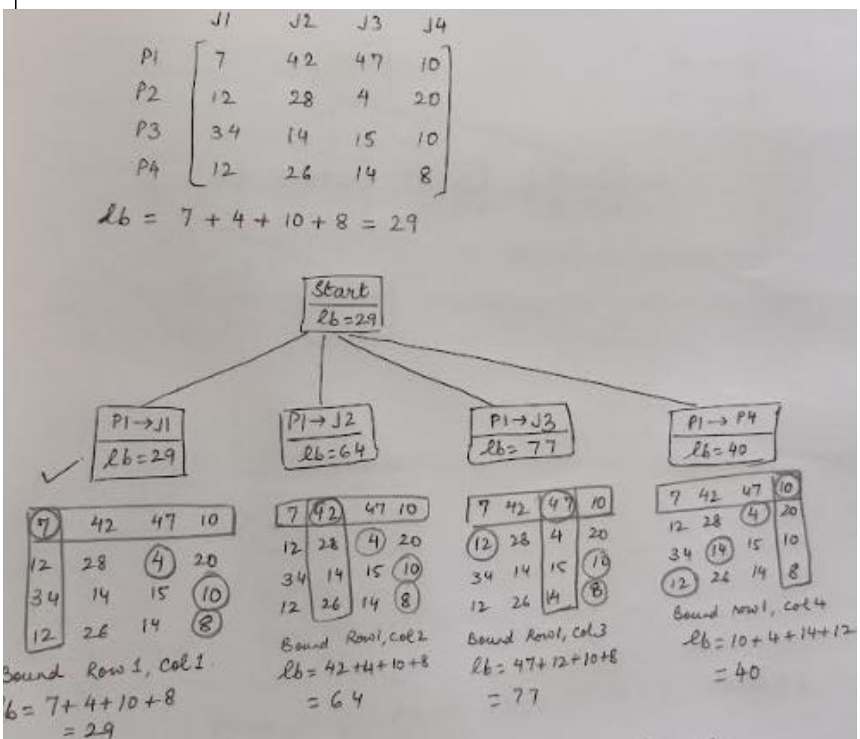

USN

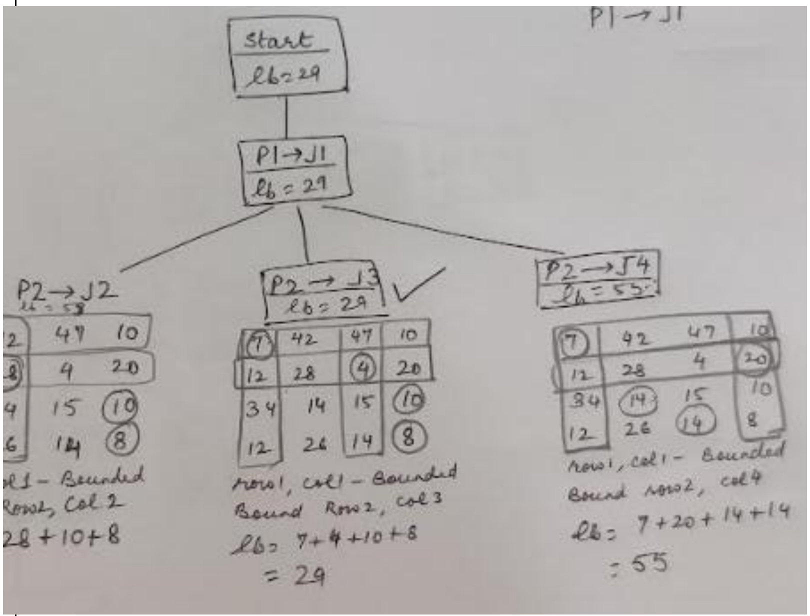
Internal Assessment Test 3 – September 2023

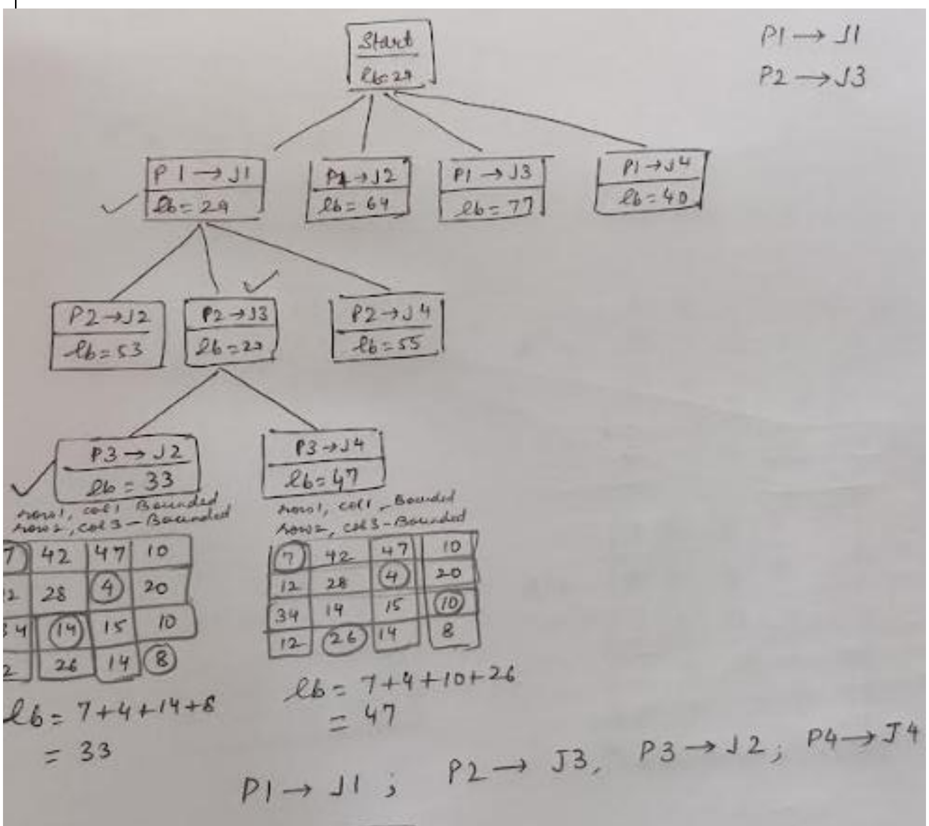
Sub	Design and Analysis of Algorithm					Sub Code	21CS42	Branch:	AIML &AIDS
Date	11/09/2023	Duration	90 mins	Max Marks	50	Sem/Sec	4 A		OBE

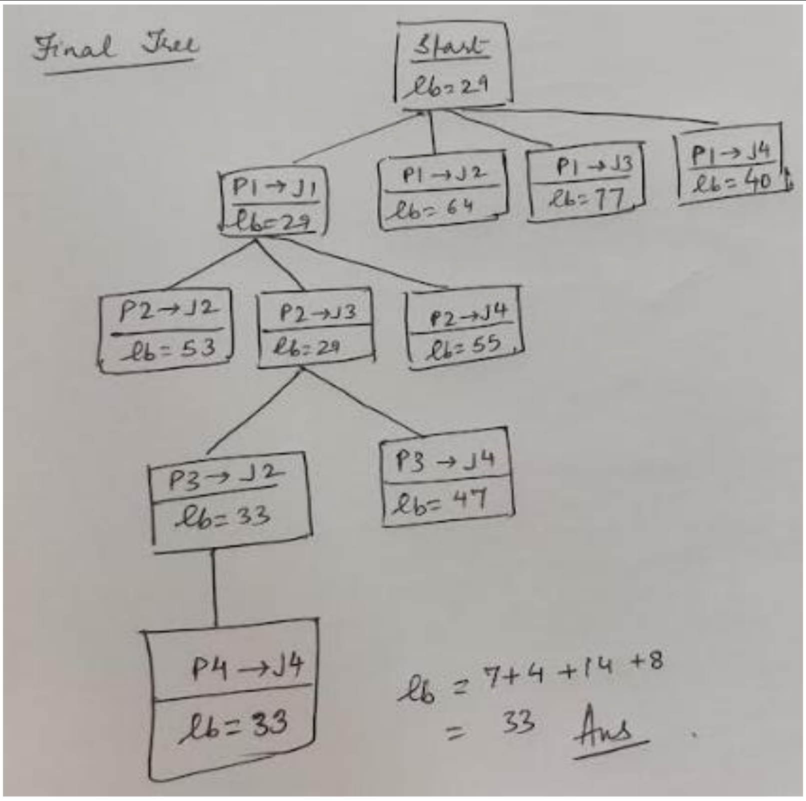
	Answer any FIVE FULL Questions	MAR KS	CO	RBT
1	Apply Branch and Bound to the following instance of assignment problem and obtain optimal solution.	[10] 1+ 2+ 7	CO 5	L 3

	J1	J2	J3	J4
Person1	7	42	47	10
Person2	12	28	4	20
Person3	34	14	15	10
Person4	12	26	14	8




P1 → J1
Start
lb=29
P1→J1
lb = 29
P2→J2
P2 → J3
lb= 29
P2 → J4
lb = 55
row1, col1 - Bounded
Bound Row2, col3
lb= 7+4+10+8
= 29
row1, col1 - Bounded
Bound row2, col4
lb= 7+20+14+14
= 55


Start
lb=29
P1 → J1
P2 → J3
P1→J1
lb= 29
P1→J2
lb= 64
P1 → J3
lb= 77
P1→J4
lb=40
P2→J2
lb=53
P2→J3
P2→J4
lb=55
P3 → J2
lb = 33
P3→J4
lb=47
row1, col1 - Bounded
row2, col3 - Bounded
lb= 7+4+14+8
= 33
lb= 7+4+10+26
= 47
P1→ J1 ; P2→ J3, P3→J2; P4→J4



2 (a) Give Warshall's algorithm for transitive closure. Find the transitive closure matrix for the graph whose adjacency matrix is given below: [6] 2+4 CO 4 L 2

1	0	0	1	0
0	1	0	0	0
0	0	0	1	1
1	0	0	0	0
0	1	0	0	1

•To find the existence of path between all the pair of vertices in a given weighted connected graph. Applicable to both directed and undirected weighted graph Warshall's Algorithm is to determine Transitive Closure of a Directed graph or all paths in a directed graph using adjacency matrix. Generate Transitive Closure of a digraph with the help of DFS or BFS

Warshall's Algorithm (pseudocode and analysis)

ALGORITHM *Warshall*($A[1..n, 1..n]$)
//Implements Warshall's algorithm for computing the transitive closure
//Input: The adjacency matrix A of a digraph with n vertices
//Output: The transitive closure of the digraph
$R^{(0)} \leftarrow A$
for $k \leftarrow 1$ **to** n **do**
  for $i \leftarrow 1$ **to** n **do**
    for $j \leftarrow 1$ **to** n **do**
      $R^{(k)}[i, j] \leftarrow R^{(k-1)}[i, j]$ **or** $(R^{(k-1)}[i, k]$ **and** $R^{(k-1)}[k, j])$
return $R^{(n)}$

Time efficiency: $\Theta(n^3)$

Space efficiency: Matrices can be written over their predecessors

Warshall's Algorithm (matrix generation)

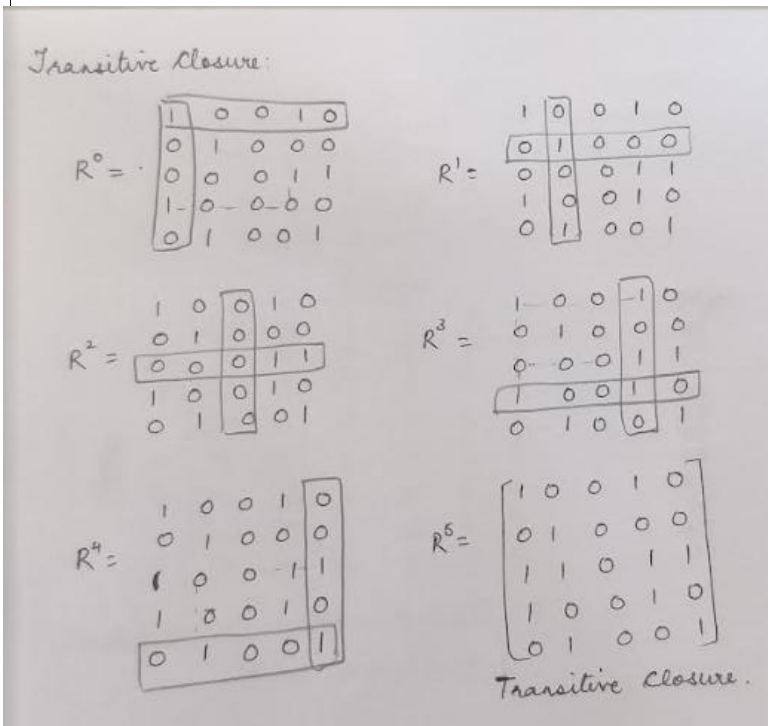
Recurrence relating elements $R^{(k)}$ to elements of $R^{(k-1)}$ is:

$$R^{(k)}[i,j] = R^{(k-1)}[i,j] \text{ or } (R^{(k-1)}[i,k] \text{ and } R^{(k-1)}[k,j])$$

It implies the following rules for generating $R^{(k)}$ from $R^{(k-1)}$:

Rule 1 If an element in row i and column j is 1 in $R^{(k-1)}$, it remains 1 in $R^{(k)}$

Rule 2 If an element in row i and column j is 0 in $R^{(k-1)}$, it has to be changed to 1 in $R^{(k)}$ if and only if the element in its row i and column k and the element in its column j and row k are both 1's in $R^{(k-1)}$

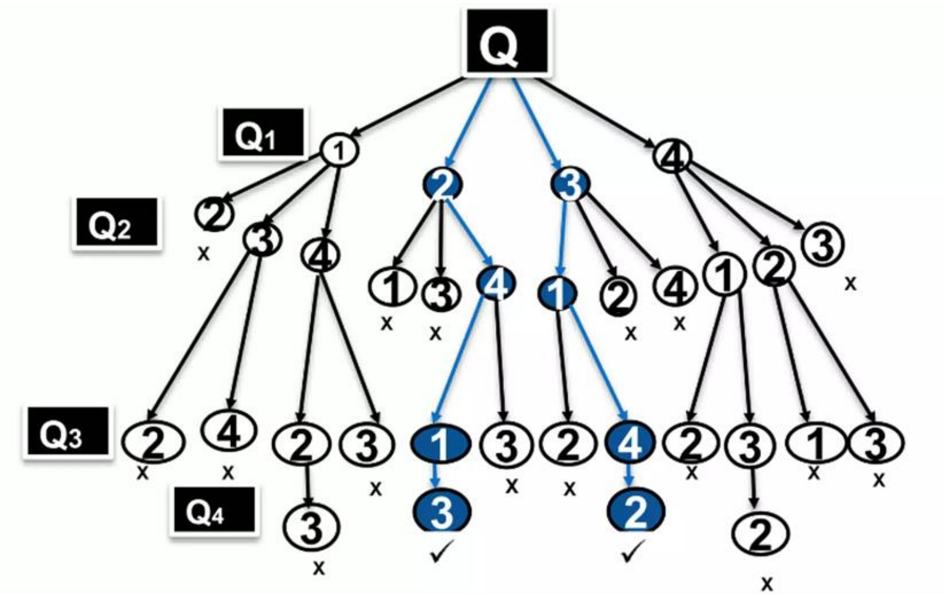


(b)	Give the control abstraction (General Algorithm) for Backtracking. Give two advantages of Backtracking.	[4] 2+2	CO 5	L 1

Algorithm backtrack(u)
//Input: node u, starts with the root of the state space tree
// Output: Result of the problem

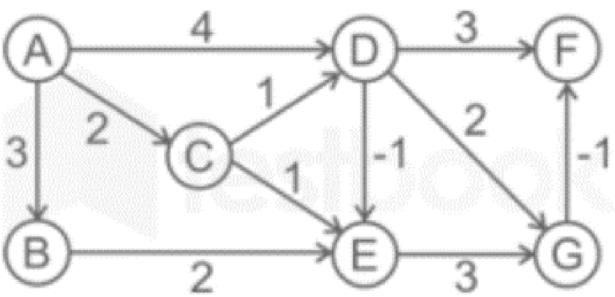
```
{
  if promising (u) then
    if (u is a goal) then
      print the solution
    else
      for each v, v belongs to child(u) ) do
         backtrack(v)
      end for
    end if
  end if
```

	}
	
	OR
	
	Algorithm backtrack_enhanced(u)
	//Input: node u, starts with the root of the state space tree
	// Output: Result of the problem
	{
	generate children v of node u
	for each v, v belongs to child(u)) do
	if promising (u) then
	if (u is a goal) then
	print the solution
	else
	backtrack_enhanced(v)
	end if
	end if
	end for
	}
	Advantages of Backtracking
	•It has a brute-force nature; due to this reason, it can solve maximum problems.
	•Backtracking problems are very intuitive to code.
	•The step-by-step representation of the backtracking solution is straightforward to understand.
	•You can easily debug backtracking code.
	•The backtracking code contains less Lines of code (LOC). Most backtracking codes are generally a few lines of recursive function code.			

3 (a)	Construct state space tree for solving 4 Queen's problem using backtracking  The solution for 4 queen problem are : 2 4 1 3 & 3 1 4 2	[6]	CO 5	L 2
(b)	Compare Backtracking design technique with branch and bound technique.	[4]	CO 5	L 1

Backtracking	Branch and Bound
Backtracking is normally used to solve decision problems	Branch and bound is used to solve optimization problems
Nodes in the state-space tree are explored in depth-first order in the backtracking method	Nodes in the tree may be explored in depth-first or breadth-first order in branch and bound method
It realizes that it has made a bad choice & undoes the last choice by backing up.	It realizes that it already has a better optimal solution that the pre-solution leads to so it abandons that pre-solution.
The feasibility function is used in backtracking.	Branch-and-Bound involves a bounding function.
The next move from the current state can lead to a bad choice	The next move is always towards a better solution
On successful search of a solution in state-space tree, the search stops	The entire state space tree is searched in order to find the optimal solution

4 Consider the following weighted graph. Bellman-Ford algorithm is implemented on the given graph with source A. Find the shortest distance from source A to vertex F. [10] CO 4 L3



Iterations

Nodes	1	2	3	4	5	6
A	0	0	0	0	0	0
B	∞	3	3	3	3	3
C	∞	2	2	2	2	2
D	∞	4	3	3	3	3
E	∞	∞	3	2	2	2
F	∞	∞	7	5	4	4
G	∞	∞	6	5	5	5

Edges.
AB.
AC
AD
BE
CD
CE
DE
DG
DF
EG
GF

Shortest Distance between A and F = 4

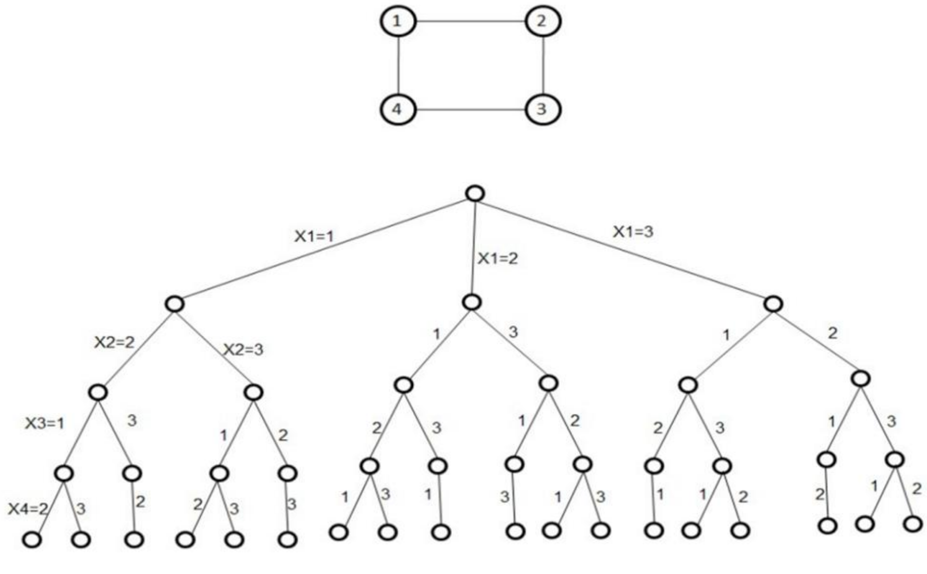
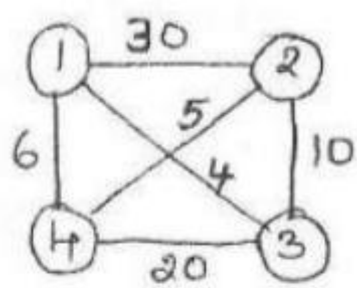
A → C → D → G → F

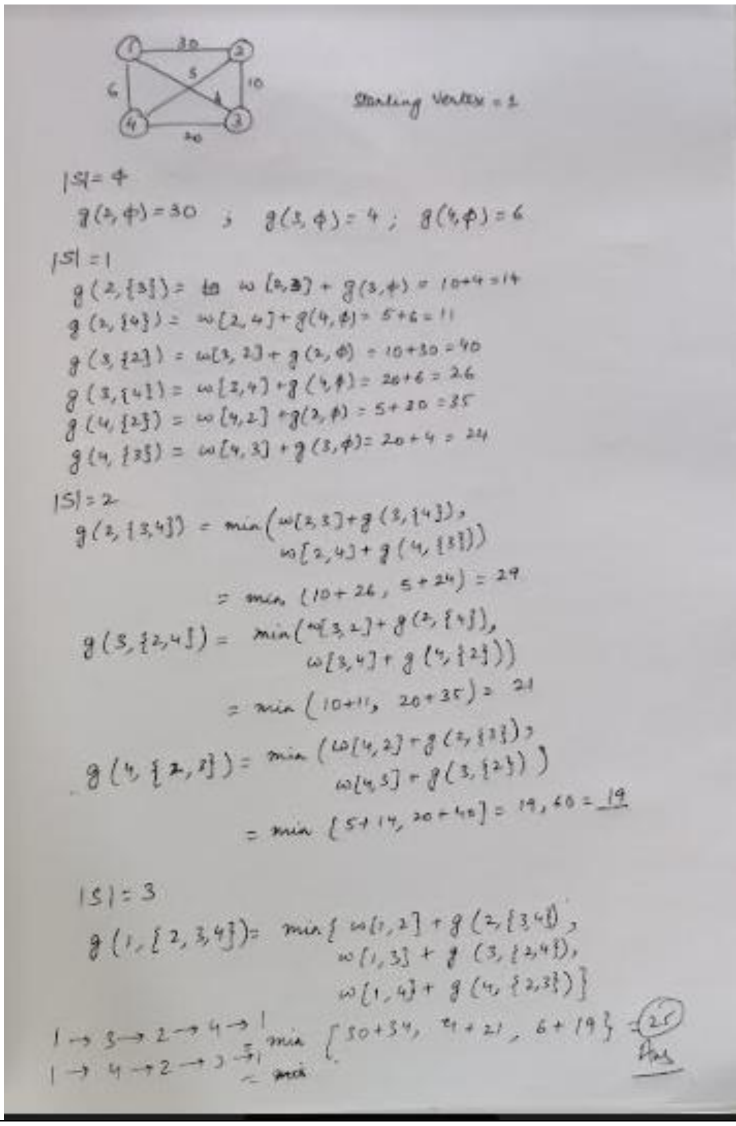
5. a) Explain the following concepts: [4] 2+2 CO 5 L2

i) State Space Tree

	ii) NP Hard problems
	
	State Space Tree:
	• Represent the solution space as a tree
	  • Each edge represents a choice of one x_i
	    • Level 0 to Level 1 edges show choice of x_1
	    • Level 1 to Level 2 edges show choice of x_2
	    • Level $i-1$ to Level i edges show choice of x_i
	  • Each internal node represents a partial solution
	    • Partitions the solution space into disjoint subspaces
	  • Leaf nodes represent the complete solution (may or may not be feasible)
	• Models the complete solution being built by choosing one component at a time
	
	NP Hard Problems: A problem is NP-hard if all problems in NP can be reduced to it in poly-time. We can see that NP-hard problems are ”harder” than all problems in NP. By reduction, or more specifically reducing problem B to problem A, we mean that given a “blackbox” solver that solves A, we can also solve B by transforming the instance of B to an instance of A, and then transform the solver’s solution to the instance of A to a solution to the instance of B. Observe that if A can be reduced to B and B can be reduced to C, then A can be reduced to C.
	
	What does NP-hard mean? – A lot of times you can solve a problem by reducing it to a different problem. I can reduce Problem B to Problem A if, given a solution to Problem A, I can easily construct a solution to Problem B. (In this case, "easily" means "in polynomial time.“).
	
	A problem is NP-hard if all problems in NP are polynomial time reducible to it, ...
	
	Ex:- Hamiltonian Cycle
	
	Every problem in NP is reducible to HC in polynomial time. Ex:- TSP is reducible to HC.
	
	B= lcm A= gcd
	
	Example: **lcm(m, n) = m * n / gcd 13 (m, n),**
	
	Here, we are reducing the problem of **lcm** to already solved problem **gcd**.			

b)	Draw the portion of the state space tree for m- colorings of a graph when n =4 and m = 3	[6]	CO 5	L3

	 A 4-node graph and all possible 3-colorings			
6. (a)	Give the differences between Bellman-Ford algorithm and Dijkstra's algorithm. (***) check the next page	[4]	CO 4	L2
b)	Solve the following Travelling Salesperson which is represented as a graph using dynamic programming. Start city is 1.  	[6]	CO 4	L3



CI **CCI** **HOD/AIML**

Bellman Ford	Dijkstra's
Deals with single source shortest path	Deals with single source shortest path
Allows negative length edges but does not allow negative cycle	Does not allow both negative weights and negative cycle
It is slower than dijkstra's algorithm if more edges are present	It is faster than bellman-ford algorithm
Can be easily implemented in a distributed way	cannot be implemented in a distributed way