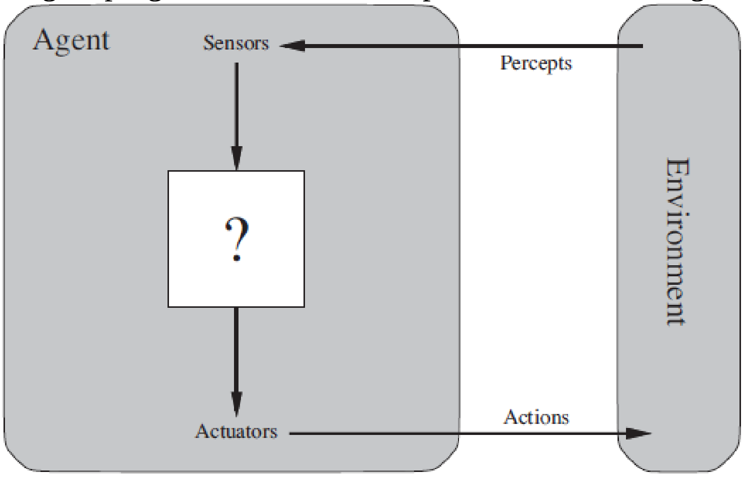
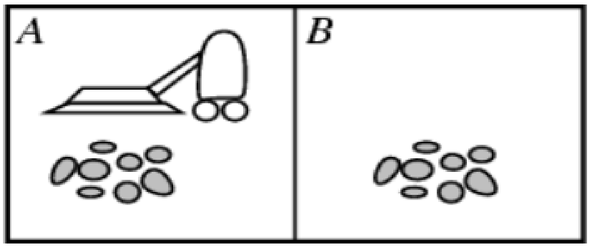
1.	**Attempt *any three* of the following:**	**15**
a.	Explain Artificial Intelligence with Turing Test approach.	
	Turing Test, proposed by Alan Turing (1950). -To provide a satisfactory operational definition of intelligence. A computer passes the test if a human interrogator, after posing some written questions, cannot tell whether the written responses come from a person or from a computer. Programming a computer to pass a rigorously applied test provides plenty to work on. The computer would need to possess the following capabilities: • **Natural Language Processing** to enable it to communicate successfully in English; • **Knowledge Representation** to store what it knows or hears; • **Automated Reasoning** to use the stored information to answer questions and to draw new conclusions; • **Machine Learning** to adapt to new circumstances and to detect and extrapolate patterns. Turing's test deliberately avoided direct physical interaction between the interrogator and the computer, because *physical* simulation of a person is unnecessary for intelligence. To pass the total Turing Test, the computer need • **Computer vision** to perceive objects, and • **Robotics** to manipulate objects and move about.	**2** **2** **1**
b.	Describe the contribution of Philosophy and Mathematics to Artificial Intelligence.	
	Philosophy: **-**Governing the rational part of mind • **Rationalism** –power of reasoning in understanding the world. • **Dualism** -There is a part of the human mind (or soul or spirit) that is outside of nature,exempt from physical laws. • **Materialism –**Holds the brain's operation according to the laws of physics *constitutes* the mind. The perception of available choices appears to the choosing entity. • **Induction -** The general rules are acquired by exposure to repeated associations between their elements. • **Logical Positivism -** This can be characterized by logical theories connected to observation sentences that correspond to sensory inputs; logical positivism combines rationalism and empiricism. • **confirmation theory - T**o analyze the acquisition of knowledge from experience. **Mathematics:** -Fundamental ideas of AI required a level of mathematical formalization in three fundamental areas: logic, computation,and probability. • A**lgorithm -** to determine the limits of what could be done with logic and computation. • **Incompleteness theorem –**shows that in any formal theory , there are true statements that are undecidable • **Computable -**This fundamental result can also be interpreted as showing that some functions on the integers cannot be represented by an algorithm that is, they cannot be computed. This motivated to characterize exactly which functions *are* capable of being computed. • **Tractability –** Problem is called intractable if the time required to solve instances of the problem grows exponentially with the size of the instances. • **Probability** - invaluable part of all the quantitative sciences, helping to deal with uncertain measurements and incomplete theories.	**2½** **2½**
c.	State the relationship between agents and environment.	
	Agent: An Agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.	**3**

Percept: We use the term percept to refer to the agent's perceptual inputs at any given instant. Percept Sequence: An agent's percept sequence is the complete history of everything the agent has ever perceived.

Agent function: Mathematically an agent's behavior is described by the agent function that maps any given percept sequence to an action. The agent function for an artificial agent will be implemented by an agent program. The agent function is an abstract mathematical description; The agent program is a concrete implementation, running on the agent architecture.



To illustrate these, we will use a example-the vacuum-cleaner world. This particular world has just two locations: squares A and B. The vacuum agent perceives which square it is in and whether there is dirt in the square. It can choose to move left, move right, suck up the dirt, or do nothing. One very simple agent function is the following: if the current square is dirty, then suck, otherwise move to the other square. **2**



Percept sequence	Action
[*A*, *Clean*]	*Right*
[*A*, *Dirty*]	*Suck*
[*B*, *Clean*]	*Left*
[*B*, *Dirty*]	*Suck*
[*A*, *Clean*], [*A*, *Clean*]	*Right*
[*A*, *Clean*], [*A*, *Dirty*]	*Suck*
⋮	⋮
[*A*, *Clean*], [*A*, *Clean*], [*A*, *Clean*]	*Right*
[*A*, *Clean*], [*A*, *Clean*], [*A*, *Dirty*]	*Suck*
⋮	⋮

d. What is PEAS description? Explain with two suitable examples.

PEAS (**P**erformance, **E**nvironment, **A**ctuators, **S**ensors) **1**

Agent Type	Performance Measure	Environment	Actuators	Sensors
Taxi driver	Safe, fast, legal, comfortable trip, maximize profits	Roads, other traffic, pedestrians, customers	Steering, accelerator, brake, signal, horn, display	Cameras, sonar, speedometer, GPS, odometer, accelerometer, engine sensors, keyboard
Medical diagnosis system	Healthy patient, reduced costs	Patient, hospital, staff	Display of questions, tests, diagnoses, treatments, referrals	Keyboard entry of symptoms, findings, patient's answers
Satellite image analysis system	Correct image categorization	Downlink from orbiting satellite	Display of scene categorization	Color pixel arrays
Part-picking robot	Percentage of parts in correct bins	Conveyor belt with parts; bins	Jointed arm and hand	Camera, joint angle sensors
Refinery controller	Purity, yield, safety	Refinery, operators	Valves, pumps, heaters, displays	Temperature, pressure, chemical sensors
Interactive English tutor	Student's score on test	Set of students, testing agency	Display of exercises, suggestions, corrections	Keyboard entry

Any 2

2X2=4

e. Explain following task environments:
i) Single Agent vs. Multiagent
ii) Episodic vs. Sequential

i) Single Agent vs. Multiagent

An agent solving a crossword puzzle by itself is clearly in a single-agent environment, whereas an agent playing chess is in a two agent environment.
chess is a **competitive** multiagent environment.
Taxi-driving environment avoiding collisions maximizes the performance measure of all agents, so it is a partially **cooperative** multiagent environment. **2½**

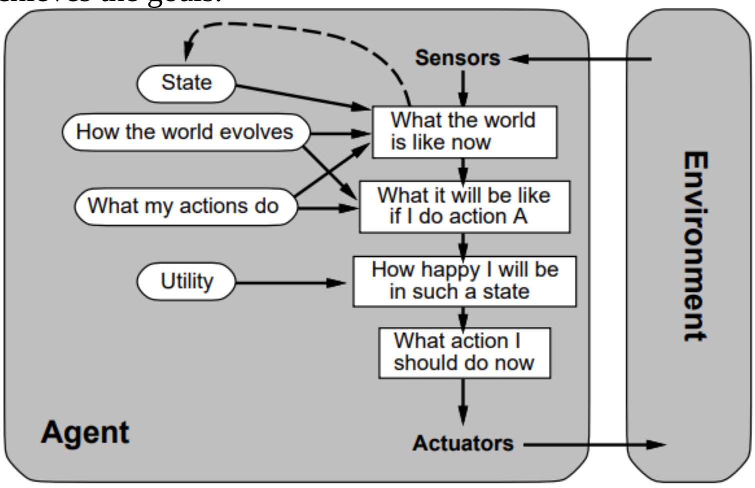
ii) Episodic vs. Sequential

In an episodic task environment, the agent's experience is divided into atomic episodes. The agent receives a percept and then performs a single action. For example, an agent that has to spot defective parts on an assembly line bases each decision on the current part. The current decision doesn't affect whether the next part is defective. In sequential environments, on the other hand, the current decision could affect all future decisions. Chess and taxi driving are sequential: in both cases, short-term actions can have long-term consequences. Episodic environments are much simpler than sequential environments because the agent does not need to think ahead. **2½**

f. Describe the structure of Utility based Agent.

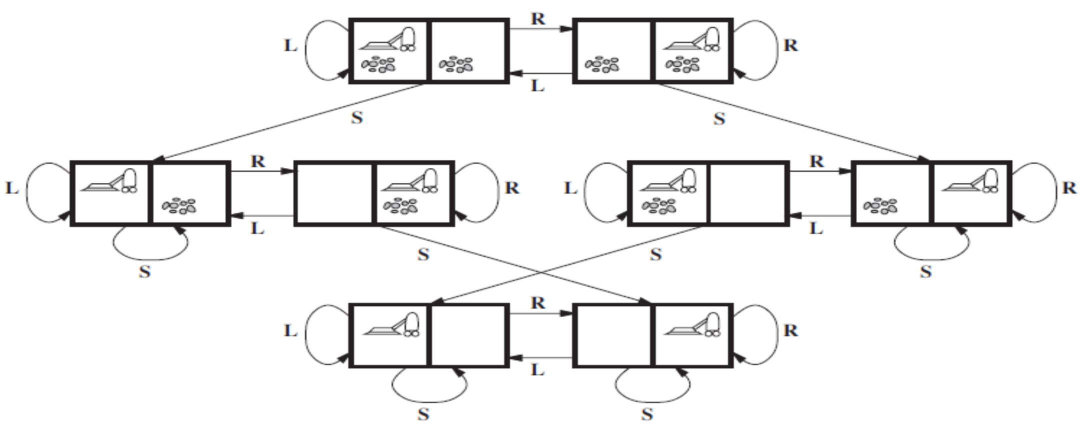
- These agents are similar to the goal-based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state. **3**
- Utility-based agent act based not only goals but also the best way to achieve the goal.
- The Utility-based agent is useful when there are multiple possible alternatives, and an agent has to choose in order to perform the best action.
- The utility function maps each state to a real number to check how efficiently each action achieves the goals.

2



2.	**Attempt *__any three__* of the following:**	**15**
a.	Describe the problem formulation of Vacuum World problem.	

2



3

States: The state is determined by both the agent location and the dirt locations. The agent is in one of two locations, each of which might or might not contain dirt. Thus,there are $2 \times 2^2 = 8$ possible world states. A larger environment with n locations has $n \times 2^n$ states.
• **Initial state**: Any state can be designated as the initial state.
• **Actions**: In this simple environment, each state has just three actions: *Left*, *Right*, and *Suck*. Larger environments might also include *Up* and *Down*.
• **Transition model**: The actions have their expected effects, except that moving *Left* in the leftmost square, moving *Right* in the rightmost square, and *Suck*ing in a clean square have no effect. The complete state space is shown in figure.
• **Goal test**: This checks whether all the squares are clean.
• **Path cost**: Each step costs 1, so the path cost is the number of steps in the path.

b. Define the following terms:

i) State Space of problem ii) Path in State Space

iii) Goal Test iv) Path Cost

iv) Optimal Solution to problem

i) State Space of problem : The set of all states reachable from the initial state by executing any sequence of actions.State is the representation of all possible outcomes. **1**

ii) Path in State Space:A sequence of states connected by a sequence of actions,in a givenstate space. **1**

iii) Goal Test: Test to deteermine whether the current state is the goal state or not.It can be carried out by comparing currnt state with the defined goal state. **1**

iv) Path Cost: The cost associated with each step to be taken to reach to reach to the goal state.Cost function chosen by the problem solving agent is used to find the cost. **1**

v) Optimal Solution to problem: The solution with least path cost among all solutions. **1**

c. Give the outline of Breadth First Search algorithm with respect to Artificial Intelligence.

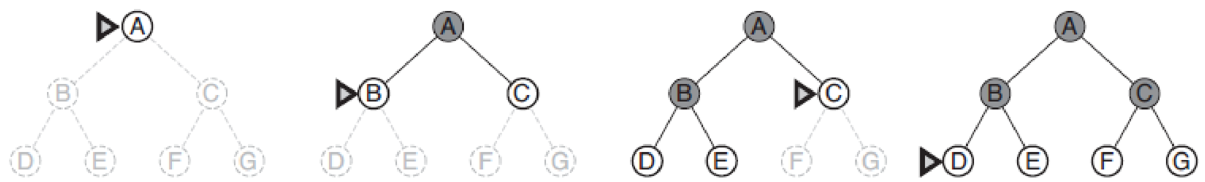
Breadth-First Search (BFS) **2**

- Proceeds level by level down the search tree
- Starting from the root node (initial state) explores all children of the root node, left to right
- If no solution is found, expands the first (leftmost) child of the root node, then expands the second node at depth 1 and so on …
- Process
  - **i)** Place the start node in the queue
  - **ii)** Examine the node at the front of the queue
    - a) If the queue is empty, stop
    - b) If the node is the goal, stop
- Otherwise, add the children of the node to the end of the queue

2

```
function BREADTH-FIRST-SEARCH(problem) returns a solution, or failure
  node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
  if problem.GOAL-TEST(node.STATE) then return SOLUTION(node)
  frontier ← a FIFO queue with node as the only element
  explored ← an empty set
  loop do
     if EMPTY?(frontier) then return failure
     node ← POP(frontier)  /* chooses the shallowest node in frontier */
     add node.STATE to explored
     for each action in problem.ACTIONS(node.STATE) do
        child ← CHILD-NODE(problem, node, action)
        if child.STATE is not in explored or frontier then
           if problem.GOAL-TEST(child.STATE) then return SOLUTION(child)
           frontier ← INSERT(child, frontier)
```

Example (Find path from A to D) **1**

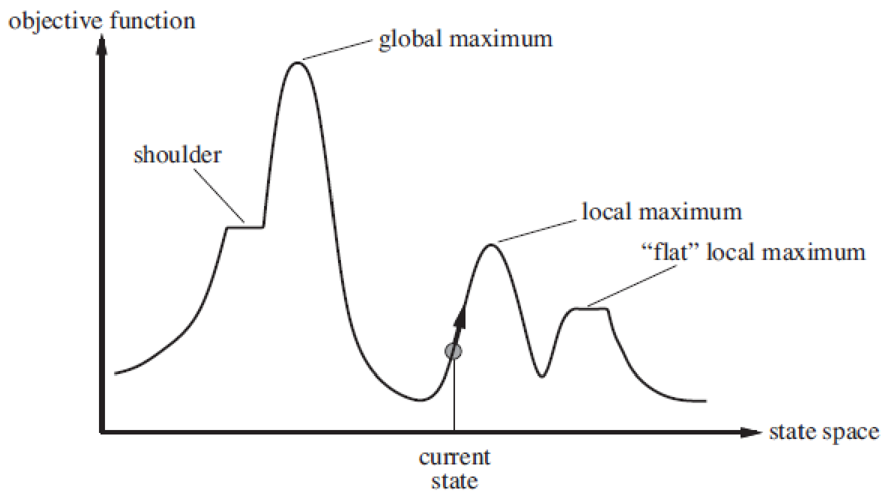
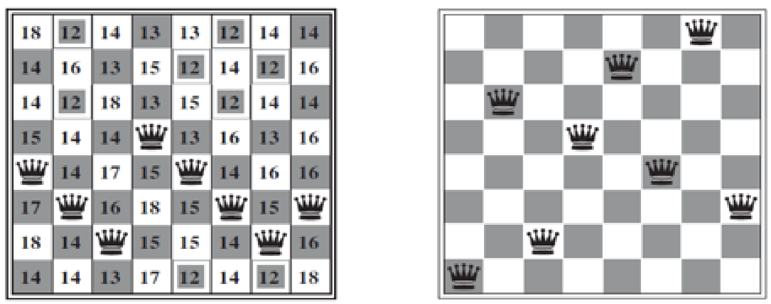


d. With the Local Search algorithm explain the following concepts:

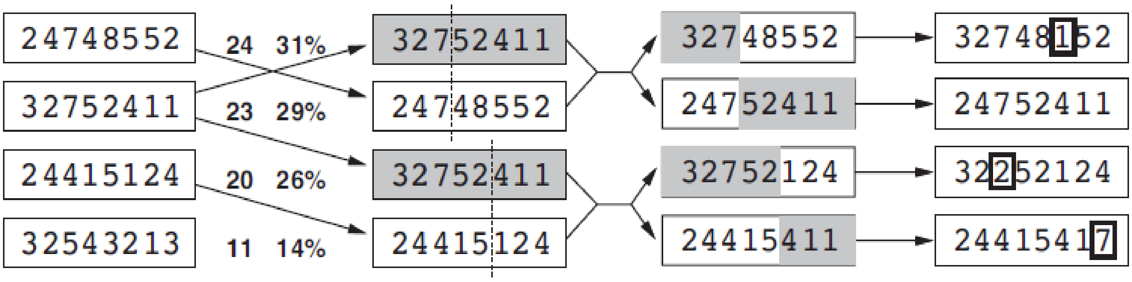
i) Shoulder ii) Global Maximum iii) Local Maximum

i) Shoulder: A plateau is a flat area of the state-space landscape. It can be a flat local maximum, from which no uphill exit exists, or a shoulder, from which progress is possible. **1**

1

	ii) Global Maximum: To understand local search, we find it useful to consider the state-space landscape. A landscape has both "location" (defined by the state) and "elevation" (defined by the value of the heuristic cost function or objective function). If elevation corresponds to an objective function, then the aim is to find the highest peak—a global maximum. **Local Maximum**: A local maximum is a peak that is higher than each of its neighboring states but lower than the global maximum. Hill-climbing algorithms that reach the vicinity of a local maximum will be drawn upward toward the peak but will then be stuck with nowhere else to go. 	1 2
e.	Illustrate Hill Climbing algorithm using 8 queen problem.	
	• Local search algorithms typically use a **complete-state formulation**, where each state has 8 queens on the board, one per column. • The successors of a state are all possible states generated by moving a single queen to another square in the same column (so each state has 8×7=56 successors). • The heuristic cost function h is the number of pairs of queens that are attacking each other, either directly or indirectly. The global minimum of this function is zero, which occurs only at perfect solutions. Figure shows a state with h=17. The figure also shows the values of all its successors, with the best successors having h=12. • Hill-climbing algorithms typically choose randomly among the set of best successors if there is more than one. • Hill climbing is sometimes called **greedy local search** because it grabs a good neighbor state without thinking. Hill climbing often makes rapid progress toward a solution because it is usually quite easy to improve a bad state. For example, from the state in figure(a), it takes just five steps to reach the state in figure (b), which has h=1 and is very nearly a solution.  (a) (b) (a) An 8-queens state with heuristic cost estimate $h = 17$, showing the value of h for each possible successor obtained by moving a queen within its column. The best moves are marked. (b) A local minimum in the 8-queens state space; the state has $h = 1$ but every successor has a higher cost.	3 2
f.	Explain the mechanism of Genetic Algorithm.	

- A **genetic algorithm** (or **GA**) is a variant of stochastic beam search in which successor states generated by combining *two* parent states rather than by modifying a single state. **2**
- GAs begin with a set of k randomly generated states, called the **population**. Each state, or **individual**, is represented as a string over a finite alphabet most commonly, a string of 0s and 1s.
- Each state is rated by the objective function, or (in GA terminology) the **fitness function**.
- Two pairs are selected at random for reproduction, in accordance with the probabilities.
- One individual is selected twice and one not at all. For each pair to be mated, a **crossover** point is chosen randomly from the positions in the string.
- The offspring themselves are created by crossing over the parent strings at the crossover point.
- Each location is subject to random **mutation** with a small independent probability. One digit was mutated in the first, third, and fourth offspring.



1

(a) Initial Population | (b) Fitness Function | (c) Selection | (d) Crossover | (e) Mutation

2

```
function GENETIC-ALGORITHM(population, FITNESS-FN) returns an individual
  inputs: population, a set of individuals
          FITNESS-FN, a function that measures the fitness of an individual

  repeat
     new_population ← empty set
     for i = 1 to SIZE(population) do
        x ← RANDOM-SELECTION(population, FITNESS-FN)
        y ← RANDOM-SELECTION(population, FITNESS-FN)
        child ← REPRODUCE(x, y)
        if (small random probability) then child ← MUTATE(child)
        add child to new_population
     population ← new_population
  until some individual is fit enough, or enough time has elapsed
  return the best individual in population, according to FITNESS-FN
```

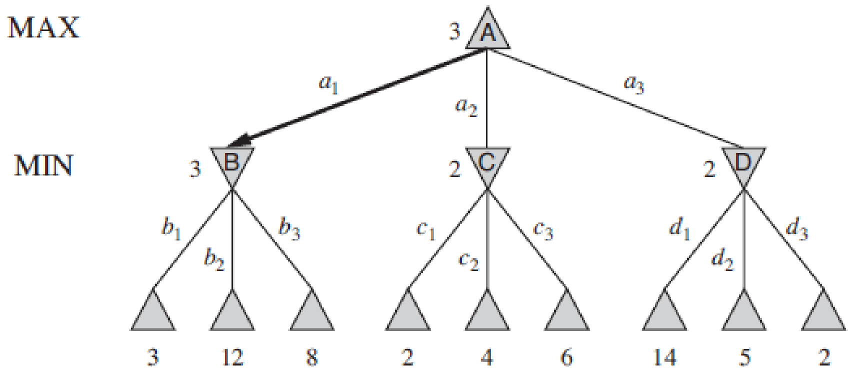
```
function REPRODUCE(x, y) returns an individual
  inputs: x, y, parent individuals

  n ← LENGTH(x); c ← random number from 1 to n
  return APPEND(SUBSTRING(x, 1, c), SUBSTRING(y, c + 1, n))
```

3.	**Attempt *any three* of the following:**	**15**
a.	Explain Minimax algorithm in detail.	

The **minimax algorithm** computes the minimax decision from the current state. It uses a simple recursive computation of the minimax values of each successor state, directly implementing the defining equations. The recursion proceeds all the way down to the leavesof the tree, and then the minimax values are **backed up** through the tree as the recursionunwinds. **2**

2



$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

For example, the algorithm first recurses down to the three bottomleft nodes and uses the UTILITY function on them to discover that their values are 3, 12, and 8, respectively. Then it takes the minimum of these values, 3, and returns it as the backedup value of node B. A similar process gives the backed-up values of 2 for C and 2 for D.

Finally, we take the maximum of 3, 2, and 2 to get the backed-up value of 3 for the root node.

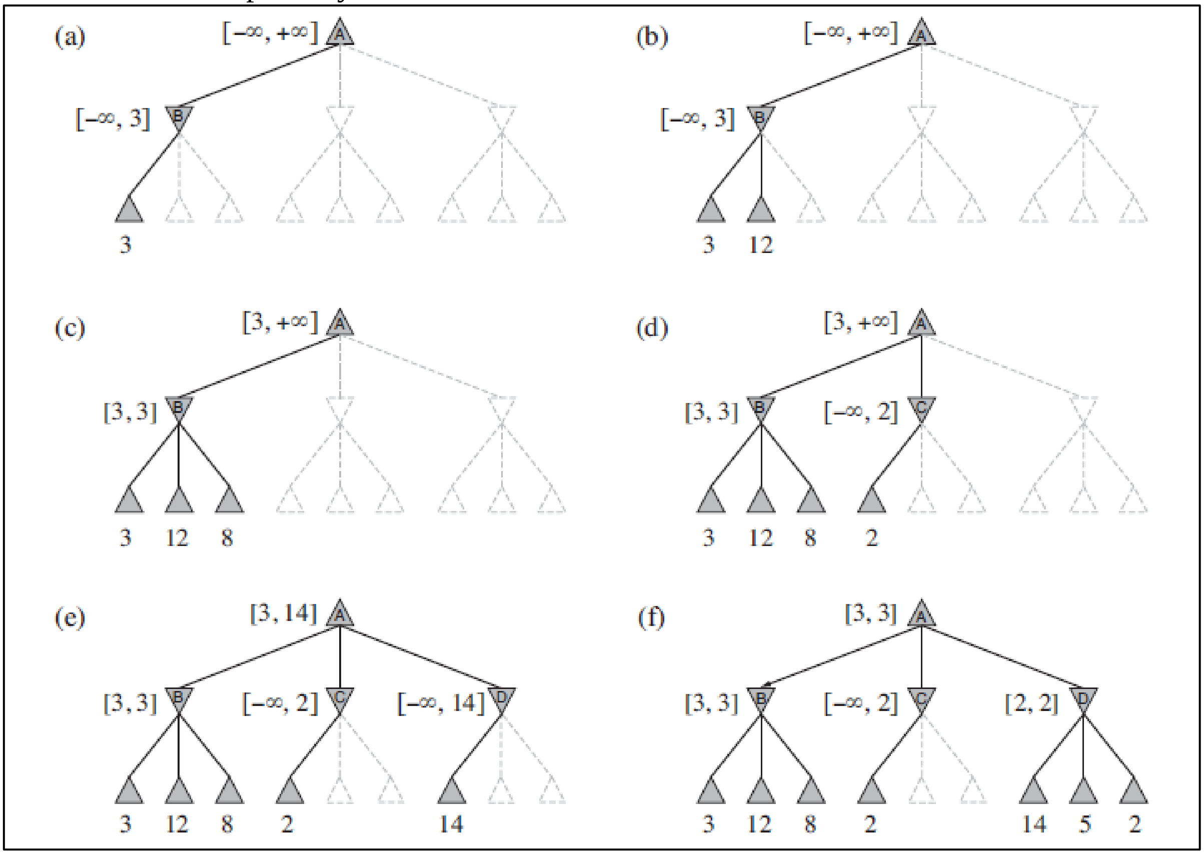
If the maximum depth of the tree is m and there are b legal moves at each point, then the time complexity of the minimax algorithm is $O(b^m)$. The space complexity is O(bm) for an algorithm that generates all actions at once, or O(m) for an algorithm that generates actions one at a time.

1

b. Describe the technique of Alpha-Beta Pruning. **1**

- The problem with minimax search is that the number of game states it has to examine is exponential in the depth of the tree. When alpha beta pruning applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

2



MINIMAX(root) = max(min(3, 12, 8), min(2, x, y), min(14, 5, 2)) **2**

= max(3, min(2, x, y), 2)

= max(3, z, 2) where z = min(2, x, y) ≤ 2

= 3

Alpha–beta pruning can be applied to trees of any depth, and it is often possible to prune entire subtrees rather than just leaves.

If Player has a better choice **m** either at the parent node of **n** or at any choice point further up, then **n** *will never be reached in actual play*. So once we have found out enough about **n** (by examining some of its descendants) to reach this conclusion, we can prune it.

α = the value of the best (i.e., highest-value) choice at any choice point along the path for MAX.

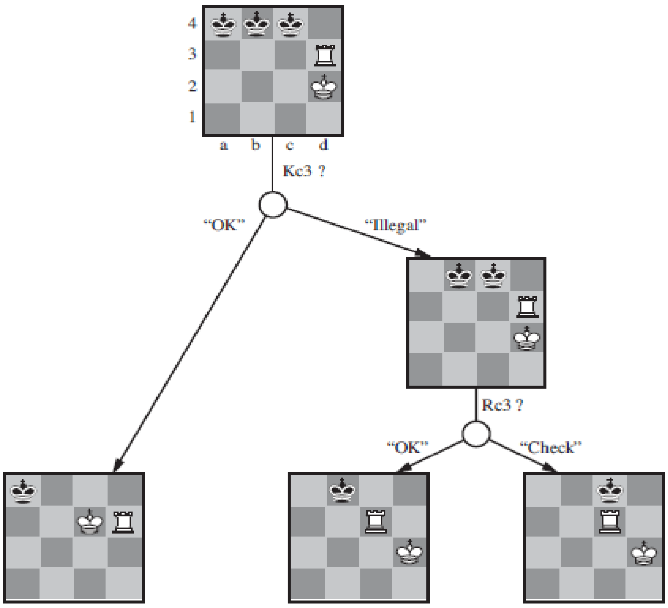
β = the value of the best (i.e., lowest-value) choice at any choice point along the path for MIN.

c. Write a short note on Kriegspiel's Partially observable chess.

- **Kriegspiel**, a partially observable variant of chess in which pieces can move but are completely invisible to the opponent. **1**
- The rules of Kriegspiel are as follows:
  White and Black each see a board containing only their own pieces. A referee, who can see all the pieces, adjudicates the game and periodically makes announcements that are heard by both players. On his turn, White proposes to the referee any move that would be legal if there were no black pieces. If the move is in fact not legal (because of the black pieces), the referee announces "illegal." Whitemay keep proposing moves until a legal one is found and learns more about the location of Black's pieces in the process. Once a legal move is proposed, the referee announces one or more of the following: "Capture on square *X*" if there is a capture, and "Check by *D*" if the black king is in check, where *D* is the direction of the check, and can be one of "Knight","Rank," "File," "Long

diagonal," or "Short diagonal". If Black is checkmated or stalemated, the referee says so; otherwise, it is Black's turn to move.

- **Belief State is** the set of all *logically possible* board states given percepts. Initially, White's belief state is a singleton because Black's pieces haven't moved yet. After White makes a move and Black responds,White's belief state contains 20 positions because Black has 20 replies to any White move. Keeping track of the belief state as the game progresses is exactly the problem of **state estimation**. Kriegspiel state estimation mapped onto the partially observable, nondeterministic .If we consider the opponent as the source of nondeterminism; that is, the RESULT of White's move are composed from the (predictable) outcome of White's own move and the unpredictable outcome given by Black's reply. **2**
- Given a current belief state, White may ask, "Can I win the game?" For a partially observable game, the notion of a **strategy** is altered; instead of specifying a move to make for each possible *move* the opponent might make, we need a move for every possible *percept sequence* that might be received. For Kriegspiel, a winning strategy, or **guaranteed checkmate**,is one that, for each possible percept sequence, leads to an actual checkmate for every possible board state in the current belief state, regardless of how the opponent moves. Figure shows part of a guaranteed checkmate for the KRK (king and rook against king) endgame. In this case, Black has just one piece (the king), so a belief state for White can be shown in a single board by marking each possible position of the Black king.



2

- The general AND-OR search algorithm can be applied to the belief-state space to find guaranteed checkmates. Kriegspiel admits an entirely new concept that makes no sense in fully observable games: **probabilistic checkmate**. Such checkmates are still required to work in every board state in the belief state; they are probabilistic with respect to randomization of the winning player's moves. To get the basic idea, consider the problem of finding a lone black king using just the white king. Simply by moving randomly, the white king will *eventually* bump into the black king even if the latter tries to avoid this fate,since Black cannot keep guessing the right evasive moves indefinitely. In the terminology of probability theory, detection occurs *with probability* 1.

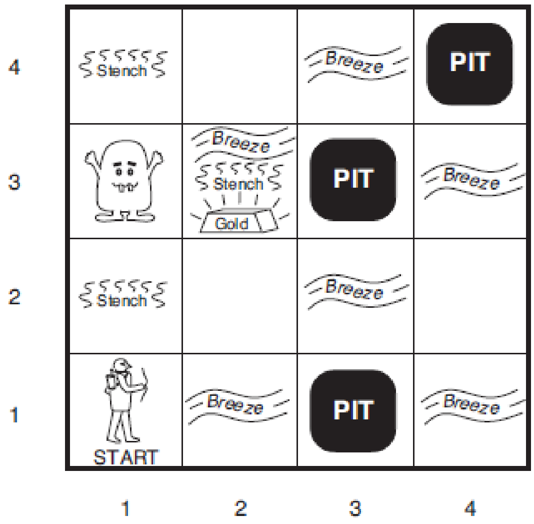
d. What is knowledge based agent? Explain its importance in problem solving techniques.

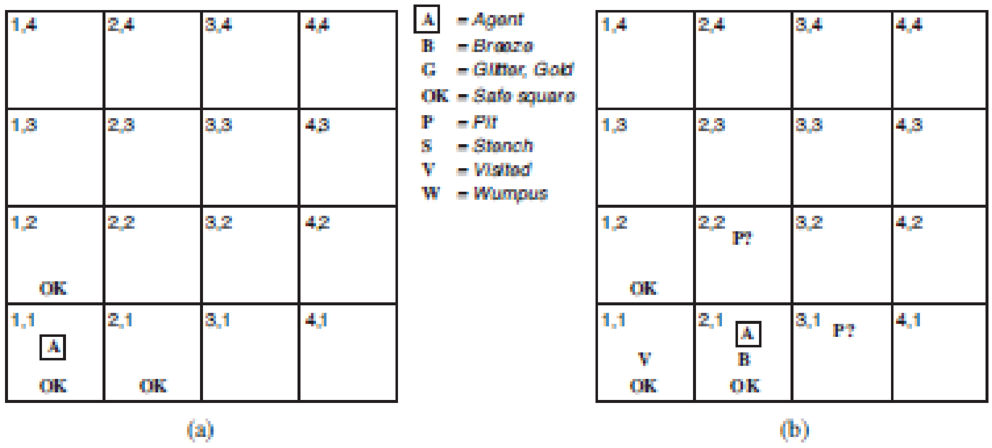
2

- The central component of a knowledge-based agent is its **knowledge base**, or KB.
- A knowledge base is a set of **sentences**.
- Each sentence is expressed in a language called a **knowledge representation language** and represents some assertion about the world.
- A sentence dignified with the name **axiom**, when the sentence is taken as given without being derived from other sentences.
- To add new sentences to the knowledge base , query operations are TELL and ASK used. Both operations may involve **inference**—that is, deriving new sentences from old.

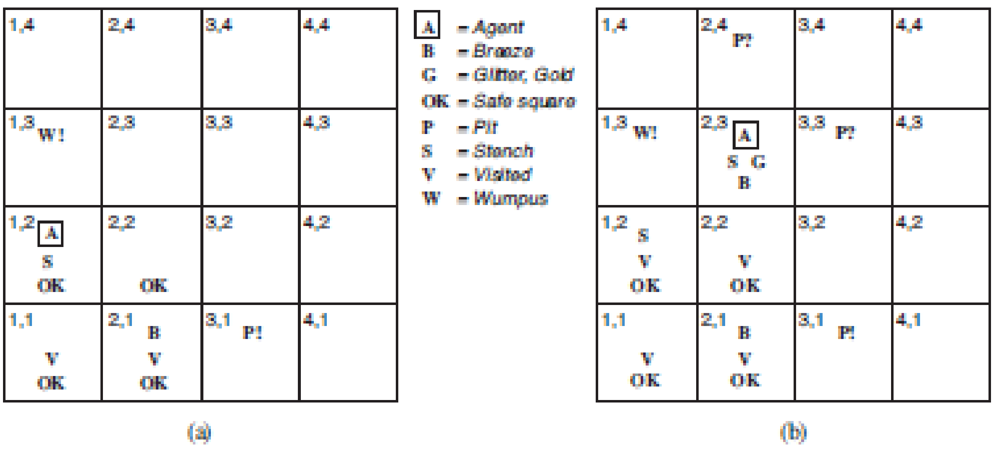
	• Like all our agents,it takes a percept as input and returns an action. The agent maintains a knowledge base, KB, which may initially contain some **background knowledge**.
	• Each time the agent program is called, it does three things.
	    • First, it TELLS the knowledge base what it perceives.
	    • Second, it ASKS the knowledge base what action it should perform.
	    • Third, the agent program TELLS the knowledge base which action was chosen, and the agent executes the action.
	
	MAKE-PERCEPT-SENTENCE constructs a sentence asserting that the agent perceived the given percept at the given time.
	MAKE-ACTION-QUERY constructs a sentence that asks what action should be done at the current time.
	MAKE-ACTION-SENTENCE constructs a sentence asserting that the chosen action was executed.
	The details of the inference mechanisms are hidden inside TELL and ASK.
	
	• **Knowledge Level-**describes agent by saying what it knows
	• Implementation level - knows that that will achieve its goal
	There are mainly two approaches to build a knowledge-based agent:
	
	    • **Declarative approach:** Knowledge-based agent initialized with an empty knowledge base and telling the agent all the sentences with which we want to start with. This approach is called Declarative approach.
	    • **Procedural approach:** Directly encoding desired behavior as a program code.
		1
		
		2

e.	Write a short note on Wumpus world problem.	
	Wumpus eats anyone that enters its room • Wumpus can be shot by an agent, but agent has one arrow • Pits trap the agent (but not the wumpus) • Agent's goal is to pick up the gold • Performance measure: – +1000 for picking up gold, -1000 for death (meeting a live wumpus or falling into a pit) -1 for each action taken, -10 for using arrow • Environment: – 4x4 grid of rooms – Agent starts in (1,1) and faces right – Geography determined at the start: • Gold and wumpus locations chosen randomly • Each square other than start can be a pit with probability 0.2 • Actuators: – Movement: • Agent can move forward • Turn 90 degrees left or right – Grab: • pick up an object in same square – Shoot: fire arrow in straight line in the direction agent is facing • Sensors: – Returns a 5-tuple of five symbols eg. [stench, breeze, glitter, bump, scream] – In squares adjacent to the wumpus, agent perceives a stench – In squares adjacent to a pit, agent perceives a breeze – In squares containing gold, agent perceives a glitter – When agent walks into a wall, it perceives a bump – When wumpus is killed, it emits a woeful scream that is perceived anywhere • Initial knowledge base contains: – Agent knows it is in [1,1] – Agent knows it is a safe square	**3** **2**





The first step taken by the agent in the wumpus world. (a) The initial situation, after percept [*None*, *None*, *None*, *None*, *None*]. (b) After one move, with percept [*None*, *Breeze*, *None*, *None*, *None*].



Two later stages in the progress of the agent. (a) After the third move, with percept [*Stench*, *None*, *None*, *None*, *None*]. (b) After the fifth move, with percept [*Stench*, *Breeze*, *Glitter*, *None*, *None*].

v

f. Explain Forward-Chaining algorithm for Propositional definite Clauses.

3

```
function PL-FC-ENTAILS?(KB, q) returns true or false
  inputs: KB, the knowledge base, a set of propositional definite clauses
          q, the query, a proposition symbol
  count ← a table, where count[c] is the number of symbols in c's premise
  inferred ← a table, where inferred[s] is initially false for all symbols
  agenda ← a queue of symbols, initially symbols known to be true in KB

  while agenda is not empty do
     p ← POP(agenda)
     if p = q then return true
     if inferred[p] = false then
        inferred[p] ← true
        for each clause c in KB where p is in c.PREMISE do
           decrement count[c]
           if count[c] = 0 then add c.CONCLUSION to agenda
  return false
```

The forward-chaining algorithm for propositional logic. The *agenda* keeps track of symbols known to be true but not yet "processed." The *count* table keeps track of how many premises of each implication are as yet unknown. Whenever a new symbol p from the agenda is processed, the count is reduced by one for each implication in whose premise p appears (easily identified in constant time with appropriate indexing.) If a count reaches zero, all the premises of the implication are known, so its conclusion can be added to the agenda. Finally, we need to keep track of which symbols have been processed; a symbol that is already in the set of inferred symbols need not be added to the agenda again. This avoids redundant work and prevents loops caused by implications such as $P \Rightarrow Q$ and $Q \Rightarrow P$.

2

$P \Rightarrow Q$

$L \wedge M \Rightarrow P$

$B \wedge L \Rightarrow M$

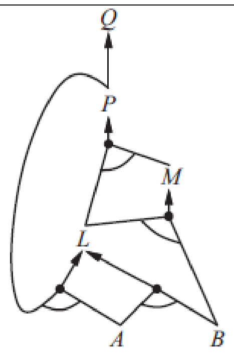
$A \wedge P \Rightarrow L$

$A \wedge B \Rightarrow L$

A

B

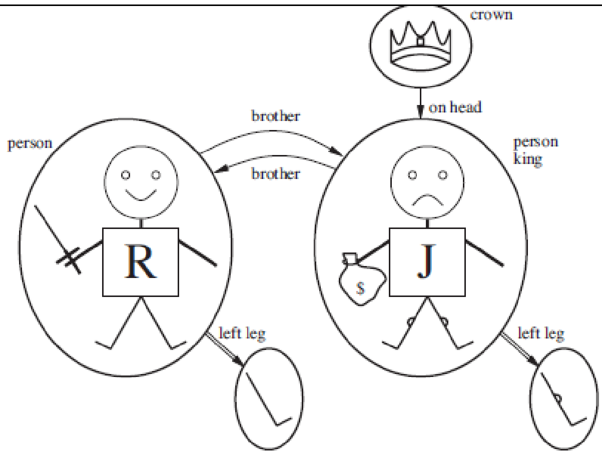
(a)



(b)

4.	**Attempt *any three* of the following:**	**15**
a.	What is meant by First Order Logic? Explain syntax and semantics of First Order Logic.	
	First-Order Logic is more expressive to represent a good deal of our commonsense knowledge. A **term** is a logical expression that refers to an object. First Order Logic symbol can be a constant term,a variable term or a function. Constant Term:Fixed value which belongs to the domain. Variable Term:Term which can be assigned values in the domain. Function: t1,t2 … are the terms then f(t1,t2…)is also a term.	**1**

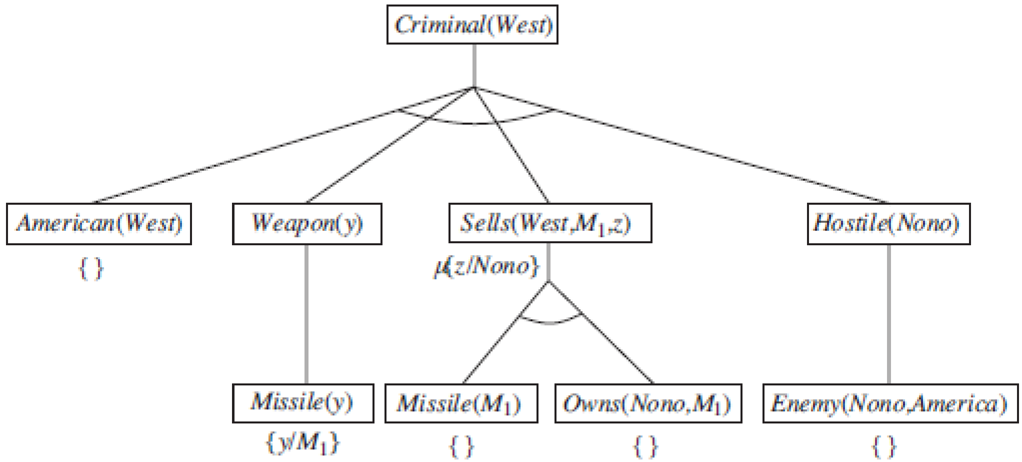
	$$\begin{aligned} Sentence &\rightarrow AtomicSentence \mid ComplexSentence \\ AtomicSentence &\rightarrow Predicate \mid Predicate(Term, \ldots) \mid Term = Term \\ ComplexSentence &\rightarrow (\ Sentence\ ) \mid [\ Sentence\ ] \\ &\mid \neg\ Sentence \\ &\mid Sentence \land Sentence \\ &\mid Sentence \lor Sentence \\ &\mid Sentence \Rightarrow Sentence \\ &\mid Sentence \Leftrightarrow Sentence \\ &\mid Quantifier\ Variable, \ldots\ Sentence \\ \\ Term &\rightarrow Function(Term, \ldots) \\ &\mid Constant \\ &\mid Variable \\ \\ Quantifier &\rightarrow \forall \mid \exists \\ Constant &\rightarrow A \mid X_1 \mid John \mid \cdots \\ Variable &\rightarrow a \mid x \mid s \mid \cdots \\ Predicate &\rightarrow True \mid False \mid After \mid Loves \mid Raining \mid \cdots \\ Function &\rightarrow Mother \mid LeftLeg \mid \cdots \\ \text{OPERATOR PRECEDENCE} &: \neg, =, \land, \lor, \Rightarrow, \Leftrightarrow \end{aligned}$$	2
	• An **atomic sentence** (or **atom**) is formed from a predicate symbol optionally followed by a ATOM parenthesized list of terms, such as Brother (Richard , John). Atomic sentences can have complex terms as arguments. Thus, Married(Father (Richard),Mother (John)) An atomic sentence is **true** in a given model if the relation referred to by the predicate symbol holds among the objects referred to by the arguments. • **Complex sentences** **logical connectives** to construct more complex sentences, with the same syntax and semantics as in propositional calculus. Here are four sentences that are true in the model ¬Brother (LeftLeg(Richard), John) Brother (Richard , John) ∧ Brother (John,Richard) King(Richard) ∨ King(John) ¬King(Richard) ⇒ King(John)	2

		
b	Give a short note on Universal and Existential quantifier with suitable example.	
	A quantifier is a language element which generates quantification, and quantification specifies the quantity of specimen in the universe of discourse. These are the symbols that permit to determine or identify the range and scope of the variable in the logical expression. There are two types of quantifier: Universal Quantifier, (for all, everyone, everything) Existential quantifier, (for some, at least one).	**1**
	Universal Quantifier: Universal quantifier is a symbol of logical representation, which specifies that the statement within its range is true for everything or every instance of a particular thing. The Universal quantifier is represented by a symbol ∀, which resembles an inverted A. "All kings are persons," is written as $\forall$ x King(x) $\Rightarrow$ Person(x) "For all x, if x is a king, then x is a person."	**2**
	Existential Quantifier: Existential quantifiers are the type of quantifiers, which express that the statement within its scope is true for at least one instance of something. It is denoted by the logical operator ∃, which resembles as inverted E. When it is used with a predicate variable then it is called as an existential quantifier. If x is a variable, then existential quantifier will be ∃x or ∃(x). And it will be read as There exists a 'x.' For some 'x.' For at least one 'x.' King John has a crown on his head, we write $\exists$ x Crown(x) $\wedge$ OnHead(x, John) . ∃x is pronounced "There exists an x such that . . ." or "For some x . . .". The sentence ∃x P says that P is true for at least one object x.	**2**
c.	Explain the steps of Knowledge Engineering projects in First Order Logic.	
	1. Identify the task: The knowledge engineer must delineate the range of questions that the knowledge base will support and the kinds of facts that will be available for each specific problem instance. **2. Assemble the relevant knowledge:** The knowledge engineer might already be an expert in the domain, or might need to work with real experts to extract . **3. Decide on a vocabulary of predicates, functions, and constants**: That is, translate the important domain-level concepts into logic-level names. This involves many questions of knowledge-engineering style. **4. Encode general knowledge about the domain**. The knowledge engineer writes down the axioms for all the vocabulary terms. **5. Encode a description of the specific problem instance**. It will involve writing simple atomic sentences about instances of concepts that are already part of the ontology.	**5**

	6. Pose queries to the inference procedure and get answers. The inference procedure operate on the axioms and problem-specific facts to derive the facts we are interested in knowing. **7. Debug the knowledge base**. The answers will be correct for the knowledge base as written, assuming that the inference procedure is sound, but they will not be the ones that the user is expecting.	
d	Write a short note on Unification Process.	
	Lifted inference rules require finding substitutions that make different logical expressions look identical. This process is called **unification** and is a key component of all first-order inference algorithms. The UNIFY algorithm takes two sentences and returns a **unifier** for them if one exists: UNIFY(p, q)=θ where SUBST(θ, p)= SUBST(θ, q) . by finding all sentences in the knowledge base that unify with Knows(John, x). Here are the results of unification with four different sentences that might be in the knowledge base: UNIFY(Knows(John, x), Knows(John, Jane)) = {x/Jane} UNIFY(Knows(John, x), Knows(y, Bill)) = {x/Bill, y/John} UNIFY(Knows(John, x), Knows(y,Mother (y))) = {y/John, x/Mother (John)} UNIFY(Knows(John, x), Knows(x, Elizabeth)) = fail . The last unification fails because x cannot take on the values John and Elizabeth at the same time. Knows(x, Elizabeth) means "Everyone knows Elizabeth," This infers that John knows Elizabeth. The problem arises only because the two sentences happen to use the same variable name, x. The problem can be avoided by **standardizing apart** one of the two sentences being unified, which means renaming its variables to avoid name clashes. For example, we can rename x in Knows(x, Elizabeth) to x_{17} (a new variable name) without changing its meaning. Now the unification will work: UNIFY(Knows(John, x), Knows(x_{17}, Elizabeth)) = {x/Elizabeth, x_{17}/John}	**1** **2** **2**
e.	Explain Datalog used in first order definite clause.	
	Datalog is a language that is restricted to first-order definite clauses with no function symbols. A Datalog database is a collection of definite clauses where I Terms are just constants and variables, there are no function symbols with arity > 0. I Every variable that occurs in the head must occur in the body. Consider the following problem: The law says that it is a crime for an American to sell weapons to hostile nations. The country Nono, an enemy of America, has some missiles, and all of its missiles were sold to it by ColonelWest, who is American. We will prove that West is a criminal. First, we will represent these facts as first-order definite clauses. American(x) ∧Weapon(y) ∧ Sells(x, y, z) ∧ Hostile(z) ⇒ Criminal (x) . "Nono . . . has some missiles." The sentence ∃ x Owns(Nono, x)∧Missile(x) is transformed into two definite clauses by Existential Instantiation, introducing a new constant M1: Owns(Nono,M1) Missile(M1)	**2** **1** **2**

	"All of its missiles were sold to it by Colonel West": Missile(x) ∧ Owns(Nono, x) ⇒ Sells(West, x, Nono) We will also need to know that missiles are weapons: Missile(x) ⇒ Weapon(x) and we must know that an enemy of America counts as "hostile": Enemy(x,America) ⇒ Hostile(x) . "West, who is American . . .": American(West) "The country Nono, an enemy of America . . .": Enemy(Nono,America) . (9.10) This knowledge base contains no function symbols and is therefore an instance of the class of **Datalog** knowledge bases.	
f.	Describe Backward-Chaining algorithm for First Order definite Clauses.	
	FOL-BC-ASK(KB,goal) is true if the knowledge base contains a clause of the form lhs ⇒ goal, where lhs (left-hand side) is a list of conjuncts. An atomic fact like American(West) is considered as a clause whose lhs is the empty list. For example, the query Person(x) could be proved with the substitution {x/John} as well as with {x/Richard }. FOL-BC-ASK as a **generator**—a function that returns multiple times, each time giving one possible result.Backward chaining is a kind of AND/OR search—the OR part because the goal query can be proved by any rule in the knowledge base, and the AND part because all the conjuncts in the lhs of a clause must be proved. FOL-BC-OR works by fetching all clauses that might unify with the goal, standardizing the variables in the clause to be brand-new variables, and then, if the rhs of the clause does indeed unify with the goal, proving every conjunct in the lhs, using FOL-BC-AND. Backward chaining, as we have written it, is a depth-first search algorithm.	2

	function FOL-BC-ASK(*KB*, *query*) **returns** a generator of substitutions
	  return FOL-BC-OR(*KB*, *query*, { })
	
	generator FOL-BC-OR(*KB*, *goal*, θ) **yields** a substitution
	  for each rule (*lhs* ⇒ *rhs*) in FETCH-RULES-FOR-GOAL(*KB*, *goal*) **do**
	    (*lhs*, *rhs*) ← STANDARDIZE-VARIABLES((*lhs*, *rhs*))
	    for each θ' **in** FOL-BC-AND(*KB*, *lhs*, UNIFY(*rhs*, *goal*, θ)) **do**
	      yield θ'
	
	generator FOL-BC-AND(*KB*, *goals*, θ) **yields** a substitution
	  if θ = *failure* **then return**
	  else if LENGTH(*goals*) = 0 **then yield** θ
	  else do
	    first,*rest* ← FIRST(*goals*), REST(*goals*)
	    for each θ' **in** FOL-BC-OR(*KB*, SUBST(θ, *first*), θ) **do**
	      for each θ'' **in** FOL-BC-AND(*KB*, *rest*, θ') **do**
	        yield θ''
		2
		
		1



5.	**Attempt *<u>any three</u>* of the following:**	**15**
a.	Explain Planning Domain Definition Language description for an Air Cargo planning problem.	

$Init(At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK)$
$\wedge\ Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$
$\wedge\ Airport(JFK) \wedge Airport(SFO))$
$Goal(At(C_1, JFK) \wedge At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg At(c, a) \wedge In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c, a) \wedge \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg At(p, from) \wedge At(p, to))$

2

It shows an air cargo transport problem involving loading and unloading cargo and flying it from place to place. The problem can be defined with three actions: Load , Unload,and Fly. The actions affect two predicates: In(c, p) means that cargo c is inside plane p, and At(x, a) means that object x (either plane or cargo) is at airport a. **2**

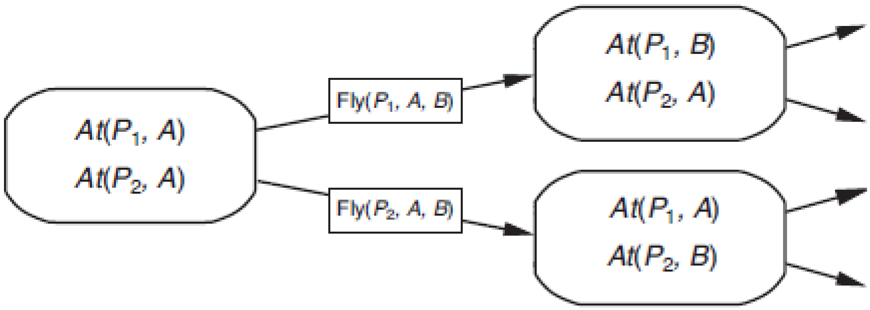
When a plane flies from one airport to another, all the cargo inside the plane goes with it. Basic PDDL does not have a universal quantifier, so we need a different solution. The approach we use is to say that a piece of cargo ceases to be At anywhere when it is In a plane; the cargo only becomes At the new airport when it is unloaded. So At really means "available for use at a given location."

The following plan is a solution to the problem: **1**

[Load (C1, P1, SFO), Fly(P1, SFO, JFK),Unload(C1, P1, JFK),
Load (C2, P2, JFK), Fly(P2, JFK, SFO),Unload(C2, P2, SFO)] .

b.	Describe Forward (Progression) State-Space Search algorithm with an example.	

2



	(a) Forward (progression) search through the space of states, starting in the initial state and using the problem's actions to search forward for a member of the set of goal states. Planning problems often have large state spaces. Consider an air cargo problem with 10 airports, where each airport has 5 planes and 20 pieces of cargo. The goal is to move all the cargo at airport A to airport B. There is a simple solution to the problem: load the 20 pieces of cargo into one of the planes at A, fly the plane to B, and unload the cargo. Finding the solution can be difficult because the average branching factor is huge: each of the 50 planes can fly to 9 other airports, and each of the 200 packages can be either unloaded (if it is loaded) or loaded into any plane at its airport (if it is unloaded). So in any state there is a minimum of 450 actions (when all the packages are at airports with no planes) and a maximum of 10,450 (when all packages and planes are at the same airport). On average, let's say there are about 2000 possible actions per state, so the search graph up to the depth of the obvious solution has about 2000^{41} nodes.	**3**
c.	Explain in brief about hierarchical planning.	
	Hierarchical Planning is an Artificial Intelligence (AI) problem solving approach for a certain kind of *planning problems* -- the kind focusing on *problem decomposition,* where problems are step-wise refined into smaller and smaller ones until the problem is finally solved. A solution hereby is a sequence of actions that's executable in a given initial state . AI systems will probably have to do what humans appear to do: plan at higher levels of abstraction. A reasonable plan for the Hawaii vacation might be "Go to San Francisco airport; take Hawaiian Airlines flight 11 to Honolulu; do vacation stuff for two weeks; take Hawaiian Airlines flight 12 back to San Francisco; go home." Given such a plan,the action "Go to San Francisco airport" can be viewed as a planning task in itself, with a solution such as "Drive to the long-term parking lot; park; take the shuttle to the terminal." Each of these actions can be decomposed , until we reach the level of actions that can be executed without deliberation to generate the required motor control sequences. In this example planning can occur both before and during the execution of the plan; for example, one would probably defer the problem of planning a route from a parking spot in long-term parking to the shuttle bus stop until a particular parking spot has been found during execution. Thus, that particular action will remain at an abstract level prior to the execution phase. For example, complex software is created from a hierarchy of subroutines or object classes; armies operate as a hierarchy of units; is reduced to a small number of activities at the next lower level, so the computational cost of finding the correct way to arrange those activities for the current problem is small. Nonhierarchical methods reduce a task to a large number of individual actions; for large-scale problems, this is completely impractical.	**2** **2** **1**
d.	Write a short note on Sensorless Planning Problem.	
	Sensorless planning (also called conformant planning). – Handles domains where the state of the world is not fully known. – Comes up with plans that work in all possible cases. • Handles domains where the state of the world is not fully known. • Comes up with plans that work in all possible cases. • Example: – You have a wall made of bricks. – You have a can of white paint. – Action: Paint(brick), effect: Color(brick, white). – Goal: every brick should be painted white In a fully observable domain, you could: – Know the initial color of every brick. – Make a plan to paint all the bricks that are not white initially. – No need to paint bricks that are already white.	**1** **2**

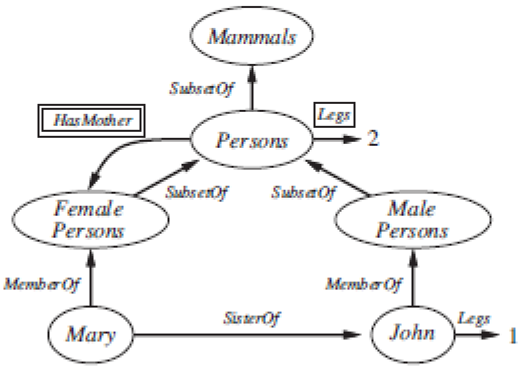
	Suppose the world is not fully observable. – We actually cannot observe the color of a brick. Suppose that the world is deterministic. – The effects of an action are known in advance. What plan would ensure achieving the goal? – Paint all bricks, regardless of their initial color (which we don't know anyway). – It may be overkill, since some bricks may already be white, but it is the only plan that guarantees achieving the goal. Limitations: – While there are a few domains simple enough to allow for sensorless planning – Many real world domains are too complicated for this approach, and you can't come up with plans that work regardless of what the state of the world is.	**2**
e.	What are events? Explain its importance.	
	Event calculus, which is based on points of time rather than on situations. Event calculus reifies fluents and events. The fluent At(Shankar , Berkeley) is an object that refers to the fact of Shankar being in Berkeley, but does not by itself say anything about whether it is true. To assert that a fluent is actually true at some point in time we use the predicate T, as in T(At(Shankar , Berkeley), t). Events are described as instances of event categories. The event E1 of Shankar flying from San Francisco to Washington, D.C. is described as	**1**
	E1 ∈ Flyings ∧ Flyer (E1, Shankar) ∧ Origin(E1, SF) ∧ Destination(E1,DC) . If this is too verbose, we can define an alternative three-argument version of the category of flying events and say E1 ∈ Flyings(Shankar , SF,DC) . Happens(E1, i) to say that the event E1 took place over the time interval i, and we say the same thing in functional form with Extent(E1)=i. We represent time intervals by a (start, end) pair of times; that is, i = (t1, t2) is the time interval that starts at t1 and endsat t2. The complete set of predicates for one version of the event calculus is T(f, t) Fluent f is true at time t Happens(e, i) Event e happens over the time interval i Initiates(e, f, t) Event e causes fluent f to start to hold at time t Terminates(e, f, t) Event e causes fluent f to cease to hold at time t Clipped(f, i) Fluent f ceases to be true at some point during time interval i Restored (f, i) Fluent f becomes true sometime during time interval i	**2**
	We assume a distinguished event, Start , that describes the initial state by saying which fluents are initiated or terminated at the start time. We define T by saying that a fluent holds at a point in time if the fluent was initiated by an event at some time in the past and was not made false (clipped) by an intervening event. A fluent does not hold if it was terminated by an event and not made true (restored) by another event. Formally, the axioms are: Happens(e, (t_1, t_2)) ∧ Initiates(e, f, t_1) ∧ ¬Clipped(f, (t_1, t)) ∧ t_1 < t ⇒T(f, t) Happens(e, (t_1, t_2)) ∧ Terminates(e, f, t_1)∧ ¬Restored (f, (t_1, t)) ∧ t_1 < t ⇒¬T(f, t) where Clipped and Restored are defined by Clipped(f, (t_1, t_2)) ⇔∃ e, t, t_3 Happens(e, (t, t_3)) ∧ t_1 ≤ t < t_2 ∧ Terminates(e, f, t) Restored (f, (t_1, t_2)) ⇔ ∃ e, t, t_3 Happens(e, (t, t_3)) ∧ t_1 ≤ t < t_2 ∧ Initiates(e, f, t) It is convenient to extend T to work over intervals as well as time points; a fluent holds over an interval if it holds on every point within the interval:	**2**

T(f, (t1, t2)) ⇔ [∀ t (t1 ≤ t < t2) ⇒ T(f, t)]
Fluents and actions are defined with domain-specific axioms that are similar to successorstate axioms. For example, we can say that the only way a wumpus-world agent gets an arrow is at the start, and the only way to use up an arrow is to shoot it:
Initiates(e, HaveArrow(a), t) ⇔ e = Start
Terminates(e, HaveArrow(a), t) ⇔ e ∈ Shootings(a)

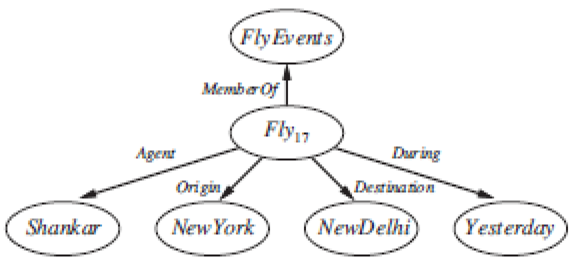
f. What is semantic network? Show the semantic network representation with a suitable example.

semantics networks **2**
—at least those with well-defined semantics *are* a form of logic.
-It provides for certain kinds of sentences is often more convenient, but if we strip away the "human interface" issues, the underlying concepts
—objects, relations, quantification, are the same.
There are many variants of semantic networks, but all are capable of representing
individual objects,
categories of objects, and
relations among objects
A typical graphical notation displays object or category names in ovals or boxes, and connects them with labeled links.

3



A semantic network with four objects (John, Mary, 1, and 2) and four categories. Relations are denoted by labeled links.



A fragment of a semantic network showing the representation of the logical assertion *Fly*(*Shankar*, *NewYork*, *NewDelhi*, *Yesterday*).