CMR
INSTITUTE OF
TECHNOLOGY

USN

Internal Assessment Test II – April 2023

Sub:	**Design and Analysis of Algorithms**							**Sub Code:**		**22MCA15**
Date:	**26.04.23**	**Duration:**	**90 min's**	**Max Marks:**	**50**	**Sem:**	**I**	**Branch:**		**MCA**

Note : Answer FIVE FULL Questions, choosing ONE full question from each Module

	PART I	MARKS	OBE CO	OBE RBT
1	Implement fractional Knapsack problem for the following data and write the algorithm to identify the items that should be kept in the sack: Weights: {1,3,4,5}, Profits: {1,4,5,7}, The maximum weight capacity is 7 kg	[4+6]	CO1	L4

OR

2 Consider the five-symbol alphabet {A, B, C, D, _} with the following occurrence frequencies in a text made up of these symbols:

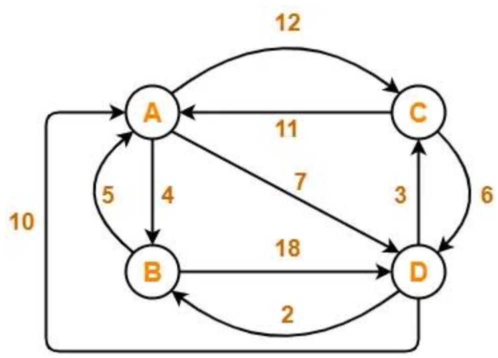
symbol	A	B	C	D	_
frequency	0.4	0.1	0.2	0.15	0.15

Create a Huffman tree and construct the codes for the symbols. Then encode ABACABAD and decode 100010111001010 using the code of question.

[6+2+2] CO1 L4

PART II

3 Write the Traveling Salesman Algorithm and explain using the following graph.



[4+6] CO1 L4

OR

4 Write Floyd's algorithm and explain with the following graph:

{ {0, 5, INF, 10},
{INF, 0, 3, INF},
{INF, INF, 0, 1},
{INF, INF, INF, 0} }

Mention the time complexity with justification.

[3+5+2] CO1, CO2 L3, L4

PART III

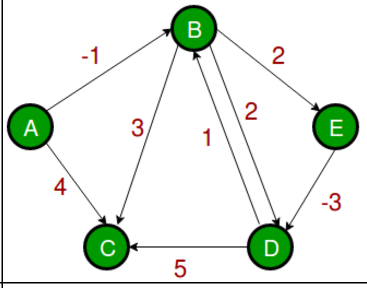
5 Find the optimal Binary Search Tree from the given keys and their frequencies:

Keys	10	20	30	40
Frequencies	4	2	6	3

[10] CO1 L4

OR

6 Implement Bellman Ford algorithm on the graph below and discuss the drawback of the algorithm: [8+2] CO1 L4



PART IV

7 Write the algorithm for job sequencing with deadline and implement the same on the data given below: [4+6] CO1 L4

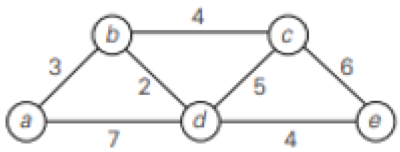
Job	J1	J2	J3	J4	J5
Deadline	2	2	1	3	4
Profit	20	60	40	100	80

OR

8 Differentiate between Greedy approach and Dynamic Programming in algorithms. Apply Coin Change problem (greedy method) on an input sum of Rs.5493 and use denominations of 1, 2, 5, 10, 20, 50, 100, 200, 500, 2000. Mention the time complexity with justification. [3+5+2] CO1, CO2 L3, L4

PART V

9 Write Dijkstra's algorithm and apply it on the graph below. [4+6] CO1 L4



OR

10 Apply Multistage Graph algorithm on [10] CO1 L4

```
graph[N][N] =
      {{INF, 1, 2, 5, INF, INF, INF, INF},
       {INF, INF, INF, INF, 4, 11, INF, INF},
       {INF, INF, INF, INF, 9, 5, 16, INF},
       {INF, INF, INF, INF, INF, INF, 2, INF},
       {INF, INF, INF, INF, INF, INF, INF, 18},
       {INF, INF, INF, INF, INF, INF, INF, 13},
       {INF, INF, INF, INF, INF, INF, INF, 2},
      {INF, INF, INF, INF, INF, INF, INF, INF}}
```

DESIGN AND ANALYSIS OF ALGORITHMS
IAT-2[26/04/2023]

Q1. <u>Fraction_Knapsack(W[],P[],B,n)</u>

1. Create an empty array X[] of size n
2. Repeat step 3 for n times for i in 0 to n-1.
3. X[i] <- P[i]/W[i]
4. Sort P[],W[] and X[] in descending order w.r.t the values in X[], set p,i to 0.
5. Repeat step 6 to 10 as long as B>0 and i<n
6. If W[i]<=B then go to 7 else go to 9
7. p<-p+P[i]
8. B<-B-W[i]
9. p <- p+(B/W[i])*P[i], go to 11
10. i <- i+1
11. Return p

Weight(W[])	1	3	4	5
Profit(P[])	1	4	5	7
X[](P[]/W[])	1	1.33	1.25	1.4

After sorting

Weight(W[])	5	3	4	1
Profit(P[])	7	4	5	1
X[](P[]/W[])	1.4	1.33	1.25	1

B= 7, n=4

Profit = 7 + (0.67*4) = 9.67

Q2.

0100011101000110

A: 0

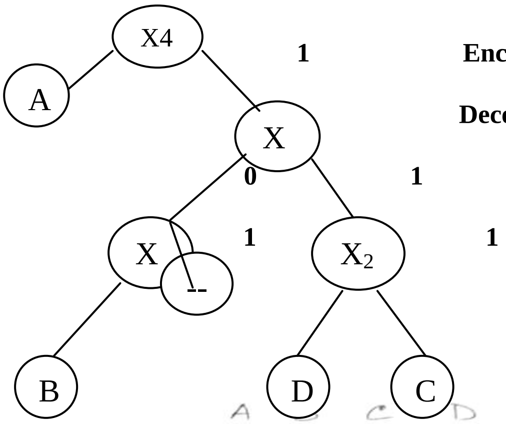
BA_DA_A

B: 100

C: 111

D: 110

_: 101

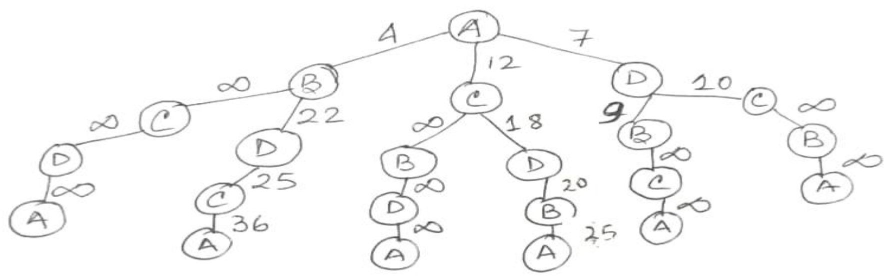


Encoding: ABACABAD ->

Decoding: 100010111001010 ->

Q3.

	A	B	C	D
A	0	4	12	7
B	5	0	∞	18
C	11	∞	0	6
D	10	2	3	0



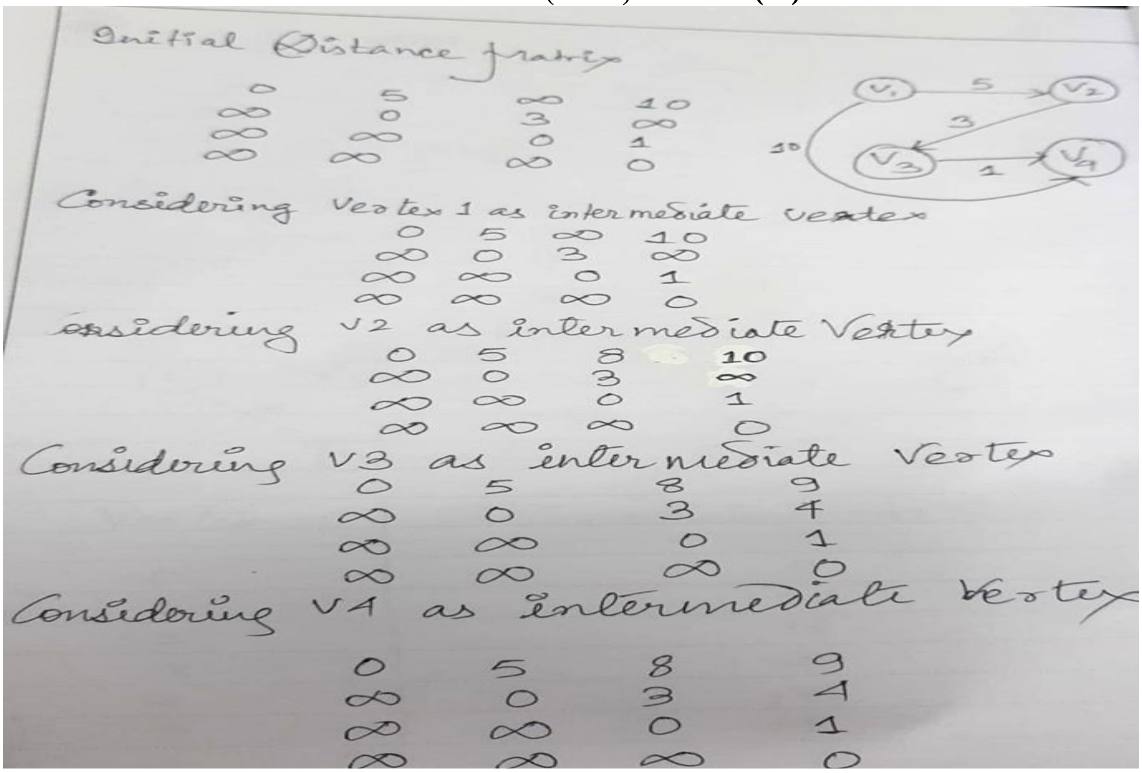
$\boxed{A - C - D - B - A}$

Q4. **<u>Floyd's Algorithms</u>**

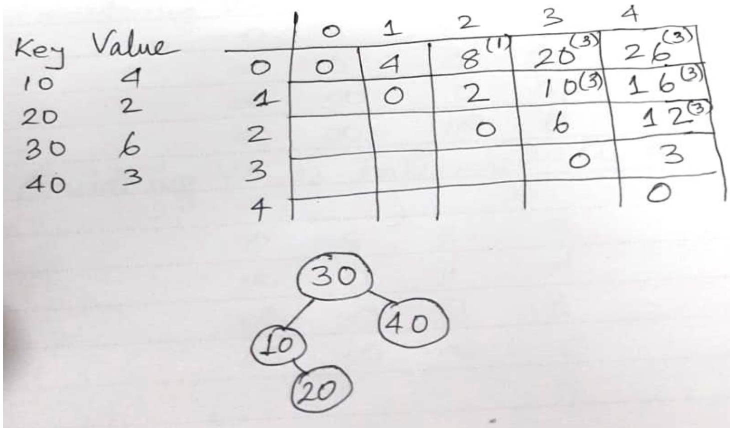
- Initialize the solution matrix same as the input graph matrix as a first step.
- Then update the solution matrix by considering all vertices as an intermediate vertex.
- The idea is to one by one pick all vertices and updates all shortest paths which include the picked vertex as an intermediate vertex in the shortest path.
- When we pick vertex number k as an intermediate vertex, we already have considered vertices {0, 1, 2, .. k-1} as intermediate vertices.

- For every pair (i, j) of the source and destination vertices respectively, there are two possible cases.
    - k is not an intermediate vertex in shortest path from i to j. We keep the value of dist[i][j] as it is.
    - k is an intermediate vertex in shortest path from i to j. We update the value of dist[i][j] as dist[i][k] + dist[k][j] if dist[i][j] > dist[i][k] + dist[k][j]

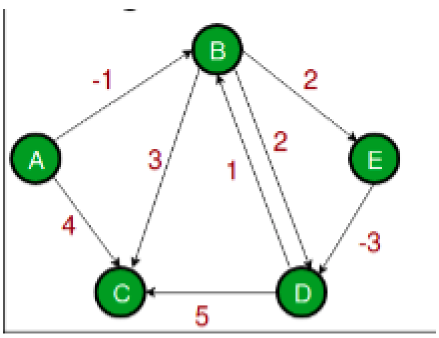
Time Complexity: If there are V vertices, then to find the shortest distance from one vertex to all the remaining vertices (V-1) will take time in the order of $O(V^2)$. This is basically the worst case for Dijkstra's. Floyd Warshall is the Dijkstra's applied to all the vertices so the time taken will be of the order $O(V . V^2)$ which is $\mathbf{O(V^3)}$.



Q5.



Q6.



ITERATION	A	B	C	D	E
0	0	INF	INF	INF	INF
1	0	-1	INF	INF	INF
	0	-1	4	INF	INF
	0	-1	2	INF	INF
2	0	-1	2	INF	1
	0	-1	2	1	1
	0	-1	2	-2	1

Q7. Job Scheduling(P[],D[])

Sort all jobs in decreasing order of P[].
Iterate on jobs in decreasing order of profit. For each job k, do the following :

- Find a time slot i, such that slot is empty and i < D[k] and i is greatest.
  Put the job in this slot and mark this slot filled.
- If no such i exists, then ignore the job.

Slot	1	2	3	4
Job Number	J3	J2	J4	J5
Deadline	1	2	3	4
Profit	40	60	100	80

Total Profit = 280
8.

Feature	Greedy method	Dynamic programming
Feasibility	In a greedy Algorithm, we make whatever choice seems best at the moment in the hope that it will lead to global optimal solution.	In Dynamic Programming we make decision at each step considering current problem and solution to previously solved sub problem to calculate optimal solution .
Optimality	In Greedy Method, sometimes there is no such guarantee of getting Optimal Solution.	It is guaranteed that Dynamic Programming will generate an optimal solution as it generally considers all possible cases and then choose the best.
Recursion	A greedy method follows the problem solving heuristic of making the locally optimal choice at each stage.	A Dynamic programming is an algorithmic technique which is usually based on a recurrent formula that uses some previously calculated states.
Memoization	It is more efficient in terms of memory as it never look back or revise previous choices	It requires Dynamic Programming table for Memoization and it increases it's memory complexity.
Time complexity	Greedy methods are generally faster. For example, Dijkstra's shortest path algorithm takes O(ELogV + VLogV) time.	Dynamic Programming is generally slower. For example, Bellman Ford algorithm takes O(VE) time.
Fashion	The greedy method computes its solution by making its choices in a serial forward fashion, never looking back or revising previous choices.	Dynamic programming computes its solution bottom up or top down by synthesizing them from smaller optimal sub solutions.
Example	Fractional knapsack .	0/1 knapsack problem

Amount = 5493

Denomination	Count	Value
2000	2	4000
500	2	1000
200	2	400
100	0	0
50	1	50
20	2	40
10	0	0
5	0	0
2	1	2
1	1	1

Time Complexity: First to sort the denominations in descending order it shall take O(nlogn), where n is the number of denominations, in this problem, 10. Further process can take at most O(n). Thus finally it is **O(nlogn).**

9. <u>Dijkstra's()</u>

- Create a set **sptSet** (shortest path tree set) that keeps track of vertices included in the shortest path tree, i.e., whose minimum distance from the source is calculated and finalized. Initially, this set is empty.

- Assign a distance value to all vertices in the input graph. Initialize all distance values as **INFINITE**. Assign the distance value as 0 for the source vertex so that it is picked first.
- While **sptSet** doesn't include all vertices
  - Pick a vertex **u** that is not there in **sptSet** and has a minimum distance value.
  - Include u to **sptSet**.
  - Then update the distance value of all adjacent vertices of u.
    - To update the distance values, iterate through all adjacent vertices.
    - For every adjacent vertex v, if the sum of the distance value of u (from source) and weight of edge u-v, is less than the distance value of v, then update the distance value of v.

If we consider the Vertex A as the source, then

Vertex	**Distance from the Source**
A	0
B	3
C	7
D	5
E	9

10.