Ans-(1)

=> In order to choose the most appropriate problem solving method, it is necessary to analyze the problem.

=> The problem is analyzed along various key dimensions.

=> These key dimensions are referred to as AI problem characteristics.

=> AI Problem characteristic is discussed below:-

(1) Is the problem decomposible into a set of independent smaller or easier sub problems?

=> A very large and composite problem can be easily solved if it can be broken into smaller problems.

=> Here, Recursion could be used.

=> For example, if we want to solve:-

$$\int (x^2 + 3x)\,dx$$

=> This can be done by breaking into two smaller problems and solving each by applying specific Rules.

=> Adding the result will give complete solution.

⇒ $\int x^2 + 3x\,dx = \int x^2\,dx + \int 3x\,dx$

$= \frac{x^3}{3} + \frac{3x^2}{2}$

⇒ But, certain problems cannot be decomposed.

(2) Can Solution steps be ignored or at least undone if they prove unwise?

⇒ The program falls under 3 classes :-

(1) ignorable
(2) recoverable
(3) irrecoverable

⇒ Consider theorem proving.

⇒ This is example of ignorable solution as we may later find that it is of no use.

⇒ Consider 8-puzzle problem which is example of recoverable solution.

⇒ Here, while moving from start to goal state, we may make wrong move, but we can backtrack and undo the move.

=) Consider game of chess ; where if wrong move made, it cannot be recovered also it cannot be ignored.

=) Above is example of irrecoverable step.

(3) Is problem's universe predictable?

=) Problems can be classified into those with certain outcome like water jug and those with uncertain outcomes like playing cards.

=) In certain outcome problems, planning could be done to lead to solution.

=) In uncertain outcome problems, planning can give a high probability of solution.

(4) Is a good solution to problem obvious without comparison to all other possible solutions?

=) There are 2 categories of problems :-
(1) Any path
(2) Best path

=) In any path problem like water jug; we are satisfied with solution, not considering the path taken

=) In Best path problem like TSP, we want the best solution.

(5) Is the desired solution, a state of world or a path to a state?

=> In problem like NLP, we need to find the interpretation but not the record or path of processing.

=> In water jug problem, along with the solution, we need to provide the path.

(6) What is the role of knowledge?

=> In game of playing chess, just the rules for determining legal moves and simple controlling is sufficient to get solution.

=>

(7) Does the task require intraction with person?

=> There are 2 categories:-

(1) Solitary -> no intermediate communication required

=> e.g. theorem proving

(2) Conversational -> intermediate communication Required

=> e.g. medical diagnosis, NLP

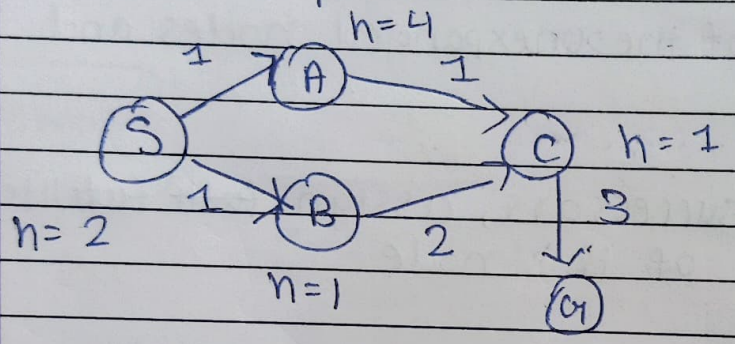
Ans-(2)

=> We need to use A* Algorithm to find path between S and G.

=> We have COST of traversing the edge.

=> Also, each node represents a heuristic estimate of distance of node to goal G.

=> Given Graph:



=> We will maintain the Queue.

=> Initially, the Queue will contain the root node.

=> Queue: <S>

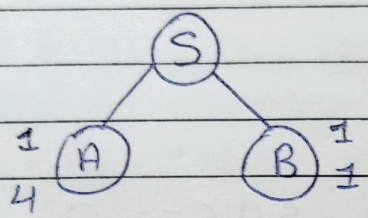
=> We will sort the Queue by f.

f = accumulated cost + heuristic.

=> Remove first path, and create all paths to children.

=> Sort the Queue by f.

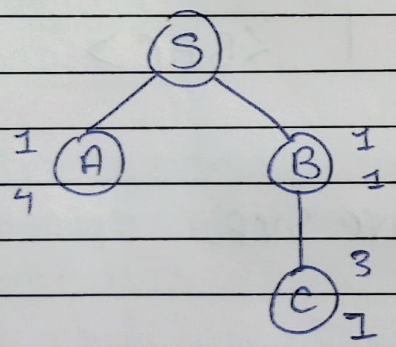
=> Queue: <SB, SA>



=> As; B has lower value of f, so it will be ahead of SA in Queue.

=> Again performing the above steps.

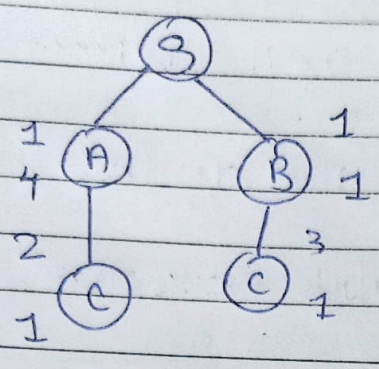
Queue: <SA, SBC>



=> Cost to reach till C => 1+2=3;

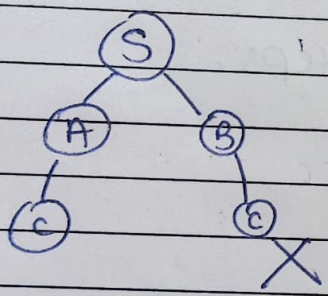
=> Again Performing above step.

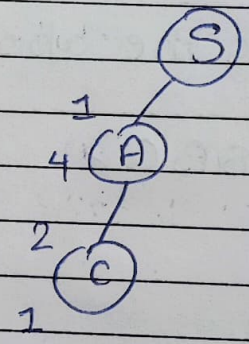
=> Queue: <SBC, SAC>



=> cost of SBC => 3; SAC => 2

=> So, we will remove SBC

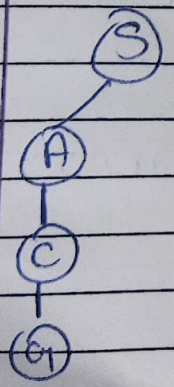
=> 

=> 

Queue: <~~SB~~
<SAC>

=> Again performing above step;

Queue: <SAC G4>

=> 

=> Here, Goal is Reached and no other alternative is there; so SUCCESS is achieved.

=> Path from S to G4

=> <SACG4> / S→A→C→G4

cost = 4

Q1 Explain Ant Colony optimization in Detail.

=> Ant Colony optimization techniave was introduced by Marco Dorigo in 1990s

=> Ants communicate with each other using touch, pheromone and sound.

=> We know that pheromones are organic chemical compound which are secreted by ants that trigger responses in peer ant members.

=> We can design an algorithm on this behaviour of ants.

=> For explaining the concept, I am using single ant colony and a single food source.

=> We will imagine the scenario using graphs.

=> Ants will be node.

=> food source will also be node or vertice.

=> The path serves as edges.

=> The amount of pheromone released will be weight of the edge.

=> Suppose graph $G_1 = (V, E)$ $V \rightarrow$ vertices and $E \rightarrow$ edges.

=> Here, $V_s \Rightarrow$ source vertice
$V_d \Rightarrow$ destination vertice

=> E_1 and E_2 are edges of lengths L_1 and L_2

=> Associated pheromone values $\Rightarrow R_1$ and R_2

=> Now, for each ant, starting probability of selection of path can be :-

$$P_i' = \frac{R_i}{R_1 + R_2} \quad \text{where, } i = 1, 2$$

=> If $R_1 > R_2$, probability of choosing E_1 is higher.

=> Now, if returning takes place from E_1, then pheromone value will be updated.

=> This updation will be made on 2 criterias :-
(1) evaporation rate
(2) Length of path

=> Update with consideration of evaporation rate :-

$$R_i \Leftarrow (1 - V) * R_i$$

=> V => interval that regulates pheromone evaporation.

⇒ For path length: →

$$R_i \leftarrow R_i + \frac{K}{L_i}$$

⇒ K → model parameter.

⇒ At each iteration, all ants are at V_s.

⇒ Then, ants will move from V_s to V_d.

⇒ At last, all ants will make return journey.

=> consider example of Traveling salesmen Problem.

=> To apply ACO to TSP, we consider the graph defined by associated set of cities within the set of vertices of graph.

=> In TSP, it is possible to move from any given city to any other city.

=> The pheromone values are modified at runtime and represent the cumulated experience of ant colony.

=> Each ant starts from randomly selected city.

=> It moves along the edges of graph.

=> Each ant keeps memory of the path, and all the steps it chooses that does not lead to path.

=> Ant has constructed the solution once it has visited all the vertices.

=> The pheromone on the edges is updated.

=> Each pheromone value is decreased by certain amount.

=) Each edge then receives an amount of extra pheromone on the quality of solutions to which it belongs.

=) This procedure is repeated until a termination criteria occurs.

Page: WALES
Date:

Ans- (4)(a) Representing following Sentences in first-order logic :->

(1) Some Students took English Subject

$\exists x \exists y : Student(x) \wedge English(y)$

=> Student(x): x is a student
=> English(y): y is basically course of English

(2) Every Student who takes English passes it.

$\forall x \forall y : (Student(x) \wedge English(y) \wedge Take(x,y)) \Rightarrow Pass(x,y)$

=> Student(x): x is a student
English(y): y is course in English
Take(x,y): x takes y (course)
Pass(x,y): x passes y (course)

(3) Every Person who buys policy is smart.

$\forall x : Person(x) \wedge (\exists y,z\ Policy(y) \wedge Buys(x,y,z)) \Rightarrow Smart(x)$

(4) No Person buys an expensive Policy :-

$\forall x,y,z : Person(x) \wedge Policy(y) \wedge Expensive(y) \Rightarrow \neg Buys(x,y,z)$

Ans- 4(b)

* Semantic Nets *

=> They are alternative of predicate logic for knowledge Representation

=> Here, we represent our knowledge in terms / in the form of graphical networks.

=> The network consists of nodes that represent objects and arcs that describes the relationship.

=> This Representation is mainly of two types :->

(1) IS-A ~~Red~~ Relation (Inheritance)

(2) Kind-of-relation

=> These are simple and easy to understand

=> They are natural representation of knowledge.

Ans-(5) Knowledge Representation is made up of 2 words:-
(1) Knowledge
(2) Representation

=> They are two distinct entities

=> Knowledge is a description of world.

=> It determines a system's competence by what it knows.

=> Representation is the way the knowledge is encoded.

=> It defines system's performance

=> Different types of knowledge have different kinds of Representation

=> Knowledge ~~Represent~~ types:-
(1) Tacit / implicit

(2) Explicit

* Implicit Knowledge:-

=> It exists in human being
=> It is embodied
=> Hard to steal / copy
=> Difficult to share
=> Drawn from experience

* Explicit Knowledge:-

=> exists outside human being
=> It is embedded
=> Can be articulated formally
=> Easy to steal/ copy
=> Can be shared

=> There are 2 different entities we are dealing with:

(1) Facts
(2) Representation

=> The entities are structured at 2 levels:
(1) Knowledge level
(2) Symbol level

=> Knowledge Representation System have following properties:-

(1) Representational adequacy
(2) Inferential adequacy
(3) Inferential Efficiency
(4) Acquisional Efficiency

Ans-

(6)

ANN	CNN
=> Stands for "Artificial Neural Network"	=> Stands for "Convolutional Neural Network"
=> ANN uses fully connected layers.	=> CNN uses Partially Connected Layers where size of connection depends on size of filter
=> ANN can be used only for small images	=> CNN can be used for any image
=> In ANN, number of parameters will be very high.	=> In CNN, number of parameters will be very less
=> ANN has lower Scalability	=> CNN has higher Scalability
=> ANN has high success Rate.	=> CNN has relatively more success rate than ANN
=> It has two basic layers:- Input and Output	=> It has input and output layer with convolution and Pooling layer
=> It has Low Computational Cost	=> It has high computational cost
=> It is not Translation Invariant	=> It is Translation Invariant

Ans -(3)

```
  LOGIC
+ LOGIC
-------
PRO LOG
```

=> As per the rule; max. sum = 9.

=> There should be one carry.

So; **P = 1**

=> Now, for O ; O + O = O ; This means, additions gives same output; so, only one case is possible; i.e. O (zero)

∴ **O = 0** (zero)

=>

```
  L O G I C
  L O G I C
-------------
+1 R O L O G
```

=> Here, I + L = R
G + G = L
C + C = G
I + I = O

=> As; O = O ; I + I = O
=> Now; I ≠ O.

∴ Carry should be there.

∴ I + I = O => **I = 5**

∴ 5 + 5 = 10
↳ carry.

(2) 19DCS048

Page: WALES
Date:

=>
```
  L O G 5 C
  L O G 5 C
+ 1 R O L O G
```

=> Here, C+C = G;

as I=5 so C+C = G ≤ 9.

=> Now, sum of 2 odd or 2 even number = even.

=> as L+L = R; and G+G = L;

let G = 2 or 4

=> as L+L = 1R so; G ≠ 2.

∴ G = 4

=> as G = 4; so C = 2 (only option)

=>
```
    1
  L O 4 5 2
  L O 4 5 2
+ 1 R O L O 4
```

=> As; I = 5; I+I = 10;

=> 1 will be carried to G+G = L

=> G+G+1 = L; L = 4+4+1

L = 9

=> L+L = R => 9+9 = 18

=> R ≤ 9; so; R == 8 as 1 will be carried forward

∴
$$\begin{array}{r} 90452 \\ +\ 90452 \\ \hline 180904 \end{array}$$

=> 90452 + 90452 = 180904

P = 1	L = 9	C = 2
A = 8	G = 4	
O = 0	T = 5	

19DCS048

WAVES

Page:
Date:

Ans -(8)

* Deep Neural Network *

=> A deep neural network is an artificial neural network with multiple layers between input and output layers.

=> In simple terms, a neural network with some level of complexity, usually atleast two layers, qualifies as a deep neural network

* Evolution of DNN *:-

AI → ML → ANN → DNN

=> Basically, DNN is a type of machine learning that mimics the way the brain learns.

=> The general idea of DNN is that it learns through repetitive actions from a collection of samples.

=> For example, a dog has a black nose and floppy ears.

=> DNN learns the same way as human brain does; i.e. through practice and making mistakes.

* Working of DNN *

=> A computer is given a piece of information like an image or sound.

=> Let's say it's given a trumpet sound.

=> Now, computer didn't know this.

=> So, computer passes the sound through its DNN.

=> Here, it will recognize what it can and by sorting elements of it for example high pitch or low pitch.

=> When it reaches the end, of the process, it decides if the sound is a trumpet or not.

=> Feedback is given on the answer, which computer uses to strengthen its decision making.

=> The process is repeated over and over with lot's of different trumpet sounds, until the computer can learn to recognize it instantly.

=> The above process is similar to brain.

Ans- (a) * Problem Reduction Using "AND-OR" graph.

=> * AND-OR * Graph

=> And-OR graph is useful for representing the solutions of problems that can be solved by decomposing them into a set of smaller problems.

=> In the set of smaller problems, all of the problems must be solved.

=> This decomposition or reduction generates arcs that are called AND arcs.

=> Here, one AND arc may point to any number of successor nodes.

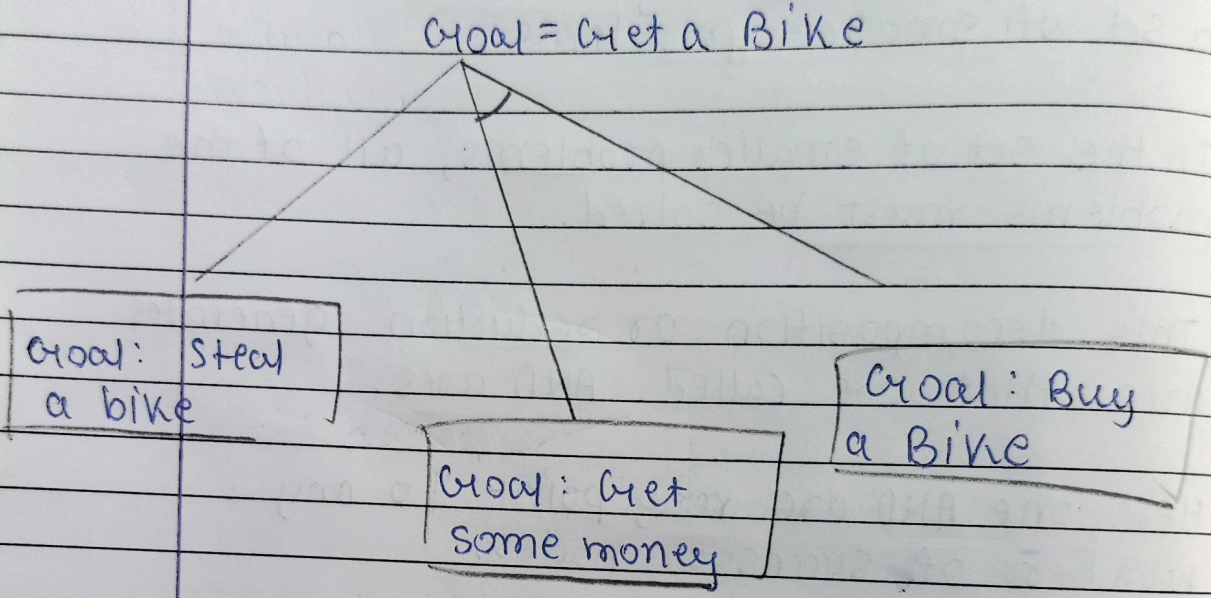
=> But all of the nodes must be solved in order for the arc to point solution.

=> In order to find solution in AND-OR graph we need an algorithm similar to best-first Search

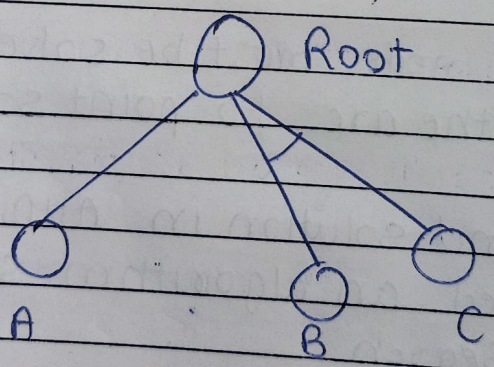
=> But that algorithm must have the ability to handle the AND arcs appropriately

=> We define Futility, if the estimated cost of solution becomes greater than value of futility, then we abandon the search.

* Example:-



=>



* Algorithm:-

(1) Initialize the graph to starting node

(2) Loop until the starting node is labelled SOLVED or cost goes above FUTILITY

(i) Traverse the graph, from initial node and following the current best path and record the set of nodes that are on that path and not been expanded.

(ii) Pick one of the unexpanded nodes and expands it.

=> If no successors, assign ~~futti~~ futility as value of this node.

=> else, add its successors to graph and compute f'(n)

=> if f'(n)==0; node is marked SOLVED

(iii) change the f'(n) estimate to reflect the new info by successors.

=> If in backward propagation, node contains successor arc whose descendents are all solved, then label the node SOLVED