1. Describe different types of environments applicable to AI agents

An environment in artificial intelligence is the surrounding of the agent. The agent takes input from the environment through sensors and delivers the output to the environment through actuators. There are several types of environments:

- Fully Observable vs Partially Observable
- Deterministic vs Stochastic
- Competitive vs Collaborative
- Single-agent vs Multi-agent
- Static vs Dynamic
- Discrete vs Continuous

1. Fully Observable vs Partially Observable

- When an agent sensor is capable to sense or access the complete state of an agent at each point of time, it is said to be a fully observable environment else it is partially observable .
- Maintaining a fully observable environment is easy as there is no need to keep track of the history of the surrounding.
- An environment is called **unobservable** when the agent has no sensors in all environments.
- **Example:**
    - **Chess –** the board is fully observable, so are the opponent's moves
    - **Driving –** the environment is partially observable because what's around the corner is not known

2. Deterministic vs Stochastic

- When an uniqueness in the agent's current state completely determines the next state of the agent, the environment is said to be deterministic.
- Stochastic environment is random in nature which is not unique and cannot be completely determined by the agent.
- **Example:**
  Chess – there would be only few possible moves for a coin at the current state and these moves can be determined
  Self Driving Cars – the actions of a self driving car are not unique, it varies time to time

3. Competitive vs Collaborative

- An agent is said to be in a competitive environment when it competes against another agent to optimize the output.
- Game of chess is competitive as the agents compete with each other to win the game which is the output.
- An agent is said to be in a collaborative environment when multiple agents cooperate to produce the desired output.
- When multiple self-driving cars are found on the roads, they cooperate with each other to avoid collisions and reach their destination which is the output desired.

4. Single-agent vs Multi-agent

- An environment consisting of only one agent is said to be a single agent environment.
- A person left alone in a maze is an example of single agent system.
- An environment involving more than one agent is a multi agent environment.
- The game of football is multi agent as it involves 10 players in each team.

5. Dynamic vs Static

- An environment that keeps constantly changing itself when the agent is up with some action is said to be dynamic.
- A roller coaster ride is dynamic as it is set in motion and the environment keeps changing every instant.
- An idle environment with no change in it's state is called a static environment.
- An empty house is static as there's no change in the surroundings when an agent enters.

6. Discrete vs Continuous

- If an environment consists of a finite number of actions that can be deliberated in the environment to obtain the output, it is said to be a discrete environment.
- The game of chess is discrete as it has only a finite number of moves. The number of moves might vary with every game, but still, it's finite.
- The environment in which the actions performed cannot be numbered ie. is not discrete, is said to be continuous.
- Self-driving cars are an example of continuous environments as their actions are driving, parking, etc. which cannot be numbered.

2. Define blind search and informed search. Hence discuss the merits and demerits of each.

Informed Search: Informed Search algorithms have information on the goal state which helps in more efficient searching. This information is obtained by a function that estimates how close a state is to the goal state.
Example: Greedy Search and Graph Search

Uninformed Search or blind search : Uninformed search algorithms have no additional information on the goal node other than the one provided in the problem definition. The plans to reach the goal state from the start state differ only by the order and length of actions.
Examples: Depth First Search and Breadth-First Search

Informed Search vs. Uninformed Search:

Informed Search	Uninformed Search
It uses knowledge for the searching process.	It doesn't use knowledge for searching process.
It finds solution more quickly.	It finds solution slow as compared to informed search.
It is highly efficient.	It is mandatory efficient.
Cost is low.	Cost is high.
It consumes less time.	It consumes moderate time.
It provides the direction regarding the solution.	No suggestion is given regarding the solution in it.

Informed Search	Uninformed Search
It is less lengthy while implementation.	It is more lengthy while implementation.
Greedy Search, A* Search, Graph Search	Depth First Search, Breadth First Search

3. Compose your opinion about heuristic function

A **heuristic function**, also called simply a **heuristic**, is a **function** that ranks alternatives in search algorithms at each branching step based on available information to decide which branch to follow. For example, it may approximate the exact solution.

The heuristic function is a way to inform the search about the direction to a goal.

It provides an informed way to guess which neighbor of a node will lead to a goal. It must use only information that can be readily obtained about a node

The objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem at hand.

This solution may not be the best of all the actual solutions to this problem, or it may simply approximate the exact solution. But it is still valuable because finding it does not require a prohibitively long time.

The representation may be approximate cost of the path from the goal node or number of hopes required to reach to the goal.

The heuristic function that we are considering, for a node n is, h (n) =estimated cost of the cheapest path from the state at node n to a goal.

Example: For Travelling Salesman Problem, the sum of the distances traveled so far can be a simple heuristic function.

It is of two types: Maximize or Minimize function.

In maximization, greater the cost of node, better is node while in minimization, lower the cost better is the node.

4. Explain any two Informed Search Strategies

5) Define Artificial Intelligence and list the task domains of Artificial Intelligence.

Artificial intelligence (or AI) is basically the capability of a machine or a computer program to think and learn like a human, in short, mimicking human intelligence processes. Algorithms are created and applied into a dynamic computing environment, enabling machines to learn and make decisions with the data.

As there are many different types of AI, it can be classified into 3 main categories.

Narrow Artificial Intelligence

Also known as Weak AI, it is developed and designated for a particular task. The single task it performs is within a limited context, being excellent at routine jobs, be it physical or cognitive.

Artificial General Intelligence

Also known as AGI, it has generalized cognitive ability to analyse and decide, in which the AI provides solutions for an unfamiliar task it encounters. It can applies understanding and reasoning like how a normal human would in the specific environment.

Artificial Super Intelligence

Also known as ASI, it exceed human abilities, being be able to mimic human thoughts and achieve an accuracy higher than normal humans. ASI proves superior to humans in terms cognitive ability.

Now, AI is dependent on the tasks they carry out. It can range from simple task, to complex analytics. There are a few domains on how AI carry out their functions.

Formal Tasks- Tasks which requires logic and constraints to function.

(For eg. Mathematics, Games, verification)

Mundane tasks- Routine everyday tasks that requires common sense reasoning

(For eg. Perception, robotics, natural language, Vision, Speech)

Expert tasks- Tasks which requires high analytical and thinking skills, a job only a professionals can do

(For eg. Financial analysis, medical diagnostics, engineering, scientific analysis, consulting)

So having these types of domains, there are different types of works and information AI can interpret. Usually an AI can approach them in the following ways.

Symbolic approach is how an AI application interprets human intelligence through symbol manipulation and interpretation. Basic concepts are broken down into symbols and the AI processes them into something understandable to solve the problem. These symbols are arranged in such a way that the structures pertaining them can relate to each other. It can be through lists, hierarchies, or even networks to determine it.

Sub-symbolic approach is how an AI recognizes things or patterns without any specific knowledge. Using a connectionist system method, the AI processes a cluster of high-dimensional quantitative sensory data to determine the relevancy of each node of data and determine the output of it. It functions on limited information, gradually adjusting to finally to best fit a solution.

Statistical approach is how AI solve specific problems using mathematics as tools. The AI will gather, organize, analyze, synthesize and interpret numerical information from data and deliver measurable forumlative results. This helps retrieve important data and trends which are vital and necessary.

6) State and explain algorithm for the Best First search with an example

Best-first Search Algorithm (Greedy Search):

Greedy best-first search algorithm always selects the path which appears best at that moment. It is the combination of depth-first search and breadth-first search algorithms. It uses the heuristic

function and search. Best-first search allows us to take the advantages of both algorithms. With the help of best-first search, at each step, we can choose the most promising node. In the best first search algorithm, we expand the node which is closest to the goal node and the closest cost is estimated by heuristic function, i.e.

1. f(n)= g(n).

Were, h(n)= estimated cost from node n to the goal.

The greedy best first algorithm is implemented by the priority queue.

1)Best first search algorithm:

- **Step 1:** Place the starting node into the OPEN list.
- **Step 2:** If the OPEN list is empty, Stop and return failure.
- **Step 3:** Remove the node n, from the OPEN list which has the lowest value of h(n), and places it in the CLOSED list.
- **Step 4:** Expand the node n, and generate the successors of node n.
- **Step 5:** Check each successor of node n, and find whether any node is a goal node or not. If any successor node is goal node, then return success and terminate the search, else proceed to Step 6.
- **Step 6:** For each successor node, algorithm checks for evaluation function f(n), and then check if the node has been in either OPEN or CLOSED list. If the node has not been in both list, then add it to the OPEN list.
- **Step 7:** Return to Step 2.

Advantages:

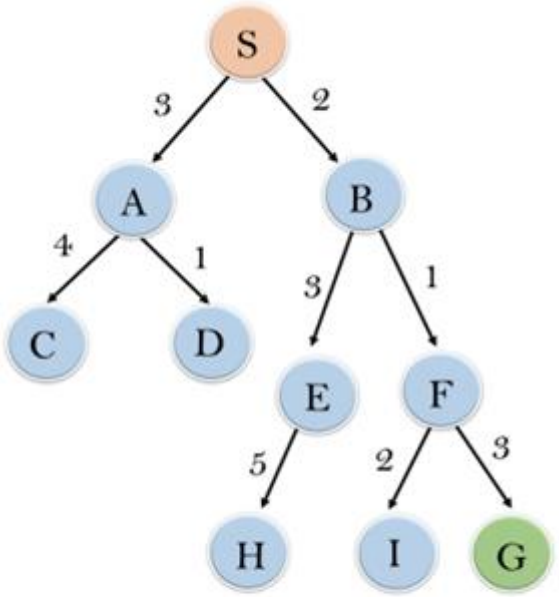
- Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
- This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

- It can behave as an unguided depth-first search in the worst case scenario.
- It can get stuck in a loop as DFS.
- This algorithm is not optimal.

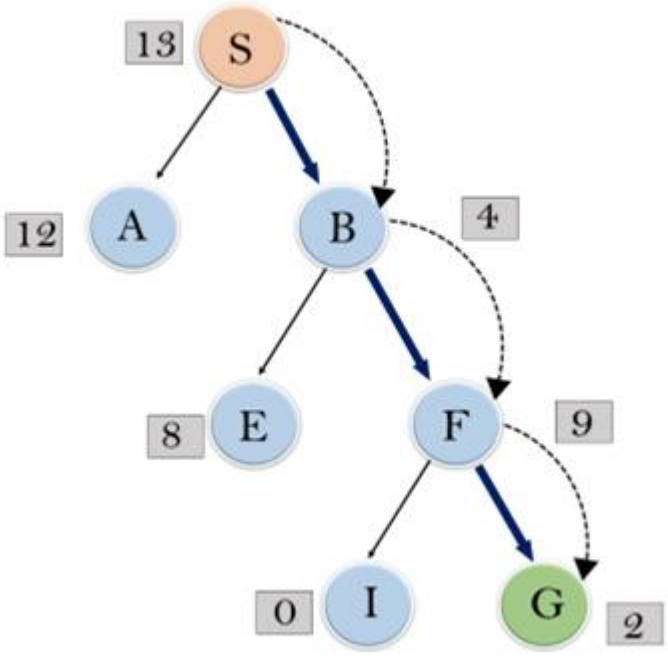
Example:

Consider the below search problem, and we will traverse it using greedy best-first search. At each iteration, each node is expanded using evaluation function f(n)=h(n) , which is given in the below table.



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0

In this search example, we are using two lists which are **OPEN** and **CLOSED** Lists. Following are the iteration for traversing the above example.



Expand the nodes of S and put in the CLOSED list

Initialization: Open [A, B], Closed [S]

Iteration 1: Open [A], Closed [S, B]

Iteration 2: Open [E, F, A], Closed [S, B]
: Open [E, A], Closed [S, B, F]

Iteration 3: Open [I, G, E, A], Closed [S, B, F]
: Open [I, E, A], Closed [S, B, F, G]

Hence the final solution path will be: **S----> B----->F----> G**

Time Complexity: The worst case time complexity of Greedy best first search is $O(b^m)$.

Space Complexity: The worst case space complexity of Greedy best first search is $O(b^m)$. Where, m is the maximum depth of the search space.

Complete: Greedy best-first search is also incomplete, even if the given state space is finite.

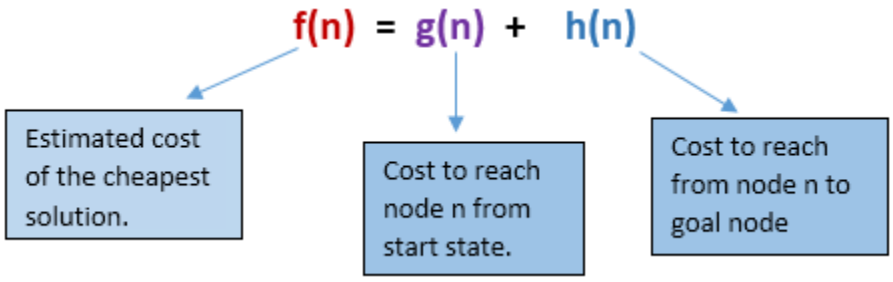
Optimal: Greedy best first search algorithm is not optimal.

7) Explain A* algorithm and write its pseudo code.

A* Search Algorithm:

A* search is the most commonly known form of best-first search. It uses heuristic function h(n), and cost to reach the node n from the start state g(n). It has combined features of UCS and greedy best-first search, by which it solve the problem efficiently. A* search algorithm finds the shortest path through the search space using the heuristic function. This search algorithm expands less search tree and provides optimal result faster. A* algorithm is similar to UCS except that it uses g(n)+h(n) instead of g(n).

In A* search algorithm, we use search heuristic as well as the cost to reach the node. Hence we can combine both costs as following, and this sum is called as a **fitness number**.



At each point in the search space, only those node is expanded which have the lowest value of f(n), and the algorithm terminates when the goal node is found.

Algorithm of A* search:

Step1: Place the starting node in the OPEN list.

Step 2: Check if the OPEN list is empty or not, if the list is empty then return failure and stops.

Step 3: Select the node from the OPEN list which has the smallest value of evaluation function (g+h), if node n is goal node then return success and stop, otherwise

Step 4: Expand node n and generate all of its successors, and put n into the closed list. For each successor n', check whether n' is already in the OPEN or CLOSED list, if not then compute evaluation function for n' and place into Open list.

Step 5: Else if node n' is already in OPEN and CLOSED, then it should be attached to the back pointer which reflects the lowest g(n') value.

Step 6: Return to **Step 2**.

Advantages:

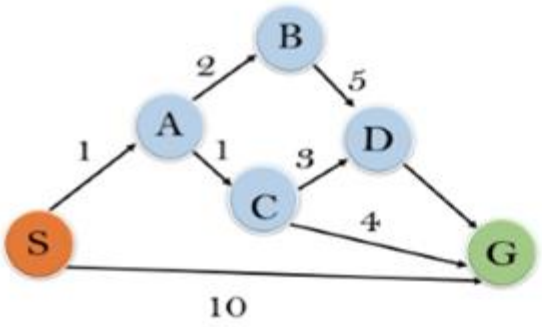
- A* search algorithm is the best algorithm than other search algorithms.
- A* search algorithm is optimal and complete.
- This algorithm can solve very complex problems.

Disadvantages:

- It does not always produce the shortest path as it mostly based on heuristics and approximation.
- A* search algorithm has some complexity issues.
- The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

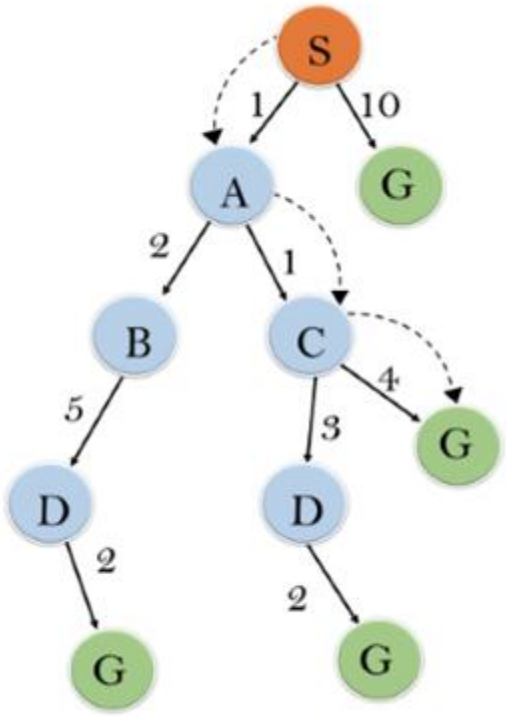
Example:

In this example, we will traverse the given graph using the A* algorithm. The heuristic value of all states is given in the below table so we will calculate the f(n) of each state using the formula f(n)= g(n) + h(n), where g(n) is the cost to reach any node from start state.
Here we will use OPEN and CLOSED list.



State	h(n)
S	5
A	3
B	4
C	2
D	6
G	0

Solution:



Initialization: {(S, 5)}

Iteration1: {(S--> A, 4), (S-->G, 10)}

Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

Iteration3: {(S--> A-->C--->G, 6), (S--> A-->C--->D, 11), (S--> A-->B, 7), (S-->G, 10)}

Iteration 4 will give the final result, as **S--->A--->C--->G** it provides the optimal path with cost 6.

Points to remember:

- A* algorithm returns the path which occurred first, and it does not search for all remaining paths.
- The efficiency of A* algorithm depends on the quality of heuristic.
- A* algorithm expands all nodes which satisfy the condition f(n)<="" li="">

Complete: A* algorithm is complete as long as:

- Branching factor is finite.
- Cost at every action is fixed.

Optimal: A* search algorithm is optimal if it follows below two conditions:

- **Admissible:** the first condition requires for optimality is that h(n) should be an admissible heuristic for A* tree search. An admissible heuristic is optimistic in nature.
- **Consistency:** Second required condition is consistency for only A* graph-search.

If the heuristic function is admissible, then A* tree search will always find the least cost path.

Time Complexity: The time complexity of A* search algorithm depends on heuristic function, and the number of nodes expanded is exponential to the depth of solution d. So the time complexity is O(b^d), where b is the branching factor.

Space Complexity: The space complexity of A* search algorithm is **O(b^d)**

8) Explain hill climbing algorithm. Explain plateau, ridge, local maxima and global maxima.

hill Climbing is a heuristic search used for mathematical optimization problems in the field of Artificial Intelligence.
Given a large set of inputs and a good heuristic function, it tries to find a sufficiently good solution to the problem. This solution may not be the global optimal maximum.

- In the above definition, **mathematical optimization problems** implies that hill-climbing solves the problems where we need to maximize or minimize a given real function by choosing values from the given inputs. Example-Travelling salesman problem where we need to minimize the distance traveled by the salesman.
- 'Heuristic search' means that this search algorithm may not find the optimal solution to the problem. However, it will give a good solution in **reasonable time.**
- A **heuristic function** is a function that will rank all the possible alternatives at any branching step in search algorithm based on the available information. It helps the algorithm to select the best route out of possible routes.

Features of Hill Climbing

1. **Variant of generate and test algorithm :** It is a variant of generate and test algorithm. The generate and test algorithm is as follows :

   1. Generate possible solutions.
   2. Test to see if this is the expected solution.
   3. If the solution has been found quit else go to step 1.

   Hence we call Hill climbing as a variant of generate and test algorithm as it takes the feedback from the test procedure. Then this feedback is utilized by the generator in deciding the next move in search space.

2. **Uses the** Greedy approach **:** At any point in state space, the search moves in that direction only which optimizes the cost of function with the hope of finding the optimal solution at the end.

Types of Hill Climbing

1. **Simple Hill climbing :** It examines the neighboring nodes one by one and selects the first neighboring node which optimizes the current cost as next node.
   Algorithm for Simple Hill climbing :
   Step 1 : *Evaluate the initial state. If it is a goal state then stop and return success. Otherwise, make initial state as current state.*
   Step 2 : *Loop until the solution state is found or there are no new operators present which can be applied to the current state.*
   a) Select a state that has not been yet applied to the current state and apply it to produce a new state.

   b) Perform these to evaluate new state
   i. If the current state is a goal state, then stop and return success.
   ii. If it is better than the current state, then make it current state and proceed further.
   iii. If it is not better than the current state, then continue in the loop until a solution is found.

   Step 3 : *Exit.*

2. **Steepest-Ascent Hill climbing:** It first examines all the neighboring nodes and then selects the node closest to the solution state as of next node.

   Step 1 : *Evaluate the initial state. If it is goal state then exit else make the current state as initial state*
   Step 2 : *Repeat these steps until a solution is found or current state does not change*
   i. Let 'target' be a state such that any successor of the current state will be better than it;

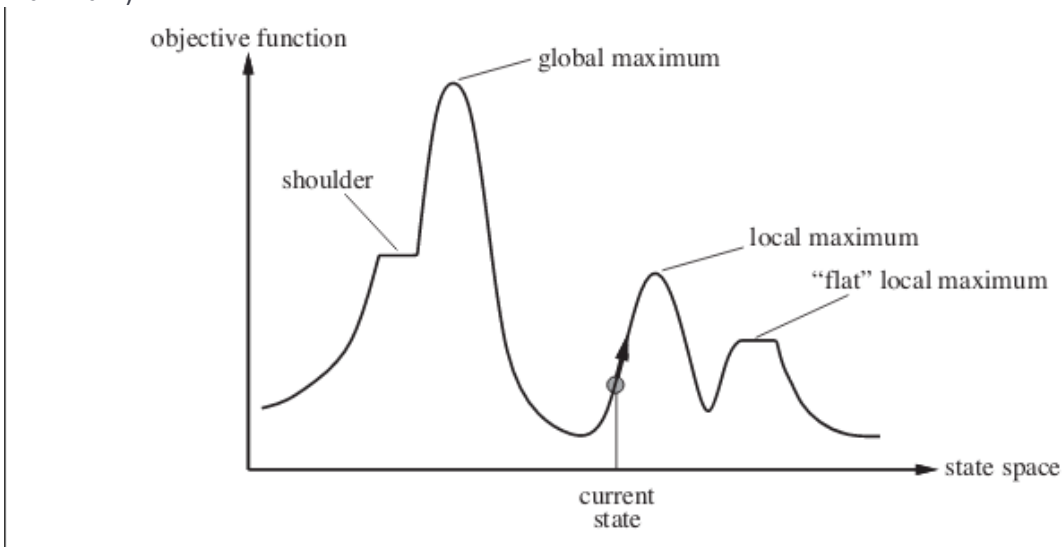
   ii. for each operator that applies to the current state
   a. apply the new operator and create a new state
   b. evaluate the new state
   c. if this state is goal state then quit else compare with 'target'
   d. if this state is better than 'target', set this state as 'target'
   e. if target is better than current state set current state to Target

Step 3 : *Exit*

3. **Stochastic hill climbing :** It does not examine all the neighboring nodes before deciding which node to select .It just selects a neighboring node at random and decides (based on the amount of improvement in that neighbor) whether to move to that neighbor or to examine another.

State Space diagram for Hill Climbing

State space diagram is a graphical representation of the set of states our search algorithm can reach vs the value of our objective function(the function which we wish to maximize).
X-axis : denotes the state space ie states or configuration our algorithm may reach.
Y-axis : denotes the values of objective function corresponding to a particular state.
The best solution will be that state space where objective function has maximum value(global maximum).



Different regions in the State Space Diagram

1. **Local maximum:** It is a state which is better than its neighboring state however there exists a state which is better than it(global maximum). This state is better because here the value of the objective function is higher than its neighbors.
2. **Global maximum :** It is the best possible state in the state space diagram. This because at this state, objective function has highest value.
3. **Plateua/flat local maximum :** It is a flat region of state space where neighboring states have the same value.
4. **Ridge :** It is region which is higher than its neighbours but itself has a slope. It is a special kind of local maximum.
5. **Current state :** The region of state space diagram where we are currently present during the search.
6. **Shoulder :** It is a plateau that has an uphill edge.

Problems in different regions in Hill climbing

Hill climbing cannot reach the optimal/best state(global maximum) if it enters any of the following regions :

1. **Local maximum :** At a local maximum all neighboring states have a values which is worse than the current state. Since hill-climbing uses a greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.
   To overcome local maximum problem : Utilize backtracking technique. Maintain a list

of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.

2. **Plateau :** On plateau all neighbors have same value . Hence, it is not possible to select the best direction.
   To overcome plateaus : Make a big jump. Randomly select a state far away from the current state. Chances are that we will land at a non-plateau region
3. **Ridge :** Any point on a ridge can look like peak because movement in all possible directions is downward. Hence the algorithm stops when it reaches this state.
   To overcome Ridge : In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

9) Explain simulated annealing.

Simulated Annealing (SA) is an effective and general form of optimization. It is useful in finding global optima in the presence of large numbers of local optima. "Annealing" refers to an analogy with thermodynamics, specifically with the way that metals cool and anneal. Simulated annealing uses the objective function of an optimization problem instead of the energy of a material.

Implementation of SA is surprisingly simple. The algorithm is basically hill-climbing except instead of picking the best move, it picks a random move. If the selected move improves the solution, then it is always accepted. Otherwise, the algorithm makes the move anyway *with some probability* less than 1. The probability decreases exponentially with the "badness" of the move, which is the amount deltaE by which the solution is worsened (i.e., energy is increased.)

Prob(accepting uphill move) ~ 1 - exp(deltaE / kT))

A parameter T is also used to determine this probability. It is analogous to temperature in an annealing system. At higher values of T, uphill moves are more likely to occur. As T tends to zero, they become more and more unlikely, until the algorithm behaves more or less like hill-climbing. In a typical SA optimization, T starts high and is gradually decreased according to an "annealing schedule". The parameter k is some constant that relates temperature to energy (in nature it is Boltzmann's constant.)

Simulated annealing is an algorithm that combines hill-climbing with a random walk in someway that yields both efficiency and completeness.
Choosing based on condition.
Here try to minimize value of objective function

- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their size and frequency.

The schedule input determines the value of the temperature T as a function of time.

- At fixed "temperature" T, state occupation probability reaches Boltzman distribution

$$p(x) = \alpha e^{\frac{E(x)}{kT}}$$

- When T is decreased slowly enough it always reaches the best state x^* because $e^{\frac{E(x^*)}{kT}} / e^{\frac{E(x)}{kT}} = e^{\frac{E(x^*)-E(x)}{kT}} \gg 1$ for small T.
(Is this necessarily an interesting guarantee?)

Delta E = +VE moving up, right path ,current state is good state

The probability that a transaction to a higher energy state will occur and so given by a function:

$$P = e^{-\Delta E/kT}$$

- E is the +ve level in the energy level
- T is the temperature
- k is Boltzmann's constant

To explain simulated annealing, we switch our point of view from hill climbing to **gradient descent** (i.e., minimizing cost) and imagine the task of getting a ping-pong ball into the deepest crevice in a bumpy surface. If we just let the ball roll, it will come to rest at a local minimum. If we shake the surface, we can bounce the ball out of the local minimum. The trick is to shake just hard enough to bounce the ball out of local minima but not hard enough to dislodge it from the global minimum. The simulated-annealing solution is to start by shaking hard (i.e., at a high temperature) and then gradually reduce the intensity of the shaking (i.e., lower the temperature).

Simulated annealing

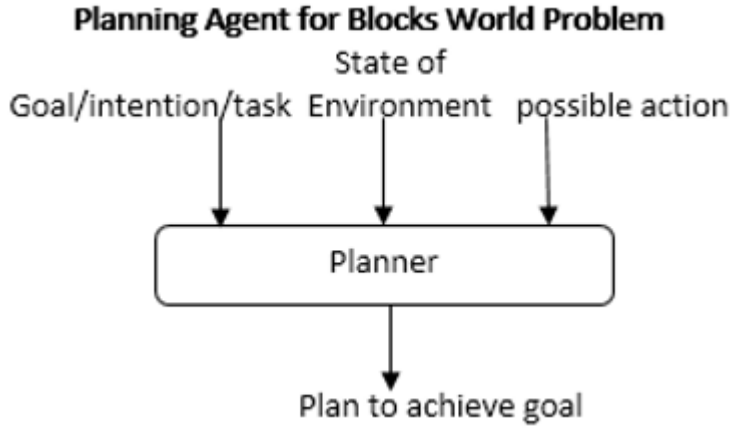
function SIMULATED ANNEALING (*problem*, *schedule*)
returns a solution state

inputs:
problem, a problem
schedule, a mapping from time to "temperature"
local variables:
current, a node
next, a node

current ← MAKE-NODE(*problem*.INITIAL-STATE)
for $t = 1$ **to** ∞ **do**
$T \leftarrow$ *schedule*(*t*)
if *T=0* **then return** *current*
next ← a randomly selected successor of *current*
$\Delta E \leftarrow$ *next*.VALUE - *current*.VALUE
if $\Delta E > 0$ **then** *current* ← *next*
else *current* ← *next* only with probability $e^{\Delta E/T}$

10**. Explain problem reduction with respect to AND-OR graphs.**

11. Design a planning agent for a Blocks World problem. Assume suitable initial state and final state for the problem



Designing the Agent

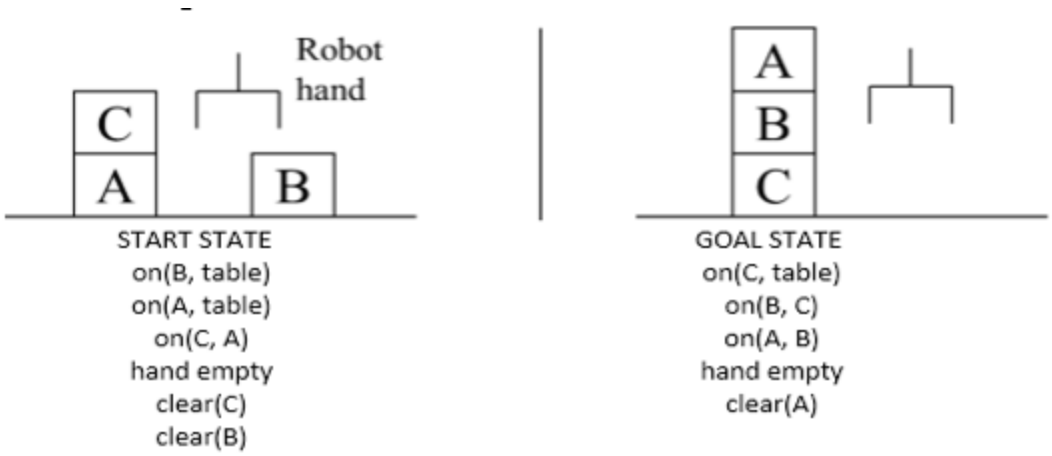
Idea is to give an agent:

- Representation of goal/intention to achieve
- Representation of actions it can perform; and
- Representation of the environment;

  Then have the agent generate a plan to achieve the goal.

  The plan is generated entirely by the planning system, without human intervention.

  Assume start & goal states as below:



a. STRIPS : A planning system – Has rules with precondition deletion list and addition list

Sequence of actions :

b. Grab C

c. Pickup C

d. Place on table C

e. Grab B

f. Pickup B

g. Stack B on C

h. Grab A

i. Pickup A

j. Stack A on B

Rules:

k. R1 : pickup(x)

1. Precondition & Deletion List : hand empty, on(x,table), clear(x)
2. Add List : holding(x)

l. R2 : putdown(x)

1. Precondition & Deletion List : holding(x)
2. Add List : hand empty, on(x,table), clear(x)

m. R3 : stack(x,y)

1. Precondition & Deletion List :holding(x), clear(y)
2. Add List : on(x,y), clear(x)

n. R4 : unstack(x,y)

1. Precondition & Deletion List : on(x,y), clear(x)
2. Add List : holding(x), clear(y)

Plan for the assumed blocks world problem

For the given problem, Start →→ Goal can be achieved by the following sequence:

1. Unstack(C,A)
2. Putdown(C)
3. Pickup(B)
4. Stack(B,C)
5. Pickup(A)
6. Stack(A,B)

12. Compare forward and backward state search algorithms with figure.

Forward Chaining:
Forward Chaining the Inference Engine goes through all the facts, conditions and derivations before deducing the outcome i.e When based on available data a decision is taken then the

process is called as Forwarding chaining, It works from an initial state and reaches to the goal(final decision).

Example:
A
A -> B
B
————————————
He is running.
If he is running, he sweats.
He is sweating.

Backward Chaining:

In this, the inference system knows the final decision or goal, this system starts from the goal and works backwards to determine what facts must be asserted so that the goal can be achieved, i.e it works from goal(final decision) and reaches the initial state.

Example:
B
A -> B
A
————————————
He is sweating.
If he is running, he sweats.
He is running.

Difference between Forwarding Chaining and Backward Chaining:

	Forward Chaining	Backward Chaining
1.	When based on available data a decision is taken then the process is called as Forward chaining.	Backward chaining starts from the goal and works backward to determine what facts must be asserted so that the goal can be achieved.
2.	Forward chaining is known as data-driven technique because we reaches to the goal using the available data.	Backward chaining is known as goal-driven technique because we start from the goal and reaches the initial state in order to extract the facts.
3.	It is a bottom-up approach.	It is a top-down approach.
4.	It applies the Breadth-First Strategy.	It applies the Depth-First Strategy.
5.	Its goal is to get the conclusion.	Its goal is to get the possible facts or the required data.
6.	Slow as it has to use all the rules.	Fast as it has to use only a few rules.

7.	It operates in forward direction i.e it works from initial state to final decision.	It operates in backward direction i.e it works from goal to reach initial state.
8.	Forward chaining is used for the planning, monitoring, control, and interpretation application.	It is used in automated inference engines, theorem proofs, proof assistants and other artificial intelligence applications.

13) Explain planning and acting in nondeterministic domains.

15. Describe Hill climbing search algorithm, what are the problems face by Hill climbing search?

- Hill climbing algorithm is a local search algorithm which continuously moves in the direction of increasing elevation/value to find the peak of the mountain or best solution to the problem. It terminates when it reaches a peak value where no neighbor has a higher value.
- Hill climbing algorithm is a technique which is used for optimizing the mathematical problems. One of the widely discussed examples of Hill climbing algorithm is Traveling-salesman Problem in which we need to minimize the distance traveled by the salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state and not beyond that.
- A node of hill climbing algorithm has two components which are state and value.
- Hill Climbing is mostly used when a good heuristic is available.
- In this algorithm, we don't need to maintain and handle the search tree or graph as it only keeps a single current state.

Features of Hill Climbing:

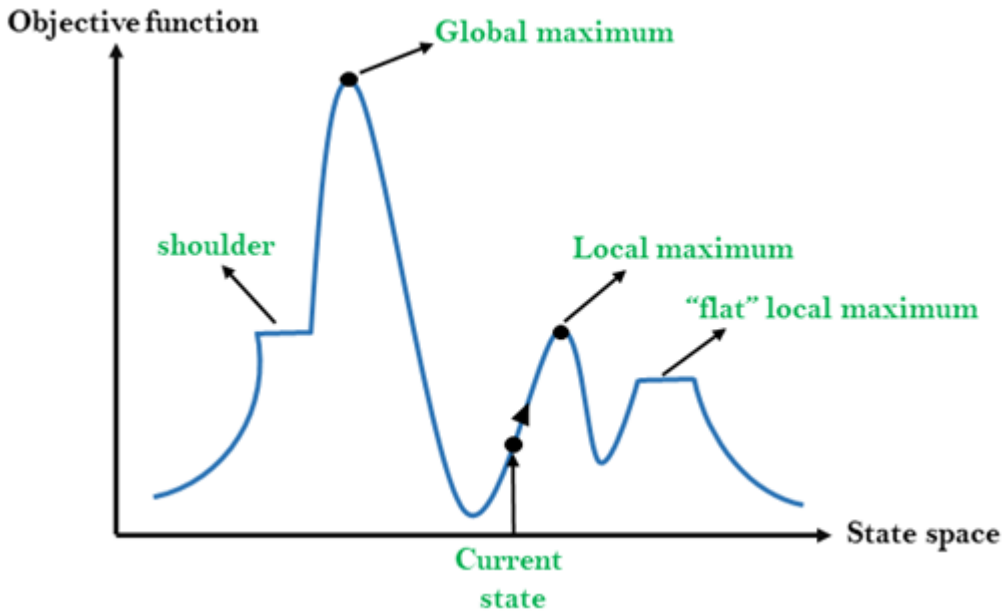
Following are some main features of Hill Climbing Algorithm:

- **Generate and Test variant:** Hill Climbing is the variant of Generate and Test method. The Generate and Test method produce feedback which helps to decide which direction to move in the search space.
- **Greedy approach:** Hill-climbing algorithm search moves in the direction which optimizes the cost.
- **No backtracking:** It does not backtrack the search space, as it does not remember the previous states.

State-space Diagram for Hill Climbing:

The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.

On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis. If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum. If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.



Different regions in the state space landscape:

Local Maximum: Local maximum is a state which is better than its neighbor states, but there is also another state which is higher than it.

Global Maximum: Global maximum is the best possible state of state space landscape. It has the highest value of objective function.

Current state: It is a state in a landscape diagram where an agent is currently present.

Flat local maximum: It is a flat space in the landscape where all the neighbor states of current states have the same value.

Shoulder: It is a plateau region which has an uphill edge.

Types of Hill Climbing Algorithm:

- Simple hill Climbing:
- Steepest-Ascent hill-climbing:
- Stochastic hill Climbing:

1. Simple Hill Climbing:

Simple hill climbing is the simplest way to implement a hill climbing algorithm. **It only evaluates the neighbor node state at a time and selects the first one which optimizes current cost and set it as a current state**. It only checks it's one successor state, and if it finds better than the current state, then move else be in the same state. This algorithm has the following features:

- Less time consuming
- Less optimal solution and the solution is not guaranteed

Algorithm for Simple Hill Climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and Stop.
- **Step 2:** Loop Until a solution is found or there is no new operator left to apply.
- **Step 3:** Select and apply an operator to the current state.
- **Step 4:** Check new state:
    a. If it is goal state, then return success and quit.
    b. Else if it is better than the current state then assign new state as a current state.
    c. Else if not better than the current state, then return to step2.
- **Step 5:** Exit.

2. Steepest-Ascent hill climbing:

The steepest-Ascent algorithm is a variation of simple hill climbing algorithm. This algorithm examines all the neighboring nodes of the current state and selects one neighbor node which is closest to the goal state. This algorithm consumes more time as it searches for multiple neighbors

Algorithm for Steepest-Ascent hill climbing:

- **Step 1:** Evaluate the initial state, if it is goal state then return success and stop, else make current state as initial state.
- **Step 2:** Loop until a solution is found or the current state does not change.
    a. Let SUCC be a state such that any successor of the current state will be better than it.
    b. For each operator that applies to the current state:
        a. Apply the new operator and generate a new state.
        b. Evaluate the new state.
        c. If it is goal state, then return it and quit, else compare it to the SUCC.
        d. If it is better than SUCC, then set new state as SUCC.

e. If the SUCC is better than the current state, then set current state to SUCC.

- **Step 5:** Exit.

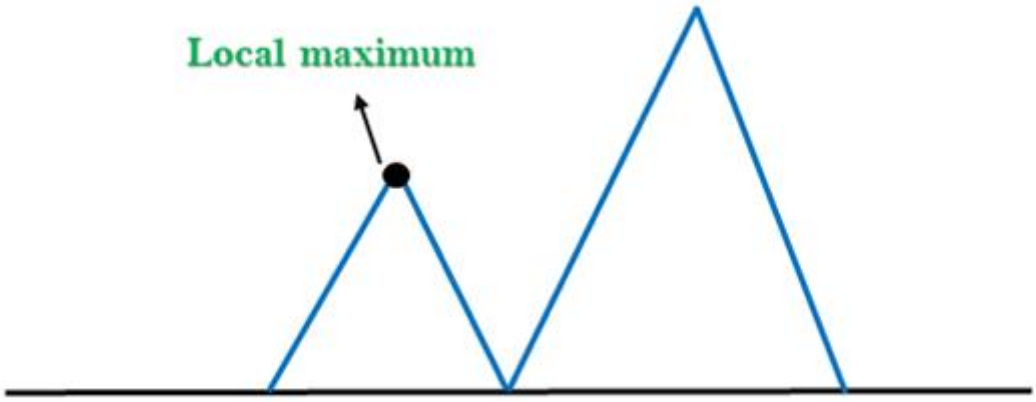
3. Stochastic hill climbing:

Stochastic hill climbing does not examine for all its neighbor before moving. Rather, this search algorithm selects one neighbor node at random and decides whether to choose it as a current state or examine another state.

Problems in Hill Climbing Algorithm:

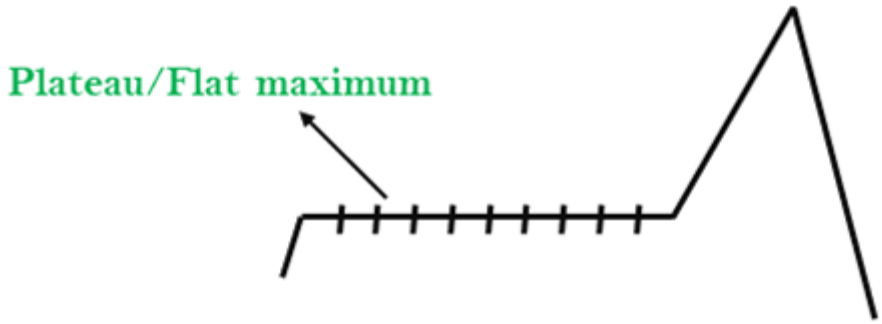
1. Local Maximum: A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than the local maximum.

Solution: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of the promising path so that the algorithm can backtrack the search space and explore other paths as well.



2. Plateau: A plateau is the flat area of the search space in which all the neighbor states of the current state contains the same value, because of this algorithm does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

Solution: The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.



3. Ridges: A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.

Solution: With the use of bidirectional search, or by moving in different directions, we can improve this problem.

Hill climbing cannot reach the optimal/best state(global maximum) if it enters any of the following regions :

1. **Local maximum :** At a local maximum all neighboring states have a values which is worse than the current state. Since hill-climbing uses a greedy approach, it will not move to the worse state and terminate itself. The process will end even though a better solution may exist.
   To overcome local maximum problem : Utilize backtracking technique. Maintain a list of visited states. If the search reaches an undesirable state, it can backtrack to the previous configuration and explore a new path.
2. **Plateau :** On plateau all neighbors have same value . Hence, it is not possible to select the best direction.
   To overcome plateaus : Make a big jump. Randomly select a state far away from the current state. Chances are that we will land at a non-plateau region
3. **Ridge :** Any point on a ridge can look like peak because movement in all possible directions is downward. Hence the algorithm stops when it reaches this state.
   To overcome Ridge : In this kind of obstacle, use two or more rules before testing. It implies moving in several directions at once.

4.

18. Discuss about the following: i) Greedy best-first search. (ii) A* search (iii) Memory bounded heuristic search.

Refer 6th and 7th qn

Simplified Memory Bounded A* is a shortest path algorithm based on the A* algorithm. The main advantage of SMA* is that it uses a bounded memory, while the A* algorithm might need exponential memory. All other characteristics of SMA* are inherited from A

SMA* has the following properties

- It works with a heuristic, just as A*
- It is complete if the allowed memory is high enough to store the shallowest solution
- It is optimal if the allowed memory is high enough to store the shallowest optimal solution, otherwise it will return the best solution that fits in the allowed memory
- It avoids repeated states as long as the memory bound allows it
- It will use all memory available
- Enlarging the memory bound of the algorithm will only speed up the calculation
- When enough memory is available to contain the entire search tree, then calculation has an optimal speed

Like A*, it expands the most promising branches according to the heuristic. What sets SMA* apart is that it prunes nodes whose expansion has revealed less promising than expected. The approach allows the algorithm to explore branches and backtrack to explore other branches.

19. Explain Alpha-Beta pruning in Min-Max search. Why it is suitable for two player game?

Alpha-Beta Pruning

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**. This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- The two-parameter can be defined as:
    a. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-∞**.
    b. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is **+∞**.

- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is:

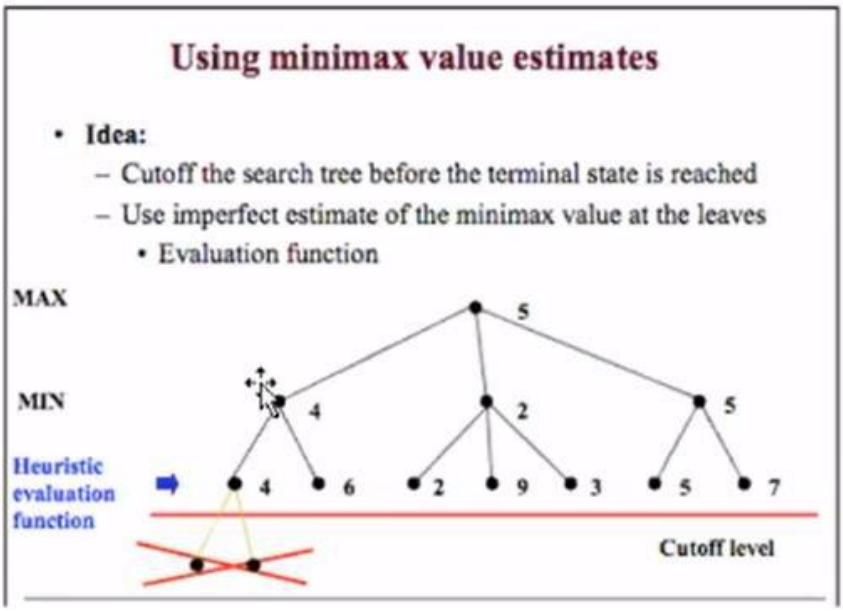
1. α>=β

Key points about alpha-beta pruning:

- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v
---------------------------------------------------
function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
---------------------------------------------------
function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Figure 5.7 The alpha–beta search algorithm. Notice that these routines are the same as the MINIMAX functions in Figure 5.3, except for the two lines in each of MIN-VALUE and MAX-VALUE that maintain α and β (and the bookkeeping to pass these parameters along).



20) Show the performance measure of various search algorithms.

23). Explain different approaches to knowledge representation.

Approaches to knowledge representation:

There are mainly four approaches to knowledge representation, which are givenbelow:

1. Simple relational knowledge:

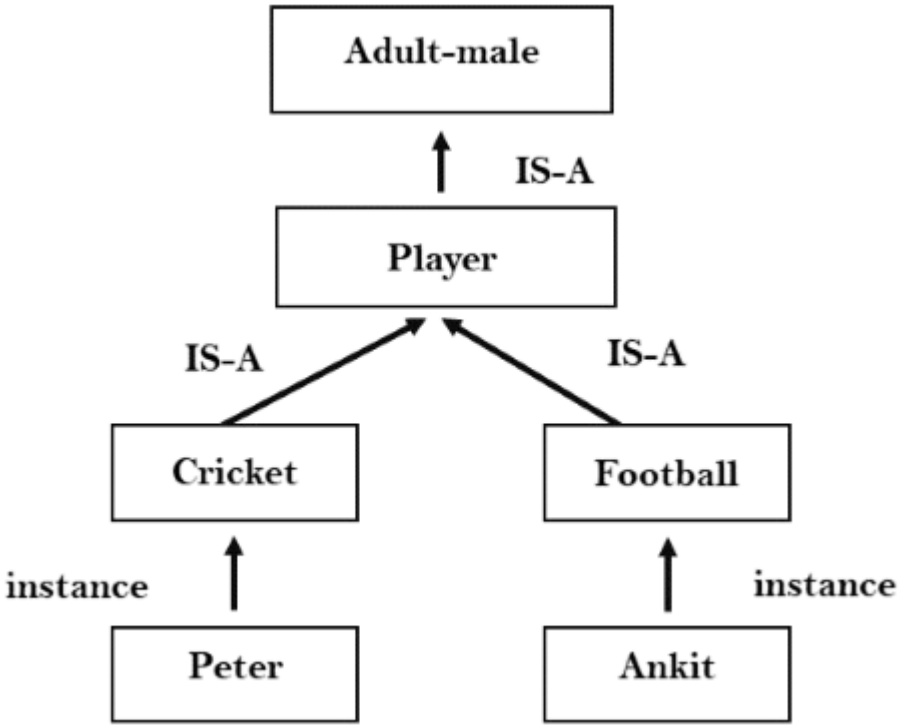
- It is the simplest way of storing facts which uses the relational method, and each fact about a set of the object is set out systematically in columns.
- This approach of knowledge representation is famous in database systems where the relationship between different entities is represented.
- This approach has little opportunity for inference.

Example: The following is the simple relational knowledge representation.

Player	Weight	Ag
Player1	65	
Player2	58	
Player3	75	

2. Inheritable knowledge:

- In the inheritable knowledge approach, all data must be stored into a hierarchy of classes.
- All classes should be arranged in a generalized form or a hierarchal manner.
- In this approach, we apply inheritance property.
- Elements inherit values from other members of a class.
- This approach contains inheritable knowledge which shows a relation between instance and class, and it is called instance relation.
- Every individual frame can represent the collection of attributes and its value.
- In this approach, objects and values are represented in Boxed nodes.
- We use Arrows which point from objects to their values.
- **Example:**



3. Inferential knowledge:

- Inferential knowledge approach represents knowledge in the form of formal logics.
- This approach can be used to derive more facts.
- It guaranteed correctness.
- **Example:** Let's suppose there are two statements:
    a. Marcus is a man

b. All men are mortal

Then it can represent as;

man(Marcus)

∀x = man (x) ----------> mortal (x)s

4. Procedural knowledge:

- Procedural knowledge approach uses small programs and codes which describes how to do specific things, and how to proceed.
- In this approach, one important rule is used which is **If-Then rule**.
- In this knowledge, we can use various coding languages such as **LISP language** and **Prolog language**.
- We can easily represent heuristic or domain-specific knowledge using this approach.
- But it is not necessary that we can represent all cases in this approach.

Requirements for knowledge Representation system:

A good knowledge representation system must possess the following properties.

1. **1. Representational Accuracy:**
   KR system should have the ability to represent all kind of required knowledge.
2. **2. Inferential Adequacy:**
   KR system should have ability to manipulate the representational structures to produce new knowledge corresponding to existing structure.
3. **3. Inferential Efficiency:**
   The ability to direct the inferential knowledge mechanism into the most productive directions by storing appropriate guides.
4. **4. Acquisitional efficiency-** The ability to acquire the new knowledge easily using automatic methods.

24. Distinguish forward and backward reasoning explain with example.

In Artificial intelligence, the purpose of the search is to find the path through a problem space. There are two ways to pursue such a search that are forward and backward reasoning. The significant difference between both of them is that forward reasoning starts with the initial data towards the goal. Conversely, backward reasoning works in opposite fashion where the purpose is to determine the initial facts and information with the help of the given results.

Definition of Forward Reasoning

The solution of a problem generally includes the initial data and facts in order to arrive at the solution. These unknown facts and information is used to deduce the result. For example, while diagnosing a patient the doctor first check the symptoms and medical condition of the body such as temperature, blood pressure, pulse, eye colour, blood, etcetera. After that, the patient symptoms are analysed and compared against the predetermined symptoms. Then the doctor is able to provide the medicines according to the symptoms of the patient. So, when a solution employs this manner of reasoning, it is known as **forward reasoning**.

Steps that are followed in the forward reasoning

The inference engine explores the knowledge base with the provided information for constraints whose precedence matches the given current state.

- In the first step, the system is given one or more than one constraints.
- Then the rules are searched in the knowledge base for each constraint. The rules that fulfil the condition are selected(i.e., IF part).
- Now each rule is able to produce new conditions from the conclusion of the invoked one. As a result, THEN part is again included in the existing one.
- The added conditions are processed again by repeating step 2. The process will end if there is no new conditions exist.

Definition of Backward Reasoning

The **backward reasoning** is inverse of forward reasoning in which goal is analysed in order to deduce the rules, initial facts and data. We can understand the concept by the similar example given in the above definition, where the doctor is trying to diagnose the patient with the help of the inceptive data such as symptoms. However, in this case, the patient is experiencing a problem in his body, on the basis of which the doctor is going to prove the symptoms. This kind of reasoning comes under backward reasoning.

Steps that are followed in the backward reasoning

In this type of reasoning, the system chooses a goal state and reasons in the backward direction. Now, let’s understand how does it happens and what steps are followed.

- Firstly, the goal state and the rules are selected where the goal state reside in the THEN part as the conclusion.
- From the IF part of the selected rule the subgoals are made to be satisfied for the goal state to be true.
- Set initial conditions important to satisfy all the subgoals.
- Verify whether the provided initial state matches with the established states. If it fulfils the condition then the goal is the solution otherwise other goal state is selected.

In Artificial intelligence, the purpose of the search is to find the path through a problem space. There are two ways to pursue such a search that are forward and backward reasoning. The significant difference between both of them is that forward reasoning starts with the initial data towards the goal. Conversely, backward reasoning works in opposite fashion where the purpose is to determine the initial facts and information with the help of the given results.

Content: Forward Vs Backward Reasoning

Definition of Forward Reasoning

The solution of a problem generally includes the initial data and facts in order to arrive at the solution. These unknown facts and information is used to deduce the result. For example, while diagnosing a patient the doctor first check the symptoms and medical condition of the body such as temperature, blood pressure, pulse, eye colour, blood, etcetera. After that, the patient symptoms are analysed and compared against the predetermined symptoms. Then the doctor is able to provide the medicines according to the symptoms of the patient. So, when a solution employs this manner of reasoning, it is known as **forward reasoning**.

Steps that are followed in the forward reasoning

The inference engine explores the knowledge base with the provided information for constraints whose precedence matches the given current state.

- In the first step, the system is given one or more than one constraints.
- Then the rules are searched in the knowledge base for each constraint. The rules that fulfil the condition are selected(i.e., IF part).
- Now each rule is able to produce new conditions from the conclusion of the invoked one. As a result, THEN part is again included in the existing one.
- The added conditions are processed again by repeating step 2. The process will end if there is no new conditions exist.

Definition of Backward Reasoning

The **backward reasoning** is inverse of forward reasoning in which goal is analysed in order to deduce the rules, initial facts and data. We can understand the concept by the similar example given in the above definition, where the doctor is trying to diagnose the patient with the help of the inceptive data such as symptoms. However, in this case, the patient is experiencing a problem in his body, on the basis of which the doctor is going to prove the symptoms. This kind of reasoning comes under backward reasoning.

Steps that are followed in the backward reasoning

In this type of reasoning, the system chooses a goal state and reasons in the backward direction. Now, let's understand how does it happens and what steps are followed.

- Firstly, the goal state and the rules are selected where the goal state reside in the THEN part as the conclusion.
- From the IF part of the selected rule the subgoals are made to be satisfied for the goal state to be true.
- Set initial conditions important to satisfy all the subgoals.
- Verify whether the provided initial state matches with the established states. If it fulfils the condition then the goal is the solution otherwise other goal state is selected.

Key Differences Between Forward and Backward Reasoning in AI

1. The forward reasoning is data-driven approach while backward reasoning is a goal driven.
2. The process starts with new data and facts in the forward reasoning. Conversely, backward reasoning begins with the results.
3. Forward reasoning aims to determine the result followed by some sequences. On the other hand, backward reasoning emphasis on the acts that support the conclusion.
4. The forward reasoning is an opportunistic approach because it could produce different results. As against, in backward reasoning, a specific goal can only have certain predetermined initial data which makes it restricted.
5. The flow of the forward reasoning is from the antecedent to consequent while backward reasoning works in reverse order in which it starts from conclusion to incipient.

Key Differences Between Forward and Backward Reasoning in AI

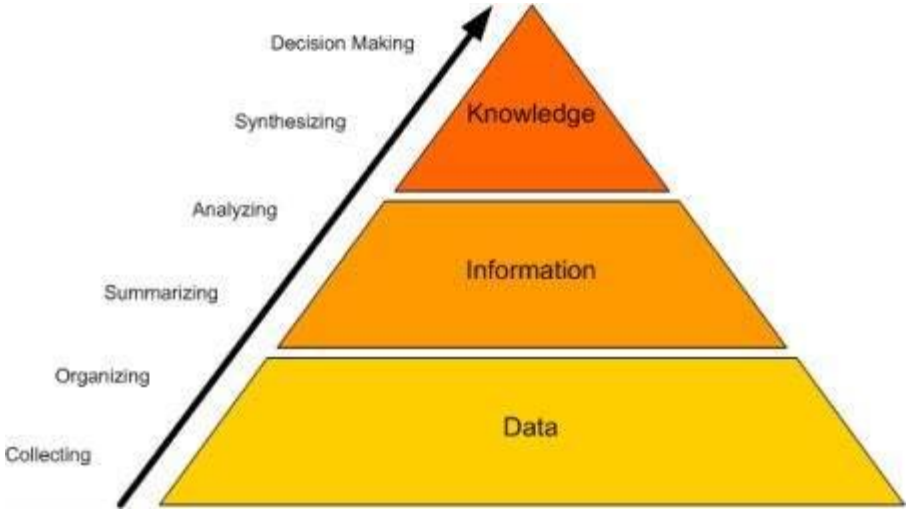
1. The forward reasoning is data-driven approach while backward reasoning is a goal driven.
2. The process starts with new data and facts in the forward reasoning. Conversely, backward reasoning begins with the results.
3. Forward reasoning aims to determine the result followed by some sequences. On the other hand, backward reasoning emphasis on the acts that support the conclusion.
4. The forward reasoning is an opportunistic approach because it could produce different results. As against, in backward reasoning, a specific goal can only have certain predetermined initial data which makes it restricted.
5. The flow of the forward reasoning is from the antecedent to consequent while backward reasoning works in reverse order in which it starts from conclusion to incipient.

25. Write in detail about the various steps of knowledge process.

The Knowledge Management Process

The process of knowledge management is universal for any enterprise. Sometimes, the resources used, such as tools and techniques, can be unique to the organizational environment.

The Knowledge Management process has six basic steps assisted by different tools and techniques. When these steps are followed sequentially, the data transforms into knowledge.



Step 1: Collecting

This is the most important step of the knowledge management process. If you collect the incorrect or irrelevant data, the resulting knowledge may not be the most accurate. Therefore, the decisions made based on such knowledge could be inaccurate as well.

There are many methods and tools used for data collection. First of all, data collection should be a procedure in knowledge management process. These procedures should be properly documented and followed by people involved in data collection process.

The data collection procedure defines certain data collection points. Some points may be the summary of certain routine reports. As an example, monthly sales report and daily attendance reports may be two good resources for data collection.

With data collection points, the data extraction techniques and tools are also defined. As an example, the sales report may be a paper-based report where a data entry operator needs to feed the data manually to a database whereas, the daily attendance report may be an online report where it is directly stored in the database.

In addition to data collecting points and extraction mechanism, data storage is also defined in this step. Most of the organizations now use a software database application for this purpose.

Step 2: Organizing

The data collected need to be organized. This organization usually happens based on certain rules. These rules are defined by the organization.

As an example, all sales-related data can be filed together and all staff-related data could be stored in the same database table. This type of organization helps to maintain data accurately within a database.

If there is much data in the database, techniques such as 'normalization' can be used for organizing and reducing the duplication.

This way, data is logically arranged and related to one another for easy retrieval. When data passes step 2, it becomes information.

Step 3: Summarizing

In this step, the information is summarized in order to take the essence of it. The lengthy information is presented in tabular or graphical format and stored appropriately.

For summarizing, there are many tools that can be used such as software packages, charts (Pareto, cause-and-effect), and different techniques.

Step 4: Analyzing

At this stage, the information is analyzed in order to find the relationships, redundancies and patterns.

An expert or an expert team should be assigned for this purpose as the experience of the person/team plays a vital role. Usually, there are reports created after analysis of information.

Step 5: Synthesizing

At this point, information becomes knowledge. The results of analysis (usually the reports) are combined together to derive various concepts and artefacts.

A pattern or behavior of one entity can be applied to explain another, and collectively, the organization will have a set of knowledge elements that can be used across the organization.

This knowledge is then stored in the organizational *knowledge base* for further use.

Usually, the knowledge base is a software implementation that can be accessed from anywhere through the Internet.

You can also buy such knowledge base software or download an open-source implementation of the same for free.

Step 6: Decision Making

At this stage, the knowledge is used for decision making. As an example, when estimating a specific type of a project or a task, the knowledge related to previous estimates can be used.

This accelerates the estimation process and adds high accuracy. This is how the organizational knowledge management adds value and saves money in the long run.