NOTES-ERA

GET OUT OF YOUR COMFORT ZONE

ARTIFICIAL INTELLIGENCE

ONE SHOT

मेडी-केप्स विश्वविद्यालय, इंदौर

Medi-Caps University, Indore

Course Code	Course Name	Hours Per Week			
		L	T	P	Credits
CS3EA01	Artificial Intelligence	3	0	0	3

Unit I: Introduction to artificial intelligence, various types of production systems, Characteristics of production systems, Study and comparison of breadth first search and depth first search techniques.

Unit II: Optimization Problems: Hill-climbing search Simulated annealing like hill Climbing, Best first Search. A* algorithm, AO* algorithms etc, and various types of control strategies, Heuristic Functions, Constraint Satisfaction Problem.

Unit III: Knowledge Representation, structures, Predicate Logic, Resolution, Refutation, Deduction, Theorem proving, Inferencing,Semantic networks, Scripts, Schemas, Frames, Conceptual dependency.

Unit IV: Uncertain Knowledge and Reasoning, forward and backward reasoning,monotonic and nonmonotonic reasoning, Probabilistic reasoning, Baye's theorem, Decision Tree, Understanding, Common sense, Planning.

Unit V: Game playing techniques like minimax procedure, alpha-beta cut-offs etc, Study of the block world problem in robotics.

Text Book:

1. Elaine Rich, Kevin Knight and Nair, Artificial Intelligence, TMH
2. S. Russel, Peter Norvig, Artificial Intelligence: A Modern Approach, Pearson.

Reference Books:

1. Saroj Kausik, Artificial Intelligence, Cengage Learning 4
2. Padhy, Artificial Intelligence and Intelligent Systems, Oxforfd University Press,
3. Nils Nilsson, Artificial Intelligence: A New Synthesis, Morgan Kaufmann.
4. David Poole, Alan Mackworth, Artificial Intelligence: Foundations for Computational Agents, Cambridge Univ. Press..

Unit 1

Artificial Intelligence is the design, study and construction of computer programs that behave intelligently.

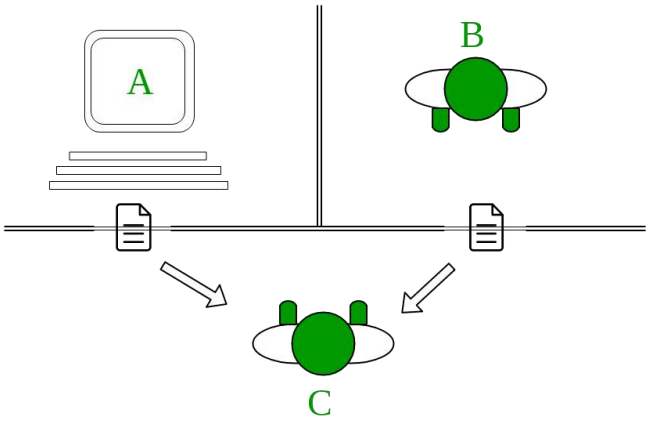
Father of AI: John McCarthy.

Turing Test:

Turing test is used to determine whether or not computer (machine) can think intelligently like human?

Imagine a game of three players having two humans and one computer, an interrogator (as human) is isolated from other two players. The interrogator job is to try and figure out which one is human and which one is computer by asking questions from both of them. To make the things harder computer is trying to make the interrogator guess wrongly. In other words, computer would try to indistinguishable from human as much as possible.

Human beings are intelligent, to be called intelligent, a machine must produce responses that are indistinguishable from those of a human.



Goals of AI:

To Create Expert Systems – The systems which exhibit intelligent behavior, learn, demonstrate, explain, and advice its users.

To Implement Human Intelligence in Machines – Creating systems that understand, think, learn, and behave like humans.

Applications of AI

1. Gaming − AI plays crucial role in strategic games such as chess, poker, tic-tac-toe, etc., where machine can think of large number of possible positions based on heuristic knowledge.
2. Natural Language Processing − It is possible to interact with the computer that understands natural language spoken by humans.
3. Expert Systems − There are some applications which integrate machine, software, and special information to impart reasoning and advising. They provide explanation and advice to the users.
4. Vision Systems − These systems understand, interpret, and comprehend visual input on the computer. For example,
    a. A spying aero plane takes photographs, which are used to figure out spatial information or map of the areas.
    b. Doctors use clinical expert system to diagnose the patient.
    c. Police use computer software that can recognize the face of criminal with the stored portrait made by forensic artist.
5. Speech Recognition − Some intelligent systems are capable of hearing and comprehending the language in terms of sentences and their meanings while a human talk to it. It can handle different accents, slang words, noise in the background, change in human's noise due to cold, etc.
6. Handwriting Recognition − The handwriting recognition software reads the text written on paper by a pen or on screen by a stylus. It can recognize the shapes of the letters and convert it into editable text.
7. Intelligent Robots − Robots are able to perform the tasks given by a human.

Advantages:

- High Accuracy.
- High-Speed.
- High reliability.
- Useful for risky areas: (defusing a bomb, exploring the ocean floor, etc.)
- Digital Assistant
- Useful as a public utility: (self-driving car)

Disadvantages:

- High Cost.
- Can't think out of the box: Even we are making smarter machines with AI, but still they cannot work out of the box, as the robot will only do that work for which they are trained, or programmed.
- No feelings and emotions.
- Increase dependency on machines.
- No Original Creativity: As humans are so creative and can imagine some new ideas but still AI machines cannot beat this power of human intelligence and cannot be creative and imaginative.

Production System

A production system consists of four basic components:

1. **A set of rules** of the form Ci → Ai where Ci is the condition part and Ai is the action part. The condition determines when a given rule is applied, and the action determines what happens when the rule is applied.

2. **A database**, which contains all the appropriate information for the particular task. Some part of the database may be permanent while some part of this may pertain only to the solution of the current problem.

3. **A control strategy** that specifies order in which the rules will applied in order to resolving the given conflicts.

4. **A rule applier** is a computational system that implements the control strategy and applies the set of rules.

Types of PS:

1. Monotonic: In this the application of one rule will never prevent the application of another rule.
2. Non-Monotonic: In this the application of one rule will prevent the application of another rule.
3. Partial Commutative: It's a type of production system in which the application of a sequence of rules transforms state X into state Y, then any permutation of those rules that is allowable also transforms state X into state Y.
4. Commutative: Monotonic + Partial Commutative.

Production System rules can be classified as:

- Deductive Inference Rules: If the original assertions are true, then the conclusion must also be true.
  Given "A" and "A implies B", we can conclude "B". (If A is true then B must be true)
  Ex: "it is raining", "the street will be wet".

- Abductive Inference Rules: It is a form of logical inference which starts with an observation or set of observations and then seeks to find the simplest and most likely explanation for the observations.
  Given "B" and "A implies B", it might be reasonable to expect "A". (If B is true then A might be True)
  Ex: "the street is wet"," it might have rained".

Example of PS:

Water-Jug Problem for 4 and 3 gallons of jugs.

Solution:

Gallons in the 4-Gallon Jug	Gallons in the 3-Gallon Jug	Rule Applied
0	0	
		2
0	3	
		9
3	0	
		2
3	3	
		7
4	2	
		5 or 12
0	2	
		9 or 11
2	0	

Control Strategy:

There are certain requirements for a good control strategy that you need to keep in mind, such as:

1. The first requirement for a good control strategy is that it should cause motion.
2. The second requirement for a good control strategy is that it should be systematic.
3. Finally, it must be efficient in order to find a good answer.

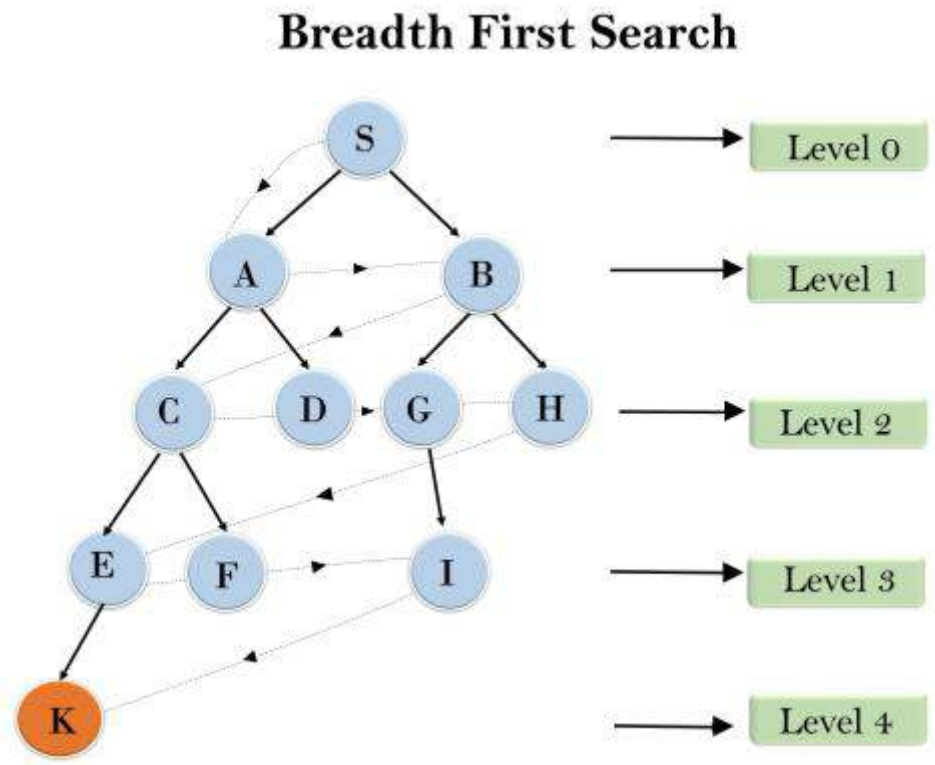
Types of CS:

1. Informed. (Unit 2)
2. Uninformed.
    a. BFS
    b. DFS

BFS: (Breadth-First Search)

Algorithm –

1. Create a variable called NODE-LIST and set it to the initial state.
2. Until a goal state is found or NODE-LIST is empty:
    a. Remove the first element from NODE-LIST and call it E. If NODE-LIST was empty, quit.
    b. For each way that each rule can match the state described in E do:
        i. Apply the rule to generate a new state,
        ii. If the new state is a goal state, quit and return this state.
        iii. Otherwise, add the new state to the end of NODE-LIST



Advantage of BFS:

1. All the nodes at 'n' level are considered before moving to 'n+1' level.
2. BFS will provide a solution if any solution exists.
3. If there are more than one solution for a given problem, then BFS will provide the minimal solution which requires the least number of steps.

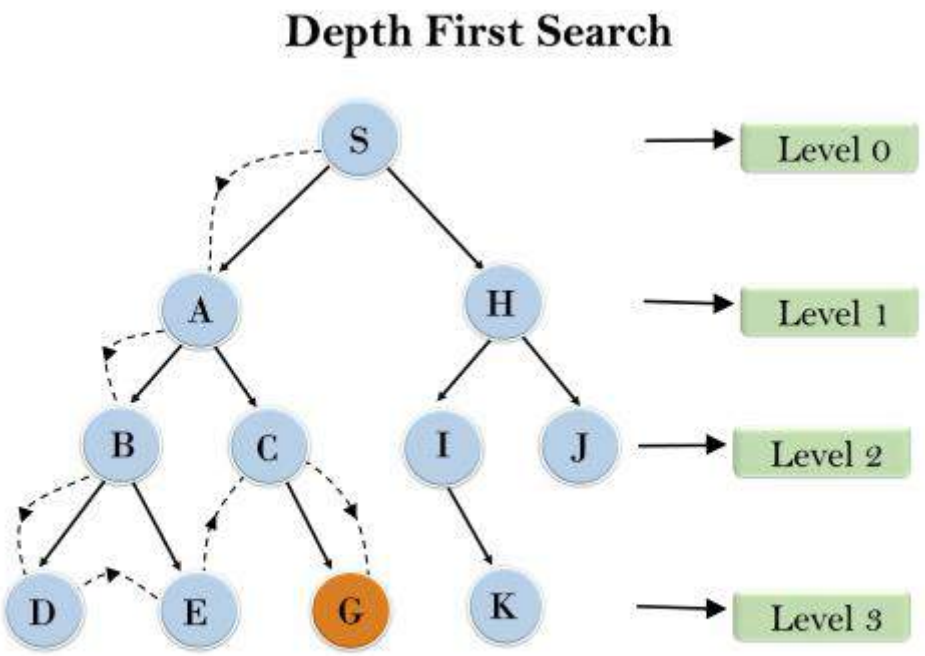
Disadvantages of BFS:

1. It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
2. BFS needs lots of time if the solution is far away from the root node.

DFS: (Depth-First Search)

Algorithm –

1. If the initial state is a goal state, quit and return success.
2. Otherwise, do the following until success or failure is signaled:
    a. Generate a successor, E, of the initial state. If there are no more successors, signal failure.
    b. Call Depth-First Search with E as the initial state.
    c. If success is returned, signal success. Otherwise continue in this loop.



Advantage of DFS:

1. It requires Less Memory since only the nodes on the current path are stored.
2. DFS may find a solution without examining much of the search space at all.
3. It takes less time to reach to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantage of DFS:

1. It may follow a single unfruitful path for a very long time, perhaps forever, before the path actually terminates in a state that has no successors.
2. It may find a long path to a solution in one part of the tree, when a shorter path exists in some other, unexplored part of the tree.

Unit 2

Heuristic Function: h(n)

Heuristics are like a tour guide.

It is a function that guides the search process in the most profitable direction by suggesting which path to follow first when more than one is available.

(Generally, its value is written above the node.)

Pure Heuristic Search:

Pure heuristic search is the simplest form of heuristic search algorithms. It expands nodes based on their heuristic value h(n). It maintains two lists, OPEN and CLOSED list. In the CLOSED list, it places those nodes which have already expanded and in the OPEN list, it places nodes which have yet not been expanded.

In the informed search we will discuss two main algorithms which are given below:

1. Best First Search Algorithm (Greedy search).
2. A* Search Algorithm.

Best-first Search Algorithm (Greedy Search):

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

$$F(n) = h(n)$$

f(n) - estimated total cost of path through n to goal. It is implemented using priority queue by increasing f(n).

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 lists, then add it to the OPEN list.

Step 7: Return to Step 2.

Advantages:

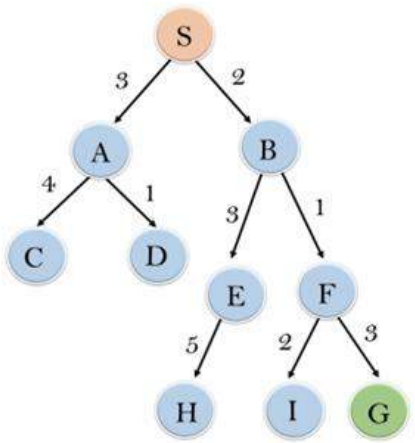
1. Best first search can switch between BFS and DFS by gaining the advantages of both the algorithms.
2. This algorithm is more efficient than BFS and DFS algorithms.

Disadvantages:

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

Example:

*ignore the values written on the edge of the graph, we only require the values of h(n).



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

Initialization: Open [A(12), B(4)], Closed [S(13)]

Iteration 1: Open [A(12)], Closed [S, B]

Iteration 2: Open [E(8), F(2), A(12)], Closed [S, B]

: Open [E(8), A(12)], Closed [S, B, F]

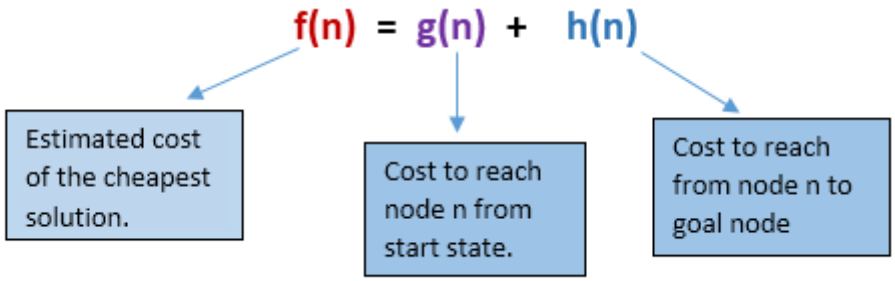
Iteration 3: Open [I(9), G(0), E(8), A(12)], Closed [S, B, F]

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

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

A* Search Algorithm:

It is similar to best-first search. It avoids expanding paths that are already expensive, but expands most promising paths first. It uses heuristic function h(n), and cost to reach the node n from the start state g(n).



Algorithm:

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.

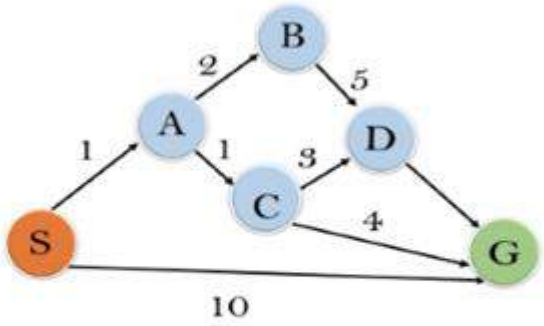
Advantages:

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

Disadvantages:

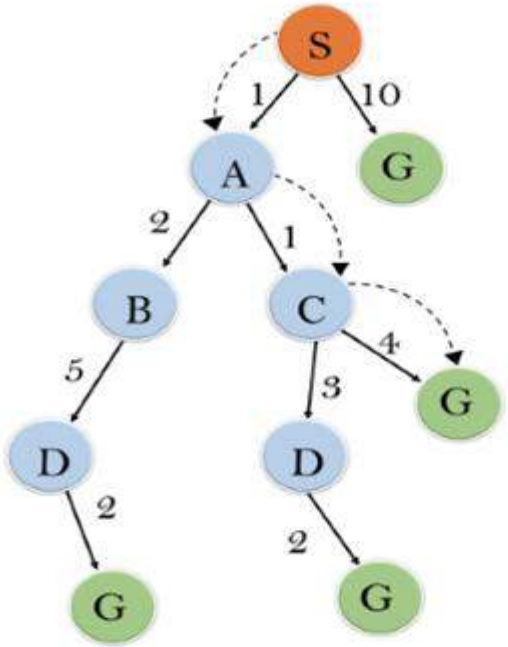
1. It does not always produce the shortest path as it mostly based on heuristics and approximation.
2. A* search algorithm has some complexity issues.
3. The main drawback of A* is memory requirement as it keeps all generated nodes in the memory, so it is not practical for various large-scale problems.

Example:



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)}

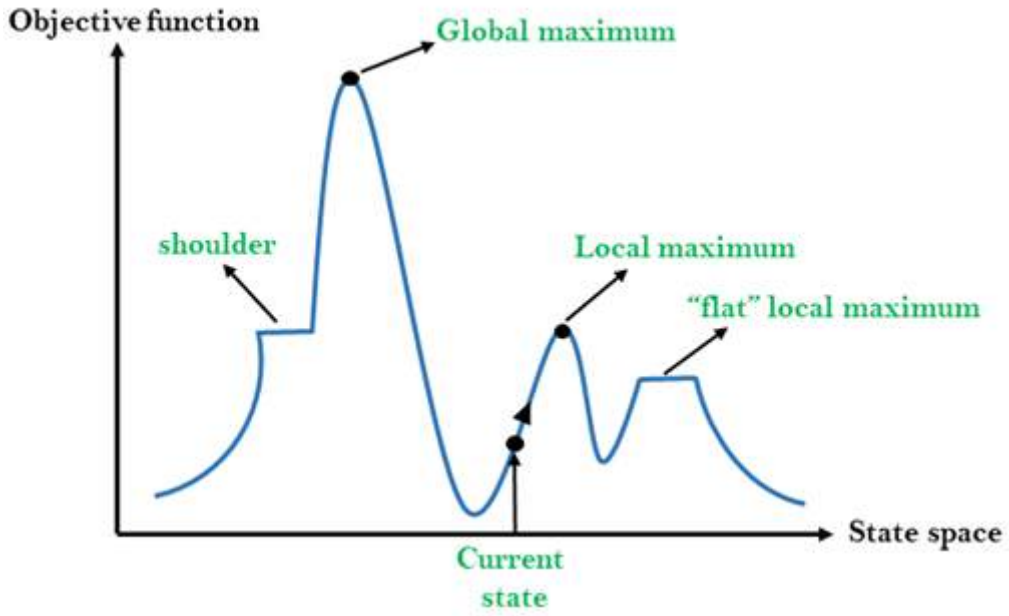
Iteration2: {(S--> A-->C, 4), (S--> A-->B, 7), (S-->G, 10)}

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.

Hill climbing:

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.



Types of Hill Climbing Algorithm:

1. Simple hill Climbing.
2. Steepest hill-climbing.

Simple Hill Climbing:

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:

1. Less time consuming.
2. Less optimal solution and the solution is not guaranteed.

Algorithm:

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:

If it is goal state, then return success and quit.

Else if it is better than the current state then assign new state as a current state.

Else if not better than the current state, then return to step2.

Step 5: Exit.

Steepest hill-climbing:

This algorithm examines all the neighboring nodes of the current state & selects one neighbor node which is closest to the goal node. Consumes more time than simple hill climbing.

Algorithm:

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.

Both the algorithm may fail to find the solution.

- Local Maximum.
- Plateau.
- Ridges.

Inference Rules

- Implication: P → Q
- Converse: Q → P
- Contrapositive: ¬ Q → ¬ P
- Inverse: ¬ P → ¬ Q

P	Q	P → Q	Q→ P	¬ Q → ¬ P	¬ P → ¬ Q.
T	T	T	T	T	T
T	F	F	T	F	T
F	T	T	F	T	F
F	F	T	T	T	T

Types of Inference rules:

1. Modus Ponens:

The Modus Ponens rule is one of the most important rules of inference, and it states that if P and P → Q is true, then we can infer that Q will be true. It can be represented as:

Notation for Modus ponens: $\frac{P \rightarrow Q, \quad P}{\therefore Q}$

Example:

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
Statement-2: "I am sleepy" ==> P
Conclusion: "I go to bed." ==> Q.
Hence, we can say that, if P→ Q is true and P is true then Q will be true.

Proof by Truth table:

P	Q	$P \rightarrow Q$
0	0	0
0	1	1
1	0	0
1	1	1

2. Modus Tollens:

The Modus Tollens rule state that if P→ Q is true and ¬ Q is true, then ¬ P will also true. It can be represented as:

Notation for Modus Tollens: $\frac{P \rightarrow Q,\ \sim Q}{\sim P}$

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
Statement-2: "I do not go to the bed."==> ~Q
Statement-3: Which infers that "**I am not sleepy**" => ~P

Proof by Truth table:

P	Q	$\sim P$	$\sim Q$	$P \rightarrow Q$
0	0	1	1	1
0	1	1	0	1
1	0	0	1	0
1	1	0	0	1

3. Hypothetical Syllogism:

The Hypothetical Syllogism rule state that if P→R is true whenever P→Q is true, and Q→R is true. It can be represented as the following notation:

Example:

Statement-1: If you have my home key then you can unlock my home. **P→Q**
Statement-2: If you can unlock my home then you can take my money. **Q→R**
Conclusion: If you have my home key then you can take my money. **P→R**

Proof by truth table:

P	Q	R	$P \rightarrow Q$	$Q \rightarrow R$	$P \rightarrow R$
0	0	0	1	1	1
0	0	1	1	1	1
0	1	0	1	0	1
0	1	1	1	1	1
1	0	0	0	1	1
1	0	1	0	1	1
1	1	0	1	0	0
1	1	1	1	1	1

4. Disjunctive Syllogism:

The Disjunctive syllogism rule state that if PVQ is true, and ¬P is true, then Q will be true. It can be represented as:

Notation of Disjunctive syllogism: $\frac{P \vee Q,\ \neg P}{Q}$

Example:

Statement-1: Today is Sunday or Monday. ==>PvQ
Statement-2: Today is not Sunday. ==> ¬P
Conclusion: Today is Monday. ==> Q

Proof by truth-table:

P	Q	$\neg P$	$P \vee Q$
0	**0**	**1**	**0**
0	**1**	**1**	**1**
1	**0**	**0**	**1**
1	**1**	**0**	**1**

5. Addition:

The Addition rule is one the common inference rule, and it states that If P is true, then PVQ will be true.

Notation of Addition: $\frac{P}{P \vee Q}$

Example:

Statement: I have a vanilla ice-cream. ==> P
Statement-2: I have Chocolate ice-cream.
Conclusion: I have vanilla or chocolate ice-cream. ==> (PvQ)

Proof by Truth-Table:

P	Q	$P \vee Q$
0	**0**	**0**
1	**0**	**1**
0	**1**	**1**
1	**1**	**1**

6. Simplification:

The simplification rule state that if P∧ Q is true, then Q or P will also be true. It can be represented as:

Notation of Simplification rule: $\frac{P \wedge Q}{Q}$ Or $\frac{P \wedge Q}{P}$

Proof by Truth-Table:

P	Q	$P \wedge Q$	
0	0	0	
1	0	0	
0	1	0	
1	1	1	←

7. Resolution:

The Resolution rule state that if P∨Q and ¬ P∧R is true, then Q∨R will also be true. It can be represented as

Notation of Resolution $\frac{P \vee Q,\ \neg P \wedge R}{Q \vee R}$

Proof by Truth-Table:

P	¬ P	Q	R	$P \vee Q$	¬ P∧R	$Q \vee R$	
0	1	0	0	0	0	0	
0	1	0	1	0	0	1	
0	1	1	0	1	1	1	←
0	1	1	1	1	1	1	←
1	0	0	0	1	0	0	
1	0	0	1	1	0	1	
1	0	1	0	1	0	1	
1	0	1	1	1	0	1	←

Unit 4 (Part-1)

Reasoning:

- The reasoning is the mental process of deriving logical conclusion and making predictions from available knowledge, facts, and beliefs.
- Or we can say, *"Reasoning is a way to infer facts from existing data."*

Uncertainty in Reasoning:

Often knowledge is imperfect (Incomplete, Inconsistent and Changing) which causes uncertainty.

AI systems must have the ability to reason under conditions of uncertainty.

Uncertainties	Desired action
‡ Incompleteness Knowledge	Compensate for lack of knowledge
‡ Inconsistencies Knowledge	Resolve ambiguities and contradictions
‡ Changing Knowledge	Update the knowledge base over time

Monotonic Reasoning:

In monotonic reasoning, once the conclusion is taken, then it will remain the same even if we add some other information to existing information in our knowledge base.

Example: Earth revolves around the Sun.

It is a true fact, and it cannot be changed even if we add another sentence in knowledge base like, "The moon revolves around the earth" Or "Earth is not round," etc.

Non-monotonic Reasoning:

Logic will be said as non-monotonic if some conclusions can become invalid when we add more knowledge into our knowledge base.

The non-monotonic reasoning is caused by the fact that our knowledge about the world is always incomplete.

Example: Let suppose the knowledge base contains the following knowledge:

- Birds can fly
- Penguins cannot fly
- Pitty is a bird

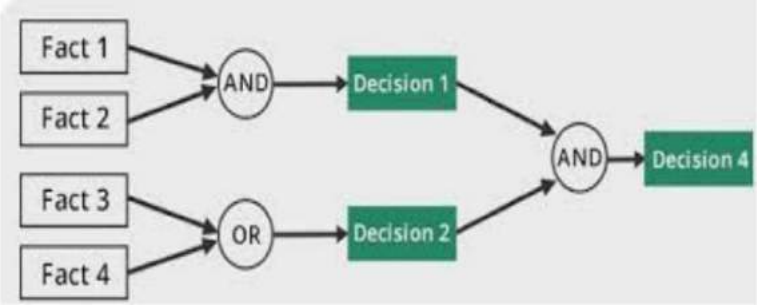
So, from the above sentences, we can conclude that Pitty can fly. However, if we add one another sentence into knowledge base "Pitty is a penguin", which concludes "Pitty cannot fly", so it invalidates the above conclusion.

Forward Chaining (or Reasoning):

- It is a process of making a conclusion based on known facts or data, by starting from the initial state and reaches the goal state.
- The Forward-chaining algorithm starts from known facts, triggers all rules whose premises are satisfied, and add their conclusion to the known facts. This process repeats until the problem is solved.
- Forward-chaining approach is also called as data-driven as we reach to the goal using available data.

Ex:

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**.

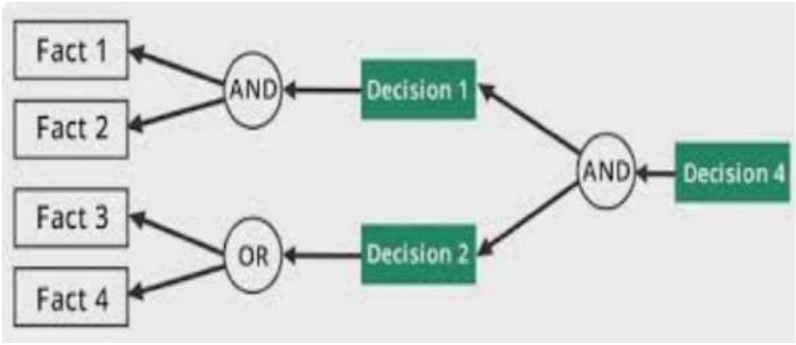


Backward Chaining:

- A backward chaining algorithm is a form of reasoning, which starts with the goal and works backward, chaining through rules to find known facts that support the goal.
- In backward chaining, the goal is broken into sub-goal or sub-goals to prove the facts true.
- It is called a goal-driven approach, as a list of goals decides which rules are selected and used.

Ex:

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.



Forward vs Backward Chaining:

S.No.	Forward Chaining	Backward Chaining
1.	Forward chaining starts from known facts and applies inference rule to extract more data unit it reaches to the goal.	Backward chaining starts from the goal and works backward through inference rules to find the required facts that support the goal.
2.	It is a bottom-up approach	It is a top-down approach
3.	data-driven	goal-driven
4.	breadth-first search strategy.	depth-first search strategy.
5.	Forward chaining tests for all the available rules	Backward chaining only tests for few required rules.
6.	Forward chaining is suitable for the planning, monitoring, control, and interpretation application.	Backward chaining is suitable for diagnostic, prescription, and debugging application.
7.	Forward chaining can generate an infinite number of possible conclusions.	Backward chaining generates a finite number of possible conclusions.
8.	It operates in the forward direction.	It operates in the backward direction.
9.	Forward chaining is aimed for any conclusion.	Backward chaining is only aimed for the required data.

Deductive reasoning:

- **Deduction**
  - Example: "When it rains, the grass gets wet. It rains. Thus, the grass is wet."
    This means in determining the conclusion; it is using rule and its precondition to make a conclusion.
  - Applying a general principle to a special case.
  - Using theory to make predictions
  - Usage: Inference engines, Theorem provers, Planning.

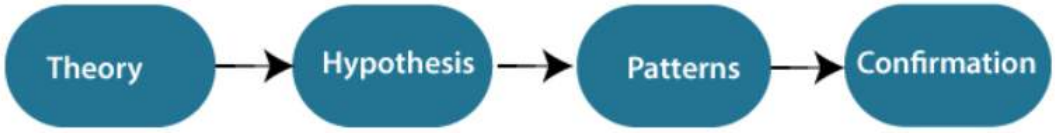
Example:

Premise-1: All the human eats veggies

Premise-2: Suresh is human.

Conclusion: Suresh eats veggies.

The general process of deductive reasoning is given below:



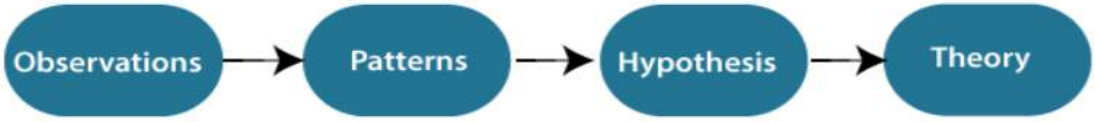
Inductive Reasoning:

- Example: "The grass has been wet every time it has rained. Thus, when it rains, the grass gets wet."
  This means in determining the rule; it is learning the rule after numerous examples of conclusion following the precondition.
- Deriving a general principle from special cases
- From observations to generalizations to knowledge
- Usage: Neural nets, Bayesian nets, Pattern recognition

Example:

Premise: All of the pigeons we have seen in the zoo are white.

Conclusion: Therefore, we can expect all the pigeons to be white.



Abductive reasoning:

- Example: "When it rains, the grass gets wet. The grass is wet, it must have rained."
  Means determining the precondition; it is using the conclusion and the rule to support that the precondition could explain the conclusion.
- Guessing that some general principle can relate a given pattern of cases
- Extract hypotheses to form a tentative theory
- Usage: Knowledge discovery, Statistical methods, Data mining.

Modus Ponens:

The Modus Ponens rule is one of the most important rules of inference, and it states that if P and P → Q is true, then we can infer that Q will be true. It can be represented as:

Notation for Modus ponens: $\frac{P \rightarrow Q, \quad P}{\therefore Q}$

Example:

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
Statement-2: "I am sleepy" ==> P
Conclusion: "I go to bed." ==> Q.
Hence, we can say that, if P→ Q is true and P is true then Q will be true.

Proof by Truth table:

P	Q	$P \rightarrow Q$
0	**0**	**0**
0	**1**	**1**
1	**0**	**0**
1	**1**	**1**

Modus Tollens:

The Modus Tollens rule state that if P→ Q is true and ¬ Q is true, then ¬ P will also true. It can be represented as:

Notation for Modus Tollens: $\frac{P \rightarrow Q, \ \sim Q}{\sim P}$

Statement-1: "If I am sleepy then I go to bed" ==> P→ Q
Statement-2: "I do not go to the bed."==> ~Q
Statement-3: Which infers that "**I am not sleepy**" => ~P

Proof by Truth table:

P	Q	$\sim P$	$\sim Q$	$P \rightarrow Q$
0	**0**	**1**	**1**	**1**
0	**1**	**1**	**0**	**1**
1	**0**	**0**	**1**	**0**
1	**1**	**0**	**0**	**1**

Common Sense Reasoning:

- Common sense reasoning is an informal form of reasoning, which can be gained through experiences.
- Common Sense reasoning simulates the human ability to make presumptions about events which occurs on every day.
- It relies on good judgment rather than exact logic and operates on heuristic knowledge and heuristic rules.

Example:

1. One person can be at one place at a time.
2. If I put my hand in a fire, then it will burn.

The above two statements are the examples of common-sense reasoning which a human mind can easily understand and assume.

Unit 4 (Part 2)

Uncertainty:

- To represent uncertain knowledge, we need uncertain reasoning or probabilistic reasoning.
- For example, we might write A→B, which means if A is true then B is true, but consider a situation where we are not sure about whether A is true or not then we cannot express this statement, this situation is called uncertainty.

Probabilistic reasoning:

We use probability in probabilistic reasoning because it provides a way to handle the uncertainty.

Need of probabilistic reasoning in AI:

1. When there are unpredictable outcomes.
2. When specifications or possibilities of predicates becomes too large to handle.
3. When an unknown error occurs during an experiment.

Probability:

- It is the numerical measure of the likelihood that an event will occur.
- The value of probability always remains between 0 and 1 that represent ideal uncertainties.

$$\textbf{Probability of occurrence} = \frac{\text{Number of desired outcomes}}{\text{Total number of outcomes}}$$

Prior probability: The prior probability of an event is probability computed before observing new information.

Posterior Probability: The probability that is calculated after all evidence or information has taken into account. It is a combination of prior probability and new information.

Conditional probability:

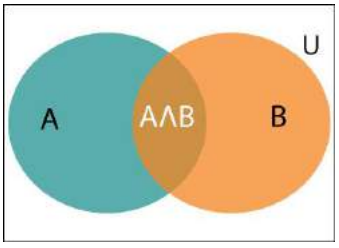
Conditional probability is a probability of occurring an event when another event has already happened.

Ex: "the probability of A under the conditions of B" is represented as,

$$P(A|B) = \frac{P(A \wedge B)}{P(B)}$$

Where $P(A \wedge B)$= Joint probability of a and B

$P(B)$= Marginal probability of B.



Bayes' theorem:

It is a way to calculate the value of P(B|A) with the knowledge of P(A|B).

$$P(A|B) = \frac{P(B|A)\,P(A)}{P(B)} \quad \text{....(a)}$$

The above equation (a) is called as *Bayes' rule or Bayes' theorem*.

- P(A|B) is known as *posterior*, which we need to calculate.
- P(B|A) is called the *likelihood*.
- P(A) is called the *prior probability*.
- P(B) is called *marginal probability*.

Generalized,

$$P(A_i|B) = \frac{P(A_i)*P(B|A_i)}{\sum_{i=1}^{k} P(A_i)*P(B|A_i)}$$

Where A1, A2, A3, ..., An is a set of mutually exclusive and exhaustive events.

Example:

Question: From a standard deck of playing cards, a single card is drawn. The probability that the card is king is 4/52, then calculate posterior probability P(King|Face), which means the drawn face card is a king card.

Solution:

$$P(king|face) = \frac{P(Face|king)*P(King)}{P(Face)} \quad \text{.....(i)}$$

P(king): probability that the card is King= 4/52= 1/13

P(face): probability that a card is a face card= 3/13

P(Face|King): probability of face card when we assume it is a king = 1

Putting all values in equation (i) we will get:

$$P(king|face) = \frac{1*(\frac{1}{13})}{(\frac{3}{13})} = 1/3,$$ **it is a probability that a face card is a king card.**

Applications:

- It is used to calculate the next step of the robot when the already executed step is given.
- Bayes' theorem is helpful in weather forecasting.
- It can solve the Monty Hall problem.

Bayesian Belief Network:

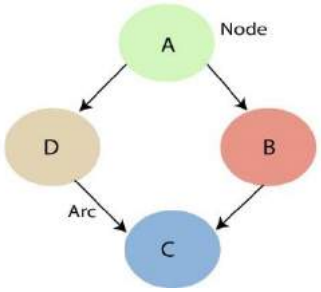
- A Bayesian network is a probabilistic graphical model which represents a set of variables and their conditional dependencies using a directed acyclic graph.
- Bayesian networks are probabilistic, because these networks are built from a probability distribution, and also use probability theory for prediction and anomaly detection.

Bayesian Network consists of two parts:

1. Directed Acyclic Graph.
2. Table of conditional probabilities.

Directed Acyclic Graph:

- Each node corresponds to the random variables.
- Arc or directed arrows represent the causal relationship or conditional probabilities between random variables.
- These links represent that one node directly influence the other node, and if there is no directed link that means that nodes are independent with each other.



- In the above diagram, A, B, C, and D are random variables represented by the nodes of the network graph.
- If we are considering node B, which is connected with node A by a directed arrow, then node A is called the parent of Node B.
- Node C is independent of node A.

Example: (zada nhi smjh me aaye ga)

Harry installed a new burglar alarm at his home to detect burglary. The alarm reliably responds at detecting a burglary but also responds for minor earthquakes. Harry has two neighbours David and Sophia, who have taken a responsibility to inform Harry at work when they hear the alarm. David always calls Harry when he hears the alarm, but sometimes he got confused with the phone ringing and calls at that time too. On the other hand, Sophia likes to listen to high music, so sometimes she misses to hear the alarm. Here we would like to compute the probability of Burglary Alarm.

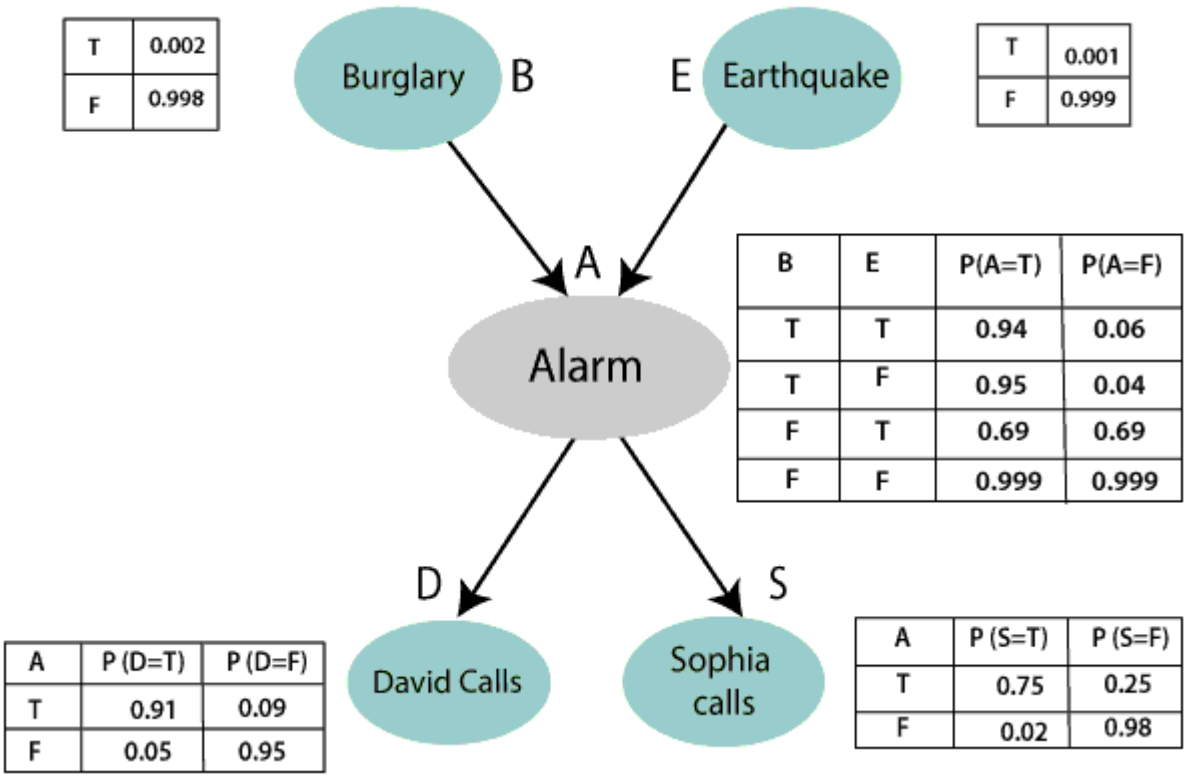
Problem:

Calculate the probability that alarm has sounded, but there is neither a burglary, nor an earthquake occurred, and David and Sophia both called the Harry.

Solution:

List of all events occurring in this network:

- Burglary (B)
- Earthquake(E)
- Alarm(A)
- David Calls(D)
- Sophia calls(S)



T	0.002
F	0.998

T	0.001
F	0.999

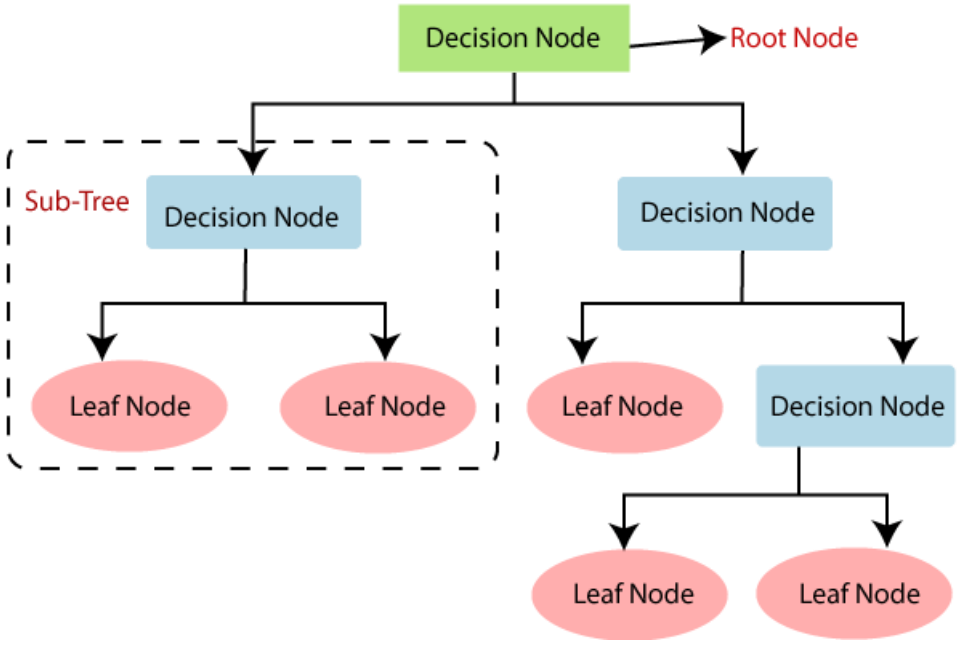
B	E	P(A=T)	P(A=F)
T	T	0.94	0.06
T	F	0.95	0.04
F	T	0.69	0.69
F	F	0.999	0.999

A	P (D=T)	P (D=F)
T	0.91	0.09
F	0.05	0.95

A	P (S=T)	P (S=F)
T	0.75	0.25
F	0.02	0.98

Decision Tree:

- It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions.
- It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome.
- There are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches.



- Root Node: Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into two or more homogeneous sets.
- Leaf Node: Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.
- Splitting: Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.
- Branch/Sub Tree: A tree formed by splitting the tree.
- Pruning: Pruning is the process of removing the unwanted branches from the tree.
- Parent/Child node: The root node of the tree is called the parent node, and other nodes are called the child nodes.

Advantages of the Decision Tree:

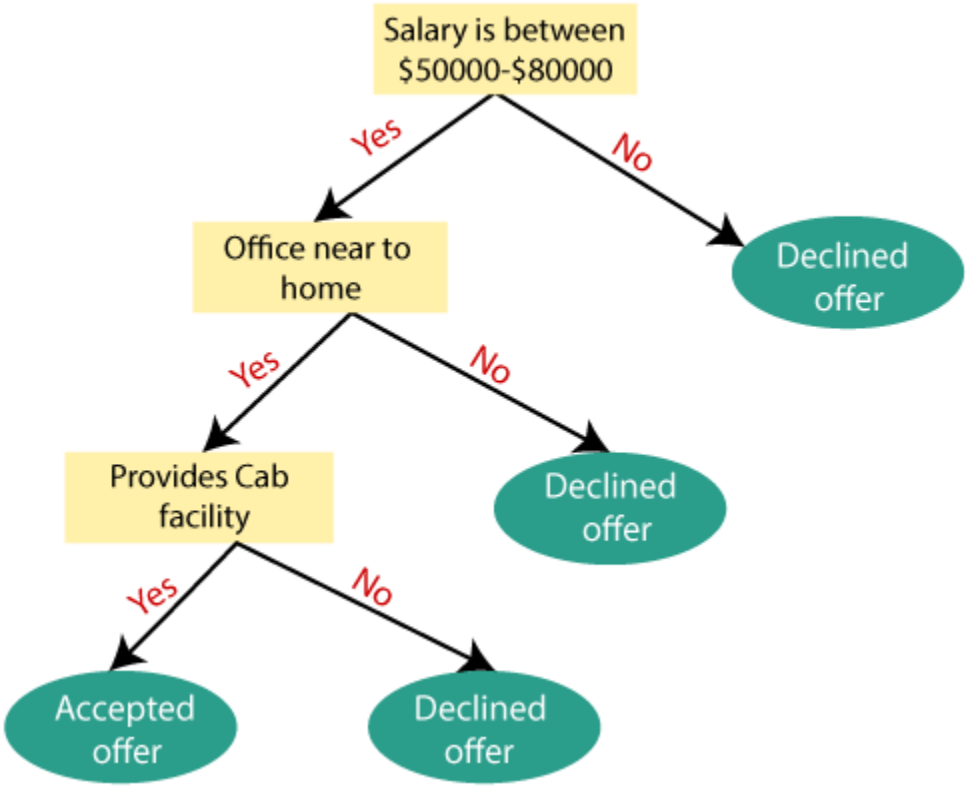
1. It is simple to understand as it follows the same process which a human follow while making any decision in real-life.
2. It can be very useful for solving decision-related problems.
3. It helps to think about all the possible outcomes for a problem.
4. There is less requirement of data cleaning compared to other algorithms.

Disadvantages of the Decision Tree:

1. The decision tree contains lots of layers, which makes it complex.
2. For more class labels, the computational complexity of the decision tree may increase.

Example:

Suppose there is a candidate who has a job offer and wants to decide whether he should accept the offer or Not. So, to solve this problem, the decision tree starts with the root node (Salary attribute by ASM). The root node splits further into the next decision node (distance from the office) and one leaf node based on the corresponding labels. The next decision node further gets split into one decision node (Cab facility) and one leaf node. Finally, the decision node splits into two leaf nodes (Accepted offers and Declined offer).



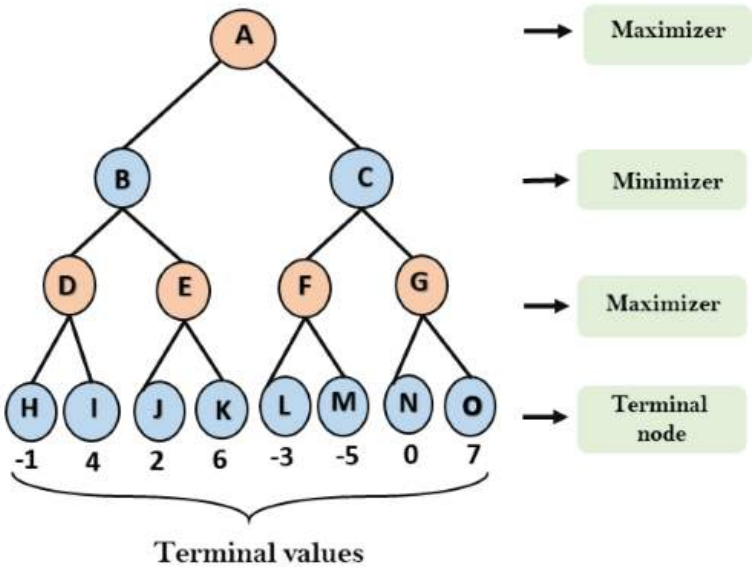
Unit 5

Mini-Max Algorithm:

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory. It provides an optimal move for the player assuming that opponent is also playing optimally.
- In this algorithm two players play the game; one is called MAX and other is called MIN. Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- The minimax algorithm performs a depth-first search algorithm.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs.
- Worst-case initial value for MAX = - infinity and for MIN = +infinity.

Example:

Step 1: let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value = - infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



Step 2: We will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values.

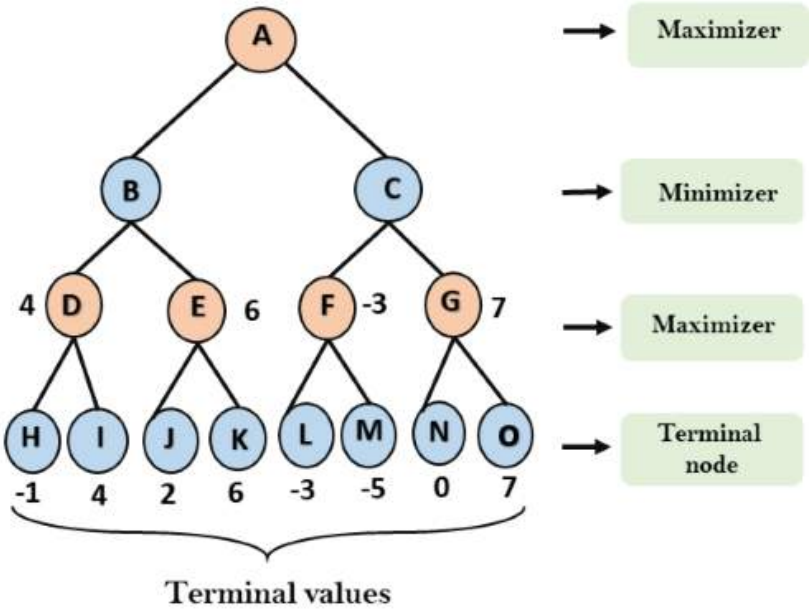
To find the value of a MAX node first we need to find, **max (left child, initial value)** and then **max (max (left child, initial value), right child)**, which is the value of the node.

For node D max (-1, -∞) => max (-1,4) = 4

For Node E max (2, -∞) => max (2, 6) = 6

For Node F max (-3, -∞) => max (-3, -5) = -3

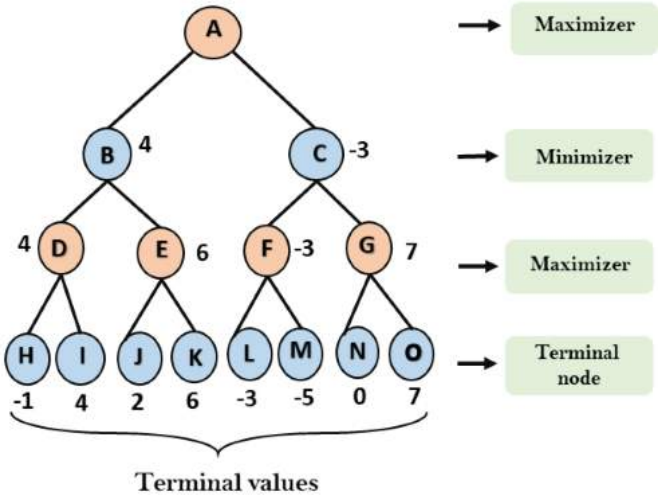
For node G max (0, -∞) => max (0, 7) = 7



Step 3: Now it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the values of the MIN nodes.

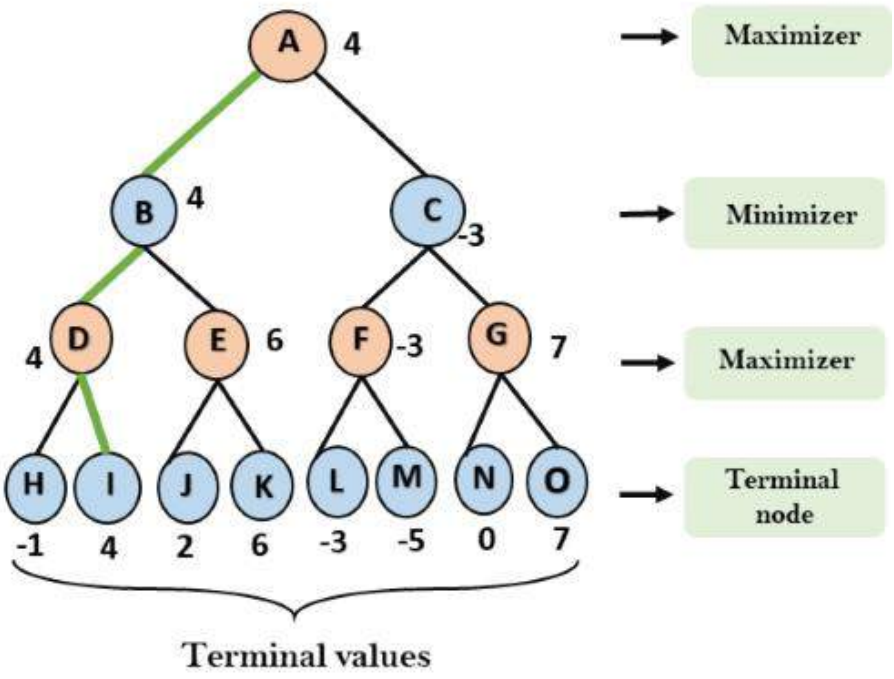
For node B min (4,6) = 4

For node C min (-3, 7) = -3



Step 4: Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node.

For node A max (4, -3) = 4



Properties of Mini-Max algorithm:

1. Complete: Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
2. Optimal: Min-Max algorithm is optimal if both opponents are playing optimally.
3. Time complexity: As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is $O(b^m)$, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
4. Space Complexity: Space complexity of Mini-max algorithm is also similar to DFS which is $O(b^m)$.

Limitation of the minimax Algorithm:

The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc. This type of games has a huge branching factor, and the player has lots of choices to decide.

This limitation of the minimax algorithm can be improved from alpha-beta pruning.

Alpha-Beta Pruning:

- Alpha-beta pruning is a modified version of the minimax algorithm. In this algorithm we don't have to search the whole tree to find the optimal solution.
- 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:
    - 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 -∞.
    - 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 main condition which required for alpha-beta pruning is:

$$\alpha >= \beta$$

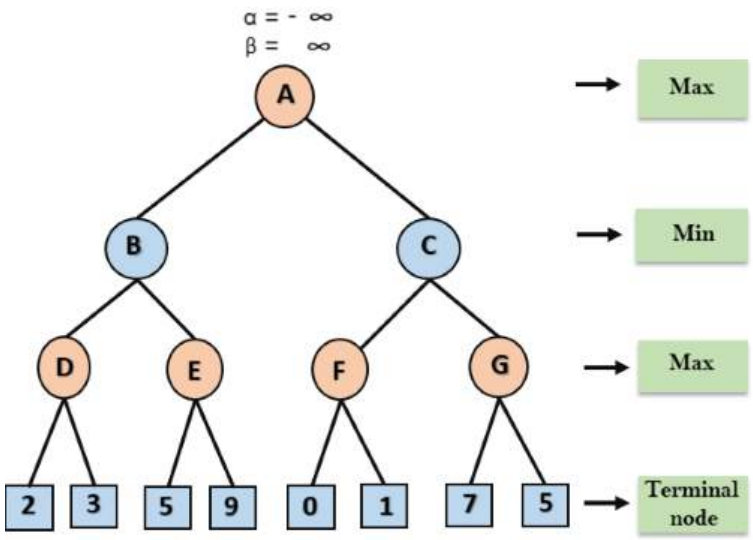
We will have to check this condition at every node. If the condition is satisfied then we will not traverse the sub-tree/child of that node.

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.
- ***Values of alpha and beta will be passed in downward direction only not in the upward direction.***

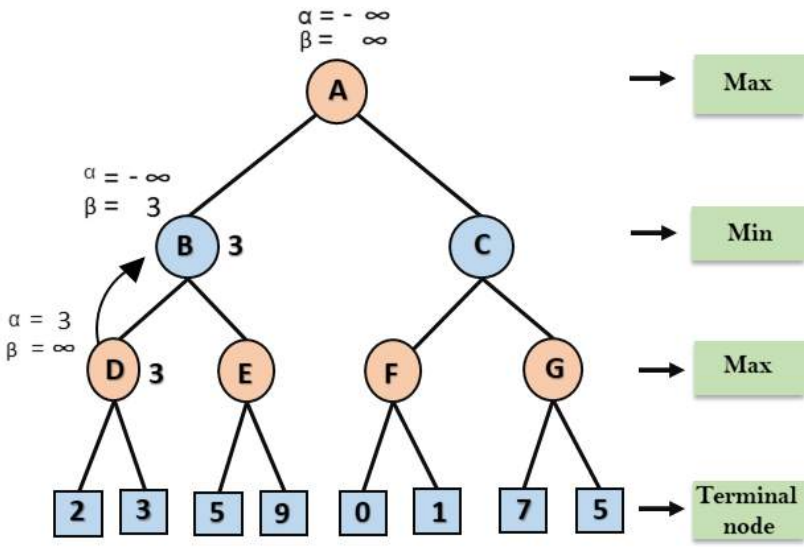
Example:

Step 1: At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



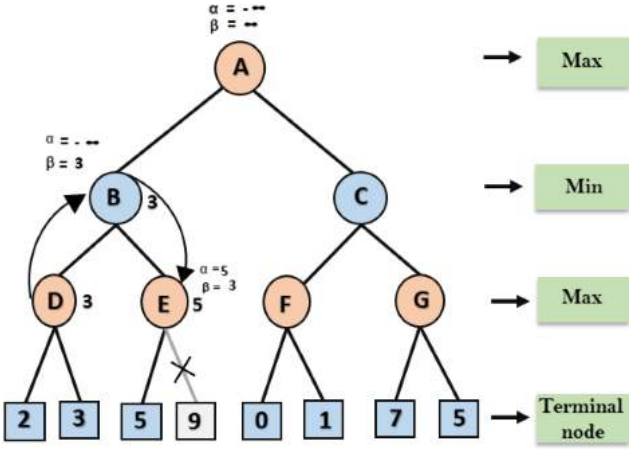
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also be 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.

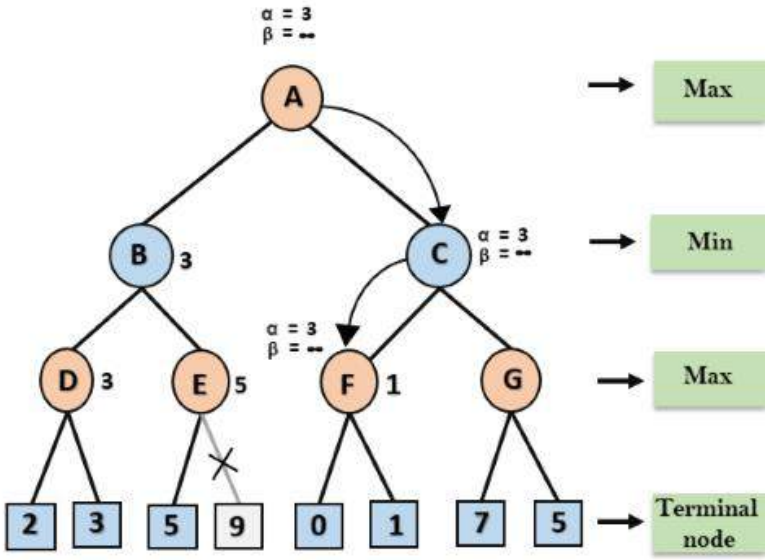
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



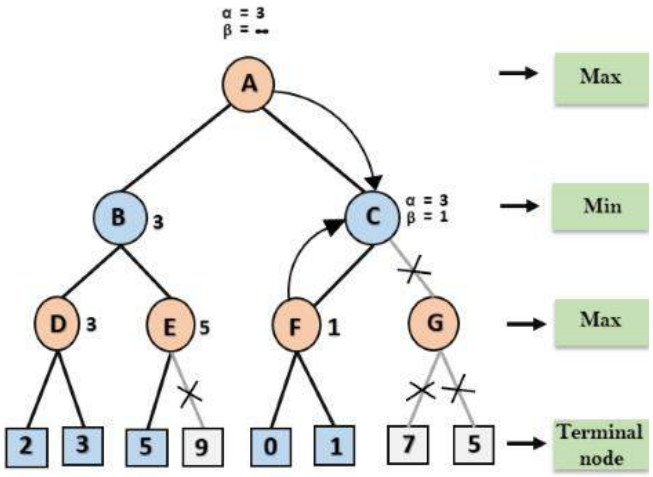
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3) = 3, and β= +∞, these two values are now passed to right successor of A which is Node C.

At node C, α=3 and β= +∞, and the same values will be passed on to node F.

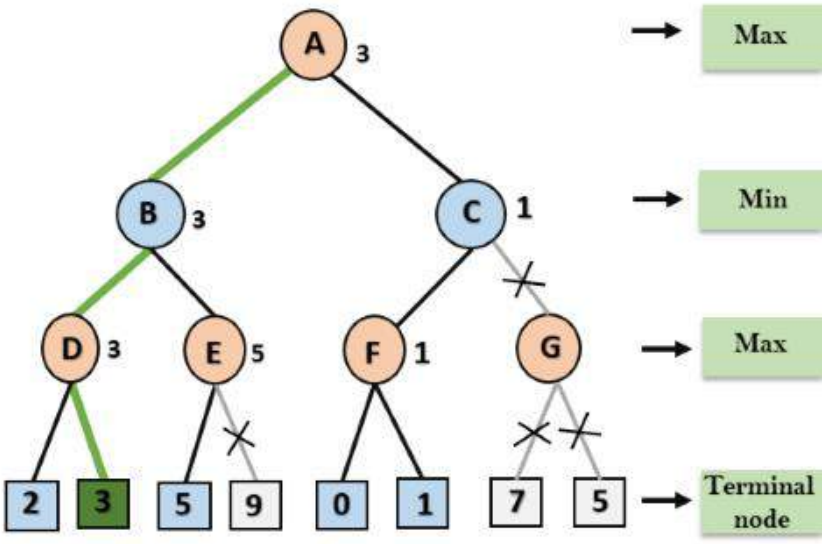
Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



Step 7: Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



Move Ordering in Alpha-Beta pruning:

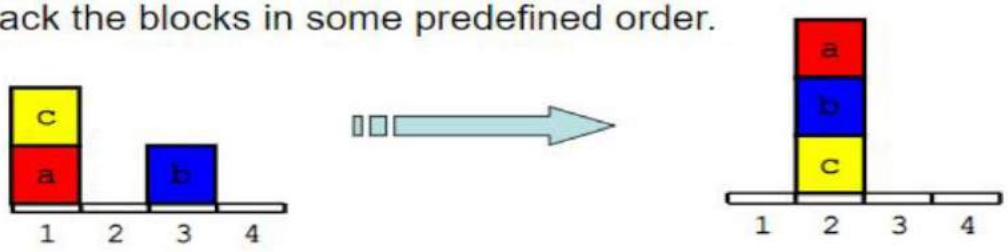
The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.

It can be of two types:

- Worst ordering: In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.
- Ideal ordering: The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.

Blocks World

- Blocks World is THE classic Toy-World problem of AI. It has been used to develop AI systems for vision, learning, language understanding, and planning.
- It consists of a set of solid blocks placed on a table top (or, more often, a simulation of a table top). The task is usually to stack the blocks in some predefined order.



- It lends itself well to the planning domain as the rules, and state of the world can be represented simply and clearly.
- Solving simple problems can often prove surprisingly difficult so it provides a robust testing ground for planning systems.

Example:

```
Plan = [move(b,3,c), move(b,c,3),move(c,a,2), move(a,1,b),
        move(a,b,1), move(b,3,c), move(a,1,b)]
```