Program: BE

Date	26/08/2024	Maximum Marks	10+50
Course Code	**CD343AI**	Duration	30+90 min
4th Sem	IV Semester	Improvement Test	

Design and Analysis of Algorithms

(Common to AIML/CSE/CD/CY/ISE)

Sl. No.	Scheme and Solutions Quiz	M
1	Spanning *tree* of an undirected connected graph is its connected acyclic subgraph (i.e., a tree) that contains all the vertices of the graph. If such a graph has weights assigned to its edges, a *minimum spanning tree* is its spanning tree of the smallest weight, where the *weight* of a tree is defined as the sum of the weights on all its edges.	2
2	Compression Ratio = $\frac{\text{(Total no of bits - Average no of bits)}}{\text{Total bits}}$ x 100 Average number of bits per character = 1x0.4 + 3x0.1 + 3x0.2 + 3x0.15 + 3x0.15 = 0.4 + 0.3 + 0.6 + 0.45 +0.45 =2.2 **Compression Ratio=** $\frac{(3-2.2)}{3}$ **x 100 = 26%**	2
3	**Dijkstra's Algorithm**: It finds the shortest path from a single source node to all other nodes in a weighted graph. It focuses on minimizing the total distance from the source to each node. **Prim's Algorithm**: It finds the Minimum Spanning Tree (MST) of a graph, connecting all nodes with the minimum total edge weight, without forming any cycles. It doesn't focus on the distance from a single source.	2
4	**State space** is a tree representation where each node represents a state or a partial solution to the problem, and the branches represent the possible choices or decisions that lead to the next state. The tree is explored to find all possible solutions, and backtracking involves pruning branches that do not lead to a valid solution.	2
5	**NP-hard problems** are a class of problems in computational complexity theory that are at least as hard as the hardest problems in NP (nondeterministic polynomial time). However, unlike NP-complete problems, NP-hard problems are not necessarily in NP, meaning they do not have to be decision problems (problems with a yes/no answer) or verifiable in polynomial time.	2

SL NO	Improvement test Scheme and Solutions	M

1 **Recursive Relation 2 marks, table 3 marks, calculations 2 , result 1 marks** **10**

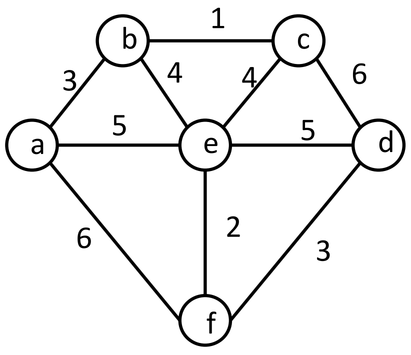
Let V[i][j] represent the maximum profit that can be obtained using the first i items with a knapsack capacity j.

The recursive relation is: $V[i,j]=\begin{cases} V[i-1, j] \text{ , if } j-wi<0 \\ \text{Max } \{ V[i-1, j], V[i-1, j-wi] + vi \} \text{ if } j-wi >=0 \end{cases}$

Items \ Knapsack Capacity	0	1	2	3	4	5
0	0	0	0	0	0	0
1	0	0	8	8	8	8
2	0	6	8	14	14	14
3	0	6	8	16	22	24
4	0	6	11	17	22	27

The value in the last cell (V[4][5] = 27) indicates the maximum profit for a knapsack capacity of 5 using the given items

2 **10**



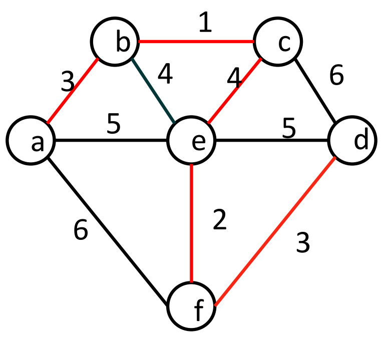
```
V={a,b,c,d,e,f}
Et=0 (Solution)
```

```
Iteration 1
Vt={a}
V-Vt={b,c,d,e,f}
{a,b},{a,e},{a,f}
  3      5      6
Et={a,b} is minimum
```

```
Iteration 2
Vt={a,b}
V-Vt={c,d,e,f}
{a,e},{a,f},{b,c},{b,e}
  5      6      1      4
  Et={{a,b},{b,c}}
```

```
Iteration 3
Vt={a,b,c}
V-Vt={d,e,f}
(a,e) (a,f) (b,e) (c,d), (c,e)
  5     6     4     6      4
Et={={(a,b),(b,c),(c,e)}
```

```
Iteration 4
Vt={a,b,c,e}
V-Vt={d,f}
(a,f)(c,d)(e,d)(e,f)
  6    6    5    2
Et={={(a,b),(b,c),(c,e),(e,f)}
```

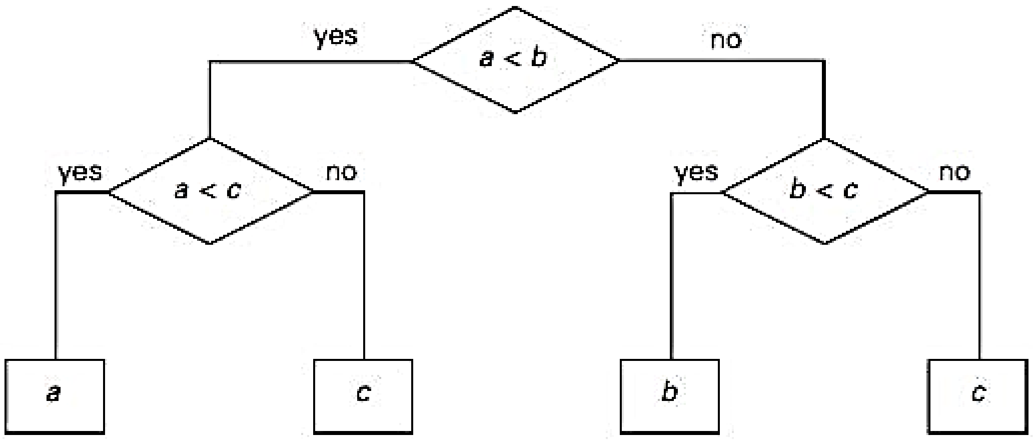


```
Iteration 5
Vt={a,b,c,e,f}
V-Vt={d}
(c,d)(e,d)(f,d)
  6    5    3
Et={={(a,b),(b,c),(c,e),(e,f),(f,d)}
Weight=13
```

3 a (4)

SL	Backtracking	Branch and Bound
1	Uses State space tree	Uses State space tree
2	Feasible solution (All combination)	Optimal Solution
3	Uses DFS Traversal	Uses Both DFS and BFS
4	No bounding function is used	Bounding Function is used (Lower & Upper)

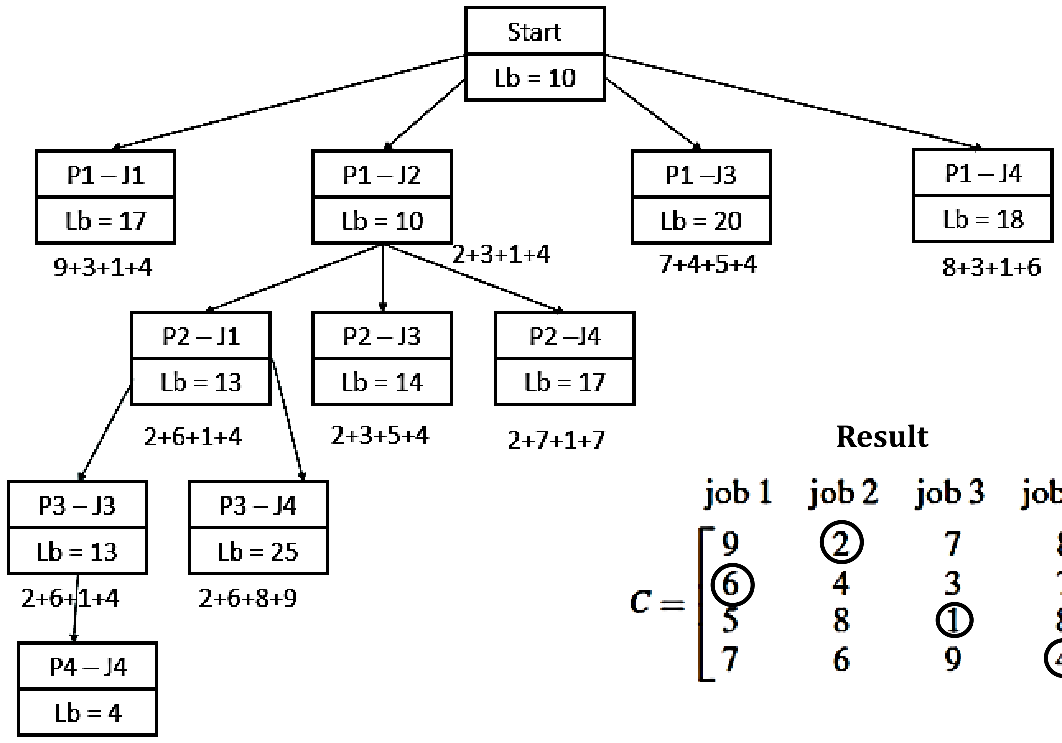
3 b (6)



4 (10)

Lower Bound 2m, state soace tree 5, result 1 m

Job/Person	Job 1	Job 2	Job 3	Job 4
Person 1	9	2	7	8
Person 2	6	4	3	7
Person 3	5	8	1	8
Person 4	7	6	9	4



Result

$$C = \begin{array}{c|cccc|l} & \text{job 1} & \text{job 2} & \text{job 3} & \text{job 4} & \\ & 9 & \textcircled{2} & 7 & 8 & \text{person } a \\ & \textcircled{6} & 4 & 3 & 7 & \text{person } b \\ & 5 & 8 & \textcircled{1} & 8 & \text{person } c \\ & 7 & 6 & 9 & \textcircled{4} & \text{person } d \end{array}$$

5a	**P (Polynomial Time) Problems:** • **Definition**: P problems are those that can be solved in polynomial time by a deterministic algorithm. This means the time it takes to solve these problems is proportional to a polynomial function of the input size (e.g., n2n^2n2, n3n^3n3, etc.). • **Examples**: Sorting algorithms like Merge Sort and searching algorithms like Binary Search are P problems. • **Significance**: Problems in P are considered "tractable" or efficiently solvable, making them practical for real-world applications. **NP (Nondeterministic Polynomial Time) Problems:** • **Definition**: NP problems are those for which a given solution can be verified in polynomial time by a deterministic algorithm, even if finding the solution might not be feasible in polynomial time. • **Examples**: The Traveling Salesman Problem (TSP) and the 0/1 Knapsack Problem are NP problems. Given a solution, you can quickly check whether it is correct, but finding the solution might require non-polynomial time. • **Significance**: NP problems are crucial because they include many real-world problems for which no efficient solution algorithm is known. The famous "P vs NP" question asks whether every problem whose solution can be quickly verified (NP) can also be quickly solved (P). This remains one of the most important open questions in computer science.	
5 b	**Define 2 marks, difference 2 marks student can write any difference from the following** The **greedy technique** is an algorithmic approach that builds up a solution piece by piece, always choosing the option that looks the best at the moment (the "greedy" choice). The idea is that by making locally optimal choices at each step, you will eventually arrive at a globally optimal solution. **Differences Between Greedy Technique and Dynamic Programming:** 1. **Approach**: ○ **Greedy**: Makes a series of choices, each of which looks best in the moment, without worrying about the consequences down the line. ○ **Dynamic Programming**: Considers the consequences of each choice by solving subproblems and building up the solution systematically. 2. **Optimality**: ○ **Greedy**: Does not always guarantee an optimal solution, unless the problem has specific properties (e.g., the greedy-choice property and optimal substructure). ○ **Dynamic Programming**: Always guarantees an optimal solution by considering all possible solutions and choosing the best one. 3. **Problem Types**: ○ **Greedy**: Works well for problems where a locally optimal choice leads to a globally optimal solution (e.g., minimum spanning tree, shortest paths in certain graphs). ○ **Dynamic Programming**: Suitable for problems where subproblems overlap and can be combined to form a global solution (e.g., knapsack, longest common subsequence).	**4**