Weight = 10) **Bound Calculation:**			

- Add item 2 → weight = 30, profit = 160
- Add 20/30 of item 3 → +80
- **Total bound = 240**
  Node A: (level = 0, profit = 60, weight = 10, bound = 240)

☑ **Step 4: Explore Node B (Include item 2 after item 1)**

- level = 1
- Add item 2 → weight = 30, profit = 160
- Remaining capacity = 20
- Add 20/30 of item 3 → +80
- **Bound = 240**

Node B: (level = 1, profit = 160, weight = 30, bound = 240)

☑ **Step 5: Explore Node C (Include item 3 after item 1 and 2)**

- level = 2
- Add item 3 → total weight = 60 (over capacity!) → Discard
  Not feasible.

☑ **Step 6: Backtrack → Exclude item 3**

Back to Node B, exclude item 3:

- level = 2
- Profit = 160, Weight = 30
- No more items to include
- **Feasible solution = 160**

☑ **Step 7: Backtrack → Exclude item 2 from Node A**

- level = 1
- Profit = 60, Weight = 10
- Only consider item 3 now

Bound Calculation:

- Add item 3 → weight = 40, profit = 180
- Bound = 180
  Node D: (level = 2, profit = 180, weight = 40)

Feasible → Better than 160

☑ **Step 8: Backtrack → Exclude item 1 from Root**

Now try starting with excluding item 1.

- level = 0
- Profit = 0, Weight = 0

Include item 2

- level = 1
- Profit = 100, Weight = 20

Bound:

- Add item 3 → weight = 50, profit = 220
- **Feasible** → Max profit = 220

☑ **Step 9: Final Best Feasible Solution**

Include:

- Item 2 (100)
- Item 3 (120)

Total Profit = 220, Total Weight = 50

☑ **Final Answer:**

Included Items	**Weights**	**Profits**
Item 2	20	100
Item 3	30	120
Total	50	220

☑ **Maximum Profit = 220**

Bounding Function
The bound is calculated using **greedy fractional knapsack**:

5. Apply the Wars hall's algorithm to find the transitive closure of the given graph, Show the matrix updates at each step. **10** CO4 L3

	A	B	C	D	E
A	1	0	0	1	0
B	0	1	0	0	0
C	0	0	0	1	1
D	1	0	0	0	0
E	0	1	0	0	1

Definition & Algorithm- [5 Marks]

Solution step by step: - [5 Marks]

Answer

☑ **Step 0: Initial Adjacency Matrix (R(0))**

	A	**B**	**C**	**D**	**E**
A	1	0	0	1	0
B	0	1	0	0	0
C	0	0	0	1	1
D	1	0	0	0	0
E	0	1	0	0	1

We will now perform updates for each intermediate node k from A to E.

☑ **Step 1: Using A as intermediate (k = A / index 0)**
Update:
For every i, j, if R[i][j] = 0 and R[i][A] = 1 and R[A][j] = 1, then set R[i][j] = 1
Updates:

- R[A][D] = 1 and R[D][A] = 1 ⇒ R[D][D] = 1
- R[D][A] = 1 and R[A][D] = 1 ⇒ R[D][D] = 1

Matrix after Step 1 (R(1)):

	A B C D E **A** 1 0 0 1 0 **B** 0 1 0 0 0 **C** 0 0 0 1 1 **D** 1 0 0 1 0 **E** 0 1 0 0 1			
	☑ **Step 2: Using B as intermediate (k = B / index 1)** Only new path: • R[E][B] = 1 and R[B][B] = 1 ⇒ R[E][B] = 1 (already 1) No update needed. **Matrix after Step 2 (R(2)) remains same.**			
	☑ **Step 3: Using C as intermediate (k = C / index 2)** From C, we have edges to D and E. Updates: • No node connects to C (all R[i][C] = 0) ⇒ **No updates** **Matrix after Step 3 (R(3)) remains same.**			
	☑ **Step 4: Using D as intermediate (k = D / index 3)** From D, we have a link to A (R[D][A] = 1) Updates: • C → D and D → A ⇒ C → A = 1 • C → D and D → D ⇒ C → D = 1 (already 1) • C → D and D → A → D ⇒ C → D = 1 (already) • C → D and D → A ⇒ C → A = 1 So: • R[C][A] = 1 **Matrix after Step 4 (R(4)):** **A B C D E** **A** 1 0 0 1 0 **B** 0 1 0 0 0 **C** 1 0 0 1 1 **D** 1 0 0 1 0 **E** 0 1 0 0 1			
	☑ **Step 5: Using E as intermediate (k = E / index 4)** From E, we have an edge to B and itself. Updates: • C → E and E → B ⇒ C → B = 1 So: • R[C][B] = 1 **Matrix after Step 5 (R(5)) — Final Transitive Closure:** **A B C D E** **A** 1 0 0 1 0 **B** 0 1 0 0 0			

	C 1 1 0 1 1 **D** 1 0 0 1 0 **E** 0 1 0 0 1			
	☑ **Final Answer: Transitive Closure Matrix** A B C D E A → 1 0 0 1 0 B → 0 1 0 0 0 C → 1 1 0 1 1 D → 1 0 0 1 0 E → 0 1 0 0 1			
6.	Given the pattern "BARBER" and the text "JIM_SAW_ME_IN_BARBERSHOP", use **Horspool's algorithm** to perform string matching. Show the shift table and each step of the matching process. **Definition & Algorithm- [5 Marks]** **Solution step by step: - [5 Marks]** **Answer** ☑ **Step 1: Build the Shift Table** **For a pattern of length m = 6 (B A R B E R), we process all but the last character (i.e., up to index 4). For each character, the shift is calculated as:** **bash** **CopyEdit** **shift[c] = length_of_pattern - 1 - index_of_c** **Characters in Pattern (excluding last):** **Positions:** **css** **CopyEdit** **0 1 2 3 4** **B A R B E** **Now build the shift table:** **Character Position Shift = 5 - i** **B 0 5** **A 1 4** **R 2 3** **B 3 2 (overwrites previous B)** **E 4 1** **Set default shift for characters not in pattern = length of pattern = 6** ☑ **Final Shift Table:** **markdown** **CopyEdit** **Character | Shift** **----------**	**10**	CO3	L3

	A | 4 **B | 2** **E | 1** **R | 3** **Others | 6**			
	☑ Step 2: Matching Process **We compare from right to left, starting from the end of the pattern aligned to position m - 1 in text.** **Pattern: B A R B E R (Length = 6)** **Text: J I M _ S A W _ M E _ I N _ B A R B E R S H O P (Length = 24)** **We'll align the pattern to positions in the text and compare characters from right to left.**			
	Attempt 1: Align pattern at text[5] to text[10] → text[5:11] = A W _ M E _ • **Compare pattern[5] (R) with text[10] (_) → Mismatch** • **Look up shift for _ → Not in table → shift = 6** • **Move pattern right by 6 positions**			
	Attempt 2: Align at text[11] to text[16] → I N _ B A R • **Compare pattern[5] (R) with text[16] = 'R' → Match** • **pattern[4] = 'E' vs text[15] = 'A' → Mismatch** • **text[16] = 'R' → shift = 3** • **Shift by 3**			
	Attempt 3: Align at text[14] to text[19] → B A R B E R • **Compare pattern[5] (R) with text[19] = 'R' → Match** • **pattern[4] (E) vs text[18] = 'E' → Match** • **pattern[3] (B) vs text[17] = 'B' → Match** • **pattern[2] (R) vs text[16] = 'R' → Match** • **pattern[1] (A) vs text[15] = 'A' → Match** • **pattern[0] (B) vs text[14] = 'B' → Match ☑** **☑ Pattern found at index 14 in the text.**			
	☑ Final Answer: **➤ Pattern "BARBER" occurs at position 14 in the text using Horspool's Algorithm.**			
	☑ Summary Table:			

Attempt	Align Text Indices	Compared	Result	Shift
1	5 to 10	R vs _	Mismatch	6
2	11 to 16	R vs R → E vs A	Mismatch	3
3	14 to 19	All match	☑ Match	-