

Artificial Intelligence....

UNIT-1

Introduction - AI Problems, foundation of AI, history of AI
intelligent agent: agent, environments, the concept of rationality
the nature of environments, structure of agents, problems.
Solving agents Problem formulation.

UNIT-2

Searching for solution, uniformed search strategies:
Breadth first search, Depth first search, search with partial
information (heuristic search), hill climbing, A^* , A_0^* algorithm
Problem reduction, game playing: adversarial search games,
min-max algorithm, optimal decision in multiplayer games,
problems in game playing, alpha Beta pruning,
evaluation functions.

UNIT-3

Representation of knowledge - knowledge representation
issues, predicate logic - logic programming, schematic nets -
frame and inheritance, constraints programming propagation.
Representing knowledge using rules, rules based deduction
system. reasoning under uncertainty, review of probability
bayes probabilistic interferences Dempsters-Shafer theory.

UNIT-4

Logic Concepts - first order logic. inference in 1st
order logic propositional vs 1st order inference, unification
and left forward chaining & backward chaining resolution,
logic form learning from observation, inductive learning
decision tree, explanation based learning, statistical
learning method, reinforcement learning.

UNIT-5

Expert Systems - Architecture of Expert systems, types of expert systems, knowledge acquisition, meta knowledge, heuristics, typical expert system MICM, DART, XCOM, expert system shells.

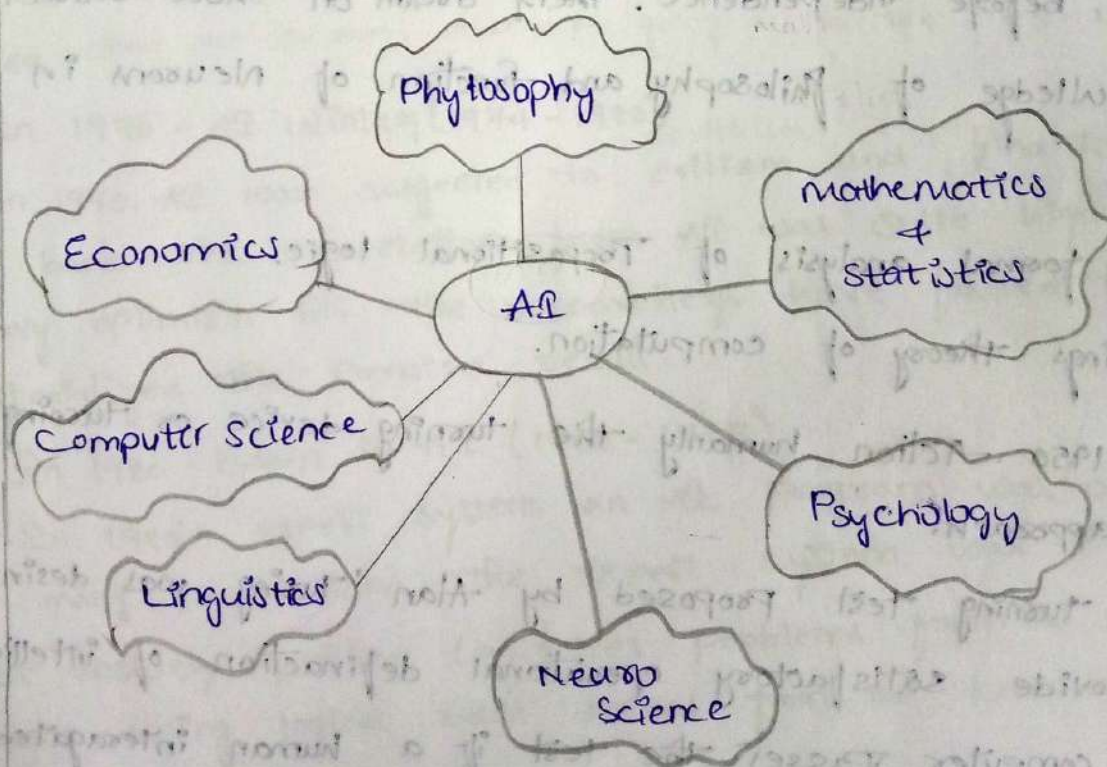
Artificial Intelligence :- Artificial means man-made.

Intelligence means thinking power

"It is a branch of computer science by which we can create intelligent machines which can behave like a human, think like human and able to make decisions".

Foundation of AI :-

The foundation provides the disciplines (subjects) that contributed, ideas, view point, techniques to AI.



* AI is like a recipe with many ingredients - each field add its own flavour

History of AI :-

During the second world war British computer scientist Alan Turing worked to track the enigma code which was used by german. Alan Turing & his team created the Bombe machine that was used to decrypt and encrypt (decipher) Enigma's messages. the enigma and Bombe machine laid the foundation for machine learning.

Alan Turing raised the question can machine think?

① In 1943, first work for in AI

The first work that is now generally recognize as AI was done by Warren McCulloch and Walter Pitts. in 1943.

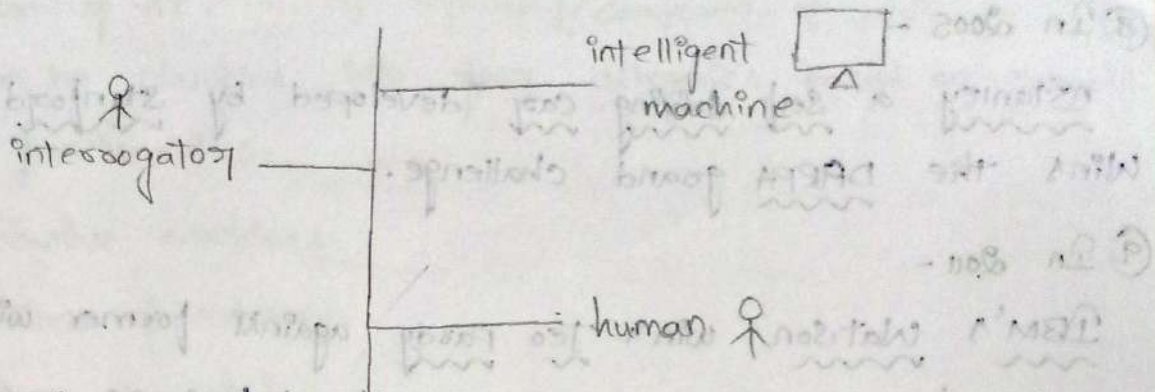
(i.e., before independence. they dream on three sources.

1. Knowledge of philosophy and function of neurons in brain.
2. A formal analysis of Propositional logic.
3. Turing's theory of computation.

② In 1950. Action humanly the turning device or turning test approach.

The turning test proposed by Alan Turing was designed to provide satisfactory operational definition of intelligence.

A computer passes the test if a human interrogated, after causing some return question, cannot tell whether the return response come from a person (or) from a computer.



③ In 1950, Marven Minsky and Dean Edmonds from Harvard built the first neural network computer in 1950 it was called SNARC.

④ In 1956, - Birth of AI

- the term AI was coined in 1956 by a computer scientist John McCarthy in Dartmouth conference organized by him. And three other scientist. He is often known as father of AI. McCarthy in 1950 developed a USP. These years 1952 to 1969. there are so many discoveries going on in the field.

⑤ In 1970 - AI Winter (1974 - 1980)

In 1970, AI was subjected to ^{criticism} and financial set back. The expectations from AI was quite high due to heavy optimism but the researchers have failed to materialized the promise result.

⑥ In 1980 - Boom in AI (1980 - 1987)

In 1980's expert system an AI program was adopted by many corporations. The expert system was a program that answers question (or) solves problems from a particular domain using logical rules derived from the knowledge from expert.

⑦ In 1997 - IBM Deep Blue: Garry

IBM Deep Blue be the world chess champion Garry Kasparov a significant milestone for AI in games.

⑧ In 2005 -

Stanley a self driving car developed by Stanford University wins the DARPA grand challenge.

⑨ In 2011 -

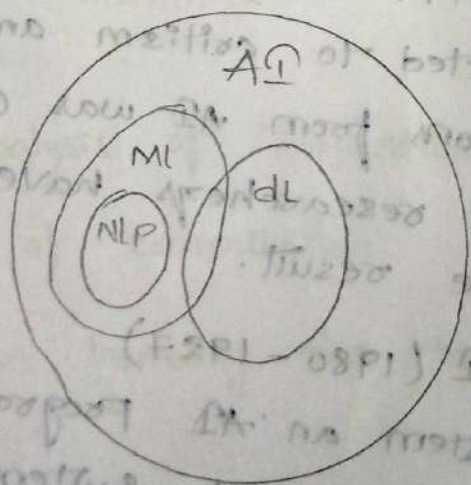
IBM's Watson wins Tes party against former winner showing the capabilities of AI in natural language processing (NLP).

⑩ In 2015 -

A breakthrough in deep learning occurs when a neural network wins image net competition significantly improving the state of image recognition.

⑪ In 2018 -

OpenAI's GPT shows a leap in NLP capability.



ML - Machine Learning

NLP - Natural Language Processing

DL - Deep Learning

Classification of AI :- [part of syllabus] / compound of AI

AI can be classified into four categories based on capability and functionalities.

1. Reactive machines

2. Limited of memory

3. Theory of mind

4. Self aware - AI

1. Reactive machines :- These are the most basic type of AI. They can react to specific situation (or) input but don't have memory or past experience to influence their decision.

Ex:- IBM's deep Blue, the chess playing AI.

2. Limited of memory :- This AI system can use past experience (or) historical data to make current decision. They have a limited memory to store past information and prediction.

Ex:- Self driving cars

3. Theory of mind :- This is a more advance type of AI. Through researches are still working to develop. These systems that would understand emotions, people, beliefs and be able to interact socially.

Ex:- More advance personal assistants that can understand human emotion and react accordingly.

4. Self-aware AI :- This is the most advanced form of AI which is still theoretically and not yet realized.

* These system would have their own consciousness, self awareness, and emotions, these would be smarter and more capable than the human mind.

An Agent and Environment :-

An agent and Environment is anything that can be viewed as perceiving the Environment through sensors and acting upon that Environment through effectors.

Ex:-

1. Human :-

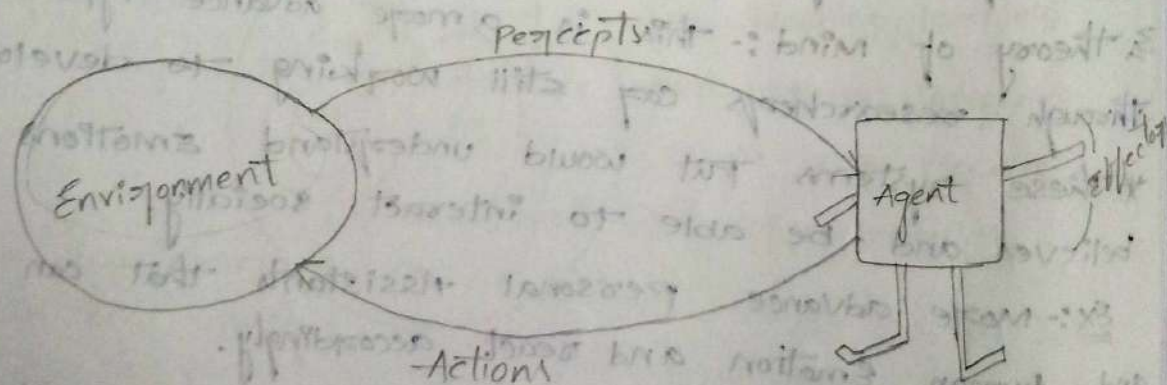
- sensors: Eyes, ears, Nose, skin
- effectors: hands, legs, mouth

2. Robot :-

- sensors: camera, Infrared sensors.
- effectors: Motors.

3. Softwares :-

- sensors: keyboard, voice receiver, data packet receiver.
- effectors: Data Packet transfer, speakers.



Example :-

1. Human is an agent.
2. Self driving car is an agent.
3. A robot is also an agent with cameras and motors.
4. A thermostat detecting room temperature.

Agent the terminology:-

1. Percept:

It is agent perceptual input at a given instance.

2. Percept sequence:

It is the history of all that an agent has perceived till date.

3. Performance measure of Agent:

It is the criteria which determines how successful an agent is.

4. Behaviour of Agent:

It is the action that agent performs after any given sequence of percept.

5. Agent function:

It is a map from the percept sequence to an action (the actions that the agent can carry out using its knowledge about the environment).

Example:-

Vacuum-cleaner

1. This particular world has just two locations square 1 and square 2.
2. The vacuum agent perceives which square it is in and whether there is dirt in the square.
3. It can choose to move left, move right, suck up the dirt (or), do nothing.
4. If the current square is dirty, the clean, it otherwise move to the other square.

Types of Environment

5m
imp We are having 8 types of Environment.

1. Fully observed and Partially observed.

2. Static and Dynamic

3. Discrete and continuous

4. Deterministic and stochastic

5. Single agent and multi agent

6. ~~Periodic~~^{episodic} and sequential

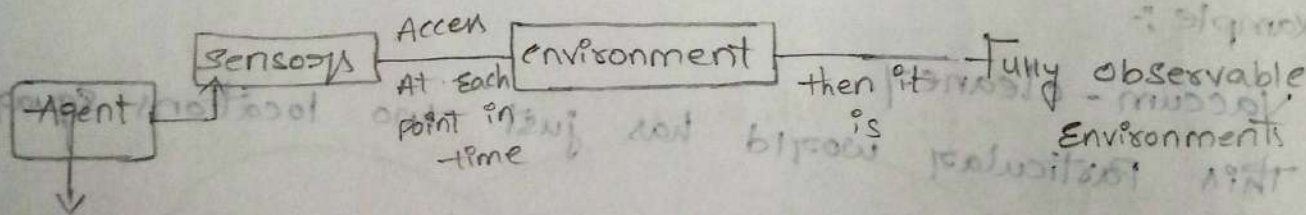
7. Known and unknown

8. Accessible and inaccessible

1) Fully observable and Partially observable:-

Fully observable:-

Every relevant aspects in the environment is visible to the agent is called fully observable.



Partially observable:-

Not all aspects are visible the agent doesn't have complete information about the Environment state because of noisy and inaccurate sensors (or) because of the states are simply missing sensors data is called Partially observable Environment.

Noisy + inaccurate sensors + state are missing = partially observable Environment.

⑤ Single Agent and Multi-Agent :-

Single Agent :- if only one agent is involved in an environment and operating by its self then such an environment is called single agent environment.

Example :-

Maze game

Multi-Agent :- if multiple agents are operating in an environment then such an environment is called multi agent

Example :- Chess and football.

⑥ Flow :-

Known and unknown :-

Known :- It is known environment for result for all actions are known to the agent.

Example :- Navigating a old city, playing video game.

Unknown :- In unknown environment agent need to learn how it works in order to perform an action.

Example :- Navigating a new city, playing a new video game.

⑦ Static and dynamic :-

Static :- The environment doesn't change while an agent is acting.

Example :- Cross word puzzle

Dynamic :- The environment make change more time.

Example :- Roller coaster ride.

⑧ Deterministic and Stochastic :-

Deterministic :- An agent current state and selected action can completely determine the next state of the environment

Example :- Tic Tac Toe.

X	O	O
O	X	O
O	O	X

Stochastic :- A stochastic environment is random in nature and cannot be determined completely by an operating.

Example :- Ludo (or) any dice game.

3) Discrete and Continuous :-

Discrete :- the environment consists of a finite number of action that can be deliberated in the environment to obtain the output.

Example :- Chess.

Continuous :- the environment in which the actions performed cannot be numbered i.e., is not discretized is said to be continuous.

Example :- Self driving car.

** Types of intelligent Agent :- There are 4 types of

intelligent agent which are as follows :-

1) Simple - reflex agent

2) Model based reflex agent

3) Goal based agent

4) Utility based agent.

1) Simple - reflex agent :-

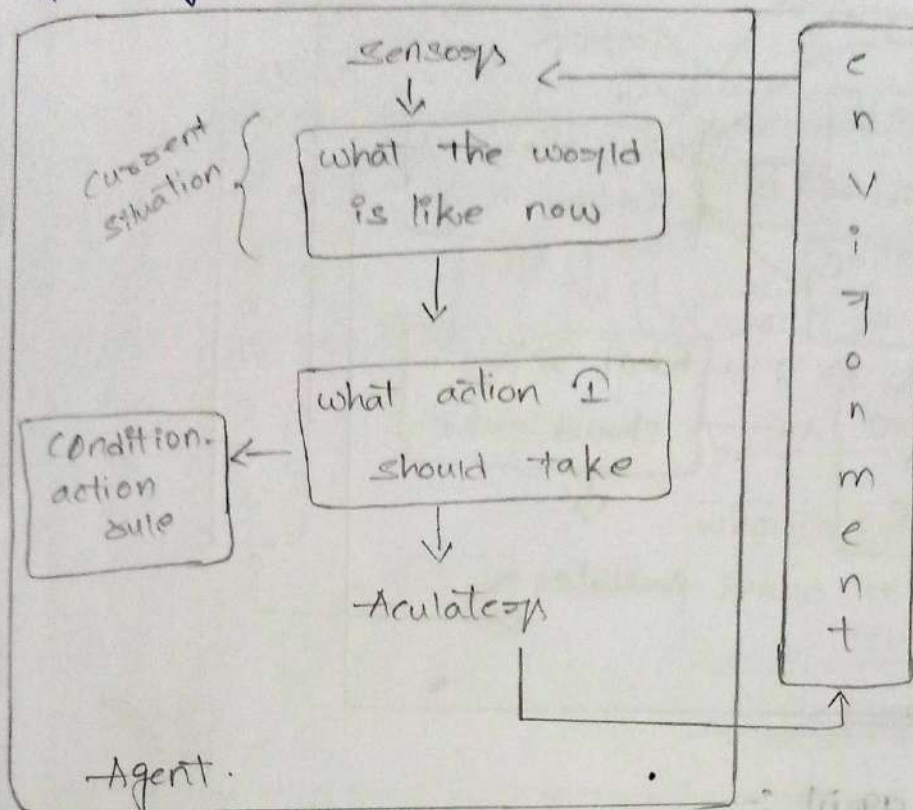
→ Simple - reflex agent act only on the basis of the current percept ignoring the rest of the percept history.

* the agent function is based on the condition action rule "If condition, the action".

* It operates in fully observable environment.

* Infinite loops often unavoidable for simple reflex agent

operating in partially observable environment.



Q. Model-based reflex agent:-

Model-based reflex agent can work in a partially observable environment and track the situation.

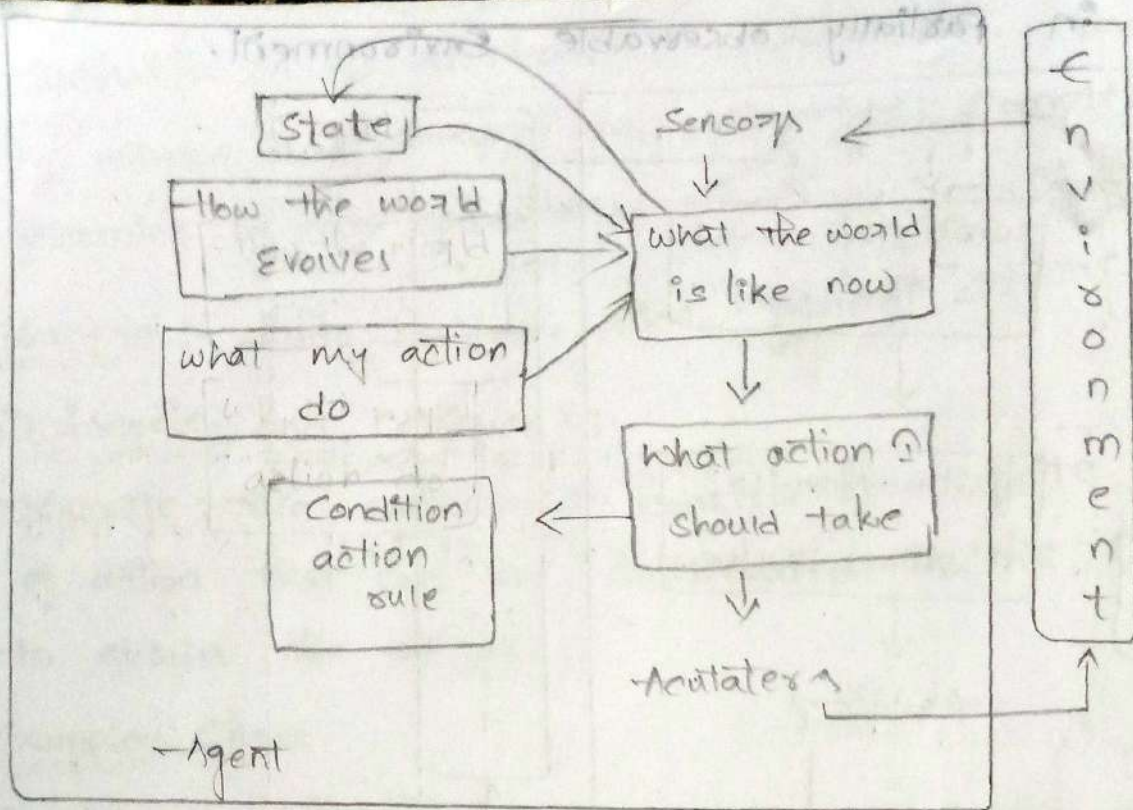
A model based agent has two important factors.

1. Model:- It is knowledge about how things happen in the world. So it is called a model based agent.
2. Internal state:- It is a representation of the current state based on percept history.

These agents have the model which is knowledge of the world and based on the model, they perform actions.

Operating the agent state requires information about:-

1. How the world evolves.
2. How the agent's action affects the world.



3. Goal based agent :-

Goal based agent Expand the capability of model based agent by having the goal information.

The knowledge of the current state Environment is not always sufficient to decide for an agent to what to do.

The agent needs to know the goal which describes desirable situation. They choose an action so that can achieve the goal. The agents may have to consider a long sequence of possible actions before deciding. Whether the goal is achieved or not such considerations of different scenarios are called searching and planning which makes an agent, pro action.

* It is upgraded version of goal based agent.

* They choose actions based on a preference (utility) for each state. Sometime, achieving the desired goal is not enough. we may look for a quicker, safer, trip to reach a destination.

* "Agent Happiness should be taken into considerations, utility describes how happy the agents is".

The concept of Rationality:-

Rationality:-

1. Status of being reasonable sensible and having good sense of judgement
2. Rationality is concerned with expected actions and results depending upon what the agent has perceived.
3. Performing actions with the aim of obtaining useful information is an important part of rationality.
4. A rational agent always performs right actions.
5. Good at Problem Solving.

Agent Action:-

1. Pluck Dark Purple flowers
2. fully Bloomed flowers
3. should have Sweet Aroma.

Rational Agent:-

Rationality of a Agent depends on the following.

1. the performance measure which determine the degree of success.

2. Agent Percepts till now

3. the Agent prior knowledge about the Environment

4. the actions that the agent can carry out.

NOTE :- AI is about creating Rational agent to use for game theory and decision theory for various real world scenarios.

Good Agent : works in all situation.

Bad Agent : works in specific task.

Rationality maximize Expected Performance

1. Omni-Science [Already knows the output]

2. Information Gathering

3. Explanation

4. Learning

The Nature of task environment

Nature : The Surrounding (or) circumstances for which AI system works.

Examples :-

1. Driving cars

2. Playing Games

3. In Industries etc...

Task Environments :-

could be defined as the problem to which rational Agent are designed to provide the solution.

The acronym to describe task environment is PEAS

Specify the task environment. These are 4 elements to take & come which designing the agent to solve a Problem.

PEAS

P - performance:-

measure the source of an Agent behaviour

E - Environment:-

where an agent operates.

A - Actuator:-

Agent act within its environment.

S - Sensors:-

The Agent sensors its environment.

Agent	Performance (4)	Environment (1)	Actuator (3)	Sensor (2)
Vaccum cleaner	cleanness, time taken in cleaning, battery life, number of moves	Room, table, floor	wheels, different brusher, vacuume, extractor.	camera, dirt detecting sensor, cliff sensors, bump sensor, infrared wall sensors.
Self driving car (or) Taxi driver	Safety, time, legal drive, comfort of customer.	Roads, other vehicles, Road sign	steering, accelerators, break, horn	camera, GPS, speedometer.
Interactive English Tutor	maximize student score (or) test.	Set of student, setting agency, previous results of that tutor.	display, exercise, suggestions, corrections.	Keyboard entry.

Properties or Types of Task Environment.

1. fully observable vs partially observable.
2. single agent vs multi Agent.
3. Deterministic vs stochastic
4. ~~Ep~~ episodic vs sequential
5. static vs Dynamic
6. Discrete vs continuous
7. known vs unknown

Structure of an Agent :-

The structure of an agent depends on 4 components :-

1. Agent behaviour
2. Agent Architecture.
3. Agent Program
4. Agent function.

Agent = Architecture + Program.

1. Agent behaviour :-
- Agent behaviour tell what the agent percepts from the environment and what action it takes.

2. Agent Architecture :-
Some set of computing devices (Sensors + Actuators)
(hardware)

3. Agent Program :-
Some function that implements the agents mapping
* The job of AI totally depends upon agent program

4. Agent Function :-
it considers the entire history of percept

Types of Agents :-

1. Simple reflex agent
2. Model base agent
3. goal base agent
4. utility agent

percept previous
percept

Assignment Questions :-

1. Structure of an agent ?
2. Problem solving agent ?

Episodic :-

- * Only the current Percept is required for the action.
- * Every episodic is independent of each other.

Example :-

Fast Picking robot

Sequential :-

- * An agent requires memory of past actions to determine the next best action.
- * The current decision could affect all future decision.

Example :-

Chess

Remarks (Out of Syllabus)

Categories of AI theories :-

- * System that thinks like humans (think like human)
- * System that act like humans (Turing Test)
- * System that think rationally.

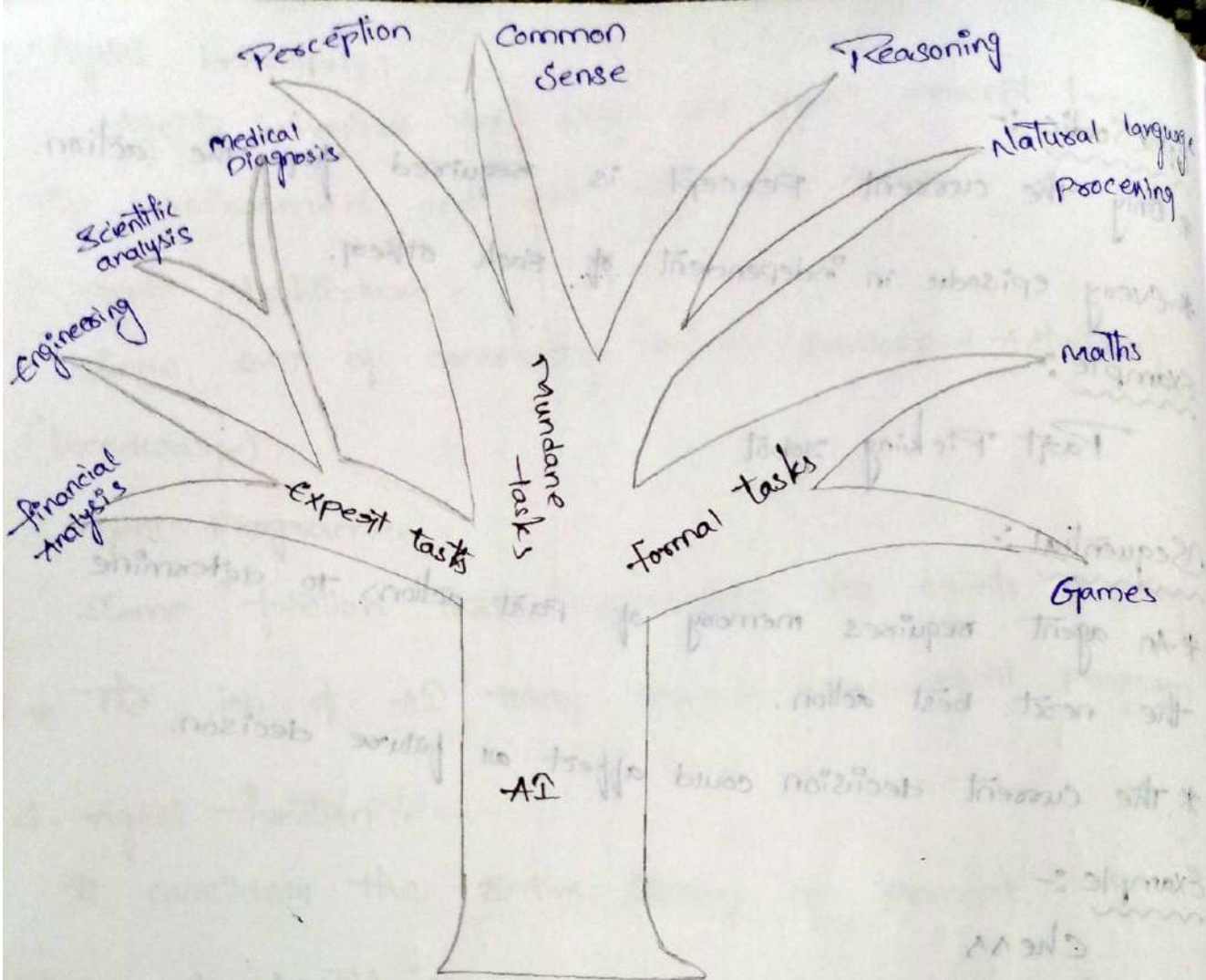
(Ex:- Sachin is an Indian, Sachin is a cricketer)

- * System that act like rationally.

(Ex:- Pulling ones hand out of the hot water)

AI Problems :-

1. Mundane Tasks
2. Formal Tasks
3. Expert Tasks



Task Domains of Artificial Intelligence

Mundane (ordinary) tasks	Formal tasks	Expert tasks
Perception <ul style="list-style-type: none"> • Computer vision • Speech, voice 	<ul style="list-style-type: none"> • Mathematics • Geometry • Logic • Integration and Differentiation. 	<ul style="list-style-type: none"> • Engineering • Fault finding • Manufacturing • Monitoring.
Natural language Processing <ul style="list-style-type: none"> • Understanding • Language Generation • Language Translation. 	Games <ul style="list-style-type: none"> • Go • Chess (Deep Blue) • Checkers 	Scientific Analysis
Common Sense	Verification	Financial Analysis
Reasoning	Theorem Proving	Medical Diagnosis
Planning		Creativity.
Robotics <ul style="list-style-type: none"> • Locomotive 		

Common-place Problems / tasks (Mundane Tasks):

- These are everyday tasks that humans and some animals do naturally and easily, but they are challenging for computers.

1. Recognizing people and objects eg. your phone unlocks by recognizing your face.
2. Communicating through natural language eg. when you ask Siri or Alexa a question and they respond.
3. Navigating around obstacles on the streets eg. self-driving cars avoiding pedestrians and other vehicles.

Formal Problems / tasks (Structured Tasks):-

- Formal tasks are structured problems that require logical thinking and precise algorithms.

1. Solving mathematical equations.

eg: Statement: solve $x + 5 = 10$

Solution: Subtract 5 from both sides to get $x = 5$.

2. Formal logic and theorem proving

eg: Statement: "If it rains, then the ground is wet."

whose logical expression: $P \rightarrow Q$.

Statement: "The sum of the angles in a triangle is 180 degrees."

which is based on: Euclidean geometry axioms.

Expert Problems / tasks:

- These are complex tasks that require specialized skills and knowledge, which only experts can do well.

1. Medical Diagnosis: AI system that help doctors by identifying diseases in medical images like X-rays.

1. Playing Games like chess

-Ex: AI Program Deep Blue that have beaten world champion in chess.

Types of AI [out of syllabus]:-

1. Narrow AI (Narrow AI):

-AI that is designed and trained for a specific task.

It operates under a limited pre-defined range of function.

-Ex:

• Siri and Alexa: these virtual assistants can perform tasks like setting reminders, providing weather updates, and playing music but they can't perform tasks outside their programming.

• Recommendation System: Netflix and Youtube music use narrow AI to suggest movies and music based on user preferences.

• Image Recognition system: Google Photos can categorize image based on visual content.

2. Strong AI (Artificial General Intelligence):

-AI that possesses the ability to understand, learn and apply knowledge across a wide range of tasks at a level comparable to human intelligence.

-Ex:-

• Theoretical concept: currently no true examples of general AI exist. It remains a concept and a goal for future.

• AI development.

• For e.g. Generalist Robot (competent in several different fields of activities).

3. Super Intelligent AI (Superhuman AI):

AI that surpasses human intelligence across all fields, including creativity, general wisdom, and problem-solving.

-Ex:-

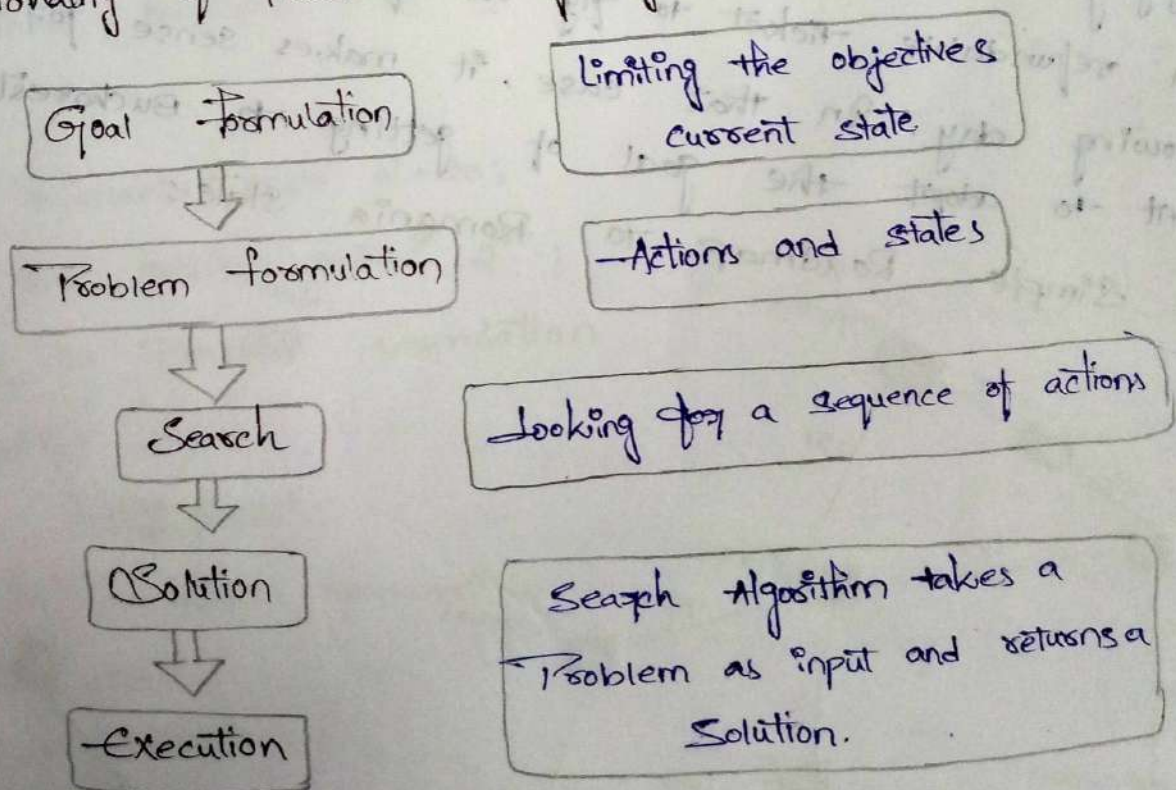
-Hypothetical Scenario: An AI that can outthink and outperform the best human minds in every field, such as Einstein in physics, Shakespeare in literature, and Bezos in business.

Problem Solving Agent :-

* Problem-solving agents are one kind of goal-based agent.

Problem-solving agents use atomic representations, that is; states of the world are considered as wholes, with no internal structure visible to the problem-solving agent.

Functionality of Problem Solving Agent.



Problem Formulation:-

- * what are the possible states of the world relevant for solving the problem.

- * what information is accessible to the agent

- * How can the agent progress from state to state

- * follow the goal-formulation.

Problem Formulation:

- * Initial State

- * Action

- * Transition model

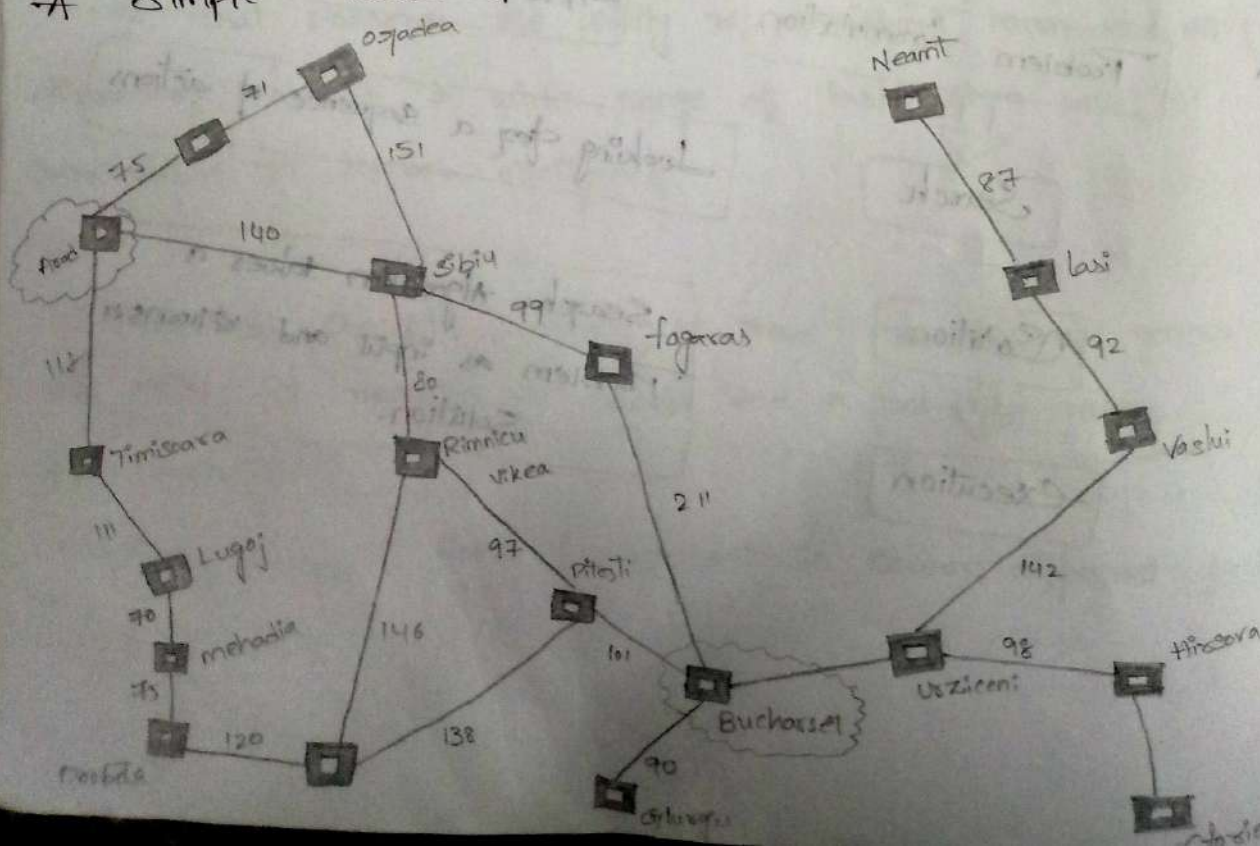
- * Goal test

- * Path cost.

A Real world Scenario (A Touring Agent Problem)

Imagine an agent in the city of Iasi, Romania, enjoying a touring holiday. Now, suppose the agent has a non refundable ticket to fly out of Bucharest the following day. In that case, it makes sense for the agent to adopt the goal of getting to Bucharest.

A Simple Roadmap to Romania state:



Problem Formulation:

* States: A state description specifies the location of each of the eight tiles and the blank in one of the nine squares.

* Initial state: Any state can be designated as the initial state.

* Actions: Left, Right, up or Down

* Transition model: Given a state and actions, this returns the resulting state; for example, if we apply left to the state start in above figure, the resulting state has the 5 and the blank switched.

* Goal test: this checks whether the state matches the goal configuration.

* Path cost: each cost step 1, so the path cost is the number of steps in the path.

Total Solutions:

$$\text{Total solutions} = N * 2^N$$

E.g:-

For Romania state $N=20$;

$$\text{Total solution} = 20 * 2^{20};$$

Steps in Problem Formulation

Initial state

IN (STATE)

Step in problem formulation

Initial state

IN (ARAD)

ACTIONS:

ACTIONS (STATE) \longrightarrow G_0 (STATES)

ACTIONS :

ACTIONS (ARAD) \Rightarrow Go (ZERIND)
Go (SIBIU)
Go (TIMISOARA)

TRANSITION MODEL

RESULT (S, a) \rightarrow IN (x)

TRANSITION MODEL

RESULT (IN (ARAD), Go (ZERIND)) = IN (ZERIND)

Goal test

IN (x) = { IN (g) }

Goal test

IN (x) = { IN (Bucharest) }

Path cost :

C (S, a, x) = P

Path cost :

C (IN (ARAD), Go (ZERIND), IN (ZERIND)) = 75

NOTE :

Solution quality is measured by the path cost function, and an optimal solution has the lowest path cost among all solutions.

Function SIMPLE - problem - solving - agent (percept) return the action.

Persistent : seq, an action sequence, initially empty.

State, some description of the current world state

goal, a goal, initially null

Problem, a problem formulation

State \leftarrow UPDATE - STATE (state, percept)

if seq is empty then

goal \leftarrow FORMULATE - GOAL (state)

Problem \leftarrow FORMULATE - PROBLEM (state, goal)

Seq \leftarrow SEARCH (Problem).

if seq = failure then return a null action

action \leftarrow FIRST (seq)

Seq \leftarrow REST (seq)

return action

Unit - 1
End

Add
& Puzzle

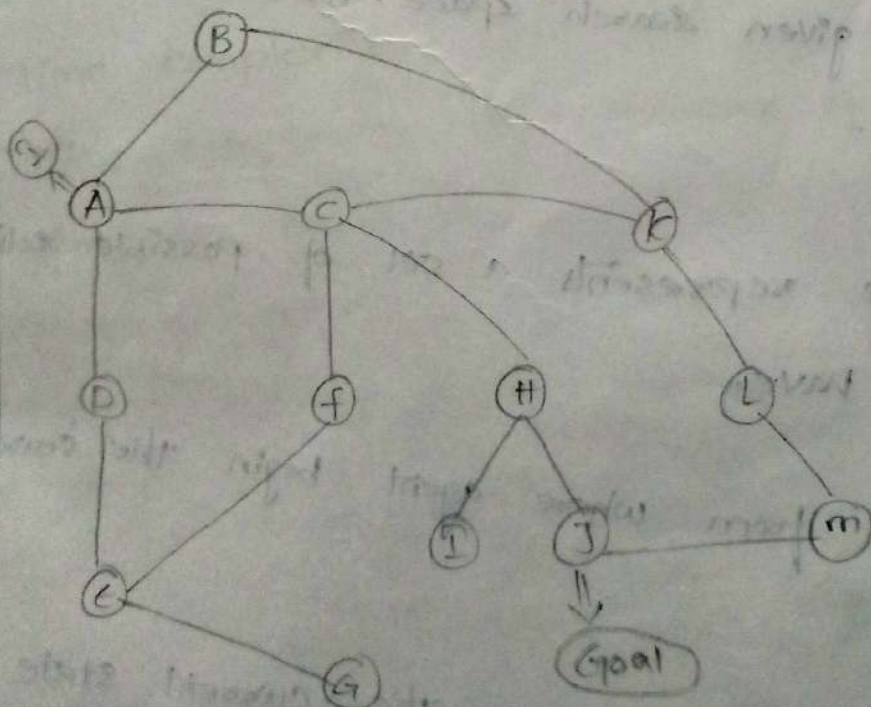
Unit - 2

UNIT - 2

Searching :-

Searching for Solution (or) Problem Solving

Having formulated we now need to solve sum problems. A solution is an actions sequence so search algorithms work by considering various possible action sequence. the possible action sequence starting at the initial state from a search tree with the initial state at the root, the branches are actions and the nodes corresponding to the states. in the state space problem, then, we need to consider taking various actions, we do this by expanding the current state i.e., applying each legal action the current state, there by generating a new set of states, in this states (or) case, we add 3 branches from the node $IN[A]$ leading to 3 new child nodes $IN[B] : IN[C] : IN[D]$.



Steps

II, Action :-

It gives the description of the all available action to the agent.

III, Transition Model :-

→ A description of what is action do can be represented as a transition model.

IV, Path cost :-

It is a function which assign a numeric cost to each path.

Properties of Search Algorithm :-

1. Completeness :-

A search algorithm is said to be complete if it guaran-
tees to return a solution if at least any solution exist
for random input.

2. Optimality :-

It a solution found for an algorithm is guaranter to be
the best solution among all other solutions.

3. Time complexity :-

Time complexity is a measure of time for an algorithm
to complete its task.

4. Space complexity :-

Maximum storage space require at any point during
the search.

Types of Search Algorithm

uniformed
Search

informed
Search

Types of search Algorithm

Uniformed Search

1. No domain knowledge
2. Search without information
3. Time consuming
4. More complexity.
5. BFS
(Breadth first search)
6. DFS
(Depth first search)
7. Uniform cost search
8. Depth limited search
9. Iterative deeping depth first Search
10. Bidirectional Search

informed Search

1. Have domain knowledge and uses the knowledge to find goal.
2. search with information.
3. Quick solution
4. less complexity.
5. Hill climbing
6. BFS (Best first search)
7. A^*
8. A_0^*
9. Branch & Bound
10. Beam Search
11. mean End Analysis.

Uniform Search :- Uniform search is also called as blind Search because this search doesn't contain any domain knowledge such as closeness, the location of the goal. It operates in a brute force where as it only include information about how to traverse the tree and how to identify the leaf and goal node.

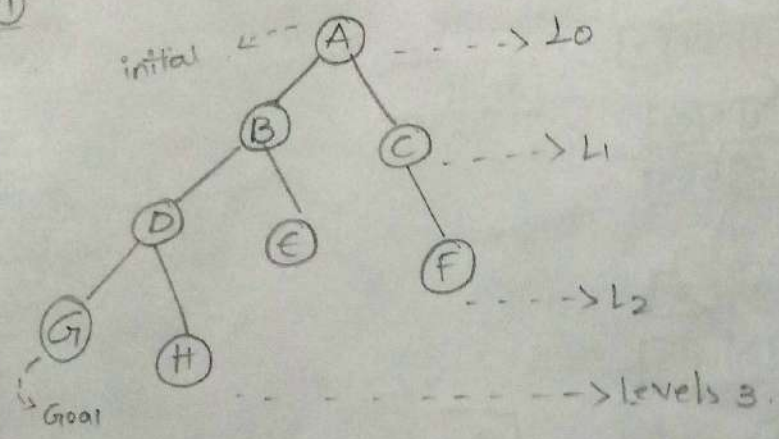
Uniform search is applies a way in which search tree is without any information about the search space, like

initial state operators and test for the goal. it Examine each node of the tree until it achieves the goal node. it can be divided into '6' main types.

- BFS [Breadth first search]
- DFS [Depth first search]
- Hill climbing
- ⇒ BFS [Best first search]
- ⇒ A*
- ⇒ AO*

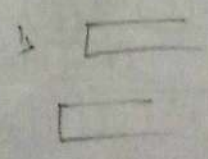
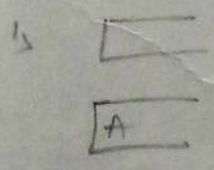
BFS [Breadth first Search]

EX: - ①

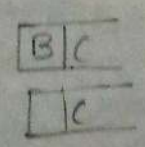
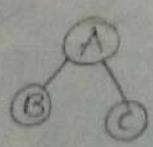
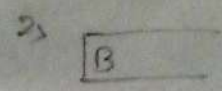


Frontier

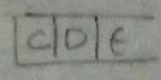
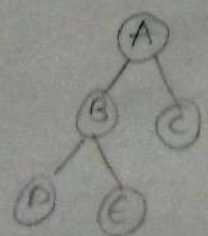
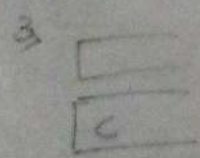
Explored



* Goal test A = G
* Explore



* Goal test B = G
* Explore



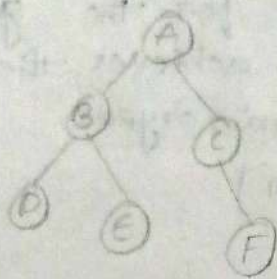
* Goal test C = G
* Explore

4) D

* Goal test

$D = G$

* Explore



D E F

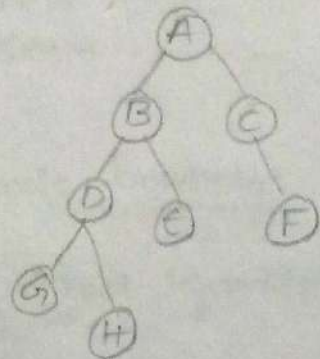
E F

5) E

* Goal test

$E = G$

* Explore



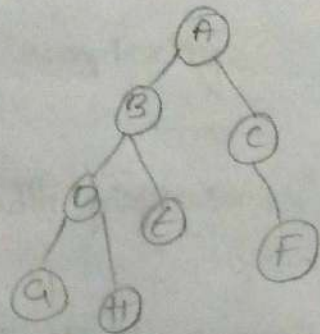
F G H

6) F

* Goal test

$F = G$

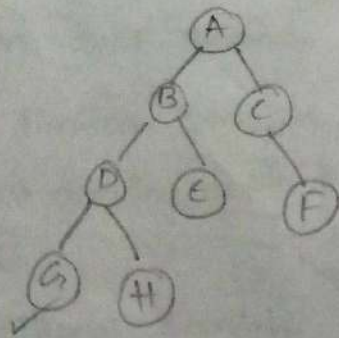
* Explore



G H

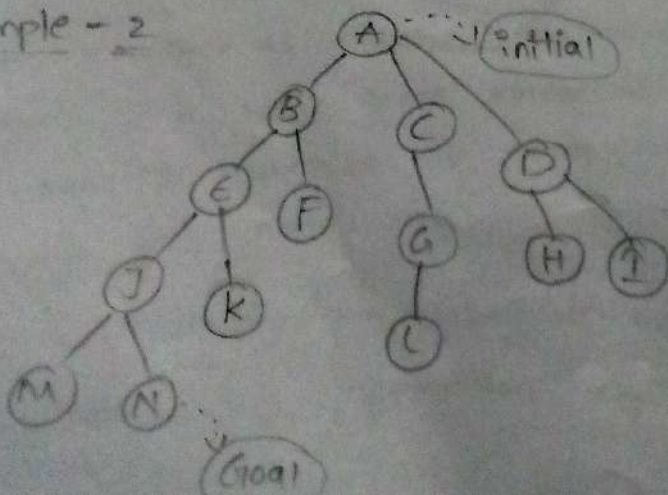
7) G

H



\Rightarrow Now G is our goal test
is successfully completed.

Example - 2



Frontier

Explores

1)

2)

Goal test

$A == N$

Explores all child

3)

Goal test

$(B == N)$

Explores

4)

5)

6)

7)

8)

9)

10)

11)

12)

13)

L	M	N
L	M	N

14)

L

M	N
M	N

15)

M

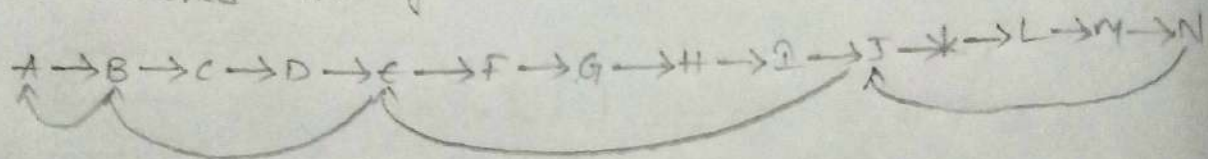
N
N

16)

N

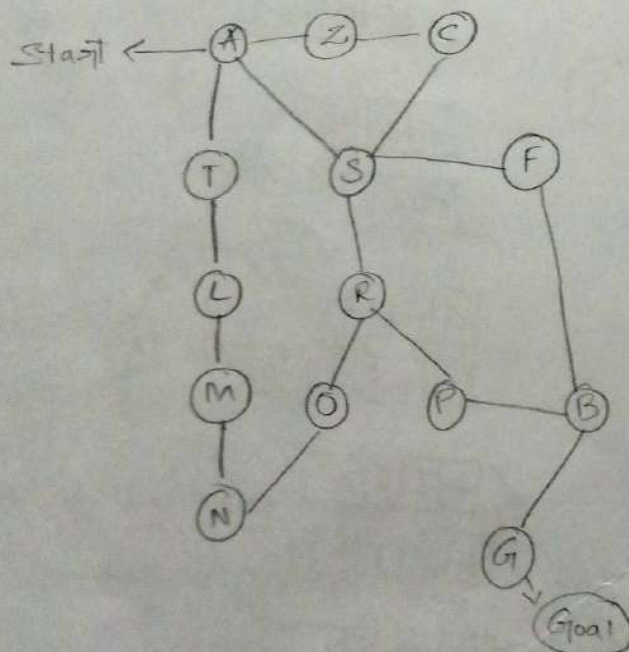
--

We reached the goal N



A → B → E → J → N

Example :- [Imp for exam]



Frontier

1)

A

Goal text

A = G X

--

Explored

S	T	Z
---	---	---

2)

S

$S = G_x$

--

T	Z
---	---

T	Z	F	R
---	---	---	---

Z	C	F	R
---	---	---	---

Z	C	F	R	L
---	---	---	---	---

C	F	R	L
---	---	---	---

C	F	R	L
---	---	---	---

F	R	L
---	---	---

F	R	L
---	---	---

3)

T

$T = G_x$

--

4)

Z

$Z = G_x$

--

5)

C

$C = G_x$

--

6)

F

$F = G_x$

--

7)

R

$R = G_x$

--

8)

L

$L = G_x$

--

9)

B

$B = G_x$

--

10)

O

$O = G_x$

--

L	B
---	---

L	B	O	P
---	---	---	---

B	O	P
---	---	---

B	O	P	M
---	---	---	---

O	P	M
---	---	---

O	P	M	G
---	---	---	---

P	M	G
---	---	---

P	M	G	N
---	---	---	---

11)

P

$P = -G_X$

--

m	G	N
---	---	---

m	G	N
---	---	---

12)

m

$m = -G_X$

--

G	N
---	---

G	N
---	---

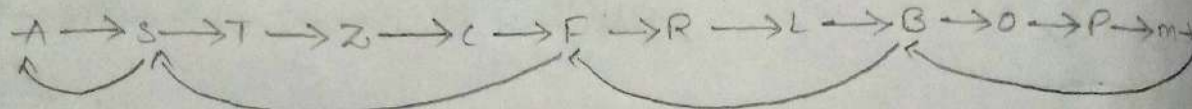
13)

G

$G = -G_X$ ✓

N

the goal is



the shortest path

$A \rightarrow S \rightarrow F \rightarrow B \rightarrow G.$

Algorithm:

function Time search (problem)

frontier = {s}

explores = { }

loop:

If frontier is empty return false

path = remove-choice (frontier)

s = path end

If s is goal return path

add s to the explores

for a in actions

add a nodes of result (s, a)

1. Breadth-first Search:

* Breadth-first search is the most common search strategy for traversing a tree or graph. This algorithm searches breadth-wise in a tree or graph, so it is called breadth-first search.

* BFS algorithm starts searching from the root node of the tree and expands all successor nodes at the current level before moving to nodes of next level.

* The breadth-first search algorithm is an example of a general-graph search algorithm.

* Breadth-first search implemented using FIFO queue data structure.

Advantages:-

- BFS will provide a solution if any solution exists.
- 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:-

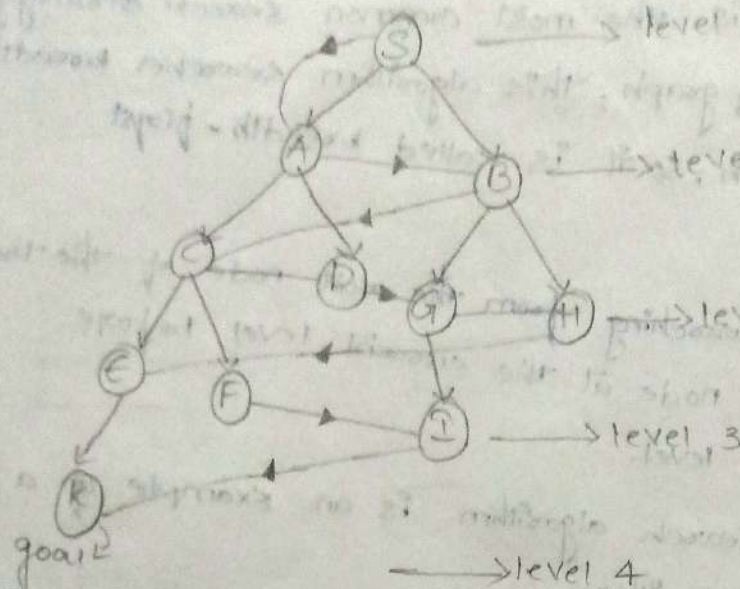
- It requires lots of memory since each level of the tree must be saved into memory to expand the next level.
- BFS needs lots of time if the solution is far away from the root node.

Example:-

In the below tree structure, we have shown the traversing of the tree using BFS algorithm from the root node S to goal node K. BFS search algorithm traverses in layers, so it will follow the path which is shown by the dotted arrow, and the traversed path will be:

S → A → B → C → D → G → H → F → E → I → K.

Breadth - First Search



Time complexity :-

Time Complexity of BFS algorithm can be obtained by the number of nodes traversed in BFS until the shallowest Node, where the d = depth of shallowest solution and b is a node at every state.

$$T(b) = 1 + b^2 + b^3 + \dots + b^d = 0 \quad (b^d)$$

ASpace complexity :-

Space complexity of BFS algorithm is given by the memory size of frontier which is $O(b^d)$.

Completeness:

BFS is complete, which means if the shallowest goal node is at some finite depth, then BFS will find a solution.

Optimality:

BFS is optimal if path cost is a non-decreasing function of the depth of the node.

2. Depth - first Search

* Depth - first search is a recursive algorithm for traversing a tree or graph data structure.

* It is called the depth - first search because it starts from the root node and follows each path to its greatest depth node before moving to the next path.

* DFS uses a stack data structure for its implementation.

* The process of the DFS algorithm is similar to the BFS algorithm.

Advantages :-

⇒ DFS requires very less memory as it only needs to store a stack of the node on the path from root node to the current node.

⇒ It takes less time to search to the goal node than BFS algorithm (if it traverses in the right path).

Disadvantages :-

⇒ * There is the possibility that many states keep re-occurring, and there is no guarantee of finding the solution.

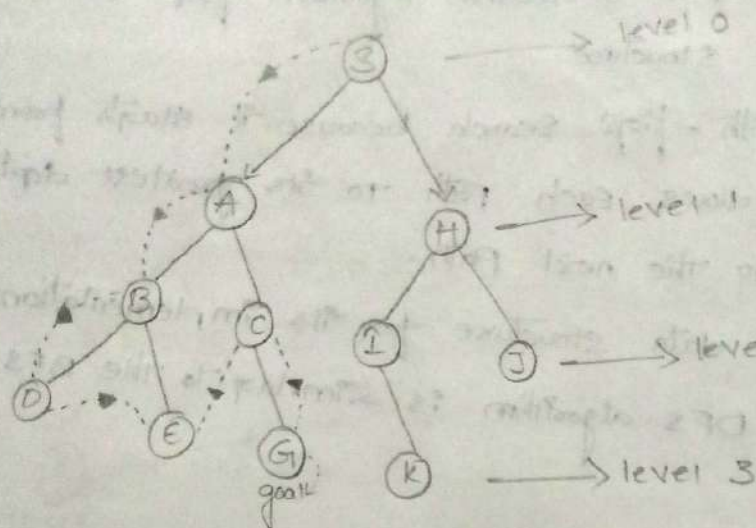
⇒ * DFS algorithm goes for deep down searching and sometime it may go to the infinite loop.

Example :-

In the below search tree, we have shown the flow of depth - first search, and it will follow the order as:

Root node ---> Left node ---> Right node.

Depth first Search



It will start searching from root node S. and traverse A, then B, then D and E, after traversing E, it will backtrack the tree as E has no other successor and still goal node is not found. After backtracking it will traverse node C and then G, and here it will terminate as it found goal node.

Completeness: DFS Search algorithm is complete within finite state space as it will expand every node within a limited search tree.

Time Complexity: Time complexity of DFS will be equivalent to the node traversed by the algorithm. It is given by:

$$T(n) = 1 + n^2 + n^3 + \dots + n^m = O(n^m)$$

Where,

m = maximum depth of any node and this can be much larger than d (shallowest solution depth)

Space complexity: DFS algorithm needs to store only single path from the root node, hence space complexity of DFS is equivalent to the size of the fringe set, which is $O(bm)$.

Optimal :- DFS search algorithm is non-optimal, as it may generate a large number of steps or high cost to reach to the goal node.

INFORMED SEARCH (Heuristic Search)

- * Informed search algorithm use domain knowledge.
- * In an informed search, problem information is available which can guide the search. Informed search strategies can find a solution more efficiently than an uninformed search strategy.
- * Informed search is also called a Heuristic Search.
- * A heuristic is a way which might not always be guaranteed for best solutions but guaranteed to find a good solution in reasonable time.

Heuristic function :-

Heuristic is a function which is used in informed search, and it finds the most promising path. It takes the current state of the agent as its input and produces the estimation of how close agent is from the goal. The heuristic method, however, might not always give the best solution, but it is guaranteed to find a good solution in reasonable time. Heuristic function estimates how close a state is to the goal. It is represented by $h(n)$, and it calculates the cost of an optimal path between the pair of states. The value of the heuristic function is always positive or positive.

Admissibility of the heuristic function is given as:

$$1. h(n) \leq h^*(n)$$

Hence

$h(n)$ is heuristic cost, and

$h^*(n)$ is the estimated cost.

Hence heuristic cost should be less than or equal to the estimated cost.

True Heuristic Search:

True heuristic search is the simplest form of heuristic search algorithm. 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 been expanded and in OPEN list, it places nodes which have yet not been expanded.

On each iteration, each node n with the lowest heuristic value is expanded and generates all its successors and n is placed to the closed list. The algorithm continues until a goal state is found.

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

- * Best first Search Algorithm (Greedy Search)

- * A* Search Algorithm.

1. Best first Search Algorithm (Greedy Search):-

Greedy best-first search algorithm always selects the path which appears best at the 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.,

$$f(n) = g(n).$$

where, $h(n)$ = estimated cost from node n to the goal

the greedy best first algorithm is implemented by the Priority queue.

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 place 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 algorithm.

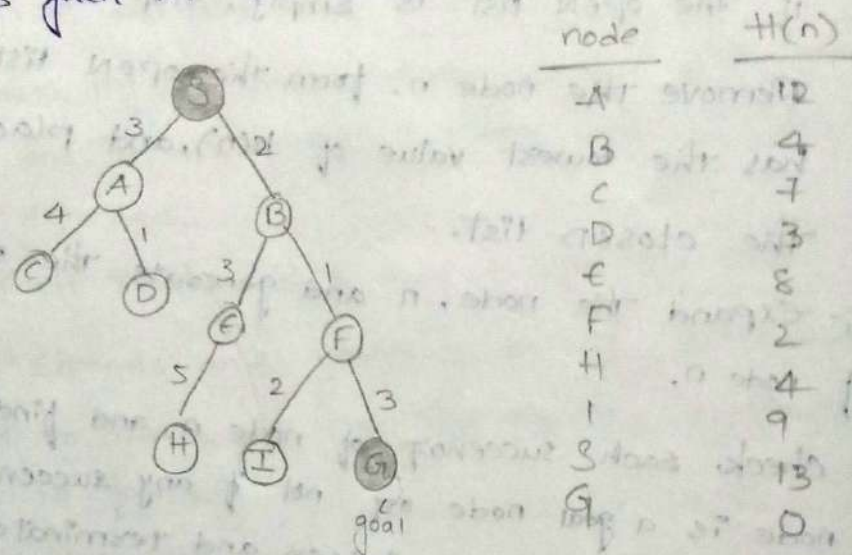
Disadvantages :-

- ⇒ it can behave as an unguided depth-first search in the worst case scenario.
- ⇒ it can get stuck in a loop as DFS.

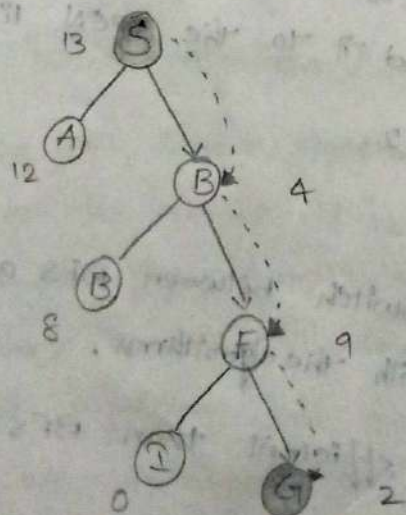
→ this algorithm is not optimal.

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.



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 $S \rightarrow B \rightarrow F \rightarrow G$.

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 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.

Q. A* Search algorithm:-

A* Search is the most commonly known form of best-first search. It uses heuristic function $h(n)$, and cost of search the node n from the start state $g(n)$. It has combined features of UCS and greedy best-first search, by which it solves the problem efficiently. A* search algorithm finds the shortest path through the search 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 close costs as following, and this sum is called as a fitness number.

$$f(n) = g(n) + h(n)$$

estimated cost
of the cheap.
set solution.

cost to reach
node n from
start state.

cost to reach
from node n
to goal node.

Algorithm of A^* Search :-

Step 1 :- 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 :-

$\Rightarrow A^*$ search algorithm is the best algorithm than other search algorithms.

$\Rightarrow A^*$ search algorithm is optimal and complete.

\Rightarrow This algorithm can solve very complex problems.

Disadvantages :-

\Rightarrow it does not always produce the shortest path as it mostly based on heuristics and approximation.

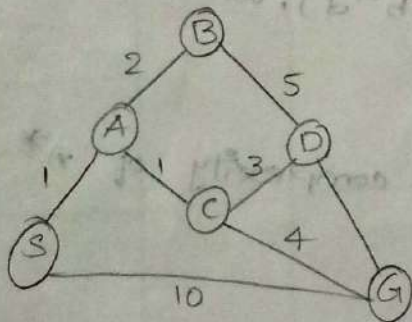
\Rightarrow A^* search algorithm has some complexity issues.

the main drawback of A^* is memory requirement as it keeps all generated nodes as the memory, so it is not practical for various large-scale problems.

Example :-

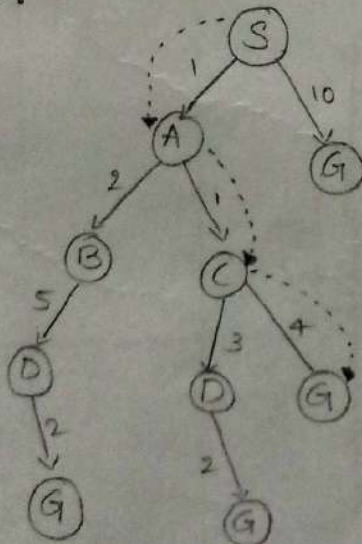
In this example, we will traverse the given graph using the A^* algorithm. The heuristic value of all states is given in the below table so we will calculate the $f(n)$ of each state using the formula $f(n) = g(n) + h(n)$, where $g(n)$ is the cost to reach any node from start state.

Hence we will use open and closed list.



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

Solution :-



Initialization : $\{(S, 5)\}$

Iteration 1 : $\{(S \rightarrow A, 4), (S \rightarrow G, 10)\}$

Iteration 2 : $\{(S \rightarrow A \rightarrow C, 4), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$

Iteration 3 : $\{(S \rightarrow A \rightarrow C \rightarrow G, 6), (S \rightarrow A \rightarrow C \rightarrow D, 11), (S \rightarrow A \rightarrow B, 7), (S \rightarrow G, 10)\}$

Iteration 4 : will give the final result, as $S \rightarrow A \rightarrow C \rightarrow G$

it provides the optimal path with cost 6.

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 $O(b^d)$.

Heuristic Search example :-

7	2	4
5	-	6
8	3	1

Start state

-	1	2
3	4	5
6	7	8

Goal state

$$h_1 = 8$$

$$h_2 = 3 + 1 + 2 + 2 + 2 + 3 + 3 + 0$$

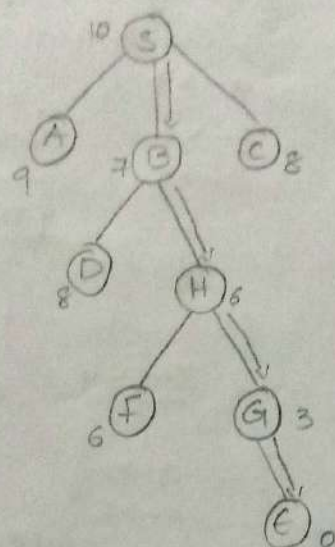
$$= 18$$

$$h_1 + h_2$$

$$8 + 18$$

$$= 26$$

Best First Search



open

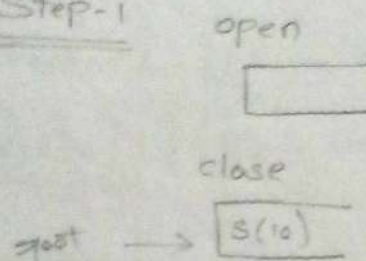
closed

Node	H(n)	Node	Parent
S	10		
B	7	S	
C	8		
A	9		
H	6	B	S
C	8		
D	8		
A	9		
G	3	H	B
F	6		
C	8		
D	8		
A	9		
E	0	G	H
F	6		
C	8		
D	8		
A	9		
		E	G

S → B → H → G → E

Method - 2

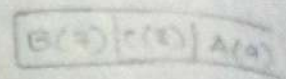
Step-1



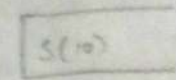
Step-2

A \rightarrow 7
B \rightarrow 8
C \rightarrow 8

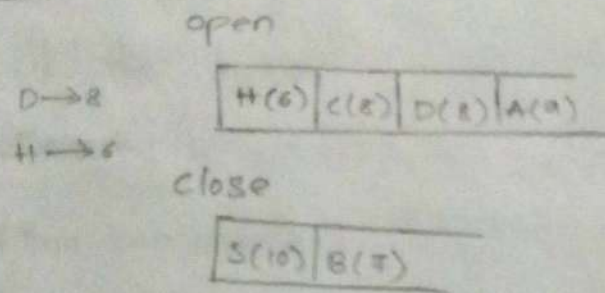
open



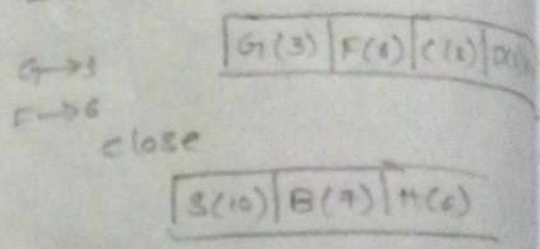
close



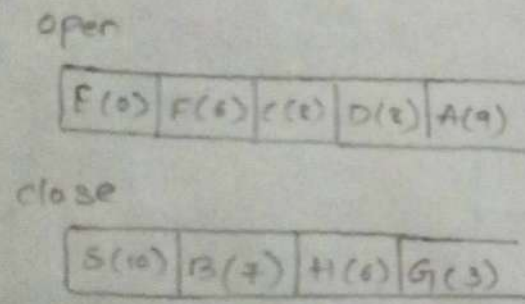
Step-3



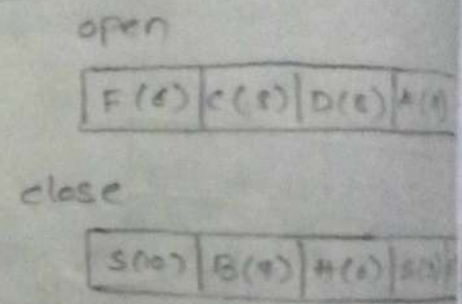
Step-4



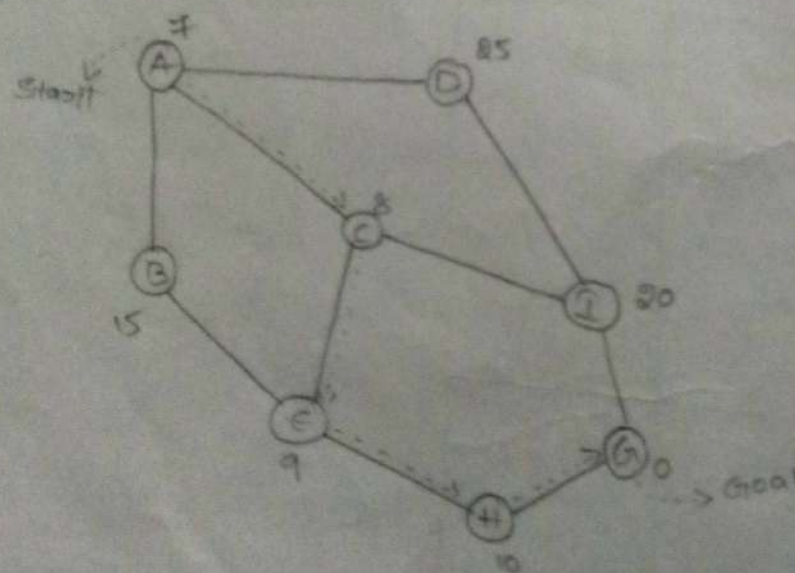
Step-5



Step-6



Example - 2



Step-1

$A \rightarrow c \rightarrow e \rightarrow H \rightarrow G$

open

close

A(7)

Step-2

open

c(8) B(15) D(25)

close

A(7)

Step-3

open

e(9) B(15) I(20) D(25)

close

A(7) c(8)

Step-4

open

H(10) B(15) I(20) D(25)

close

A(7) c(8) e(9)

Step-5

open

G(10) B(15) I(20) D(25)

close

A(7) c(8) e(9) H(10)

Step-6

open

B(15) I(20) D(25)

close

A(7) c(8) e(9) H(10) G(10)

Best first Search:-

- * DFS is good because it allows a solution to be found with the least number of nodes expanded.
- * Completing all branches having to be expanded.
- * BFS is good because it does not get trapped on deep path.
- * Best first search combines both BFS & DFS.
- * BFS search uses a heuristic function which is called an evaluation function and indicate how far the node is from the goal.

Algorithm :-

- 1) create 2 empty lists: OPEN and CLOSED
- 2) start from the initial node (say N) and put it in the 'ordered' OPEN list
- 3) Repeat the next steps until the Goal node is reached or OPEN list is empty.
 - i, select the best node (say N) in the OPEN list and move it to the closed list. Also, capture the information of the Parent node.
 - ii, If N is a Goal node then, exit the loop returning the True the solution can be found by backtracking the Path.
 - iii, If N is not the Goal node, expand node N generate the "immediate" next nodes linked to node N and add all those to the OPEN list.
 - iv, Reorder the nodes in the OPEN list is ascending order.

A* algorithm Example :-

$$h(n) = A \rightarrow 5$$

$$B \rightarrow 6$$

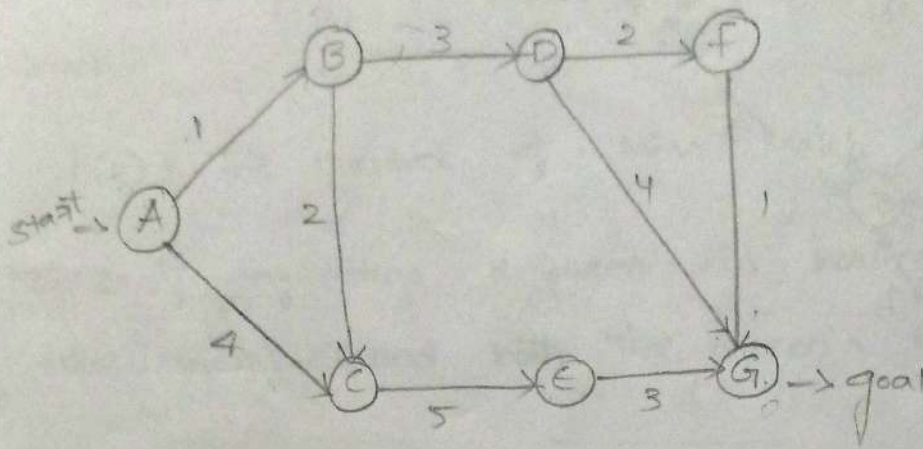
$$C \rightarrow 4$$

$$D \rightarrow 3$$

$$E \rightarrow 3$$

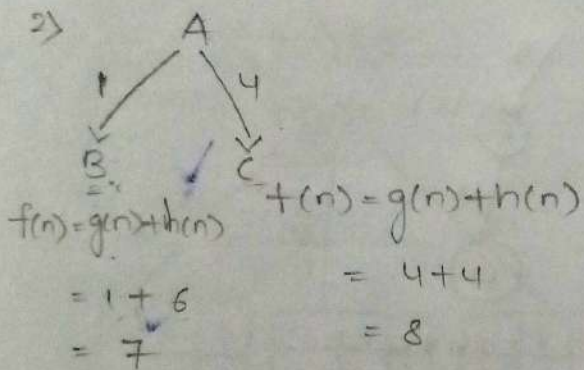
$$F \rightarrow 1$$

$$G \rightarrow 0$$

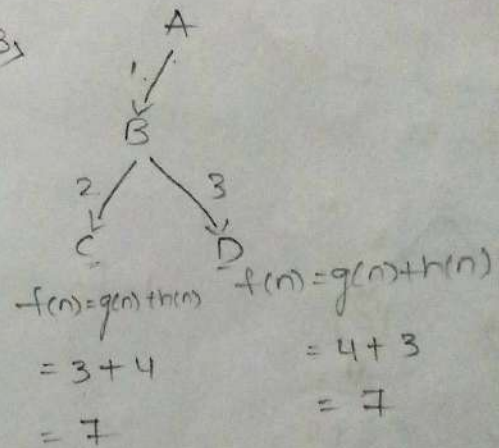


1) $A \neq \text{Goal}$

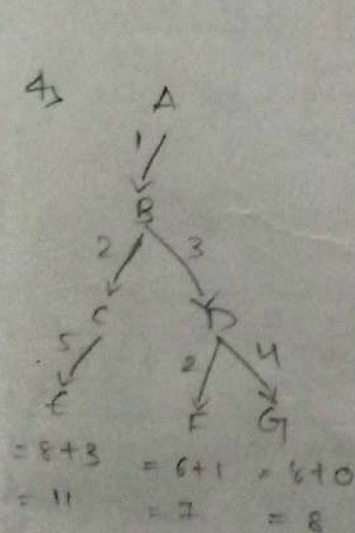
2)



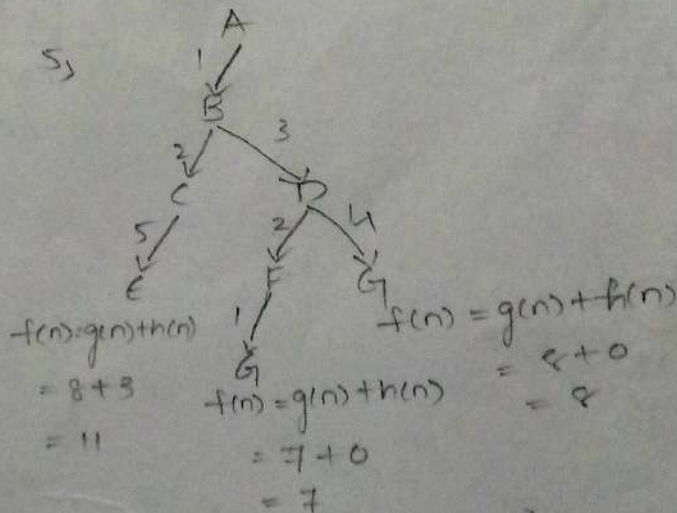
3)



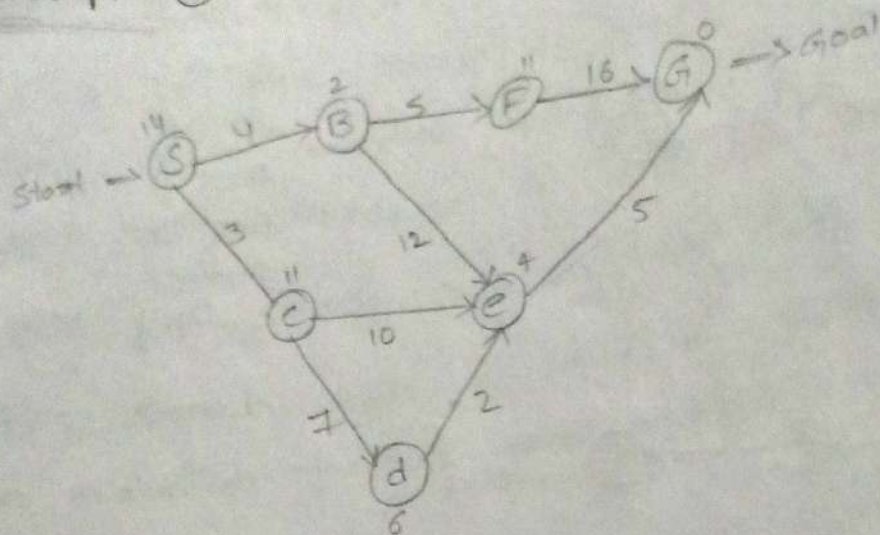
4)



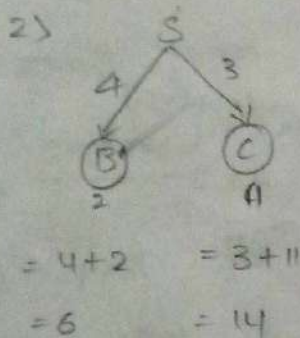
5)



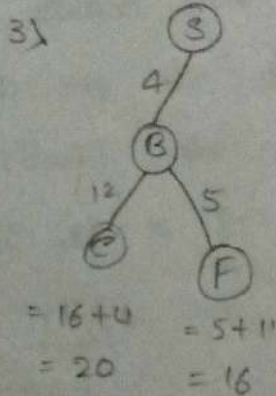
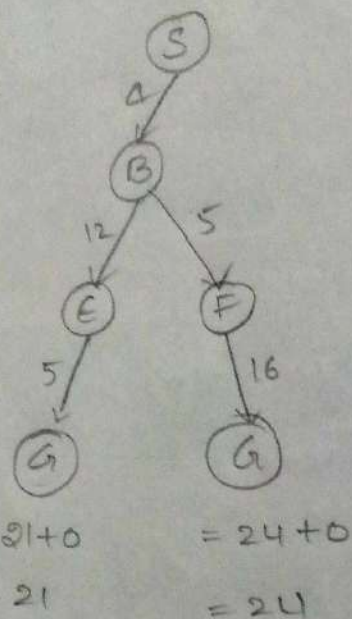
Example ② :-



1) $S = 0$



4)



Path = $S \rightarrow B \rightarrow E \rightarrow G$

Cost = 21

A * algorithm using 8 queen Problem [out of syllabus] :-

For 8 queen Problem $g(n)$ is considered as number of level.

$h(n)$ = the number of miss match cell value

Note :- if any two 8 queen's is having same heuristic value then expand both the queen's by performing

2	8	3
1	6	4
7		5

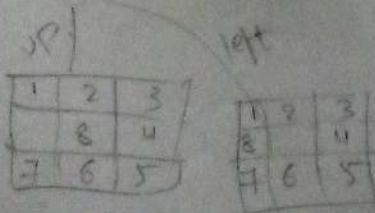
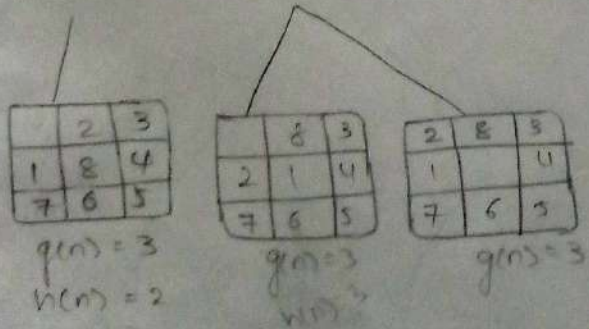
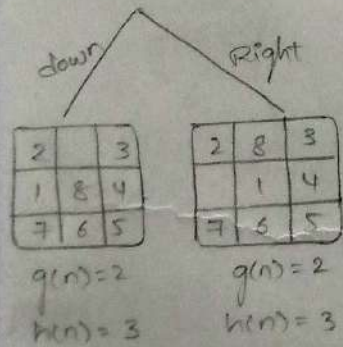
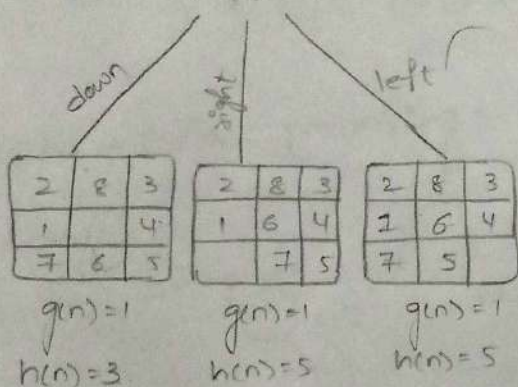
Start

$g(n) = 0$

$h(n) = 4$

1	2	3
2		4
7	6	5

Goal



A0* algorithm

A0* algorithm is based on AND & OR operation.

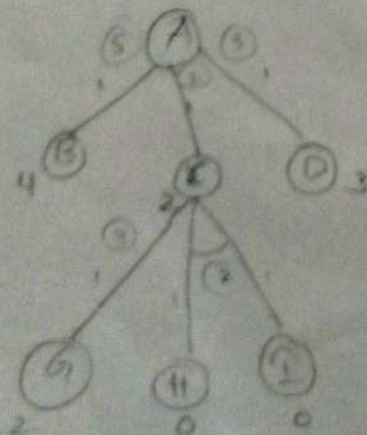
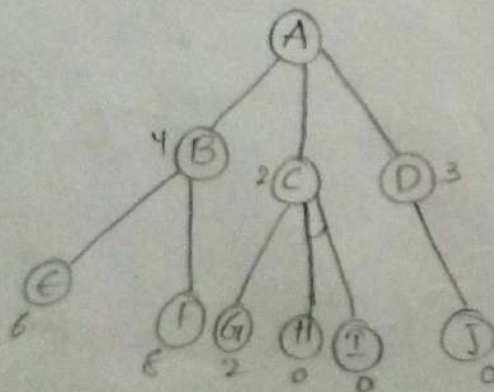
NOTE :-

1. check the heuristic values below the root node
2. Among the heuristic values select the smallest heuristic value and start teasing.
3. If you find 'A' this symbol then you have to add both the heuristic values of both the nodes by

Performing and operations.

4. the distance between all the nodes are same that is "1".

Example :-

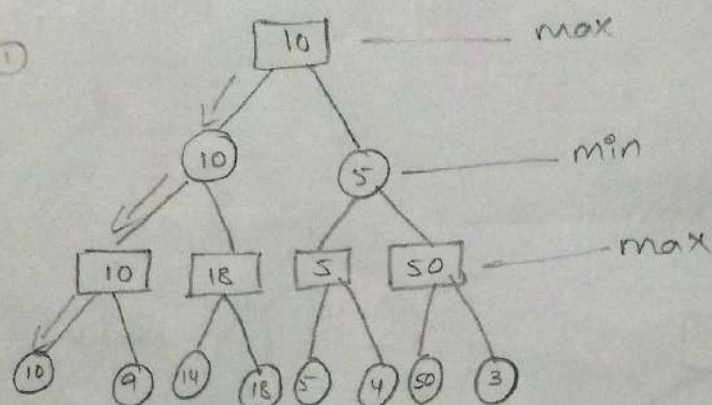


Min-max algorithm

It is a two players game tree in which static score are given from the first player point of view. It follows depth first search.

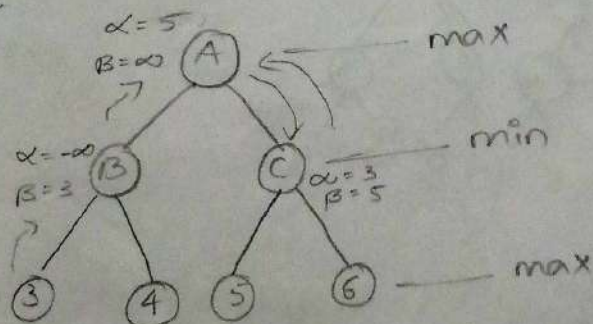
□ → Indicates maximum value
○ → Indicates minimum value

Ex: ①



Alpha-Beta pruning:-

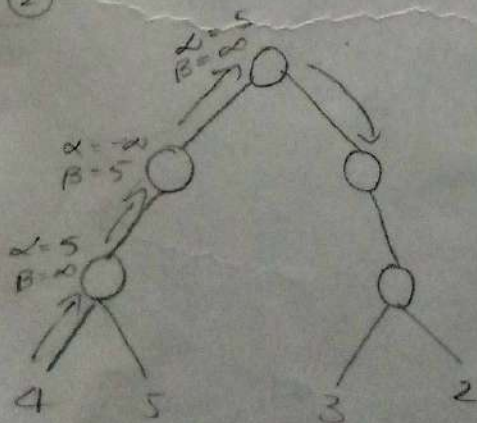
Example: 1



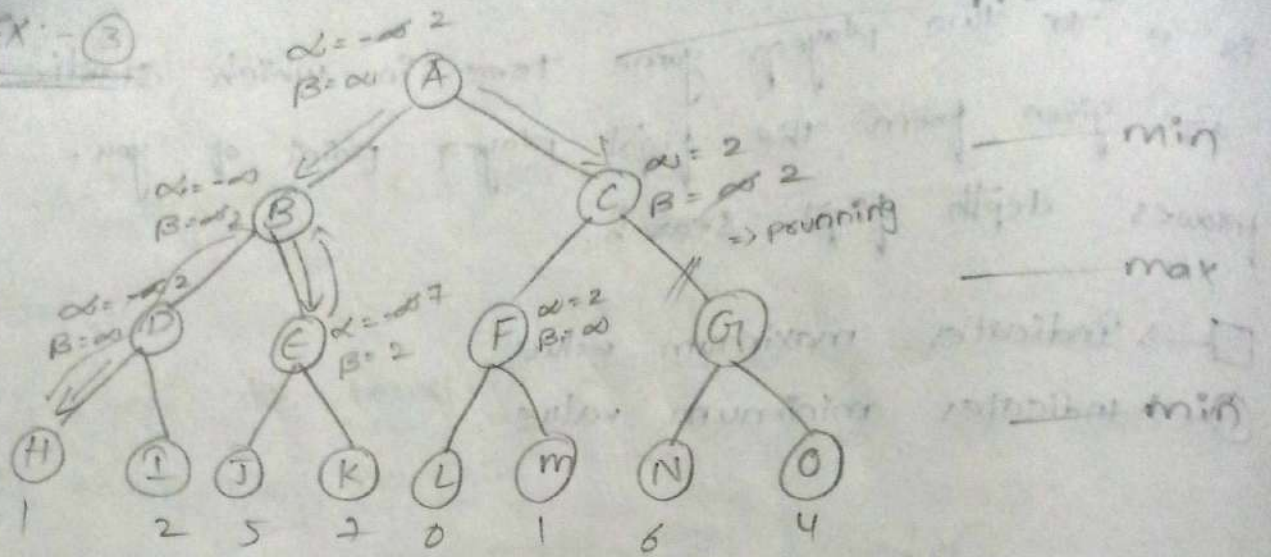
$$\alpha = -\infty$$

$$\beta = +\infty$$

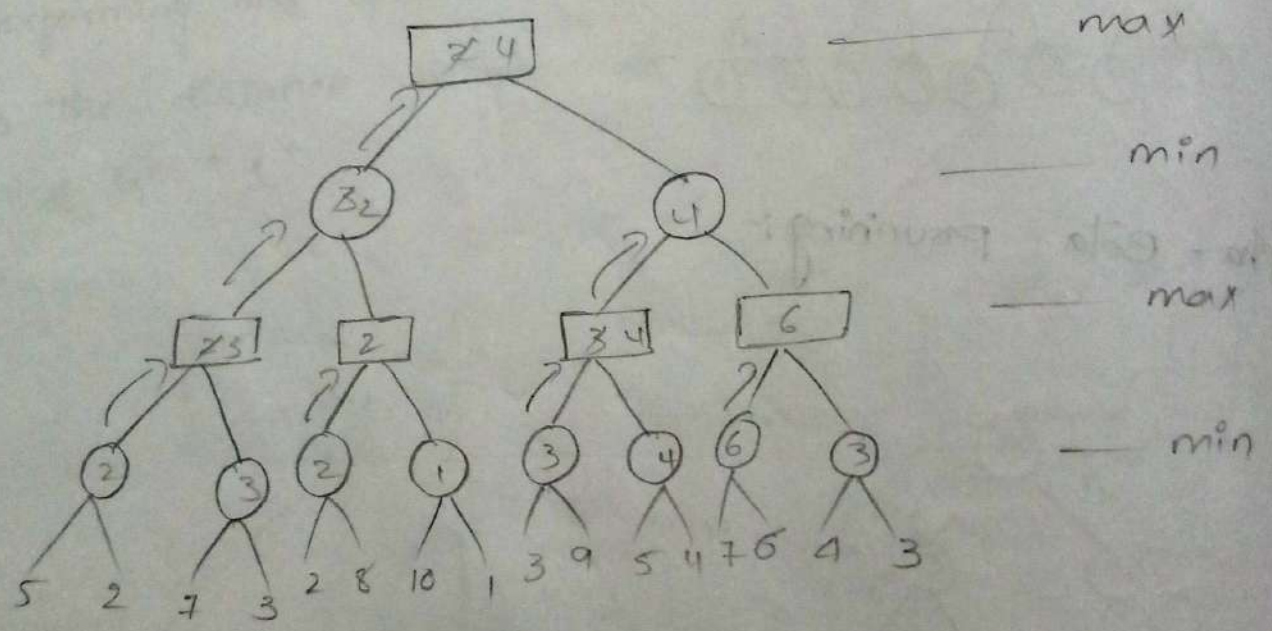
Ex: ②



Ex: - (3)



Ex: - (4)



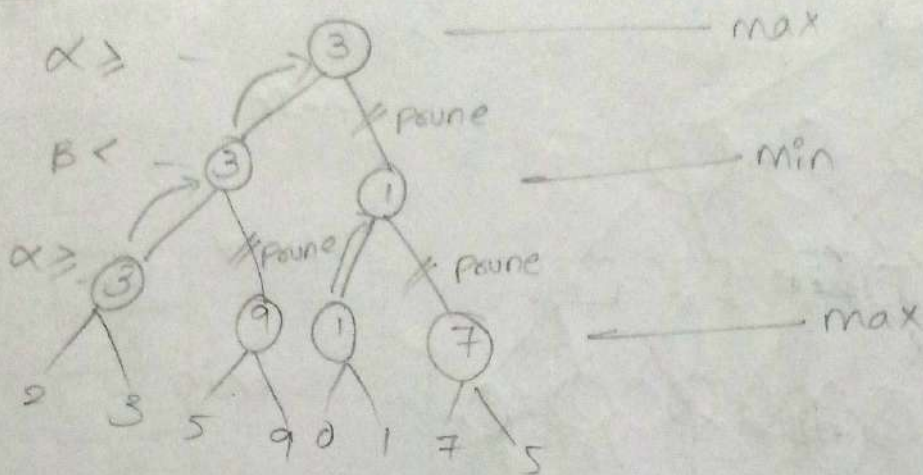
Assignment - 2

- 1> uniformed search
- 2> informed search
- 3> Alpha + Beta Pruning

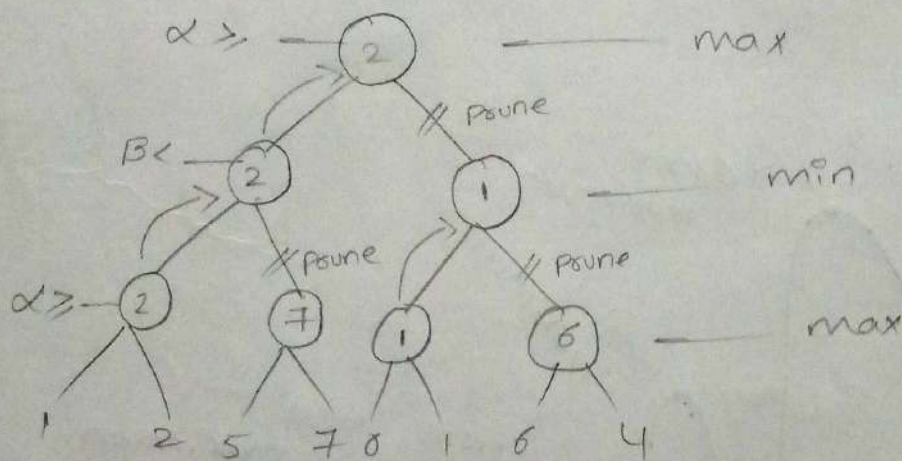
Example - 1

min - max Example

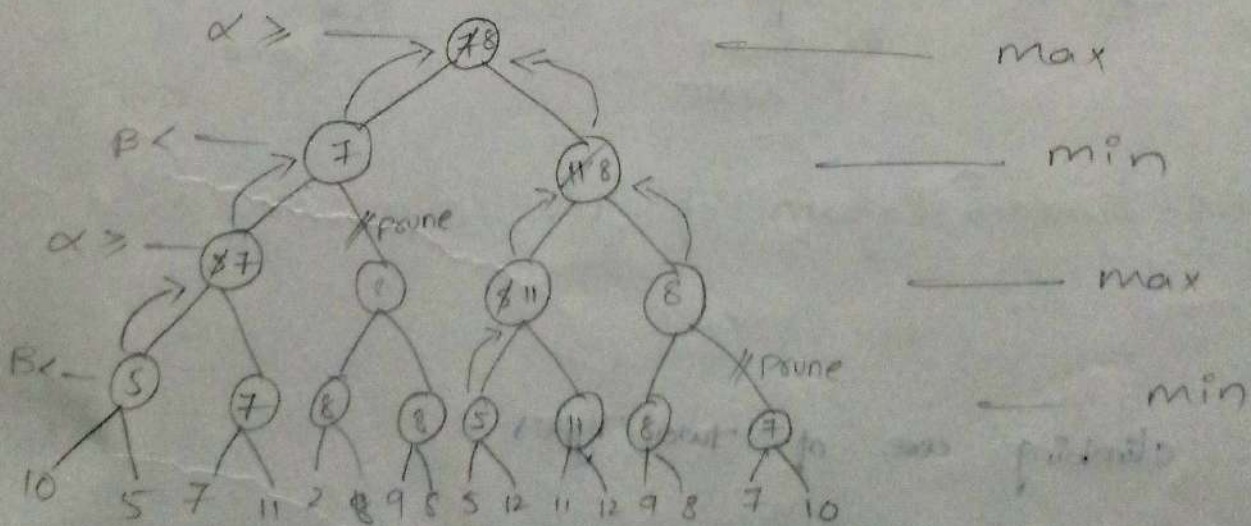
method - ②

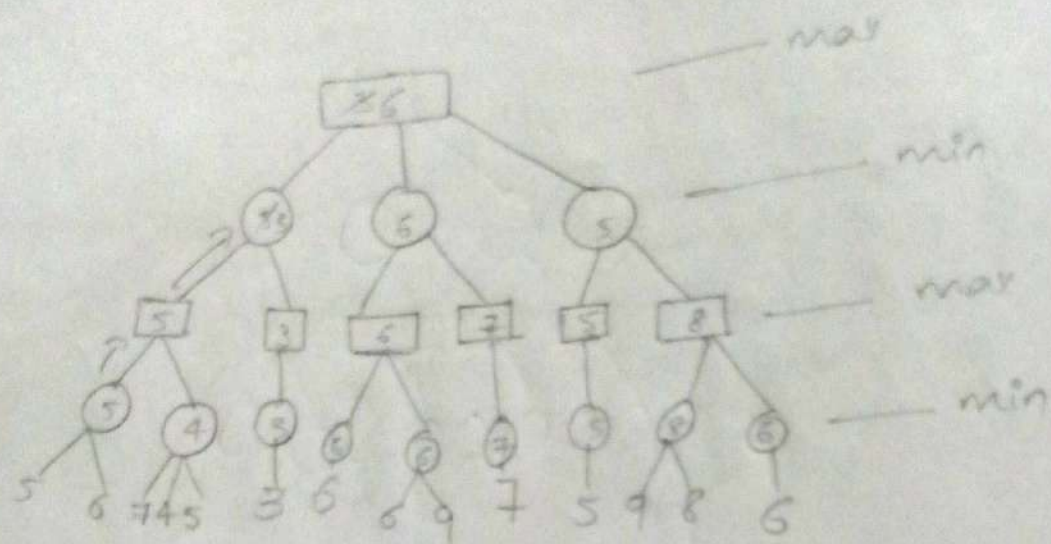


Example - 2

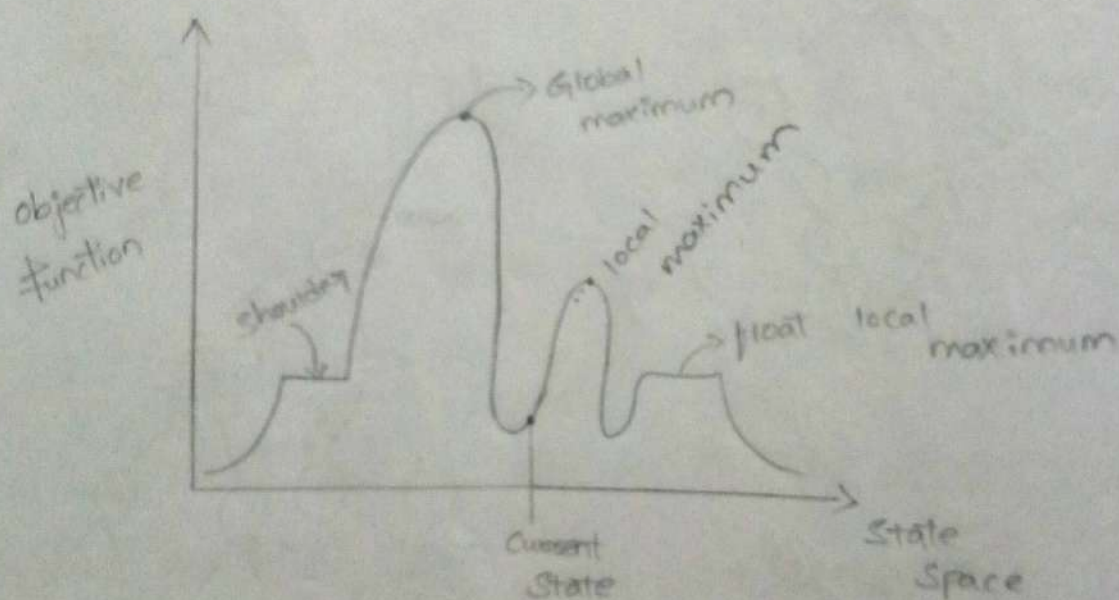


Example 3





Hill climbing



State-Space diagram of hill climbing

Hill climbing are of two types

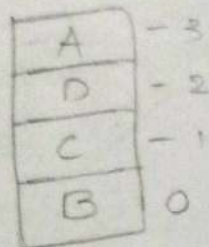
1. Simple hill climbing
2. Steepest ascent hill climbing

Rule :-

$h(x) = +1$ for all the blocks in the support structure. If the structure is correctly ~~per~~ fashioned

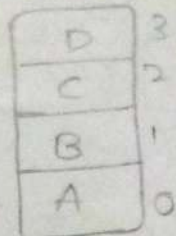
$h(x) = -1$ \Rightarrow the block are not correctly positioned.

Example



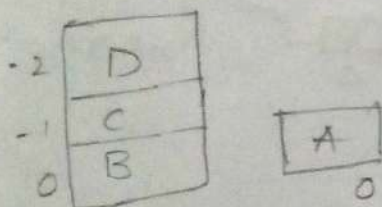
State

$$h(1) = -6$$

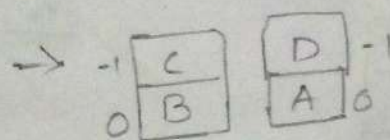


Goal

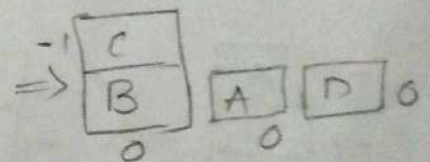
$$h(x) = 6$$



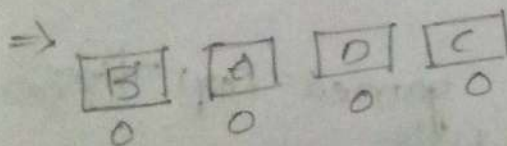
$$h(2) = -3$$



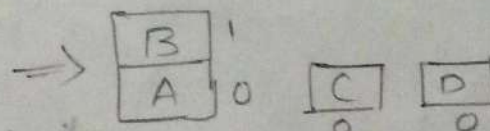
$$h(3) = -1$$



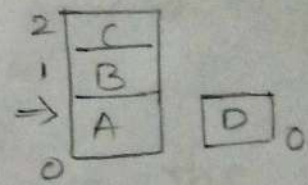
$$h(4) = -1$$



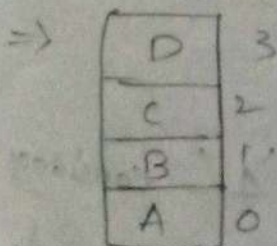
$$h(5) = 0$$



$$h(6) = 1$$



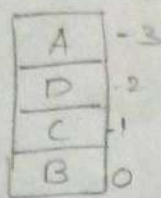
$$h(7) = 3$$



$$h(8) = 6$$

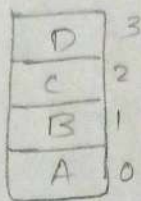
Steepest ascent :-

Steepest ascent hill climbing will choose all the directions of one go among them it will select the shortest heuristic value



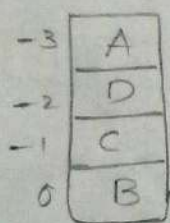
Start

$$h(1) = -6$$



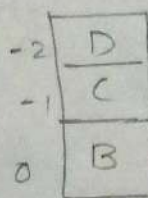
Goal

$$h(x) = 6$$

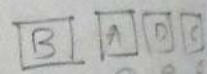
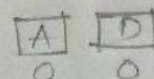
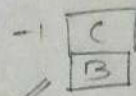
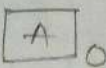


$$h(1) = -6$$

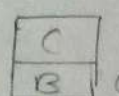
\Rightarrow



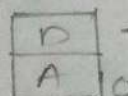
$$h(2) = -3$$



$$h(u) = 0$$



$$h(3) = -1$$



optimal dis. decisions in multiplayer games (or) three player games.

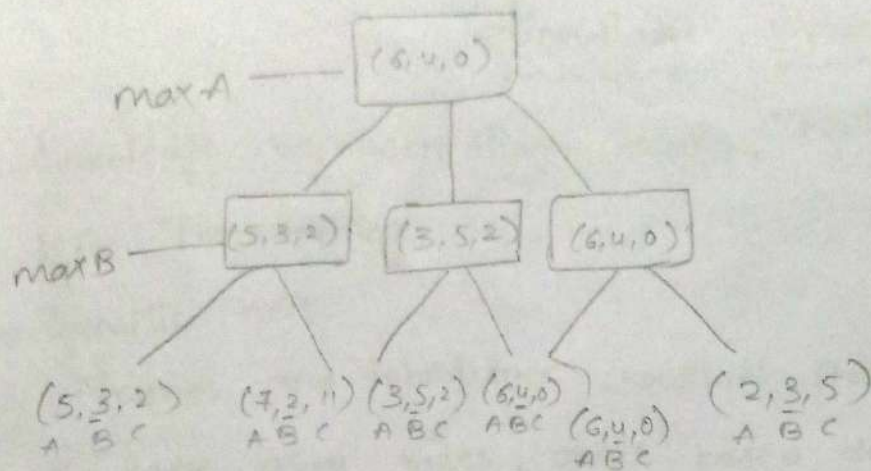
Note:

Here we have to select the maximum values in which when according to the player.

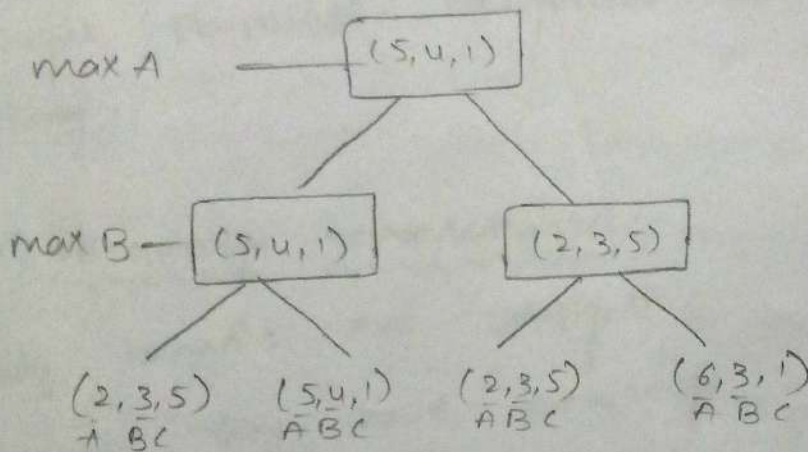
Here level '0' is considered as 'A', level '1' considered as 'B', level -3 is considered as 'C'. whatever the levels starts from A.

Because of only three player game. we are taking the help of Alphabet A, B, C.

Example:

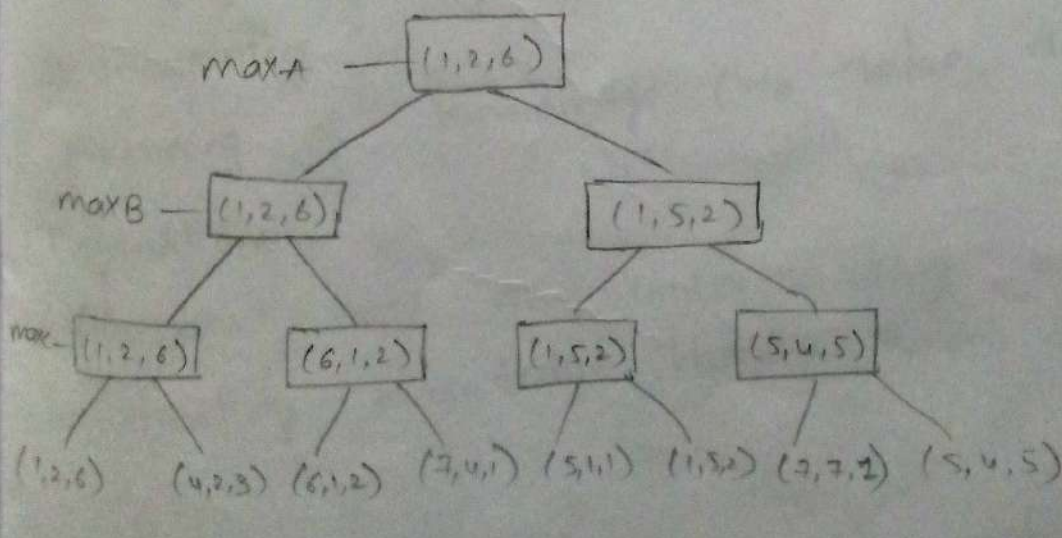


Question



Note:

* If both the child value are same then give the preference to the left child.



Knowledge representation

Knowledge representation issues, Predicate logic.

logic Programming

⇒ Semantic nets

i, Frames and inheritance, constant propagation, representing knowledge using rules, rules based deduction system.

* Reasoning under uncertainty, review of probability, bayes probabilistic inference and Dempster-Shafer

theory.

⇒ ① Knowledge representation :-

① Why human's are intelligent?

Because they have knowledge information data with these gathered through experience.

2) What is intelligence?

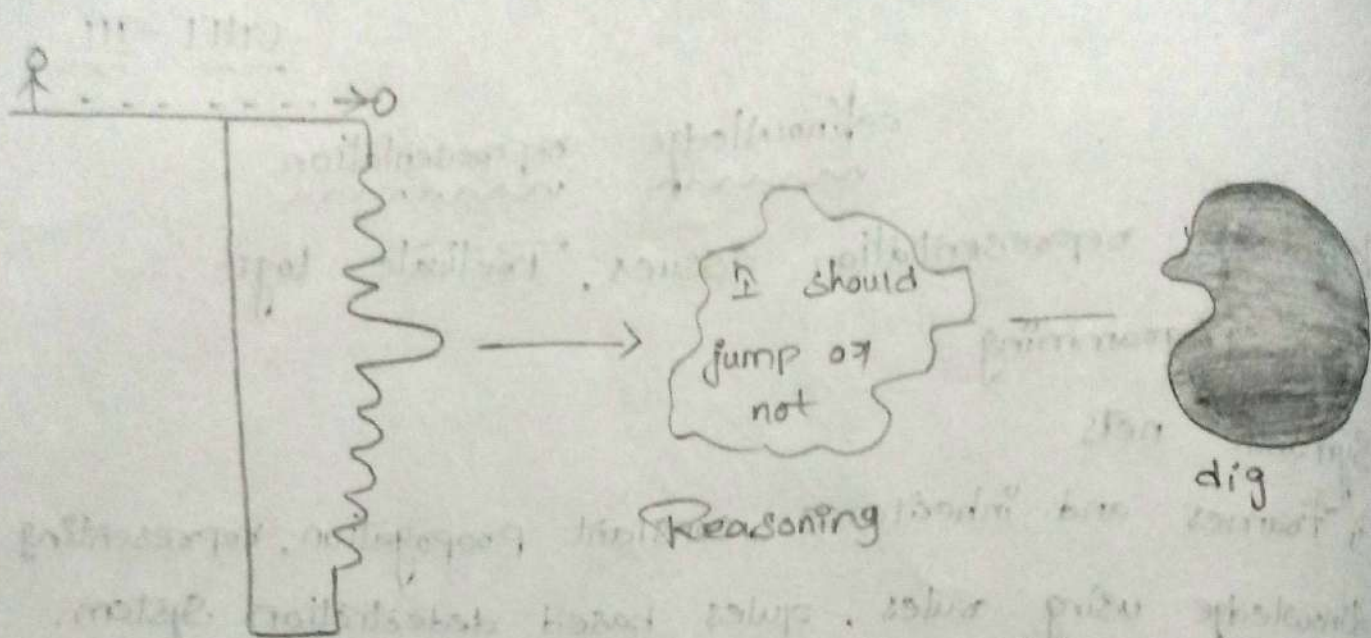
Ability to use the gathered knowledge.

3) Reasoning?

Processing of knowledge (to take decision)

Definitions :-

Humans best at understanding, reasoning and interpreting knowledge, human know things,



Knowledge based
agent

Decision: not to jump

Knowledge: if u jump u will get hurt

* But how machines do all this things comes under knowledge representation and reasoning. Hence, we can describe knowledge representation as follows.

[Knowledge representation, know

is the part of AI which is concerned with AI agent thinking and how thinking contributes through intelligent behaviour of agent. It is responsible for representing knowledge about the real world so that a computer can understand or utilize the knowledge to solve the complex real world problem such as diagnosis, a medical

what to represent :

Following are the kind of knowledge which needs to be represented in AI system.

1. Facts : facts are the truth about the real world
2. object : All the facts about object in our world domain.

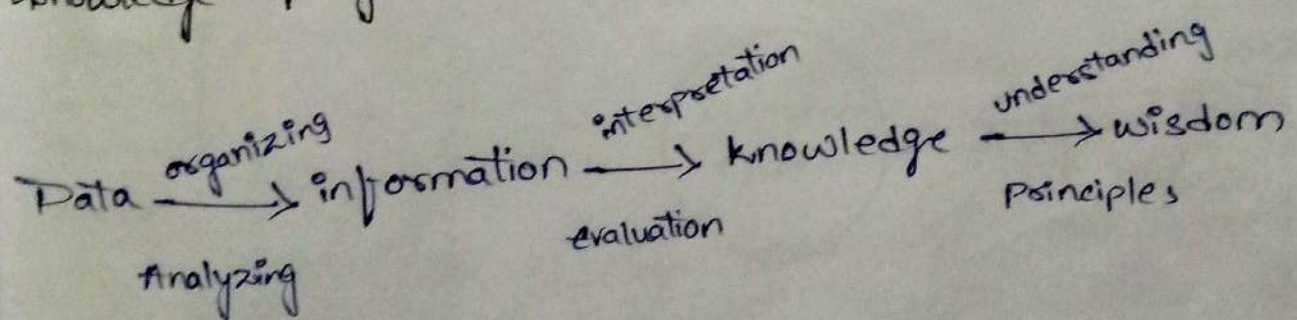
Example : Guitars contain strings.

3. Events : events are the actions which occur in our world.

4. Performance : It describes behaviour which involves knowledge about how to do things.

5. Meta knowledge : It is knowledge about what we know
[knowledge about knowledge.]

Knowledge progression :-

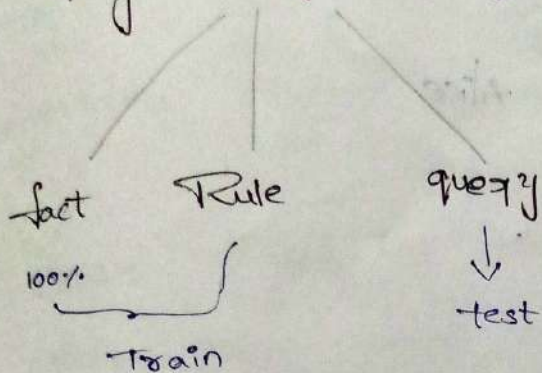


Logic Programming

(PROLOG)

↳ Programming
in logic

Logic Programming



Rules tables:-

and

or

;

;

fact:-

likes (dog, cat)

likes (cat, pig)

likes (elephant, horse)

Rules:-

friendship (x, y) :-

like (x, y);

likes (y, x).

Query:-

? likes (cat, pig) ✓ (True)

? friendship (dog, pig) ⇒ (false)

? friendship (dog, cat) ✓ ⇒ (True)

Example:-

Parent (John, Mary),

Parent (Mary, Alice),

Parent (John, Bob).

Rules:-

grandparent (x, y) :- parent (x, z)
parent (z, y)

Query:-

Is John a grandparent of Alice
? grandparent (John, Alice)

Example (2):-

Rule:-

Fact:-

raining (today)

Rule:-

wet-ground (x) - raining (x)

Query:-

? wet-ground (today)

Bayes Probabilistic interference

1. Prior knowledge $P(H)$:-

a. The probability that a person brush their teeth might be 0.9%.

2. Likelihood $P(E|H)$:-

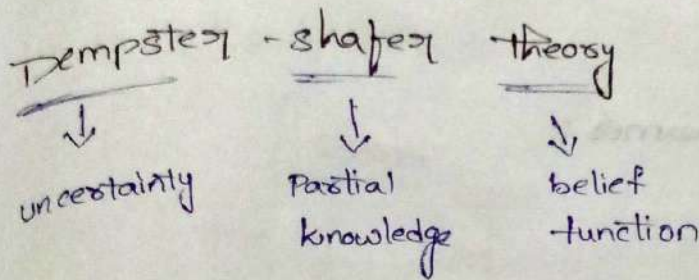
After observing some evidence like tooth brush wet and bad smell is not coming = 0.8% from mouth.

3. Marginal likelihood $P(E)$:-

By seeing the tooth paste is wet but we can't decide whether he brush or not, the probability is

$$P(H|e) = \frac{P(e|H) \cdot P(H)}{P(e)}$$

$$= \frac{0.8 * 0.9}{0.2}$$



1) Basic Probability Assignment (BPA)

this assign belief to a set of possibilities.

2) Belief function (Bel)

this represent the total belief committed to a set.

3) Plausibility function (Pl)

this represent the potential that a set could be true considering all possible possibilities.

Scenario :-

Person brush their teeth

Source 1 (witness) → They saw the person brushing

Source 2 (camera) → A camera show the person in the bathroom but not surge.

Possible outcomes :- Brush ✓ A
Brush ✗ B

BPA :- S₁(witness) $\begin{cases} A - 0.9 \\ B - 0.1 \end{cases}$

S₂(camera) $\begin{cases} A - 0.5 \\ B - 0.5 \end{cases}$

Outcomes

BPA S₁

BPA S₂

A

0.9

0.5

$$0.9 * 0.5 = 4500$$

B

0.1

0.5

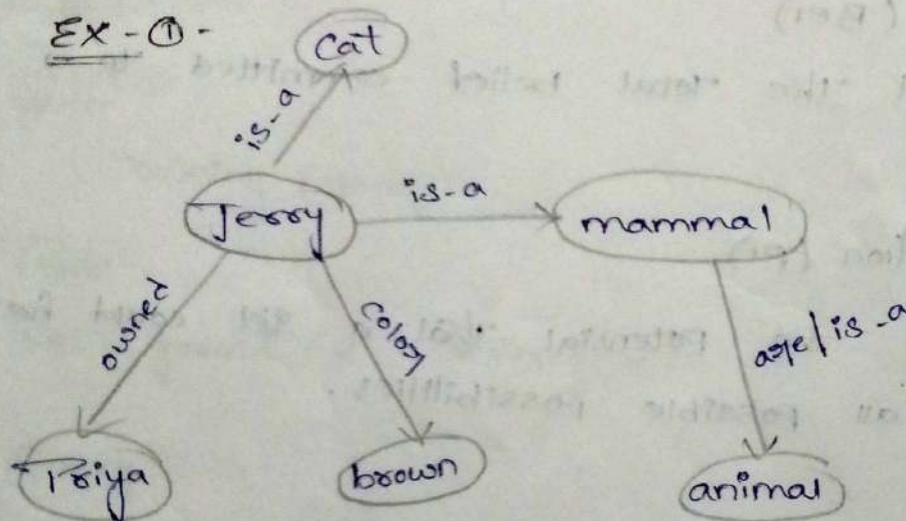
$$0.1 * 0.5 = 0.05$$

Semantic nets and frames :-

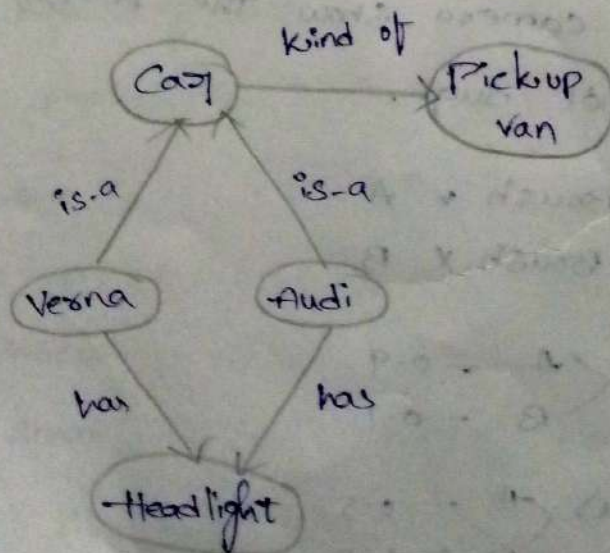
Semantic nets :-

is-a, has-a relation it will show just like object oriented programming.

EX - ① -



EX ② :-



Frames :-

Frames consist of slots and fillers

Example :-

Slot - filler

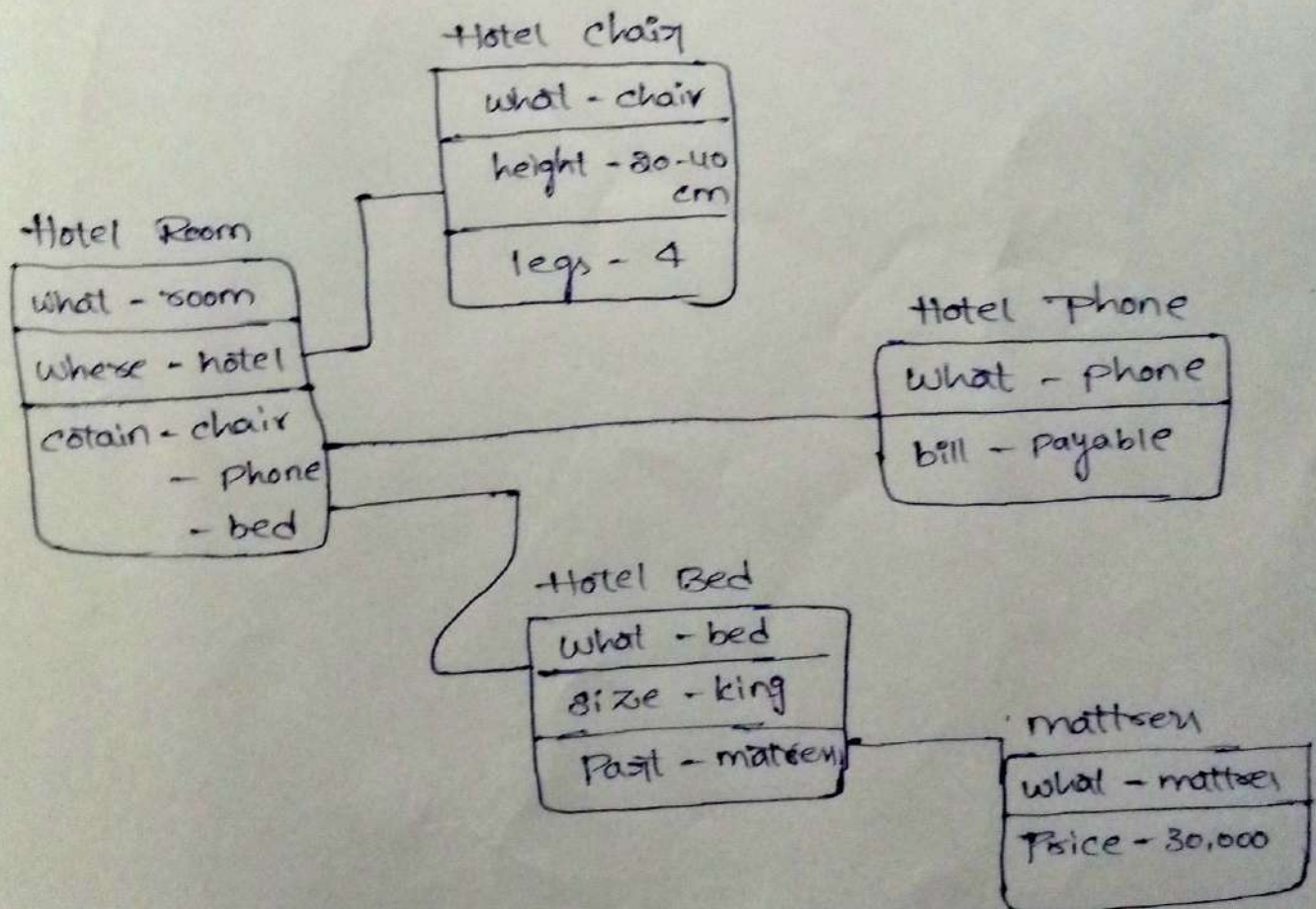
Books

Title	CN
Cost	200
written by	Armit

Class

Strength	66
girls	84
boys	42
Dropout	4

Inheritance :-



What is Knowledge Representation (KR)?

Knowledge Representation (KR) in Artificial Intelligence (AI) is the process of structuring and storing information so that an AI system can reason, learn, and make decisions. It is a bridge between human understanding and machine processing.

KR is essential for:

- Understanding complex environments (e.g., self-driving cars, medical diagnosis).
- Enabling reasoning and decision-making (e.g., expert systems).
- Storing and retrieving knowledge efficiently (e.g., chatbots, search engines).

Types of Knowledge in AI

AI systems need different types of knowledge, including:

Type of Knowledge	Example
Declarative Knowledge (facts)	"Paris is the capital of France."
Procedural Knowledge (how to do something)	"To drive, press the gas pedal."
Semantic Knowledge (meaning and relationships)	"Dogs are mammals."
Episodic Knowledge (specific events)	"I had coffee this morning."
Common Sense Knowledge	"Water is wet."
Meta-Knowledge (knowledge about knowledge)	"I know that I don't know this fact."

Approaches to Knowledge Representation

There are several ways AI systems can represent and process knowledge:

KR Approach	Description	Example
Logical Representation	Uses formal logic (propositional & first-order logic).	If it rains, the ground is wet.
Semantic Networks	Graph-based structure with entities and relationships.	"Cat → is a → Mammal"
Frames	Object-oriented structure with attributes and values.	Person(Name: Alice, Age: 30, Job: Engineer)
Production Rules	IF-THEN rules for decision-making.	IF temperature > 100°C THEN water boils.
Ontologies	Structured knowledge in hierarchical form (used in the Semantic Web).	Ontology of "Animals" includes "Mammals" and "Birds".
Fuzzy Logic	Handles uncertainty using degrees of truth (0 to 1).	"The room is warm (0.7 truth value)."
Bayesian Networks	Probabilistic models for reasoning under uncertainty.	If a patient has a cough, the probability of flu is 60%.

Example: Knowledge Representation in AI Chatbots

Consider an AI chatbot that helps with medical diagnosis.

Approach 1: Rule-Based System (Production Rules)

IF fever AND cough THEN flu.

IF sore_throat AND no_fever THEN cold.

Approach 2: Semantic Network

Flu → causes → Fever

Flu → causes → Cough

Cold → causes → Sore Throat

Approach 3: Bayesian Network (Probabilistic KR)

- $P(\text{Flu}|\text{Fever})=0.8$
 $P(\text{Flu} | \text{Fever}) = 0.8$
 $P(\text{Flu}|\text{Fever})=0.8$
- $P(\text{Cold}|\text{SoreThroat})=0.6$
 $P(\text{Cold} | \text{Sore Throat}) =$
 0.6
 $P(\text{Cold}|\text{SoreThroat})=0.6$

The chatbot can reason about symptoms and suggest possible illnesses.

challenges in Knowledge Representation

- ◆ Incomplete Knowledge – AI may lack all the facts.
 - ◆ Uncertainty – Some knowledge is probabilistic (handled by Bayesian Networks, Fuzzy Logic).
 - ◆ Ambiguity – Natural language is complex (handled by NLP techniques).
 - ◆ Scalability – Storing vast knowledge efficiently (handled by ontologies and knowledge graphs).
-

Applications of Knowledge Representation

- ✓ Expert Systems – Medical diagnosis (IBM Watson), legal reasoning
- ✓ Search Engines – Google Knowledge Graph
- ✓ Robotics – Navigation and decision-making
- ✓ AI Assistants – Siri, Alexa (understanding context)
- ✓ Self-Driving Cars – Recognizing road signs, pedestrians

Constant Propagation

In **knowledge representation (KR)**, **constant propagation** is a logical optimization technique used to simplify reasoning by replacing known constant values in a knowledge base. This helps in inference, rule-based systems, and automated reasoning.

How Constant Propagation Works in KR

When a **fact or rule contains constants**, those constants can be propagated through the system to simplify expressions, remove redundant computations, and improve efficiency.

Example in Rule-Based Systems

Consider a **Prolog-style** knowledge base:

```
age(john, 30).
```

```
age(mary, 25).
```

```
is_adult(X) :- age(X, A), A >= 18.
```

Here, instead of checking $A \geq 18$ every time, we can directly infer:

```
is_adult(john). % Directly inferred
```

```
is_adult(mary). % Directly inferred
```

This avoids unnecessary computations when querying whether someone is an adult.

Example in First-Order Logic (FOL)

Suppose we have a knowledge base:

1. $\text{father}(\text{John}) = \text{Robert}$ $\text{father}(\text{John}) = \text{Robert}$ (John's father is Robert)
2. $\text{father}(\text{Robert}) = \text{William}$ $\text{father}(\text{Robert}) = \text{William}$ (Robert's father is William)
3. $\text{grandfather}(X) = \text{father}(\text{father}(X))$ $\text{grandfather}(X) = \text{father}(\text{father}(X))$

Using **constant propagation**, we can directly derive:

```
grandfather(John) = William
```

Instead of repeatedly applying the **father()** function, we replace known constants directly.

Application in AI and Knowledge Graphs

1. **Semantic Web (RDF & OWL Reasoning)**
 - In **ontology reasoning**, constant propagation simplifies relationships.
 - Example: If **hasAge(Alice, 25)**, then **hasAge(Bob, 25)** (if Bob and Alice are the same entity).
2. **Expert Systems & Decision Trees**
 - **Precomputing constant values** reduces rule evaluations.
 - Example: A **medical expert system** with rules:

fever(X) :- temperature(X, T), T > 38.

If temperature(john, 39), it simplifies to fever(john).

3. Logic Programming (Prolog, Datalog)

- Directly substitutes values for predicates to avoid redundant computations.

Benefits of Constant Propagation in KR

- ✅ **Faster inference** – Precomputes known values, reducing real-time reasoning workload.
- ✅ **Memory optimization** – Reduces storage by eliminating redundant evaluations.
- ✅ **Simplifies rule evaluation** – Allows efficient decision-making in AI reasoning.

Representing Knowledge Using Rules

In **Knowledge Representation (KR)**, **rules** are used to encode knowledge in a structured way. Rule-based systems define relationships, constraints, and logical inferences that allow AI systems to **reason and derive conclusions**.

Types of Rules in Knowledge Representation

📄 Production Rules (IF-THEN Rules)

- **Format:** IF (condition) THEN (action/conclusion)
- Used in **expert systems** and **decision-making AI**.

✅ Example (Medical Diagnosis Rule)

IF (temperature > 38°C) AND (cough = yes)

THEN (diagnosis = "Possible flu")

🚀 Implementation in Prolog:

flu(X) :- temperature(X, T), T > 38, cough(X, yes).

👉 If **John** has **39°C temperature** and a **cough**, the system infers flu(john).

📄 Derivation Rules (Inference Rules)

- Used in **logical reasoning** and **knowledge graphs**.
- Helps derive **new facts from existing facts**.

✅ Example (Family Relationships)

less

IF (parent(X, Y)) AND (parent(Y, Z))

THEN (grandparent(X, Z))

🚀 Implementation in Prolog:

grandparent(X, Z) :- parent(X, Y), parent(Y, Z).

👉 If Alice is Bob's parent and Bob is Charlie's parent, then Alice is Charlie's grandparent.

§ Constraint Rules

- Defines what is **allowed or restricted** in a knowledge base.
- Used in **AI planning, business rules, and databases**.

✅ Example (Bank Loan Eligibility)

IF (income > \$50,000) AND (credit_score > 700)

THEN (loan_approved = yes)

🚀 Implementation in SQL-like Rule System:

sql

SELECT customer_id

FROM applicants

WHERE income > 50000 AND credit_score > 700;

🔲 Fuzzy Rules (For Handling Uncertainty)

- Used in **fuzzy logic systems** to deal with **imprecise data**.
- Instead of strict "yes/no," it considers degrees of truth.

✅ Example (Fan Speed Control Based on Temperature)

csharp

IF (temperature is high)

THEN (fan_speed is fast)

🚀 Fuzzy Logic Example:

python

import skfuzzy as fuzz

temperature = fuzz.trimf([30, 40, 50])

fan_speed = fuzz.trimf([0, 50, 100])

👉 This allows **gradual adjustment** rather than an abrupt ON/OFF decision.

Applications of Rule-Based Knowledge Representation

- ✅ **Expert Systems** (e.g., MYCIN for medical diagnosis)
- ✅ **Chatbots & Virtual Assistants** (e.g., rule-based responses in AI bots)
- ✅ **Semantic Web & Ontologies** (e.g., RDF, OWL for knowledge graphs)
- ✅ **Business Rule Management Systems** (e.g., Drools, IBM Operational Decision Manager)
- ✅ **AI Planning & Robotics** (e.g., decision-making in self-driving cars)

Rule-Based Deduction System in Knowledge Representation

A **Rule-Based Deduction System** is a **knowledge-based system** that derives new facts from existing facts using **rules** and **logical inference**. It is widely used in **expert systems**, **AI reasoning**, and **automated decision-making**.

Key Components of a Rule-Based Deduction System

- ❏ **Knowledge Base (KB)** – Contains **facts** and **rules**.
 - ❏ **Inference Engine** – Applies rules to known facts to **derive new facts**.
 - ❏ **Working Memory** – Stores **intermediate facts** generated during reasoning.
 - ❏ **User Interface** – Allows interaction with the system.
-

How Rule-Based Deduction Works

- **Rules** are written in **IF-THEN** format.
 - The **inference engine** applies **forward** or **backward chaining** to derive conclusions.
-

Types of Rule-Based Deduction

❏ **Forward Chaining (Data-Driven Inference)**

- Starts with known **facts** and applies **rules** to infer new facts.
- Used in **expert systems** and **automated reasoning**.

✅ Example: Diagnosing a Disease

Facts:

fever(john).

cough(john).

Rule:

IF (fever(X) AND cough(X)) THEN flu(X).

🚀 Prolog Implementation:

flu(X) :- fever(X), cough(X).

👉 Query: ?- flu(john).

✅ Output: Yes (because John has fever and cough).

🔍 Backward Chaining (Goal-Driven Inference)

- Starts with a **goal** and works **backwards** to find supporting facts.
- Used in **theorem proving** and **logic programming (e.g., Prolog)**.

✅ Example: Checking if John has the flu

Goal: flu(john)?

1. The system checks: Does John have **fever** and **cough**?
2. If facts exist → Conclude flu(john).
3. If facts do not exist → Ask user or perform further deductions.

🚀 Prolog Example (Backward Chaining)

flu(X) :- fever(X), cough(X).

fever(john).

cough(john).

👉 Query: ?- flu(john).

✅ Output: Yes (because Prolog checks rules recursively).

Applications of Rule-Based Deduction Systems

- ✅ **Expert Systems** – MYCIN (Medical Diagnosis), CLIPS (Decision Making)
- ✅ **AI Chatbots** – Rule-based conversational AI
- ✅ **Theorem Provers** – Prolog, Automated Reasoning Systems
- ✅ **Semantic Web & Ontologies** – RDF, OWL reasoning
- ✅ **Business Rule Engines** – Drools, IBM Operational Decision Manager

Advantages & Disadvantages

✓ Advantages:

- ✓ Transparent reasoning process
- ✓ Easy to modify rules
- ✓ Suitable for well-defined domains

✗ Disadvantages:

- ✗ Cannot handle uncertainty well (unless fuzzy logic is used)
- ✗ Computationally expensive for large knowledge bases
- ✗ Hard to scale for complex real-world scenarios

Reasoning Under Uncertainty in Knowledge Representation

In **Knowledge Representation (KR)**, **reasoning under uncertainty** deals with situations where facts and rules are **incomplete, imprecise, or uncertain**. Traditional logic-based systems (like Prolog) work well with **definite knowledge**, but real-world scenarios (e.g., medical diagnosis, self-driving cars, AI assistants) require handling **probabilistic** and **fuzzy** information.

Types of Uncertainty in Knowledge Representation

- 1 **Probability-Based Uncertainty** (e.g., Bayesian Networks)
- 2 **Fuzzy Logic** (e.g., Handling vague concepts like "tall" or "hot")
- 3 **Non-Monotonic Reasoning** (e.g., Default and belief-based reasoning)
- 4 **Dempster-Shafer Theory** (e.g., Combining evidence from multiple sources)
- 5 **Possibility Theory** (e.g., Alternative to probability for handling vagueness)

Methods for Reasoning Under Uncertainty

1 Bayesian Networks (Probabilistic Reasoning)

- Uses **probabilities** to model uncertainty.
- Based on **Bayes' Theorem**: $P(H|E) = \frac{P(E|H)P(H)}{P(E)}$ Where:
 - $P(H|E)P(H|E)P(H|E)$ = Probability of hypothesis HHH given evidence EEE.
 - $P(E|H)P(E|H)P(E|H)$ = Likelihood of evidence if HHH is true.

- $P(H)P(H)P(H)$ = Prior probability of HHH.

✅ Example: Medical Diagnosis

If **80% of flu patients have a fever**, and **5% of the general population has a fever**, we can compute:

$$P(\text{Flu}|\text{Fever}) = (0.8 * 0.1) / 0.05 = 0.16 \quad P(\text{Flu} \mid \text{Fever}) = (0.8 * 0.1) / 0.05 = 0.16$$

$$P(\text{Flu}|\text{Fever}) = (0.8 * 0.1) / 0.05 = 0.16$$

Applications: AI-based diagnosis, spam detection, robotics.

🔧 Fuzzy Logic (Handling Vagueness and Approximation)

- Used when information is **not binary (True/False)** but **gradual (0 to 1)**.
- Example:
 - **Crisp Logic:** Temperature > 30°C → "Hot" (Strict condition).
 - **Fuzzy Logic:** Temperature = 28°C → "Moderately Hot" (Partial truth).

✅ Example: Controlling a Fan Based on Temperature

IF temperature is HIGH THEN fan speed is FAST.

🚀 Python Example using skfuzzy

```
import skfuzzy as fuzz
```

```
import numpy as np
```

```
temperature = np.arange(20, 41, 1)
```

```
hot = fuzz.trimf(temperature, [30, 35, 40])
```

```
print(hot) # Membership values for "hot" category
```

Applications: AI controllers, robotics, recommendation systems.

🔧 Non-Monotonic Reasoning (Default & Belief-Based Reasoning)

- In **classical logic**, once something is proven, it **never changes**.
- In **non-monotonic reasoning**, new evidence **can retract conclusions**.

✅ Example: Bird & Flight Assumption

IF X is a bird THEN X can fly.

IF X is a penguin THEN X cannot fly.

If we first assume "Tweety is a bird," we conclude "Tweety can fly."

Later, if we learn "Tweety is a penguin," we **revise our belief** and conclude "Tweety cannot fly."

🚀 Prolog Implementation (Non-Monotonic Logic)

flies(X) :- bird(X), \+ penguin(X).

bird(tweety).

penguin(tweety).

👉 Query: ?- flies(tweety).

✅ Output: No (because Tweety is a penguin).

Applications: AI planning, knowledge graphs, dynamic decision-making.

📌 **Dempster-Shafer Theory (Evidence-Based Reasoning)**

- Used when **probabilities are unknown**, but we have **belief and doubt**.
- Allows **combining evidence** from multiple sources.

✅ **Example: Identifying an Object Based on Two Sensors**

Sensor A: 60% confidence it's a car.

Sensor B: 50% confidence it's a car.

Combined belief using Dempster's Rule: 75%.

Applications: Sensor fusion, fraud detection, AI-based surveillance.

Comparison of Uncertainty Handling Methods

Method	Best For	Example Use Case
Bayesian Networks	Probabilistic reasoning	Medical diagnosis, spam filters
Fuzzy Logic	Approximate reasoning	AI controllers, smart home systems
Non-Monotonic Logic	Revising beliefs	AI planning, expert systems
Dempster-Shafer	Combining evidence	Multi-sensor fusion, cybersecurity

Bayesian Probabilistic Inference and Dempster-Shafer Theory

Bayesian inference and Dempster-Shafer theory are two key **reasoning under uncertainty** approaches in **knowledge representation (KR)**.

📌 **Bayesian Probabilistic Inference**

Bayesian inference is based on **Bayes' theorem**, which updates **the probability of a hypothesis** as new evidence appears.

Bayes' Theorem:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}$$

Where:

- $P(H|E)P(H|E)P(H|E)$ = Probability of **hypothesis** HHH given **evidence** EEE (posterior).
- $P(E|H)P(E|H)P(E|H)$ = Probability of **evidence given hypothesis** (likelihood).
- $P(H)P(H)P(H)$ = Prior probability of the **hypothesis**.
- $P(E)P(E)P(E)$ = Total probability of the **evidence**.

Example: Medical Diagnosis

Let's say a doctor wants to determine if a patient has **flu** (HHH) based on the symptom **fever** (EEE).

- **Prior Probability** $P(H)P(H)P(H) = 10\%$ (1 in 10 people have the flu).
- **Likelihood** $P(E|H)P(E|H)P(E|H) = 80\%$ (80% of flu patients have fever).
- **Evidence Probability** $P(E)P(E)P(E) = 5\%$ (5% of the population has fever).

Using **Bayes' Theorem**:

$$P(\text{Flu}|\text{Fever}) = \frac{0.8 \times 0.1}{0.05} = 0.16$$

👉 The probability that a patient has the **flu given that they have a fever** is **16%**.

✅ **Applications:** Medical diagnosis, spam filters, recommendation systems, robotics.

Dempster-Shafer Theory (Evidence Theory)

Dempster-Shafer theory (DST) is a **generalization of probability theory** that deals with **uncertainty and incomplete knowledge**. Unlike Bayes' Theorem, DST does **not require prior probabilities** and allows for "**degrees of belief**" based on evidence.

Key Concepts in Dempster-Shafer Theory

- **Belief function** $\text{Belief}(A)$: The degree of confidence in **A being true**.
- **Plausibility function** $\text{Plausibility}(A)$: The maximum possible belief in **A** (accounts for unknowns).
- **Mass function** $m(A)$: A function that assigns **belief masses** to different subsets of the hypothesis space.

Dempster's Rule of Combination

If two independent sources give mass functions m_1 and m_2 , the combined belief is:

$$m(A)=\sum X\cap Y=Am_1(X)\cdot m_2(Y)1-Km(A)=\frac{\sum_{X\cap Y=A}m_1(X)\cdot m_2(Y)}{1-K}m(A)=1-K\sum X\cap Y=Am_1(X)\cdot m_2(Y)$$

where **K** represents the conflict between the two sources.

Example: Identifying an Object Based on Two Sensors

Imagine two sensors trying to identify whether an object is a **car (C)** or a **bike (B)**:

- **Sensor A:**
 - 60% sure it's a **car**.
 - 30% sure it's a **bike**.
 - 10% uncertainty.
- **Sensor B:**
 - 50% sure it's a **car**.
 - 40% sure it's a **bike**.
 - 10% uncertainty.

Using **Dempster’s Rule**, we combine the belief values to get a final **updated belief** for each possibility.

✅ **Applications:** Sensor fusion, AI decision-making, fraud detection, autonomous vehicles.

Comparison: Bayesian Inference vs. Dempster-Shafer Theory

Feature	Bayesian Inference	Dempster-Shafer Theory
Requires Prior Probabilities?	✅ Yes	❌ No
Handles Uncertainty?	✅ Yes, using probabilities	✅ Yes, using belief functions
Deals with Missing Information?	❌ No (needs prior data)	✅ Yes (supports unknowns)
Mathematical Complexity	Moderate	Higher
Combining Evidence	Uses conditional probabilities	Uses Dempster’s Rule
Best Used For	Probabilistic reasoning (e.g., medical AI)	Multi-source evidence fusion (e.g., sensor AI)

Note:

- ✓ Use **Bayesian Networks** if you have **prior probabilities** and want **probabilistic reasoning**.
- ✓ Use **Dempster-Shafer Theory** if you want to **combine uncertain evidence** from multiple sources **without needing priors**.

Simple example for students

Bayesian Probabilistic Inference: "Brush Teeth" Example

Bayesian **probabilistic inference** helps us make decisions based on **prior knowledge** and **new evidence**. Let's apply **Bayes' Theorem** to the everyday habit of **brushing teeth** and its relationship with having **fresh breath**.

❏ Problem Statement: Does Fresh Breath Indicate That a Person Brushed Their Teeth?

Let:

- **HHH = Person brushed their teeth**
- **EEE = Person has fresh breath**

We want to compute **$P(H|E)$** , the probability that a person brushed their teeth given that they have fresh breath.

Bayes' Theorem Formula:

$$P(H|E) = \frac{P(E|H) \cdot P(H)}{P(E)}$$

Assume the following probabilities:

1. **Prior Probability $P(H)$** (chance that a person brushed their teeth) = **0.8** (80%)
2. **Likelihood $P(E|H)$** (probability of fresh breath if they brushed) = **0.9** (90%)
3. **Total Probability of Fresh Breath $P(E)$** :
 - **Fresh breath can also occur if they used mouthwash or chewed gum**
 - Suppose **10% of people don't brush but still have fresh breath**
 - Then: $P(E) = P(E|H)P(H) + P(E|\neg H)P(\neg H)$
 $P(E) = (0.9 \times 0.8) + (0.1 \times 0.2) = 0.72 + 0.02 = 0.74$

❏ Apply Bayes' Theorem

$$P(H|E)=0.9 \times 0.80.74=0.720.74=0.973 \quad P(H|E) = \frac{0.9 \times 0.8}{0.74} = \frac{0.72}{0.74} = 0.973$$

👉 If a person has fresh breath, there is a **97.3% probability that they brushed their teeth.**

Real-Life Applications of Bayesian Inference

- ✅ **Medical AI** – Diagnosing diseases based on symptoms
- ✅ **Spam Filters** – Determining if an email is spam based on keywords
- ✅ **AI Assistants** – Making smart decisions based on user behavior
- ✅ **Self-Driving Cars** – Assessing road conditions based on sensor data

Dempster-Shafer Theory (DST) with "Brush Teeth" Example:

Dempster-Shafer Theory (DST) is used for **reasoning under uncertainty**, especially when multiple sources provide **partial or conflicting evidence**. Unlike Bayesian inference, **DST does not require prior probabilities** and allows for **degrees of belief and uncertainty**.

❏ Problem Statement: Did a Person Brush Their Teeth?

We have **two independent sources of evidence**:

1. **A camera in the bathroom** that may have recorded the person brushing.
2. **A witness (e.g., roommate)** who may have seen the person brushing.

Each source provides **some belief, doubt, and uncertainty** about whether the person brushed their teeth.

❏ Assigning Mass Functions to Evidence Sources

We define three hypotheses:

- **B (Brushed Teeth)**
- **~B (Did NOT Brush Teeth)**
- **U (Uncertainty / Don't Know)**

Belief Masses from the Two Evidence Sources

● Camera Evidence

Hypothesis Belief Mass (m1)

Brushed Teeth (B) 0.7

Did NOT Brush (~B) 0.1

Uncertainty (U) 0.2

● Witness Testimony

Hypothesis Belief Mass (m2)

Brushed Teeth (B) 0.6

Did NOT Brush (~B) 0.2

Uncertainty (U) 0.2

Each source has **some uncertainty**, as the camera might not have captured everything, and the witness might be mistaken.

Combine Evidence Using Dempster’s Rule

Dempster’s Rule of Combination:

$$m(A) = \frac{\sum_{X \cap Y = A} m_1(X) \cdot m_2(Y)}{1 - K}$$
$$m(A) = 1 - K$$

where **K** represents the **conflict** between the sources.

Step 1: Compute Raw Beliefs

Multiplying mass functions for **all possible intersections**:

Combined Hypothesis	Computation
Brushed Teeth (B)	$(0.7 \times 0.6) + (0.7 \times 0.2) + (0.6 \times 0.2) = 0.42 + 0.14 + 0.12 = 0.68$ $(0.7 \times 0.6) + (0.7 \times 0.2) + (0.6 \times 0.2) = 0.42 + 0.14 + 0.12 = 0.68$
Did NOT Brush (~B)	$(0.1 \times 0.2) = 0.02$ $(0.1 \times 0.2) = 0.02$
Uncertainty (U)	$(0.2 \times 0.2) + (0.2 \times 0.1) = 0.04 + 0.02 = 0.06$ $(0.2 \times 0.2) + (0.2 \times 0.1) = 0.04 + 0.02 = 0.06$

Combined Hypothesis	Computation
---------------------	-------------

Conflict (K)	$(0.7 \times 0.2) + (0.1 \times 0.6) = 0.14 + 0.06 = 0.2$ $(0.7 \times 0.2) + (0.1 \times 0.6) = 0.14 + 0.06 = 0.2$
---------------------	--

Step 2: Normalize by (1 - Conflict)

$K = 0.2$

Final adjusted beliefs:

- **Brushed Teeth (B) = $0.68 / (1 - 0.2) = 0.85$**
- **Did NOT Brush (~B) = $0.02 / (1 - 0.2) = 0.025$**
- **Uncertainty (U) = $0.06 / (1 - 0.2) = 0.075$**

Final Result:

✅ **85% belief** that the person **brushed their teeth**

❌ **2.5% belief** that they **did NOT brush**

👉 **7.5% uncertainty** remains

Logic Concepts: First order logic, inference in first order logic, propositional vs first order inference, unification and lift, forward chaining, backward chaining, resolution, learning from observation, inductive learning, decision trees, explanation based learning, statistical learning methods, reinforcement learning.

Propositional logic in AI:

PL is the simplest form of logic where all the statements are made by propositions

A proposition is a declarative statement which is either true or false, not both. It is a technique of knowledge representation in logical and mathematical form.

Ex:- 1. It is a Sunday

2. The sun rises from west [false proposition]

3. $3 + 3 = 7$ [false proposition]

4. 5 is a prime number [True]

Some basic facts about PL:

* PL is also called Boolean logic as it works on

0 or 1

* In PL we use symbolic variables to represent the logic and we can use any symbol for representing a proposition such as A, B, C, P, Q, R etc.

* Propositions can be either true or false but it can't be both

- * PL consists of an object, relation or function and logical connectives. these connectives are also called logical operators.
- * A propositional formula which is always true is called Tautology and it is also a valid statement / sentence.
- * A propositional formula which is always false is called Contradiction.
- * A propositional formula which has both true and false values is called Contingency.
- * Statements which are questions, commands or options are not propositions such as
 1. where is Rohini?
 2. How are you?
 3. What is your name?
 are not propositions.
- * Propositional logic are of 2 types

Syntax

- * It should be in proper structure and should be present in library

Ex: printf(" "); ✓

printf(" "); ✗

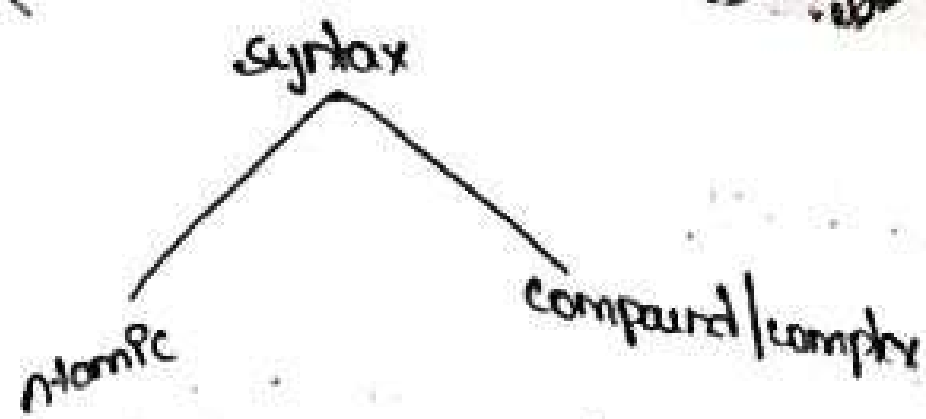
2 + 2 = 4 ✓

22 + = ✗

Semantic

- * Semantic means it should be in a proper meaningful manner, the sentence should have a proper meaning

- * The syntax is further divided into 2 types



Atomic: Atomic propositions consist of a single proposition symbol. These are the sentences which must be either true or false

examples: 1. $2 + 2 = 4$ is an atomic proposition as it is a true fact.
2. The sun is cold is also a proposition as it is false fact

Compound Proposition: Compound proposition constructed by combining simpler or atomic proposition using logical connectives

Ex: 1. It is raining and street is wet
2. Ankit is a Doctor and his clinic is in Chennai

Logical connective (or) operator :-

Connective Symbol	word	Technical term	example
\wedge	AND	conjunction	$P \wedge q$
\vee	OR	Disjunction	$P \vee q$
\rightarrow	IMPLIES	Implication (if)	$P \rightarrow q$
\leftrightarrow / \Leftrightarrow	IF and only IF	Bi-condition (iff)	$P \leftrightarrow q$
\neg / \leftarrow / \sim	NOT	Negation	$\sim p$

1. Conjunction :-

Ex:- Rohan is intelligent and hardworking

Here, P = Rohan is intelligent

q = Rohan is hardworking then we can represent the above sentence using conjunction as follows.

" $P \wedge q$ "

2. Disjunction :-

Ex:- Ritika is a doctor or Engineer

if P = Ritika is a doctor

q = Ritika is an Engineer then we can represent the above sentence as

" $P \vee q$ "

3. Negation :-

Ex:- Geetha is not a singer

if P = Geeta is a singer then negation ($\sim P$ = Geeta is not a singer. " $\sim p$ "

1. Implication (if) :-

ex:- if it is raining then the street is wet

$$P \rightarrow q$$

5. Bi-condition (iff) :-

ex:- I am breathing then I am alive

$$P \leftrightarrow q$$

(or)

$$P \iff q$$

Truth Table :-

1) Negation :-

P	$\neg P$
T	F
F	T

2) Conjunction

P	q	$P \wedge q$
T	T	T
T	F	F
F	T	F
F	F	F

3) Disjunction

P	q	$P \vee q$
T	T	T
T	F	T
F	T	T
F	F	F

4) Implication

P	q	$P \rightarrow q$
T	T	T
T	F	F
F	T	T
F	F	T

5) Bi-conditional

P	q	$P \leftrightarrow q$
T	T	T
T	F	F
F	T	F
F	F	T

1) Commutative :

$$P \wedge q \Rightarrow q \wedge P$$

$$P \vee q \Rightarrow q \vee P$$

2) Identity Element :

$$P \wedge \text{True} = P$$

$$P \vee \text{True} = P$$

3) Associativity :

$$(P \wedge q) \wedge r = P \wedge (q \wedge r)$$

$$(P \vee q) \vee r = P \vee (q \vee r)$$

1) It is raining

RAINING

2) It is hot

HOT

3) It is sunny

SUNNY

4) It is windy

WINDY

4) Distributive :-

$$P \wedge (q \vee r) = (P \wedge q) \vee (P \wedge r)$$

$$P \vee (q \wedge r) = (P \vee q) \wedge (P \vee r)$$

5) De - Morgan's law :-

$$\neg(P \wedge q) = \neg P \vee \neg q$$

$$\neg(P \vee q) = \neg P \wedge \neg q$$

6) Double negation Elimination:

$$\neg(\neg P) = P$$

5) If it is raining then it is not sunny

RAINING \rightarrow \sim SUNNY

It is humid

HUMID

It is cold

COLD

It is humid then it is not cold

HUMID \rightarrow \sim COLD

It is humid then it is hot

HUMID \rightarrow HOT

It it is raining then it is hot

RAINING \rightarrow HOT

It it is not humid then it is not raining

\sim HUMID \rightarrow \sim RAINING

If I am breathing then I am alive

BREATHING \leftrightarrow ALIVE

Limitations of Propositional logic :-

1. PL is not sufficient to represent the natural language statement
2. In PL we have seen that how to represent stmt using PL.
3. By unfortunately in PL we can only represent the fact which are either true or false.

4) PL has limit and expressive power they can't be expressed.

5) we need a language that allows us to describe Properties or Predicate of objective or a relationship among object represented by the variable.

Q the sentence which PL can't express are as follows

i) All the girls

ii) Some Apple or Sweet

iii) Sachin likes cricket

∴ To overcome from these advantages a concept is introduced i.e., Predicate logic:

Predicate logic satisfies the requirement of the language.

Predicate logic is powerful enough for expression and reasoning. It build upon the idea of PL

First order logic - Predicate logic.

→ first order logic [FOL] is known as FOL. FOL is another way of knowledge representation in AI. It is an extension to PL.

* FOL is sufficiently expressive to represent the natural language statement in a concise

* FOL is a powerful language that developed inference about the objects in a more easy way and can also in the relationship between those objects.

*FOL [like natural language] doesn't only assume that the world contain facts like PL but also assume object, relation, in the world.

→ FOL is of two types

1. Syntax
2. Symmantic / semimantic

1. Syntax of FOL :-

The basic syntactic elements of FOL are symbols.

Basic elements of FOL :-

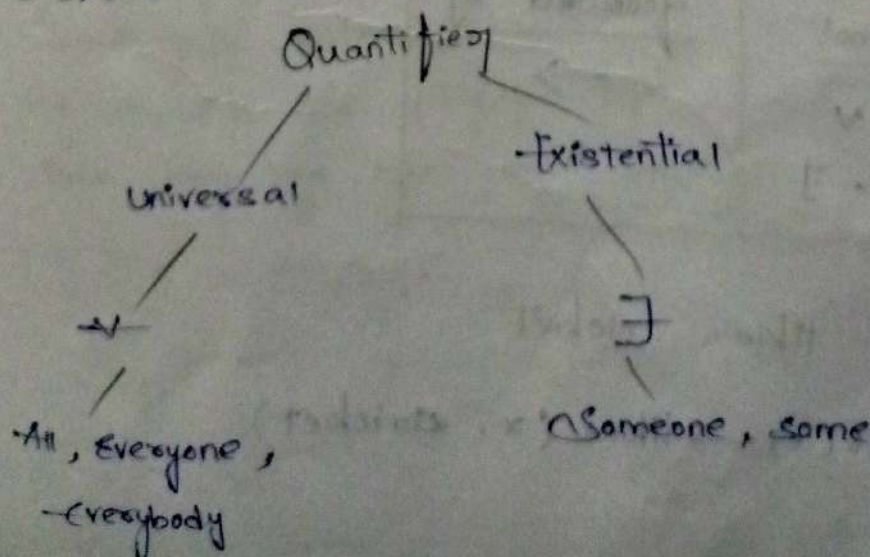
1. Constant - 1, 2, A, John, Mumbai
2. Variables - x, y, z, a, b
3. Predicates - Brother, Sister, likes,
4. Functions - mapping, father-of, sister-of, mother-of
5. Connectives - \wedge , \vee , \neg , \rightarrow , \Leftrightarrow
6. Equality - $=$

Syntax :-

Function (term₁, term₂ ... term_n)

Predicate (term₁, term₂ ... term_n)

Quantifiers



Example :-

1. Sam is tall
S P

tall (Sam)

2. Roses are red
S P

Red (Roses)

3. Pencil is very sharp
S P

Sharp (pencil)

4. Students are very Active
S P

Active (Students)

5. Chintu is a dog
S P

dog (Chintu)

With Action words:-

1. John likes tea

likes (John, tea)

3. Raju, watching TV

Watching (Raju, TV)

2. Madhu playing cricket

Playing (Madhu, cricket)

4. Kisan reading books

reading (Kisan, books)

For Quantifiers and some we have to follow this table:

Symbol	Followed by
All - \forall	\rightarrow
Some - \exists	\wedge

1) All Students like cricket

$\forall x$ Students (x) \rightarrow likes (x, cricket)

1. All girls like roses

girls(x)

All (x)

$\forall x \text{ girls}(x) \rightarrow \text{likes roses}(x, \text{roses})$

2. Everyone likes Biryani

$\forall x \text{ everyone}(x) \text{ likes Biryani}$

$\forall x \text{ everyone}(x) \rightarrow \text{likes}(x, \text{Biryani})$

3. Every body is having fun

$\forall x \text{ everybody}(x) \rightarrow (x, \text{fun})$

4. Some students like cricket

$\exists x \text{ students}(x) \wedge \text{likes}(x, \text{cricket})$

5. Some girls like roses

$\exists x \text{ girls}(x) \wedge \text{likes}(x, \text{roses})$

Examples:-

1. Mary was a woman?

2. Radha is a musician?

3. All Musicians were women?

4. Some boys like football

5. Some girls hate football

6. All students are playing chess

7. Some body boys are not intelligent

8.

1) Women (Mary)

2) Musician (Radha)

3) $\forall x$ Musicians $(x) \rightarrow$ were (x, Romans)

4) $\exists x$ boys $(x) \wedge$ like $(x, \text{football})$

5) $\exists x$ girls $(x) \wedge$ hate $(x, \text{football})$

6) $\forall x$ students $(x) \rightarrow$ Playing (x, chess)

7) $\exists x$ boys $(x) \wedge$ are $(x, \text{unintelligent})$

Statement.

1) Everyman respect his parents

$\forall x$ man $(x) \rightarrow$ respect $(x, \text{parents})$

2. only one student failed in maths

$\exists x$ students $(x) \wedge$ failed (x, maths)

3. There exist a student students

$\exists x$ student $(x) \wedge$ Smart (x) .

Types \exists (Either - or)

4. Ram takes either mathematics or physics

A. Ram takes mathematics or Ram takes physics

takes $(\text{Ram}, \text{mathematics}) \vee$ takes $(\text{Ram}, \text{physics})$

2) ~~At~~

Shyam likes playing cricket or football

Shyam likes playing cricket (or) Shyam likes playing football

Playing (Shyam)

3. All Romans were either loyal to Caesar or hated him.

→ All Romans were loyal to Caesar (or) All Romans were hated him.

$\forall x \text{ romans}(x) \rightarrow \text{loyal}(x, \text{caesar}) \vee \forall x \text{ romans}(x) \rightarrow \text{hated}(x, \text{caesar})$

$\forall x \text{ romans}(x) \rightarrow \text{loyal}(x, \text{caesar}) \vee \text{hated}(x, \text{caesar})$

4. Some students were intelligent in maths or poor in it.

$\exists x \text{ students}(x) \rightarrow \text{intelligent}(x, \text{maths}) \vee \text{poor}(x, \text{maths})$

4- Method (Neither - NOR):-

$\text{Not} = \neg$

1. Ram takes neither maths nor physics

$\neg \text{takes}(\text{Ram}, \text{maths}) \wedge \neg \text{takes}(\text{Ram}, \text{physics})$

2. Romans were neither joined in music class nor in poetry class.

$\neg \text{joined}(\text{Romans}, \text{music class}) \wedge \neg \text{joined}(\text{Romans}, \text{poetry class})$

3. Ram takes coffee if and only if Ram doesn't take tea.

$\text{takes}(\text{Ram}, \text{coffee}) \iff \neg \text{takes}(\text{Ram}, \text{tea})$

4. Ram and Shyam are brothers

$\text{brothers}(\text{Ram}, \text{shyam})$

Method - 5

① Some integers are even and some are odd.

$\exists x \text{ integers}(x) (x, \text{even}) \wedge \exists x \text{ integers}(x) (x, \text{odd})$

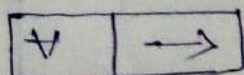
John loves Mary.

loves (John, Mary)

Universal Quantifiers :-

universal quantifiers is a symbol of logical representation which specifies that the statement within its range is true for everything or every instance of a particular things.

The universal quantifier is represented by a symbol 'V' in universal quantifiers we use implies symbol



Example :-

\rightarrow All mangoes are sweet

$\forall x \text{ mangoes}(x) \wedge \text{sweet}(x)$

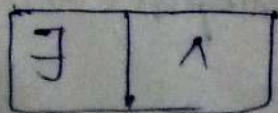
\Rightarrow All men drink coffee

$\forall x \text{ man}(x) \rightarrow \text{drink}(x, \text{coffee})$

Existential quantifiers :-

Existential quantifiers are the types of quantifiers which express that the statements within its scope is true for at least one instance of something.

It is denoted by the symbol there exist and it must follow and ' \wedge ' symbol.



Example :-

\Rightarrow Some human are lazy.

$$\exists x \text{ human}(x) \wedge \text{lazy}(x)$$

\Rightarrow Some boys are not intelligent

$$\exists x \text{ boys}(x) \wedge \neg \text{intelligent}(x).$$

Properties of quantifiers :-

1. $\forall x \forall y = \forall y \forall x$

2. $\exists x \exists y = \exists y \exists x$

3. $\forall x \exists y \neq \exists y \forall x$

Application of FOL

Applications of FOL :-

1. Knowledge Representation:

FOL provides robust framework for representing complex relationships and properties of object.

Ex: In a medical diagnosis system predicate can represent symptoms, disease and their relationship.

2. Automated theorem proving:

Automated theorem proving involves using algorithms to prove math-theorem. FOL provides the foundation structure of many theorems.

3. Natural language processing:-

In NLP, FOL is used to parse and understand natural language by representing the meaning of sentence in a formal, logical manner. This allows AI system to perform task such as Question, Answer, machine translation and text summarizing.

4. Expert System:

Expert System encodes Expert knowledge using FOL and reason about it to make decisions.

Unification :- Unification is an algorithm for determining the substitution to make two FOL expressions match.

To apply the rules of inference, an inference system must be able to determine when two expressions match. Here the unification algorithm is used.

The unify algorithm is used for unification which takes two atomic sentences and returns a unifier for those sentences. [It may exist]

Following are the basic conditions for unification.

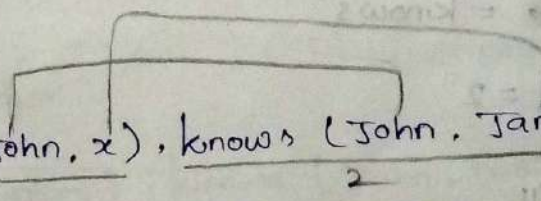
Step-1 :- Predicate symbol must be same, atoms, or expressions with different symbols can never be unified.

Step-2 :- Number of arguments in both expressions must be identical.

Step-3 :- Unification will fail if there are two similar variables present in the same expressions.

Example :-

Unify (knows(John, x), knows(John, Jane))



Predicate \rightarrow knows (both expressions are same)

argument \rightarrow 2 arguments

Substitution = $\theta (x / \text{Jane})$

Now,

Unify (knows(John, Jane),
knows(John, Jane))

Step-4 :- Substitution = $\theta (x / \text{Jane})$ is a unifier for this atom and applying this substitution to both expressions will be identical.

Example :- ②

Unify ($\text{knows}(\text{John}, x)$, $\text{knows}(y, \text{Jack})$)

1) Predicate = knows

2) argument = 2

3) x / John

$\text{knows}(\text{John}, x)$, $\text{knows}(\text{John}, \text{Jack})$

$\text{knows}(\text{John}, \text{Jack})$, $\text{knows}(\text{John}, \text{Jack})$

~~for~~

Substitution $\theta = \{x / \text{Jack}, y / \text{John}\}$.

Example ③ :-

Unify ($\text{knows}(x, \text{Raj})$, $\text{knows}(\text{Bill}, y)$)

1) Predicate = knows

2) argument = 2

3) x / Bill

Unify ($\text{knows}(\text{Bill}, \text{Raj})$, $\text{knows}(\text{Bill}, y)$)

$\text{knows}(\text{Bill}, \text{Raj})$, $\text{knows}(\text{Bill}, y)$
(y / Raj)

$\text{knows}(\text{Bill}, \text{Raj})$, $\text{knows}(\text{Bill}, \text{Raj})$

Substitutions $\theta = \{x / \text{Bill}, y / \text{Raj}\}$

Example (4) :-

unify (knows (John, x), knows (y, mother(y))).

$$\theta = \{y/John\}$$

→ Predicate = knows
Argument = 2

knows (John, x), knows (John, mother(John))

$$\theta = \{x/mother(John)\}$$

knows (John, mother(John)), knows (John, mother(John))

Example (5) :-

unify (knows (John, x), knows (x, mother(x)))

1, Predicate = knows

2, Argument = 2

$$3, \theta = x/John$$

knows (John, x), knows (John, mother(x))

$$\theta = x/mother$$

knows (John, mother), knows (John, mother)

these unification is fail.

Example (6) :-

unify { p(b, x, f(g(z))), p(z, f(y), f(y)) }

$$\theta = \{z/b\}$$

→ 1, predicate = p

2, Argument = 3

know { p(b, x, f(g(z))), p(b, f(y), f(y)) }

$$\theta = \{x/f(y)\}$$

{ p(b, f(y), f(g(b))), p(b, f(y), f(y)) }

$$\theta = \{y/g(b)\}$$

$$P(b, f(g(b), f(g(b))), P(b, f(g(b), f(g(b))))$$

$$\text{Substitution } \theta = \{z/b, x/f(y), y(g(b))\}$$

The above Expression is successfully unified.

Example ⑦:-

$$Q(a, g(x, a), f(y)), Q(a, g(f(b), a), x))$$

$$1, \text{ Predicate} = Q$$

$$2, \text{ Argument} = 3$$

$$3, \theta = \{x/f(b)\}$$

$$Q(a, g(f(b), a), f(y)), Q(a, g(f(b), a), f(b))$$

$$4, \theta = \{y/b\}$$

$$Q(a, g(f(b), a), f(b)), Q(a, g(f(b), a), f(b))$$

The above Expression is successfully unified.

Example ⑧:-

$$P(f(a), g(y)), P(x, x)$$

Example ⑨:-

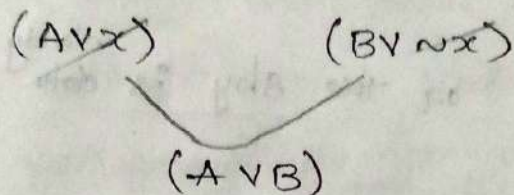
$$P(f(x), x), Q(f(x), x)$$

Example ⑩:-

$$P(g(b), y), Q(y)$$

Resolution:-

The resolution inference take two premises in the form of clauses $(A \vee x)$ $(B \vee \neg x)$ and gives $(A \vee B)$ as a conclusion.



Steps (or) Rules for resolution:-

1. Convert the English statement into first order logic
2. Convert FOL into CNF
3. Apply negative to what's need to be proven.
4. Draw resolution graph/tree.

Rules to convert FOL to CNF

$$1) x \rightarrow y \Rightarrow \neg x \vee y$$

$$2) x \wedge y \rightarrow z \Rightarrow \neg(x \wedge y) \vee z$$
$$\neg x \vee \neg y \vee z$$

$$3) \forall a \text{ dog}(a) \Rightarrow \text{dog}(a)$$

Consider the following facts and ^{prove} ~~have~~ the resolution for the facts.

1) the humidity is high or the sky is cloudy

2) If the sky is cloudy then it will rain

3) If the humidity is high, then it is hot

4) It is not hot

Prove:- It will rain

P = the humidity is high

Q = the sky is cloudy

R = It will rain

S = it is hot

a. the humidity is high or the sky is ^{cloudy} ~~data~~ $\Rightarrow P \vee Q$
 P S

b. It the sky is cloudy then it will rain $\Rightarrow Q \rightarrow R$

c. If the humidity is high, then it is hot $\Rightarrow P \rightarrow S$

d. It is hot $\rightarrow NS$

Convert FOL to CNF

1) $P \vee Q = P \vee Q$

2) $Q \rightarrow R = \neg Q \vee R$

3) $P \rightarrow S = \neg P \vee S$

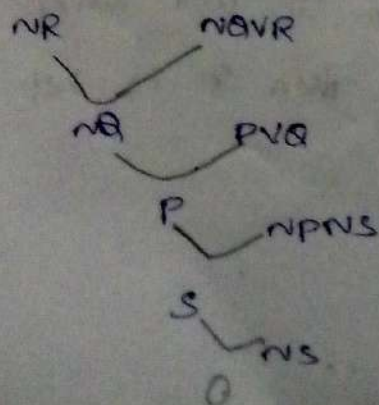
4) $NS = \neg S$

Step 3: Apply Negation to what's need to be proven

Prove: It will rain: R

$\Rightarrow \neg R$

Tree:



free search to null state which means NR. is false, which means R is true.

1. If it is Sunny and warm you will Enjoy $\Rightarrow P \wedge Q \rightarrow S$

2. If it is warm and pleasant day you will do strawberry picking. $\Rightarrow Q \wedge S \rightarrow T$

3. If it is raining then no strawberry picking. $\Rightarrow U \rightarrow \neg T$

4. If it is raining you will get wet $\Rightarrow U \rightarrow V$

5. It is warm day $\Rightarrow Q$

6. It is raining $\Rightarrow U$

7. It is Sunny $\Rightarrow P$

Qwe want to solve by Refutation you are not doing strawberry picking

② you will Enjoy.

$\Rightarrow P = \text{It is Sunny}$

S: It is sunny

w: warm

Q = warm

S = you will Enjoy

Q = It is warm

T = you will do strawberry picking

S = Pleasant

U = it is raining

V = will get wet

e: enjoy

P: pleasant

Sp: strawberry picking

R: Raining

wt: wet

convert FOL to CNF

1) $S \wedge W \rightarrow S$

2) $W \wedge P \rightarrow SP$

3) $R \rightarrow \neg SP$

4) $R \rightarrow wt$

5) W

6) P

CNF

$\neg(S \wedge W) \vee S \Rightarrow \neg S \vee \neg W \vee S$

$\neg(W \wedge P) \vee SP \Rightarrow \neg W \vee \neg P \vee SP$

NR $\neg V \neg SP$

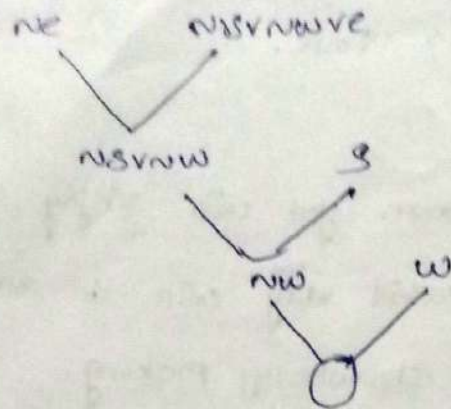
NR $V \wedge wt$

W

R

S

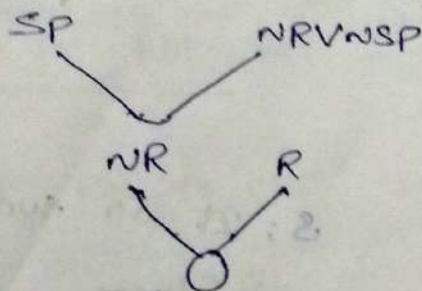
②



Tree reached to null state which means ~~ne~~^{ne} is false
which means e is true

① NSP

$$N(NSP) \Rightarrow SP$$



tree reached to null state which means SP is false
which means NSP is true.

$$1) P(f(a), g(y)), P(x, x)$$

$$1) \text{ Predicate} = P$$

$$2) \text{ Arguments} = 2$$

$$3) \theta = \{f(a), x\}$$

$$P(x, g(y)), P(x, x)$$

$$\theta = \{g(y), x\}$$

$$P(x, x), P(x, x)$$

\therefore the unification is fail.

$AP(f(x, x), Q(f(x), x))$

predicate = x

∴ the unification is failed.

$AP(g(b), y), Q(y)$

predicate = x

∴ the unification is failed.

Learning from Observation:-

the components are 1. Environment.

2. Sensors

3. Performance Element

4. Performance Standards

5. Critics

6. Learning elements

7. Problem generator

the learnings are of four types

1. Supervised learning

2. Unsupervised learning

3. Reinforcement learning

4. Semi supervised learning.

1. Supervised learning:-

* we train the data by given both input and output

Example :- election exit poll

In that exit poll we perform classification i.e.,

win or lose.

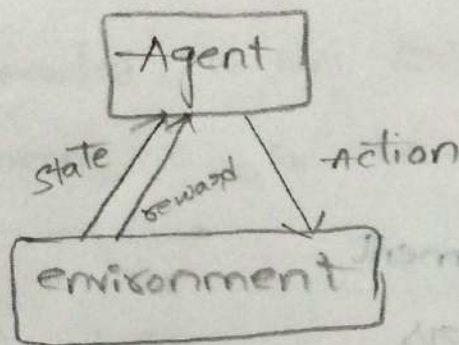
The algorithm which we use is naive bayes algorithm.

2) Unsupervised learning:-

Here we are providing only input based on that input we perform ~~the~~ clustering. the algorithm which used in * - means algorithm.

3) ~~Reinforcement~~ learning

3) Reinforcement learning



Here reinforcement learning mostly depends upon rewards. Point it take it as positive feedback in some situation if it gets penalty it will take it as negative feedback and try to change penalty from reward (negative to positive) from experience. * the algorithm which used is Q - learning.

Key terms

What is data?

- Gathering information
- Measuring information
- From different sources.
- This is **evidence**



Key terms

What is hypothesis?

- Explanation made on the basis of limited evidence
- From starting point for further investigation.



Candy bags Example

- **Candy comes in two flavors**

Candy bags Example

- Candy comes in two flavors



Cherry



Candy bags Example

- Candy comes in two flavors



Cherry

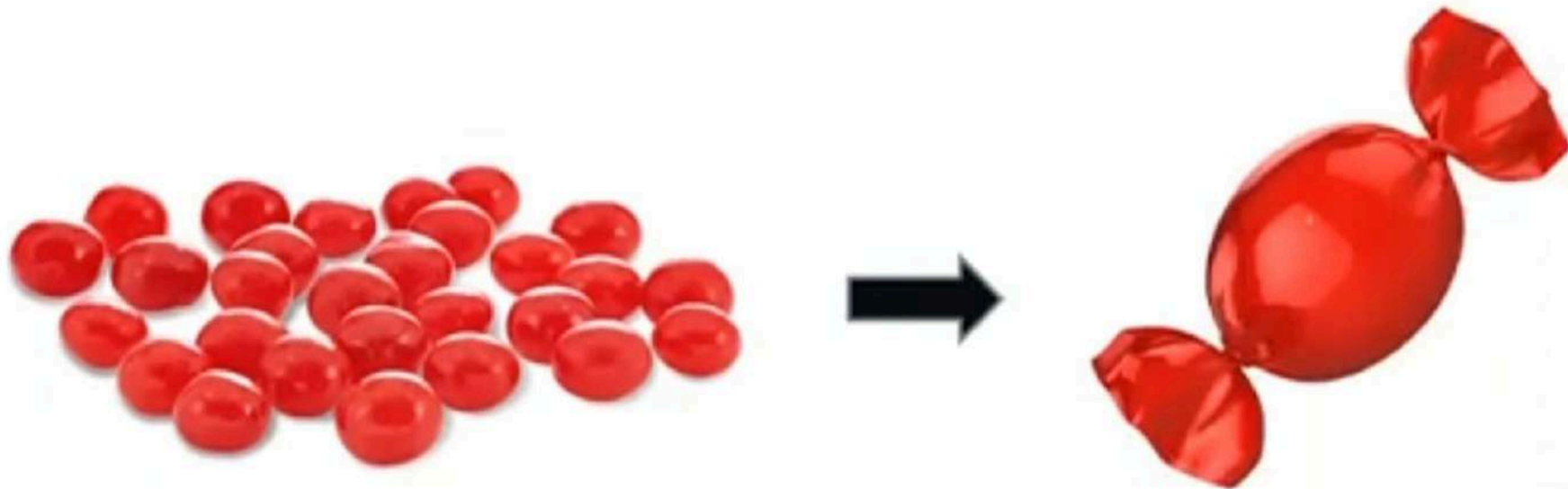


Lime



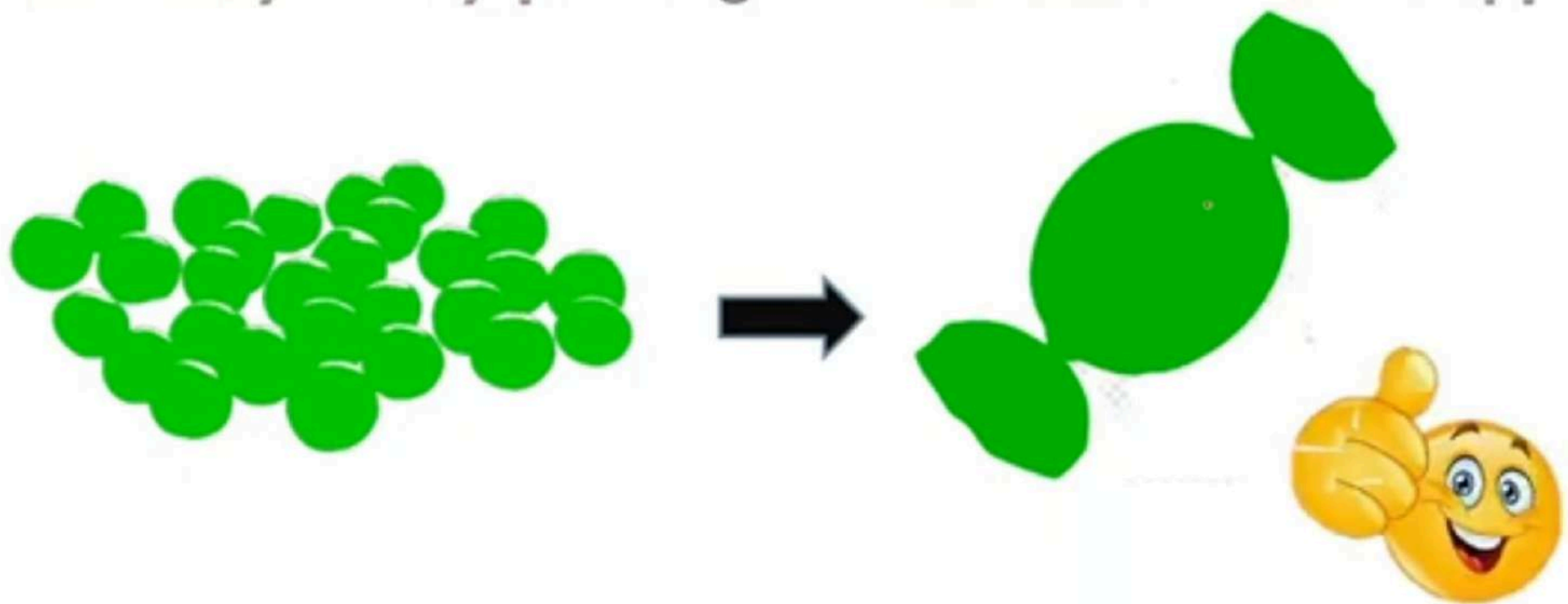
Candy bags Example

Normally candy packing is with same color wrapper



Candy bags Example

Normally candy packing is with same color wrapper

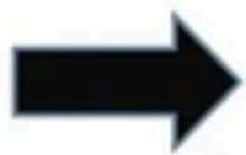


Candy bags Example

The manufacturer wraps each piece of candy in the same solid wrapper, regardless of flavor

Candy bags Example

The manufacturer wraps each piece
of candy in the same solid wrapper, regardless of flavor



Candy bags Example

**The candy is sold in very large bags
(Types of bags : 5)**



Candy bags Example

h1:
100% cherry



h2:
75% cherry
+
25% lime



h3:
50% cherry
+
50% lime



h4:
25% cherry
+
75% lime



h5:
100% lime



Candy bags Example

**For a new bag the random variable H (Hypothesis).
 H is not directly observable**





Candy bags Example

**Assume all candies are in a solid paper
(Not shown in the diagram)**



Candy bags Example

As the pieces of candy are opened and inspected, data are revealed



Candy bags Example

As the pieces of candy are opened and inspected, data are revealed (D_1, D_2, \dots)



Candy bags Example

As the pieces of candy are opened and inspected, data are revealed (D_1, D_2, \dots)



Candy bags Example

As the pieces of candy are opened and inspected, data are revealed ($D1, D2, \dots$)



D1



D2



Candy bags Example

As the pieces of candy are opened and inspected, data are revealed (D1,D2....)



D1



D2



D3



Candy bags Example

As the pieces of candy are opened and inspected, data are revealed ($D1, D2, \dots$)



D1



D2



D3



It's a Cherry bag (h1)

Candy bags Example

As the pieces of candy are opened and inspected, data are revealed



Candy bags Example

As the pieces of candy are opened and inspected, data are revealed



Candy bags Example

As the pieces of candy are opened and inspected, data are revealed



D1



D2



D3



Candy bags Example

As the pieces of candy are opened and inspected, data are revealed



D1



D2



D3



It's a Cherry bag (h1)

Candy bags Example

As the pieces of candy are opened and inspected, data are revealed



D1



D2



D3



It's a Cherry bag (h1)



Candy bags Example

**For this issue we need to learn probability
of each hypothesis
(i.e., h_1, h_2, h_3, h_4, h_5) using**

Bayesian Learning

Bayesian Learning

Let **D** represent all the data, with observed value **d**; then the probability of each hypothesis is obtained by Bayes' rule:

$$P(h_i | d) = \alpha P(d | h_i) P(h_i)$$

Hypothesis Prior

Manufacturing company announces that

10%
i.e., $P(h_{100}) = 0.1$

20%
i.e., $P(h_{75}) = 0.1$

50%
i.e., $P(h_{50}) = 0.2$

20%
i.e., $P(h_{25}) = 0.2$

10%
i.e., $P(h_0) = 0.1$



The prior distribution over h_1, \dots, h_5 is given by $\langle 0.1, 0.2, 0.4, 0.2, 0.1 \rangle$

Likelihood

likelihood of the data under each hypothesis, $P(\mathbf{d} | h_i)$.

likelihood of the data is calculated under the assumption that the observations are i.i.d.

(Independently and identically distributed)


$$P(\mathbf{d} | h_i) = \prod_j P(d_j | h_i) .$$

Likelihood

d1	d2	d3	d4	d5	d6	d7	d8





Likelihood

d1	d2	d3	d4	d5	d6	d7	d8
							






Likelihood

d1	d2	d3	d4	d5	d6	d7	d8
							



Likelihood

d1	d2	d3	d4	d5	d6	d7	d8
							



Likelihood

d1	d2	d3	d4	d5	d6	d7	d8
							



Likelihood

d1	d2	d3	d4	d5	d6	d7	d8
							



Likelihood

d1	d2	d3	d4	d5	d6	d7	d8
							



Likelihood

h₁

d1	d2	d3	d4	d5	d6	d7	d8
							



Likelihood

d1	d2	d3	d4	d5	d6	d7	d8
							



Likelihood

d1	d2	d3	d4	d5	d6	d7	d8
							



Likelihood

d1	d2	d3	d4	d5	d6	d7	d8
							



Likelihood

h4

d1	d2	d3	d4	d5	d6	d7	d8
							



UNIT-5

Expert Systems

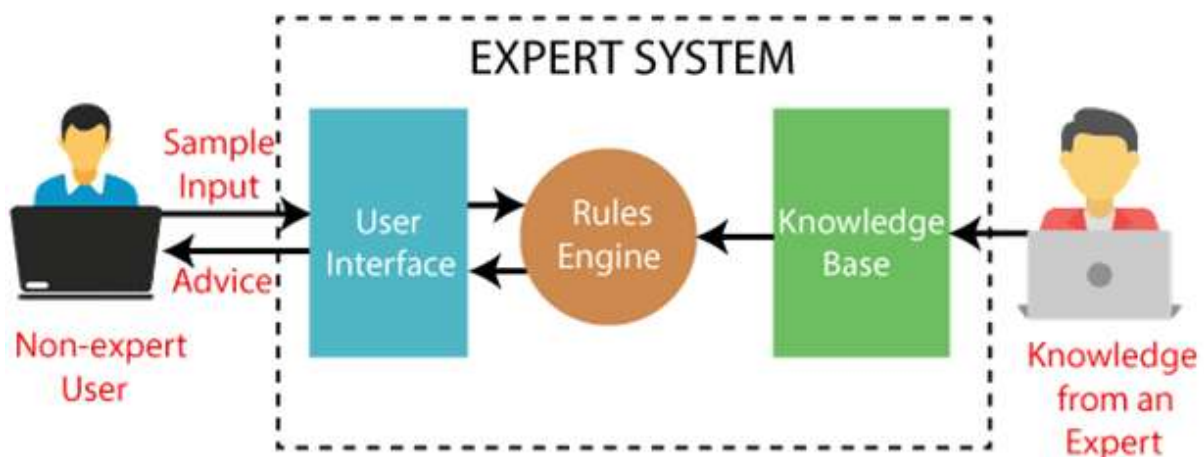
What is an Expert System?

An expert system is a computer program that is designed to solve complex problems and to provide decision-making ability like a human expert. It performs this by extracting knowledge from its knowledge base using the reasoning and inference rules according to the user queries.

The expert system is a part of AI, and the first ES was developed in the year 1970, which was the first successful approach of artificial intelligence. It solves the most complex issue as an expert by extracting the knowledge stored in its knowledge base. The system helps in decision making for complex problems using **both facts and heuristics like a human expert**. It is called so because it contains the expert knowledge of a specific domain and can solve any complex problem of that particular domain. These systems are designed for a specific domain, such as **medicine, science**, etc.

The performance of an expert system is based on the expert's knowledge stored in its knowledge base. The more knowledge stored in the KB, the more that system improves its performance. One of the common examples of an ES is a suggestion of spelling errors while typing in the Google search box.

Below is the block diagram that represents the working of an expert system:



Below are some popular examples of the Expert System:

- **DENDRAL:** It was an artificial intelligence project that was made as a chemical analysis expert system. It was used in organic chemistry to detect unknown organic molecules with the help of their mass spectra and knowledge base of chemistry.
- **MYCIN:** It was one of the earliest backward chaining expert systems that was designed to find the bacteria causing infections like bacteraemia and meningitis. It was also used for the recommendation of antibiotics and the diagnosis of blood clotting diseases.
- **PXDES:** It is an expert system that is used to determine the type and level of lung cancer. To determine the disease, it takes a picture from the upper body, which looks like the shadow. This shadow identifies the type and degree of harm.
- **CaDeT:** The CaDet expert system is a diagnostic support system that can detect cancer at early stages.

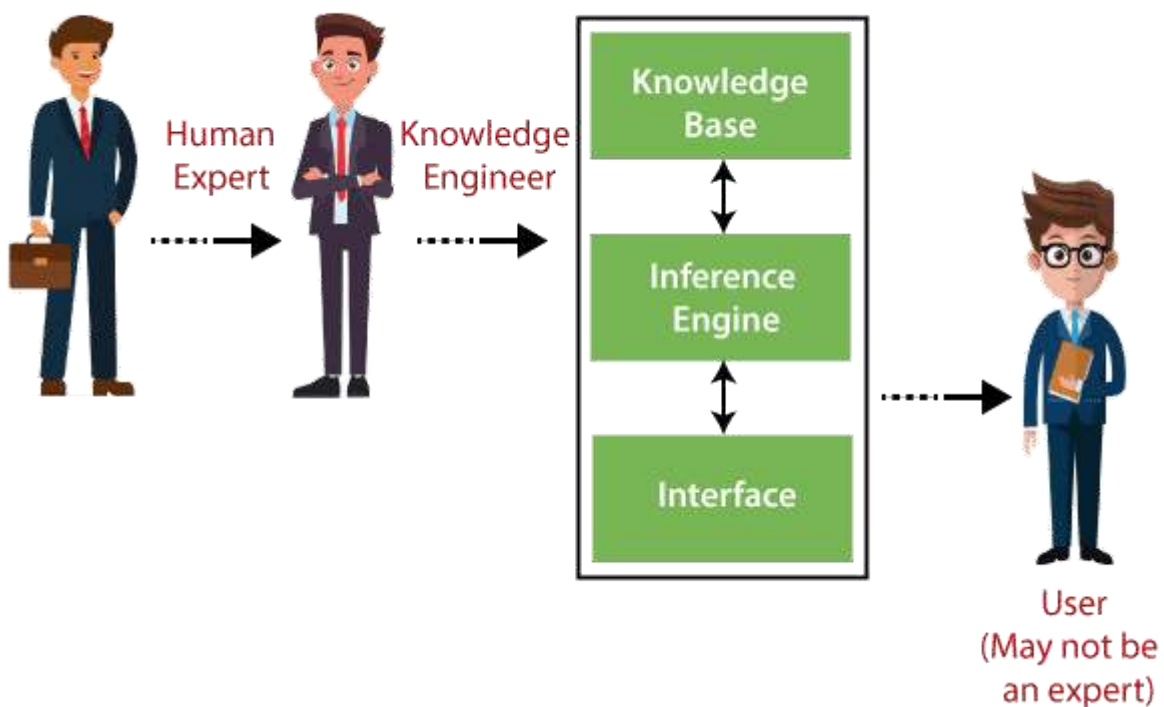
Characteristics of Expert System

- **High Performance:** The expert system provides high performance for solving any type of complex problem of a specific domain with high efficiency and accuracy.
- **Understandable:** It responds in a way that can be easily understandable by the user. It can take input in human language and provides the output in the same way.
- **Reliable:** It is much reliable for generating an efficient and accurate output.
- **Highly responsive:** ES provides the result for any complex query within a very short period of time.

Components of Expert System

An expert system mainly consists of three components:

- **User Interface**
- **Inference Engine**
- **Knowledge Base**



1. User Interface

With the help of a user interface, the expert system interacts with the user, takes queries as an input in a readable format, and passes it to the inference engine. After getting the response from the inference engine, it displays the output to the user. In other words, **it is an interface that helps a non-expert user to communicate with the expert system to find a solution.**

2. Inference Engine(Rules of Engine)

- The inference engine is known as the brain of the expert system as it is the main processing unit of the system. It applies inference rules to the knowledge base to derive a conclusion or

deduce new information. It helps in deriving an error-free solution of queries asked by the user.

- With the help of an inference engine, the system extracts the knowledge from the knowledge base.
- There are two types of inference engine:
- **Deterministic Inference engine:** The conclusions drawn from this type of inference engine are assumed to be true. It is based on **facts** and **rules**.
- **Probabilistic Inference engine:** This type of inference engine contains uncertainty in conclusions, and based on the probability.

Inference engine uses the below modes to derive the solutions:

- **Forward Chaining:** It starts from the known facts and rules, and applies the inference rules to add their conclusion to the known facts.
- **Backward Chaining:** It is a backward reasoning method that starts from the goal and works backward to prove the known facts.

3. Knowledge Base

- The knowledgebase is a type of storage that stores knowledge acquired from the different experts of the particular domain. It is considered as big storage of knowledge. The more the knowledge base, the more precise will be the Expert System.
- It is similar to a database that contains information and rules of a particular domain or subject.
- One can also view the knowledge base as collections of objects and their attributes. Such as a Lion is an object and its attributes are it is a mammal, it is not a domestic animal, etc.

Components of Knowledge Base

- **Factual Knowledge:** The knowledge which is based on facts and accepted by knowledge engineers comes under factual knowledge.
- **Heuristic Knowledge:** This knowledge is based on practice, the ability to guess, evaluation, and experiences.

Knowledge Representation: It is used to formalize the knowledge stored in the knowledge base using the If-else rules.

Knowledge Acquisitions: It is the process of extracting, organizing, and structuring the domain knowledge, specifying the rules to acquire the knowledge from various experts, and store that knowledge into the knowledge base.

Expert Systems



- **Expert Systems** solves problems that are normally solved by **human experts**
- An **expert system** is a **computer program** that represents and reasons with **knowledge of some specialist subject** with a view to solving problems or giving advice.
- To solve expert-level problems, expert systems will need efficient access to a substantial **domain knowledge base** which must be built as efficiently as possible
- They also need to exploit **one or more reasoning mechanisms** to apply their knowledge to the problems they are given.
- They will also need to be able to **explain**, to the users who rely on them, how they have reached their decisions.

Typical tasks for expert systems



Some typical existing expert system tasks include:

1. The interpretation of data Such as sonar data or geophysical measurements
2. Diagnosis of malfunctions Such as equipment faults or human diseases
3. Structural analysis or configuration of complex objects Such as chemical compounds or computer systems
4. Planning sequences of actions Such as might be performed by robots
5. Predicting the future Such as weather, share prices, exchange rates

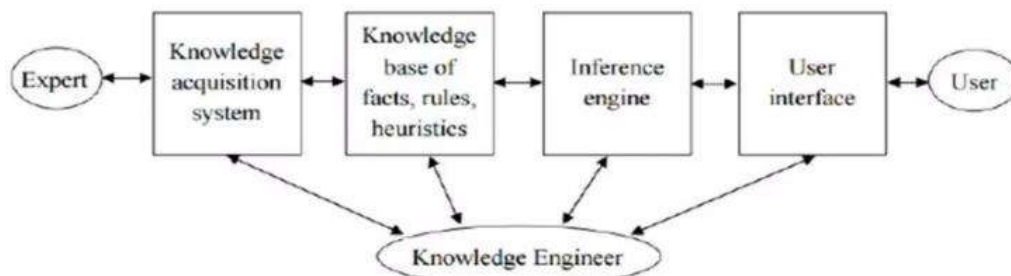
Characteristics of Expert Systems



1. They **simulate human reasoning** about the problem domain, rather than simulating the domain itself.
2. They perform **reasoning over representations of human knowledge**, in addition to doing numerical calculations or data retrieval. They have corresponding distinct modules referred to as the inference engine and the knowledge base.
3. **Problems tend to be solved using heuristics** (rules of thumb) or approximate methods or probabilistic methods which, unlike algorithmic solutions, are not guaranteed to result in a correct or optimal solution.
4. They usually have to **provide explanations** and justifications of their solutions in order to convince the user that their reasoning is correct.

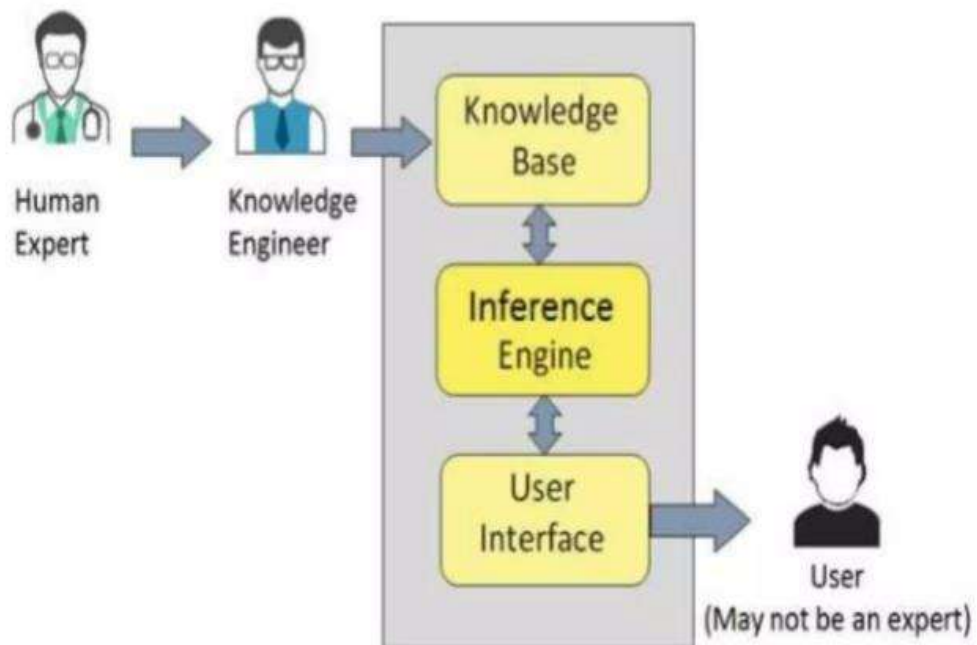
The term **Intelligent Knowledge Based System (IKBS)** is sometimes used as a synonym for Expert System

Architecture of Expert Systems



- The process of building expert systems is often called **knowledge engineering**.
- The **knowledge engineer** is involved with all components of an expert system:
- Building expert systems is generally an **iterative process**.
- The components and their interaction will be refined over the course of numerous meetings of the knowledge engineer with the experts and users.

Architecture



Knowledge acquisition



The success of any expert system majorly depends on the **quality, completeness, and accuracy of the information** stored in the **knowledge base**.

The **knowledge acquisition** component allows the expert to enter their knowledge or expertise into the expert system, and to refine it later as and when required.

Historically, **the knowledge engineer** played a major role in this process.

The knowledge acquisition process is usually comprised of three principal stages:

- **Knowledge elicitation** is the interaction between the expert and the knowledge engineer/program to elicit the expert knowledge in some systematic way.
- The knowledge thus obtained is usually stored in some form of human friendly **intermediate representation**.
- The intermediate representation of the knowledge is then compiled into an **executable form** (e.g. production rules) that the inference engine can process.



knowledge elicitation



It consists of several stages:

1. Find as much as possible about the problem and domain from books, manuals, etc. In particular, become familiar with any specialist terminology.
2. Try to characterize the types of reasoning and problem solving tasks that the system will be required to perform.
3. Find an expert (or set of experts) that is willing to collaborate on the project. Sometimes Interview the expert (usually many times during the course of building the system). Find out how they solve the problems your system will be expected to solve.
4. Have them check and refine your intermediate knowledge representation.

Inference Engine



It is the **brain** of the Expert System.

1. **Match the premise patterns** of the rules against elements in the working memory. Generally the rules will be domain knowledge built into the system, and the working memory will contain the case based facts entered into the system, plus any new facts that have been derived from them.

2. If there is more than one rule that can be applied, use a **conflict resolution strategy** to choose one to apply. Stop if no further rules are applicable.

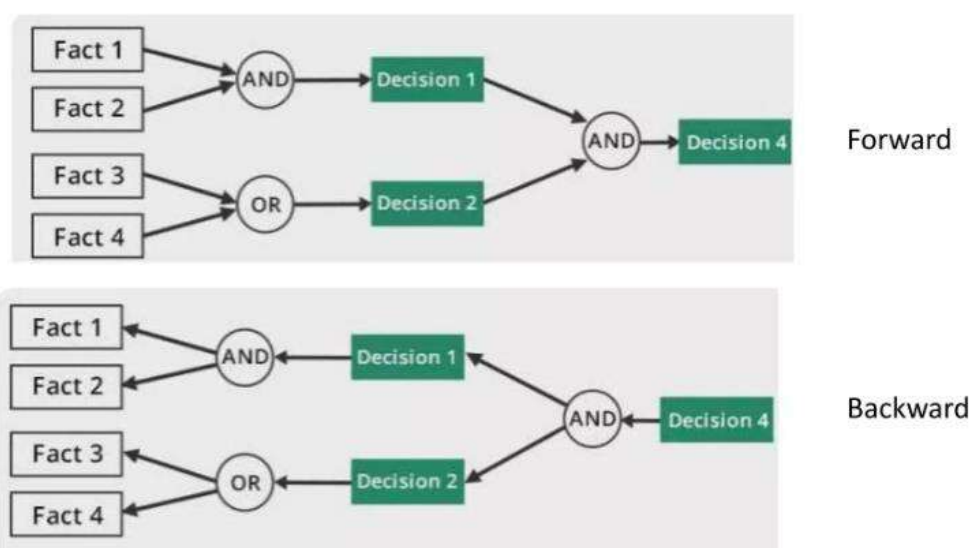
3. **Activate the chosen rule**, which generally means adding/deleting an item to/from working memory. Stop if a terminating condition is reached, or return to step 2.

- Early production systems spent over 90% of their time doing **pattern matching**.

Chaining



To recommend a solution, the Inference Engine uses the following strategies –Forward Chaining, Backward Chaining



Forward Chaining



- Forward Chaining -**What can happen next?** Here, the Inference Engine follows the chain of conditions and derivations and finally deduces the outcome.
- It considers all the facts and rules, and sorts them before concluding to a solution.
- This strategy is followed for working on conclusion, result, or effect.
- For example, **prediction of share market status** as an effect of changes in interest rates.

Backward Chaining

Backward Chaining - An expert system finds out the answer to the question, "**Why this happened?**"

On the basis of what has already happened, the Inference Engine tries to find out which conditions could have happened in the past for this result.

This strategy is followed for **finding out cause or reason**. For example, diagnosis of blood cancer in humans.



User interface

The Expert System user interface usually comprises of two basic components:

1. **The Interviewer Component** : This controls the dialog with the user and/or allows any measured data to be read into the system. For example, it might ask the user a series of questions, or it might read a file containing a series of test results.
2. **The Explanation Component** : This gives the system's solution, and also makes the system's operation transparent by providing the user with information about its reasoning process.

Meta Knowledge

Meta knowledge can be simply defined as knowledge about knowledge.

Meta knowledge is knowledge about the use and control of domain knowledge in an expert system.



Roles in an Expert System

Three fundamental roles in building expert systems are:

1. Expert

- Successful ES systems depend on the experience and application of knowledge that the people can bring to it during its development. Large systems generally require multiple experts.

2. Knowledge engineer –

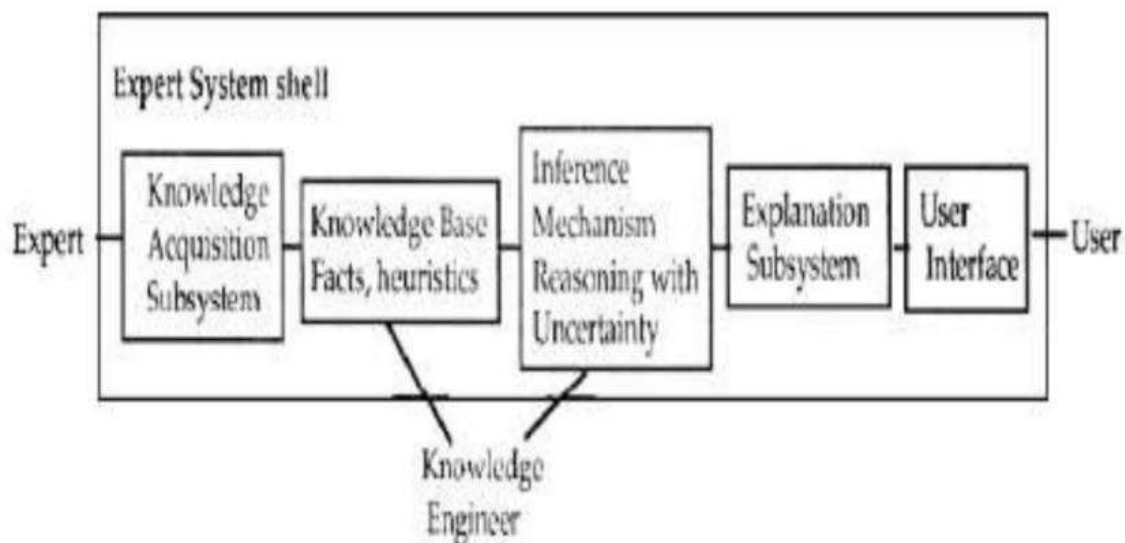
- This person should be able to elicit knowledge from the expert, gradually gaining an understanding of an area of expertise.
- Intelligence, tact, empathy, and proficiency in specific techniques of knowledge acquisition are all required of a knowledge engineer

3. User

- A system developed by an end user with a simple shell, is built rather quickly and inexpensively.



Expert System Shell



Expert System Shells



- Initially each expert system is build from **scratch (LISP)**.
- Expert System Shells are a **collection of software packages & tools** used to develop expert systems
- A shell provides the developers with **knowledge acquisition, inference engine, user interface, and explanation facility**.
- Example of shell is EMYCIN (for Empty MYCIN derived from MYCIN).
- Expert system shells CLIPS, JESS, DROOLS

Shells

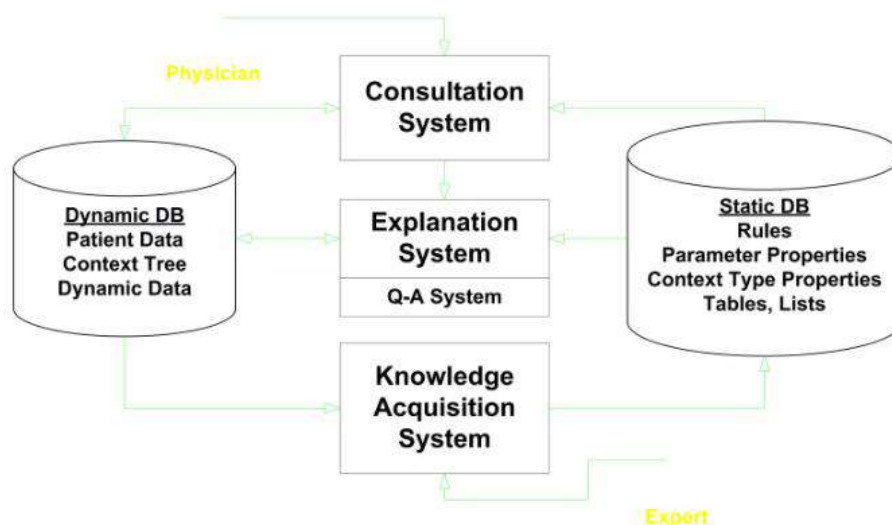


- Shells provide greater flexibility in representing knowledge and in reasoning
- Expert system shells need to integrate with other programs easily.
- Expert systems cannot operate in vacuum.
- The shells must provide an easy-to-use interface between an expert system written with the shell and programming environment.

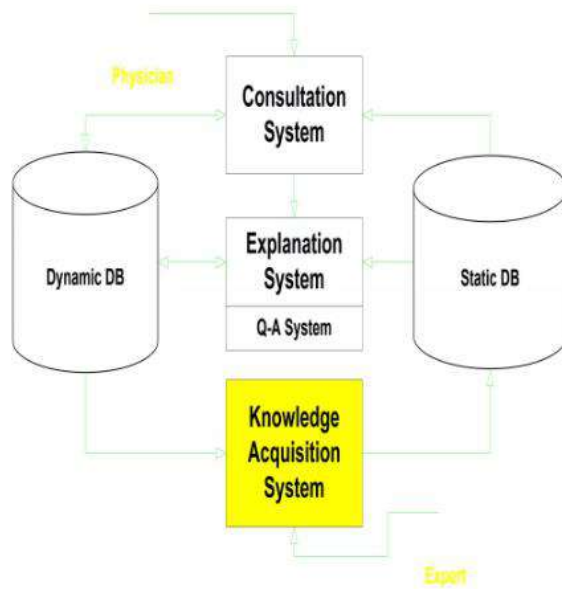
MYCIN

- MYCIN is used for **Disease DIAGNOSIS** and **Therapy SELECTION**
- MYCIN would attempt to diagnose patients based on reported symptoms and medical test results.
- The program could request further information concerning the patient, as well as suggest **additional laboratory tests**, to arrive at a probable diagnosis, after which it would recommend a course of treatment.
- If requested, MYCIN would explain the reasoning that led to its diagnosis and recommendation.
- Using about 500 production rules, MYCIN operated at roughly the **same level of competence as human specialists** in **blood infections** and rather better than general practitioners.
- MYCIN was written in LISP

MYCIN Architecture



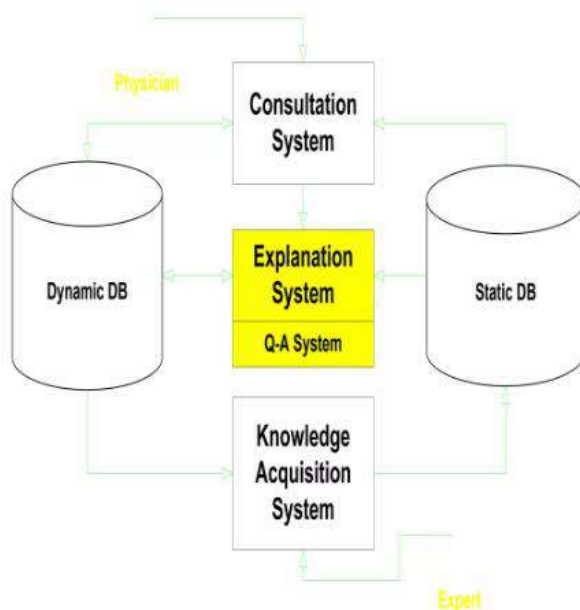
Knowledge Acquisition System



- Extends Static DB via Dialogue with Experts
- Dialogue Driven by System
- Requires minimal training for Experts

30

Explanation System



- Provides reasoning why a conclusion has been made, or why a question is being asked
- Q-A Module
- Reasoning Status Checker

DART (Diagnostic Assistant Reference Tool)

- An expert system for **computer fault diagnosis**
- DART is a joint project of stanford university and IBM
- It explores AI techniques to the diagnosis of computer faults
- DART system identifies specific system components (both hardware and software) likely to be responsible for an observed fault
- It offers a brief explanation of the major factors and evidence supporting these indictments.
- DART , was constructed using HMYCIN

DART



- It uses a **device independent language** for **describing devices** and device independent inference procedure for diagnosis
- The practical goal is the construction of an **automated diagnostician** capable of pinpointing the functional units responsible for observed malfunctions
- The current DART knowledge base consists of 300 EMYCIN parameters and 190 production rules and was constructed over a period of 8 months.
- During this period 5 specialists were interviewed about different aspects of the diagnostic process and the knowledge base reflects their composite expertise.
- DART has been implemented in common LISP.

XOON



- It is an Expert system that finds the hardware and software that are currently used in a particular company.
- It contains all the list of software and hardware configurations to be used and implemented accordingly.
- Its the first expert system in use.