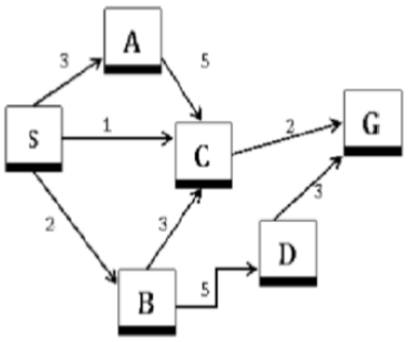
USN

Internal Assessment Test 3 – December 2022

Sub:	Artificial Intelligence and Machine Learning – Set 3				Sub Code:	18CS71	Branch:	ISE
Date:	26-12-2022	Duration:	90 Minutes	Max Marks:	50	Sem / Sec:	**7 A,B,C**	**OBE**

	Answer any FIVE FULL Questions	MARKS	CO	RBT
1	Explain the steepest ascent hill climbing search technique with an algorithm. Comment on itsdrawbacks and how to overcome these drawbacks	10	CO2	L2
2	What are AND-OR graphs, explain how the problem reduction (AO*) algorithm uses AND-OR graphs for the search procedure. Apply the O* algorithm for the below graph where S is the Initial State and G is the Goal state. The table below indicates h' values	10	CO2	L3

State	h(n)
S	6
A	2
B	4
C	1
D	2
G	0



3	Write the algorithm for Best -first search with an example. Write A* Algorithm.	10	CO2	L1

4	Define Artificial Intelligence and classify the task domains of AI	10	CO3	L1
5	Mention the different approaches used to solve the tic-tac-toe problem. Apply any two AI techniques for solving tic-tac-toe problem.	10	CO3	L3
6	Explain in detail about Constraint Satisfaction Problem with an example.	10	CO3	L2
7	Consider the following statements. Convert these statements into equivalent predicate logic form. i. Marcus was a man. ii. Marcus was a Pompeian iii. All Pompeians were Romans iv. Caesar was a ruler v. All Romans were either loyal to Caesar or hated him vi. Everyone is loyal to someone vii. People only try to assassinate rulers they are not loyal to. viii. Marcus tried to assassinate Caesar.	10	CO3	L3

CI **CCI** **HOD**

Internal Assessment Test 3 Scheme and Solution – December 2022

Sub:	Artificial Intelligence and Machine Learning – Set 3	Sub Code:	18CS71	Branch :	ISE

		MARKS	CO	RBT
1	Steepest hill Climbing Search technique (4Marks) Drawbacks and to Overcome the drawback (6 Marks)			
2	AND-OR graphs (2 Marks) Problem reduction (AO*) algorithm (3 Marks) Steps and Path to reach the Goal state (5 Marks)			
3	Algorithm for Best -first search (4 Marks) Example (2 Marks) A* Algorithm (4 Marks)			
4	Definition of Artificial Intelligence (2 Marks) Task Domains of AI (8 Marks)			
5	Definition of production system (2 Marks) Different classes of production system (4 Marks) Algorithm for production system (4 Marks)			
6	Constraint Satisfaction Problem (5 Marks) Example (5 Marks)			
7	Definition of CNF (2 Marks) Algorithm for converting proposition logic into CNF (4 Marks) Example – Conversion (4 Marks)			
1	A useful variation on simple hill climbing considers all the moves from the current state and selects the best one as the next state. This method is called *steepest-ascent hill climbing* or *gradient search.* Algorithm: 1. Evaluate the initial state. If it is also a goal state then return it and quit. Otherwise continue with the initial state as the current state. 2. Loop until a solution is found or until a complete iteration produces no change to current state: a. Let SUCC be a state such that any possible successor of the current state will be better than SUCC. b. For each operator that applies to the current state do: i. Apply the operator and generate a new state. ii. Evaluate the new state. If it is a goal state, then return it and quit. If not compare it to SUCC. If it is better, then set SUCC to this state. If it is not better, leave SUCC alone. c. IF the SUCC is better than current state, then set current state to SUCC.	10	CO2	L2

	Bothe basic and steepest-ascent hill climbing may fail to find a solution. Either algorithm may terminate not by finding a goal state but by getting a state from which no better states can be generated. This will happen if the program has reached a local maximum, a plateau or a ridge. A local maximum is a state that is better than all its neighbour's but it not better than some other states farther away. At the local maximum, all moves appear to make things worse. Local maxima are particularly frustrating because they often occur almost within sight of a solution. In this case, they are called foothills. A plateau is a flat area of the search space in which a whole set of neighbouring states has the same value. In this, it is not possible to determine the best direction in which to move by making local comparisons. A *ridge* is a special kind of maximum. It is an area of the search space that is higher than surrounding areas and that itself has a slope. There are some ways of dealing with these problems, although these methods are by no means guaranteed: □ Backtrack to some earlier node and try going in a different direction. This is particularly reasonable if at that node there was another direction that looked as promising or almost as promising as the one that was chosen earlier. This is a fairly good way to deal with local maxima. □ Make a big jump in some direction to try to get to a new section of the search space. This is a good way of dealing with plateaus. □ Apply two or more rules before doing the test. This corresponds to moving in several directions at once. This is a good strategy for dealing with ridges.			
2	What are AND-OR graphs, explain how problem reduction (AO*) algorithm uses AND-OR graphs for search procedure. Apply AO* algorithm for the below graph where S is the Initial State and G is the Goal state. The table below indicates h' values. **AND-OR Graphs:** Useful for representing the solution of problems that can be solved by **decomposing them into a set of smaller problems**, all of which must be then solved. Some problems are best represented as achieving sub goals, some of which achieved simultaneously and independently (AND) Up to now, only dealt with OR options **AO* Algorithm:** 1. Initialize the graph to start node 2. Traverse the graph following the current path accumulating nodes that have not yet been expanded or solved 3. Pick any of these nodes and expand it and if it has no successors call this value *FUTILITY* otherwise calculate only *f'* for each of the successors. 4. If *f'* is 0 then mark the node as *SOLVED* 5. Change the value of *f'* for the newly created node and let f' reflect on its predecessors by back propagation.	10	CO2	L3

6. Wherever possible use the most promising routes and if all descendants of a node is marked as *SOLVED* then mark the parent node as *SOLVED*.
If starting node is *SOLVED* or value greater than *FUTILITY*, stop, else repeat from 2.

Current state is S

- f(A)= 3+2 =5, f(B)= 2+4=6, f(C)=1+1=2
- Since, C is having smaller distance compared to other nodes, we have chosen current state as C.

2.Current State is C

- f(C)= 2+0 = 2

Path = S -> C -> G

3	Write the algorithm for Best -first search with example	10	CO2	L1

Best-First Search (BFS) is a way of combining the advantages of both depth-first search and breadth first search into a single method, i.e., is to follow a single path at a time but switch paths whenever completing path looks more promising than the current one does.

The process is to select the most promising of the new nodes we have generated so far. We then expand the chosen node by using the rules to generate its successors. If one of them is a solution, then we can quit, else repeat the process until we search goal.

In BFS, one move is selected, but others are kept around so that they can be revisited later if the selected path becomes less promising. This is not the case steepest ascent climbing.

OR Graphs

A graph is called OR graph, since each of its branches represents alternative problems solving path.

To implement such a graph procedure, we will need to use lists of nodes:

1) OPEN: nodes that have been generated and have had the heuristic function applied to them which have not yet been examined. It is a priority queue in which the elements with highest priority are those with the most promising value of the heuristic function.

2) CLOSED: nodes that have already been examined whenever a new node is generated we need to check whether it has been generated before.

3) A heuristic function f which will estimate the merits of each node we generate.

Algorithm:

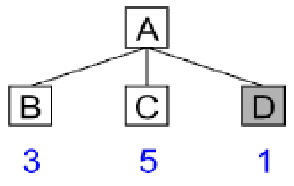
```
1. Start with OPEN containing just the initial state
2. Until a goal is found or there are no nodes left on OPEN do:
        a. Pick the best node on OPEN
        b. Generate its successors
        c. For each successor do:
                i. If it is not been generated before, evaluate it, add it to OPEN and
record its parent.
```

ii. If it has been generated before, change the parent if this new path is better than the previous one. In that case update the cost of getting to this node and to any successors that this node may already have.

Step 1:

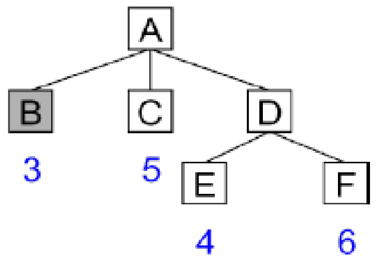
A → NIL

Step 2:



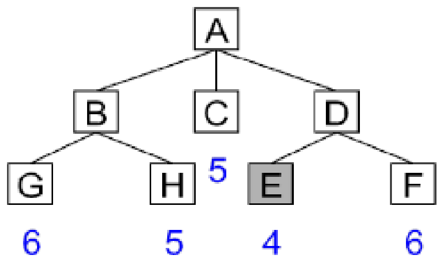
A → NIL
B → A
C → A
D → A

Step 3:



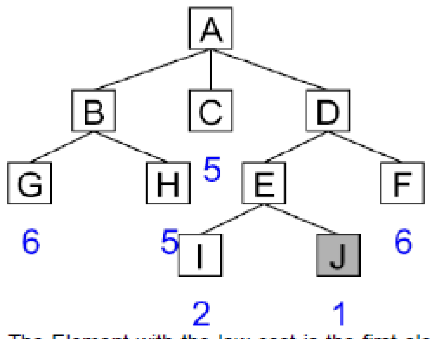
A → NIL
B → A
C → A
D → A
E → D
F → D

Step 4:



A → NIL
B → A
C → A
D → A
E → D
F → D
G → B
H → B

Step 5:



A → NIL
B → A
C → A
D → A
E → D
F → D
G → B
H → B
I → E
J → E

The Element with the low cost is the first element. The new states are added according to the cost value.

Write A* Algorithm

A* algorithm is a best first graph search algorithm that finds a least cost path from a given initial node to one goal node. The simplification of Best First Search is called A* algorithm.

This algorithm uses , functions as well as the lists OPEN and CLOSED.

For many applications, it is convenient to define function as the sum of two components
that we call g and h'.

$$f' = g + h'$$

- ***g*** **:**
– Measures of the cost of getting from the initial state to the current node.
– It is not the estimate; it is known to be exact sum of the costs.
- **h' :**
– is an estimate of the additional cost of getting from current node to goal state.

Algorithm:

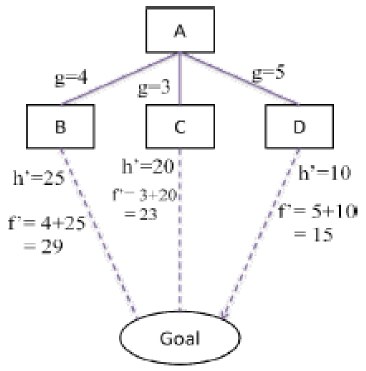
1) Start with OPEN containing only the initial state (node) set that node g value 0 its h' value to whatever it is and its ' value '+ 0 or '. Set CLOSED to the empty list.

2) Until a goal node is found repeat the following procedure: If there are no nodes on OPEN, report failure. Otherwise pick the node on OPEN with lowest ' value. CALL it BESTNODE. Remove from OPEN. Place it on CLOSED. If BESTNODE is the goal node, exit and report a solution. Otherwise, generate the successors of BESTNODE. For each successor, do the following
   a) Set successors to point back to BESTNODE this backwards links will make possible to recover the path once a solution is found.

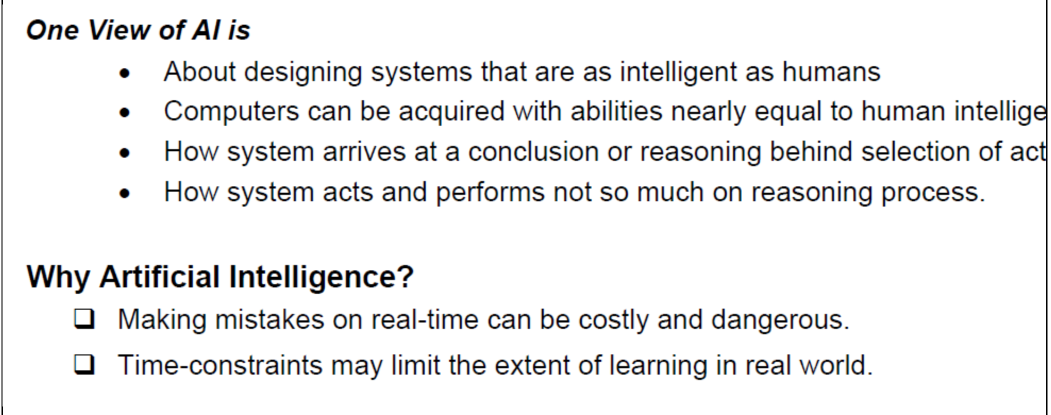
   b) Compute
   $$g(successor) = g(BESTNODE) + cost\ of\ getting\ from\ BESTNODE\ to\ successor$$
   c) If successor is already exist in OPEN call that node as OLD and we must decide whether OLD' s parent link should reset to point to BESTNODE (graphs exist in this case)

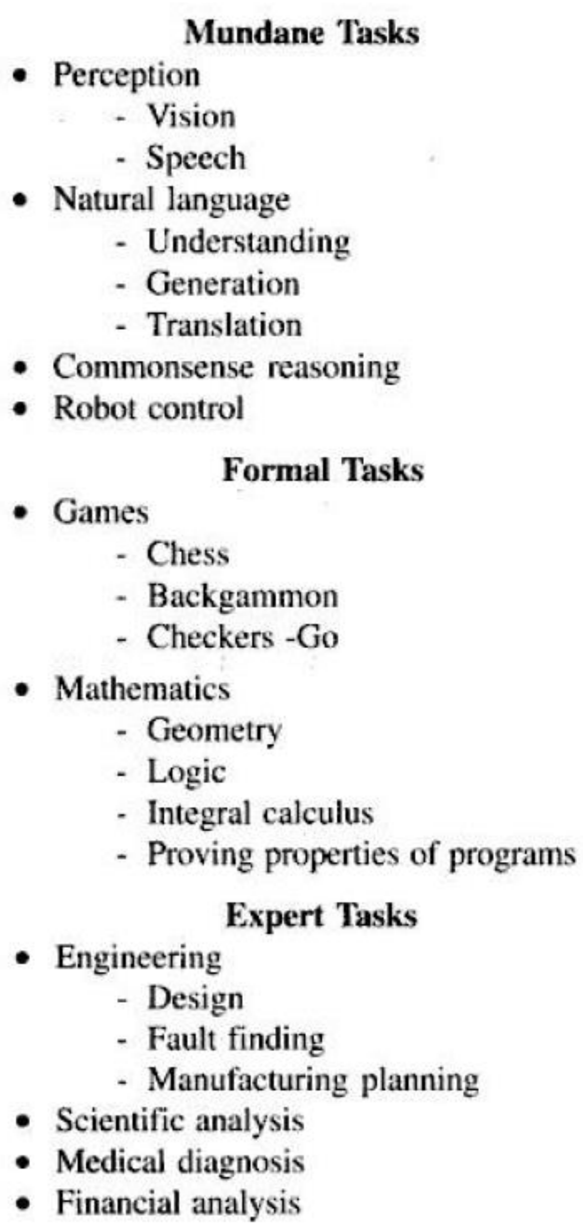
      If OLD is cheaper then we need do nothing. If successor is cheaper then reset OLD's parent link to point to BESTNODE. Record the new cheaper path in ( ) and update '( ).
   d) If SUCCESSOR was not on OPEN, see if it is on CLOSED. If so, call node on CLOSED OLD and add OLD to the list of BESTNODE successors. Calculate all the g, f' and h' values for successors of that node which is better then move that.

      So to propagate the new cost downward, do a depth first traversal of the tree starting at OLD, changing each nodes value (and thus also its ' value), terminating each branch when you reach either a node with no successor or a node which an equivalent or better path has already been found.
   e) If successor was not already on either OPEN or CLOSED, then put it on OPEN and add it to the list of BESTNODE successors. Compute
   $$f'(successor) = g(successor) + h'(successor)$$



A* algorithm is often used to search for the lowest cost path from the start to the goal location in a graph of visibility/quad tree. The algorithm solves problems like 8-puzzle problem and missionaries & Cannibals problem.

4	Define Artificial Intelligence and classify the task domains of AI  ***One View of AI is*** • About designing systems that are as intelligent as humans • Computers can be acquired with abilities nearly equal to human intellige • How system arrives at a conclusion or reasoning behind selection of act • How system acts and performs not so much on reasoning process. **Why Artificial Intelligence?** ❑ Making mistakes on real-time can be costly and dangerous. ❑ Time-constraints may limit the extent of learning in real world.	10	CO3	L1

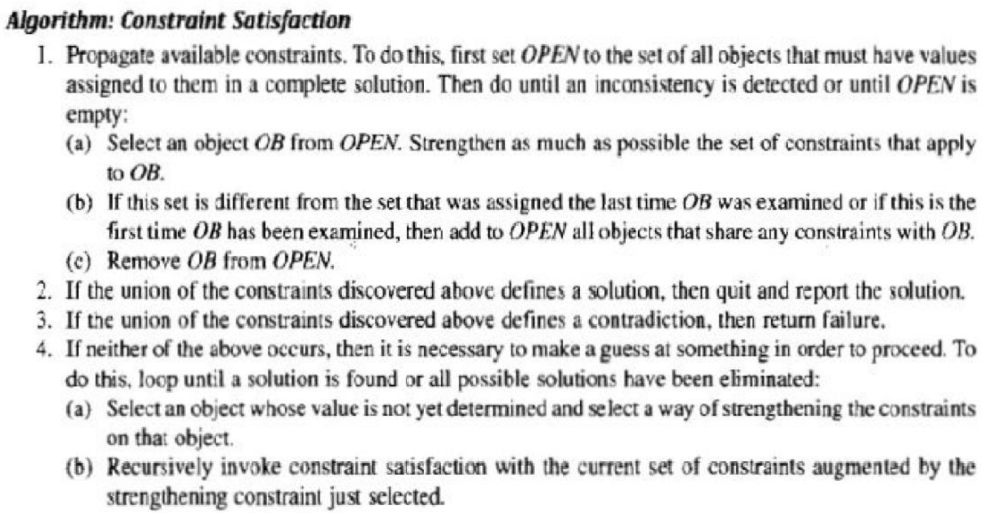
	 **Mundane Tasks** • Perception - Vision - Speech • Natural language - Understanding - Generation - Translation • Commonsense reasoning • Robot control **Formal Tasks** • Games - Chess - Backgammon - Checkers -Go • Mathematics - Geometry - Logic - Integral calculus - Proving properties of programs **Expert Tasks** • Engineering - Design - Fault finding - Manufacturing planning • Scientific analysis • Medical diagnosis • Financial analysis			
5	Mention the different approaches used to solve the tic-tac-toe problem. Apply any two AI techniques for solving tic-tac-toe problem. Program 1: Data Structures: • Board: 9 element vector representing the board, with 1-9 for each square. 1 2 3 4 5 6 7 8 9 An element contains the value 0 if it is blank, 1 if it is filled by X, or 2 if it is filled with a O • Movetable: A large vector of 19,683 elements (3^9), each element is 9-element vector. Algorithm: 1. View the vector as a ternary number. Convert it to a decimal number. 2. Use the computed number as an index into Move-Table and access the vector stored there. 3. Set the new board to that vector. Comments:	10	CO3	L3

	This program is very efficient in time. It has several disadvantages. 1. A lot of space to store the Move-Table. 2. A lot of work to specify all the entries in the Move-Table. 3. Difficult to extend. Program 2: Data Structure: A nine element vector representing the board. But instead of using 0,1 and 2 in each element, we store 2 for blank, 3 for X and 5 for O Turn: An integer indicating which move of the game is about to be played , 1 indicates the first move, 9 indicates the last move. Functions: Make2: returns 5 if the center square is blank. i.e Board[5]=2,Else any other blank non corner sq 2,4,6 or 8. Posswin(p): Returns 0 if the player p cannot win on his next move; otherwise it returns the number of the square that constitutes a winning move. If the product is 18 (3x3x2), then X can win. If the product is 50 (5x5x2) then O can win. Go(n): Makes a move in the square n. This sets Board[n] to 3 if turn is odd and 5 if turn is even. Also increments turn by 1. Strategy: Turn = 1 Go(1) Turn = 2 If Board[5] is blank, Go(5), else Go(1) Turn = 3 If Board[9] is blank, Go(9), else Go(3) Turn = 4 If Posswin(X) ≠ 0, then Go(Posswin(X)) else Go(Make2) Turn = 5 .... Turn = 9 Program 3: Board Position : A structure containing 9-element vector representing the board, a list of board positions that could result from the next move, and a number representing an estimate of how likely the board position is to lead to an ultimate win for the player to move. Assign rating for the best move. To decide on the next move, look ahead at the best board positions. 1. If it is a win, give it the highest rating. 2. Otherwise, consider all the moves the opponent could make next. Assume the opponent will make the move that is worst for us. Assign the rating of that move to the current node. 3. The best node is then the one with the highest rating.			
6	Explain in detail about Constraint Satisfaction Problem with an example. Search procedure operates in a space of constraint sets. Initial state contains the original constraints given in the problem description. A goal state is any state that has been constrained enough – Cryptarithmetic: “enough” means that each letter has been assigned a unique numeric value. Constraint satisfaction is a 2-step process:	10	CO3	L2

Constraints are discovered and propagated as far as possible.
If there is still not a solution, then search begins. A guess about is made and added as a new constraint.

To apply the constraint satisfaction in a particular problem domain requires the use of 2 kinds of rules:

Rules that define valid constraint propagation
Rules that suggest guesses when necessary



Algorithm: Constraint Satisfaction

1. Propagate available constraints. To do this, first set *OPEN* to the set of all objects that must have values assigned to them in a complete solution. Then do until an inconsistency is detected or until *OPEN* is empty:
   (a) Select an object *OB* from *OPEN*. Strengthen as much as possible the set of constraints that apply to *OB*.
   (b) If this set is different from the set that was assigned the last time *OB* was examined or if this is the first time *OB* has been examined, then add to *OPEN* all objects that share any constraints with *OB*.
   (c) Remove *OB* from *OPEN*.
2. If the union of the constraints discovered above defines a solution, then quit and report the solution.
3. If the union of the constraints discovered above defines a contradiction, then return failure.
4. If neither of the above occurs, then it is necessary to make a guess at something in order to proceed. To do this, loop until a solution is found or all possible solutions have been eliminated:
   (a) Select an object whose value is not yet determined and select a way of strengthening the constraints on that object.
   (b) Recursively invoke constraint satisfaction with the current set of constraints augmented by the strengthening constraint just selected.

Example:

→ 1. SEND
MORE
MONEY

0 1 2 3 4 5 6 7 8 9.

sol:-

$$\begin{array}{cccc} & c_3 & c_2 & c_1 \\ S & E & N & D \\ M & O & R & E \\ \hline M & O & N & E & Y \end{array}$$

0	1	2	3	4	5	6	7	8	9
↑	↑	↑			↑	↑	↑	↑	↑
O	m	y			E	N	D	R	S

1) Two single digit number sum is maximum 18 or 19.
(one is carry from previous addition)
So we conclude that m=1

2) $S+m+c_3 \geq \underline{10}$

$S+1+c_3 \geq 10$

The carry may be $c_3 = 0$ or 1.

$S+1+0 \geq 10 \Rightarrow S \geq 9 \Rightarrow S=9$.

$S+1+1 \geq 10 \Rightarrow S \geq 8 \Rightarrow S=8$ or 9

If $S=9 \Rightarrow 9+1+c_3 \Rightarrow c_3=0 \Rightarrow \underset{m}{1}\ \underset{o}{0}$ ✓

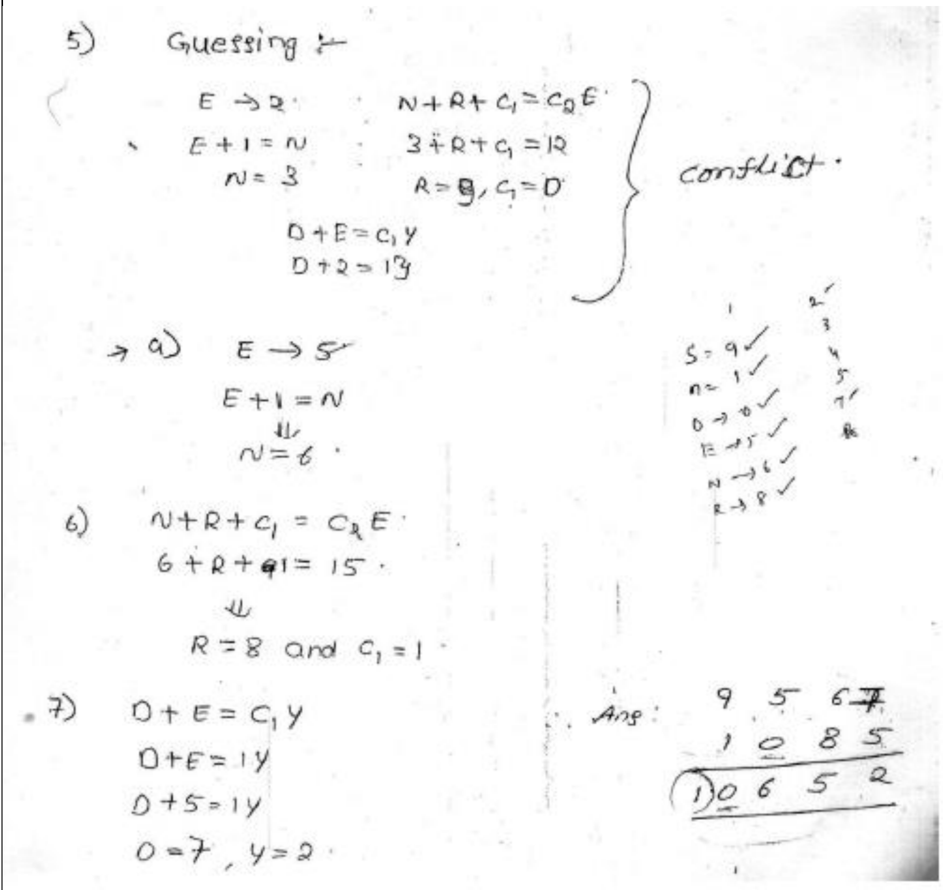
$c_3=1 \Rightarrow \underset{m}{1}\ \underset{o}{1}$ ×

$S=8 \Rightarrow 8+1+c_3 \Rightarrow \underline{c_3=1} \Rightarrow \underset{m}{1}\ \underset{o}{0}$.

3) let us consider $S=9$ ✓

4) $E+o+c_2 = N$
⇓
$E+0\cdot \cancel{c_2} = N$
$E+1=N$

it implies c_2 must be 1
$\underline{c_2=1}$

	 5) Guessing :- $E \rightarrow 2$ $N+R+C_1=C_2E$ $E+1=N$ $3+R+C_1=12$ $N=3$ $R=9, C_1=0$ $D+E=C_1Y$ $D+2=13$ conflict. → a) $E \rightarrow 5$ $E+1=N$ $\Downarrow$ $N=6$. 6) $N+R+C_1=C_2E$ $6+R+1=15$. $\Downarrow$ $R=8$ and $C_1=1$. 7) $D+E=C_1Y$ $D+E=1Y$ $D+5=1Y$ $D=7, Y=2$. $S=9$ ✓ $n=1$ ✓ $O \rightarrow 0$ ✓ $E \rightarrow 5$ ✓ $N \rightarrow 6$ ✓ $R \rightarrow 8$ ✓ 1 2 3 4 5 7 8 ∴ Ans: 9 5 6 7 1 0 8 5 1 0 6 5 2			
7	Consider the following statements. Convert these statements into equivalent predicate logic form. i. Marcus was a man. ii. Marcus was a Pompeian iii. All Pompeians were Romans iv. Caesar was a ruler v. All Romans were either loyal to Caesar or hated him vi. Everyone is loyal to someone vii. People only try to assassinate rulers they are not loyal to. viii. Marcus tried to assassinate Caesar. Solution: Converting the given statements into Predicate/Propositional Logic i. "x : food(x) ® likes (John, x) ii. food (Apple) ^ food (chicken) iii. "a : "b: eats (a, b) ^ killed (a) ® food (b) iv. eats (Bill, Peanuts) ^ alive (Bill) v. "c : eats (John, c) ® eats (Rita, c) vi. "d : alive(d) ® ~killed (d) vii. "e: ~killed(e) ® alive(e) Conclusion: likes (John, Peanuts)	10	CO3	L3