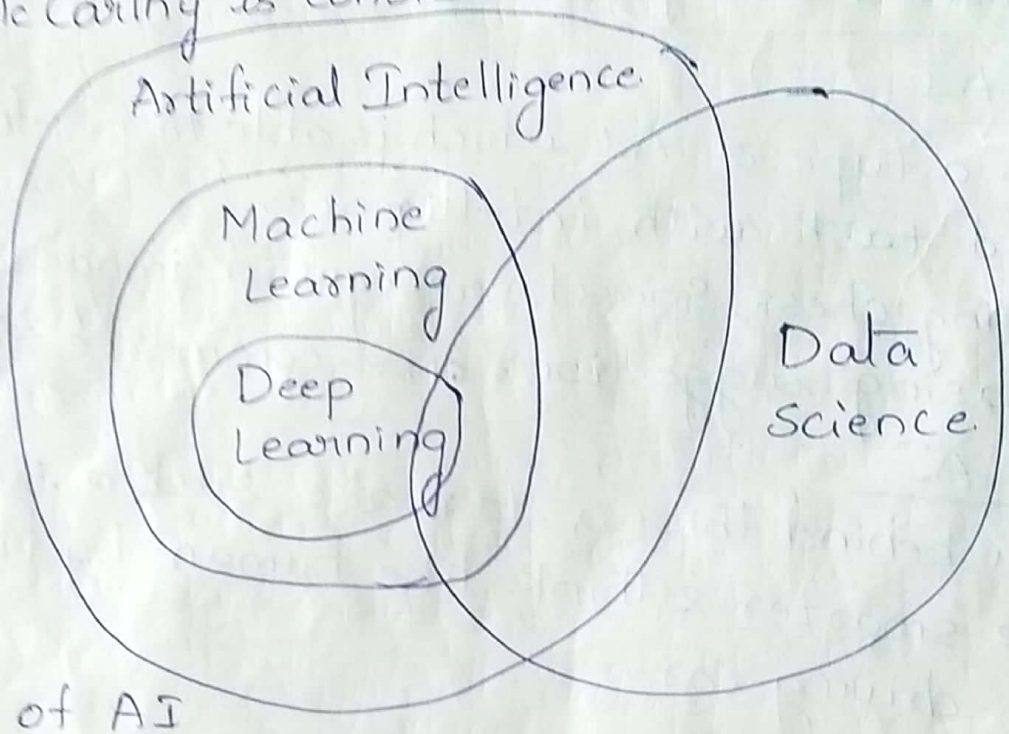
# Artificial Intelligence

## Introduction

- AI is concerned with the design of intelligence in an artificial device.
- Intelligence is the ability to acquire, understand & apply the knowledge to achieve goals in the world.
- AI refers to the development of computer systems that can perform tasks that would normally require human intelligence such as learning, problem solving, decision making & pattern recognition.
- The term 'AI' was coined in the 1950s by John McCarthy
- John McCarthy is considered the father of AI.




## Overview of AI

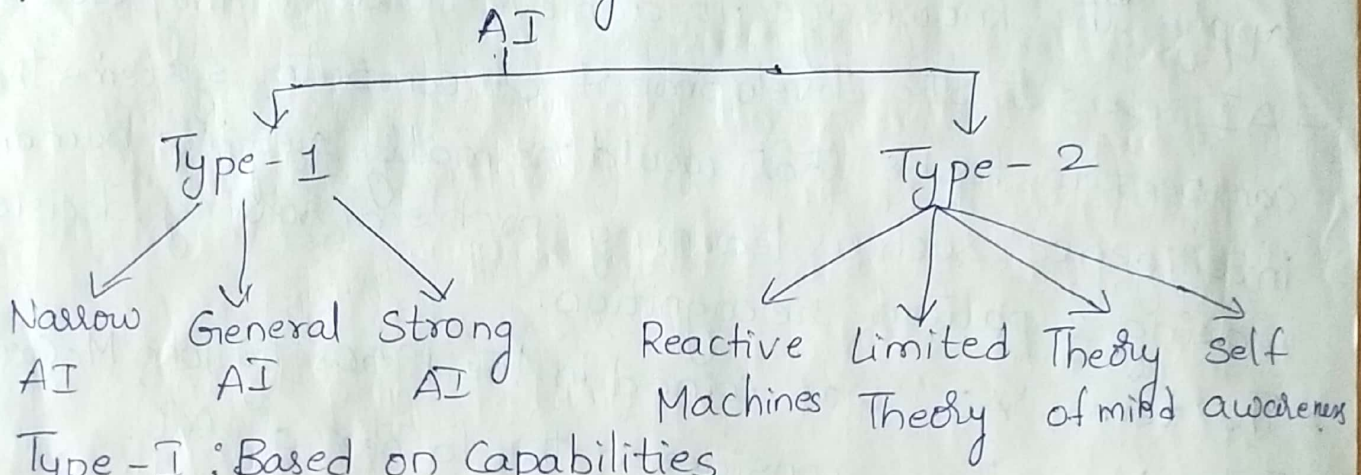
- AI is composed of two words Artificial & Intelligence where artificial ~~means~~ defines "Man-Made" & intelligence defines "Thinking Power", hence AI means "A man-made thinking power".

## Definition

AI refers to the simulation of human intelligence in machines that are programmed to think & learn.

## Types of AI

AI can be divided into TWO types based on capabilities & based on functionality of AI.




## Type-I : Based on Capabilities

- **Narrow AI**

It is a type of AI which is able to perform a dedicated task with intelligence.

Eg: Playing chess, Speech recognition, image recognition & purchasing suggestions on e-commerce site.

- **General AI**

The idea behind this is to make such a system which could be smarter & think like human by its own.

Eg Self-driving car.

- **Strong AI**

Intelligent Systems that can surpass human intelligence & can perform any task better than human with cognitive properties.

Eg - Ability to think, to reason, solve the puzzle, make judgements, plan, learn & communicate by its own.

## Type-2: Based on Functionality.

- **Reactive Machine** -
  These machines focus only on current scenarios & react on it as per possible best action.
  Eg - IBM's Deep Blue system, Google's AlphaGo
- **Limited Memory**
  These machines can store past experiences or some data for a short period of time.
  Eg - Self-driving car store recent speed of nearby cars, the distance of other cars, speed limit, info to navigate the road.
- **Theory of mind**
  This type of AI machines understand the human emotions, people, beliefs & be able to interact socially like humans. Eg - Under developing.
- **Self-Awareness**
  These machines will be super intelligent & will have their own consciousness, sentiments & self-awareness. These machines will be smarter than human mind.
  Eg - Under developing.

# History of AI

## The birth of AI (1950s)

1950: Alan Turing - introduced the Turing Test & Proposed the idea of a "Computing Machinery & Intelligence".

1956: Dartmouth Conference.

John McCarthy who coined the term in 1956, defines it as "The Science & Engineering of making intelligent machines, especially intelligent computer programs".

## The Golden Years (1960 - 1970)

- Optimization & Progress.
  - Eliza (1966) - created by Joseph Weizenbaum, a NLP computer program that simulated conversation.
  - Shakey the Robot (1966-1972) - First general-purpose mobile robot able to reason about its actions.
- Early AI applications

  Development of systems for medical diagnosis, symbolic mathematics & chess playing.

## The Rise of Modern AI (2000s - PRESENT)

- Machine Learning

  Rise of new algs & techniques for data-driven learning
- Big data & Improved algorithms

  Enhanced computational power & access to large datasets
- Key Innovations.
  - Deep Learning: Use of deep neural n/ws for image & speech recognition
  - AI in Everyday Life: Virtual assistants (Siri, Alexa), autonomous vehicles, recommendation systems.

# Applications of AI

## ① Healthcare.

- **Diagnosis.**
  - **Medical Imaging**: AI algorithms analyze X-rays, MRIs and CT scans to detect diseases like cancer, fractures & infections.
  - **Pathology**: AI systems assist in examining tissue samples for accurate diagnosis.
- **Personalized Medicine.**
  AI analyzes patient data to tailor treatments to individual genetic profiles & health histories.
- **Robotic Surgery.**
  Robotics like da Vinci Surgical System enhance precision in complex surgeries, reducing recovery times & improving outcomes.
- **Drug Discovery.**
  AI accelerates the process of discovering new drugs by predicting molecular interactions & potential side effects.

## ② Finance

- **Fraud Detection:**
  AI systems monitor transactions in real-time to identify suspicious activities & prevent fraud.
- **Algorithmic Trading.**
  AI algs analyze market data & execute trades at high speeds, optimizing investment strategies.
- **Risk Management.**
  AI assesses credit scores & loan egibility, helping financial institutions manage risks.

- Customer service

Chatbots & virtual assistants handle customer inquires providing efficient & accurate responses.

③ Agriculture

- Precision farming.

AI analyzes data from soil sensors, weather forecasts & crop health to optimize farming practices.

- Crop Monitoring.

Drones equipped with AI analyze aerial images to detect diseases & pests in crops.

- Yield Predicition

AI models predict crop yields, helping farmers make informed decisions about planting & harvesting.

④ Education

- Personalized Learning

AI systems adapt educational content to the learning pace & style of individual students.

- Tutoring Systems

AI powered tutors provide additional support & resources to students outside of the classroom.

- Administrative Tasks

AI automates grading, scheduling & other administrative tasks, freeing up time for educators.

⑤ Transportation

- Autonomous Vehicles.

AI powers self-driving cars by processing data from sensors & cameras to navigate safely.

- Traffic Management.

- Traffic Management

AI systems analyze traffic patterns to optimize signal timings & reduce congestion.

- Logistics & Supply chain

AI optimizes routes for delivery trucks, manages inventory levels & predicts demand.

(6) Smart Homes & Cities

- Home Automation

AI controls home devices such as lights, thermostats & security systems for convenience & energy efficiency

- Smart Grids

AI optimizes energy distribution & usage in power grids, enhancing sustainability.

- Urban Planning

AI analyzes data on traffic, pollution & population to improve city planning & infrastructure.

# Intelligent Agent

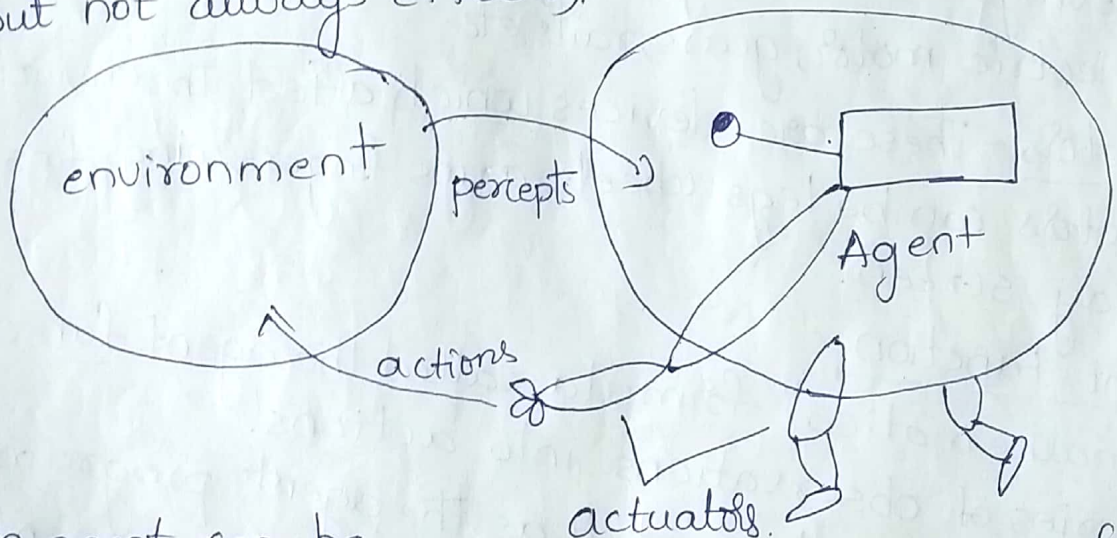
Agent: element in a system & environment that can trigger action.

- When deciding what to do, an agent makes decision based on how they perceive their surroundings.
- The perception capability is usually called a sensor.
- The most recent perception or the full history (percept sequence) may influence the actions.

## Agents

In AI, an agent is a computer program or system & anything that can be viewed as perceiving its environment through sensors & acting upon that environment through effectors.

Assumption - Every agent can perceive its own actions (but not always effects).




An agent can be

- Human agent - has eyes, ears & other organs for sensors & hands, legs, mouth & other body parts for effectors/actuators.
- Robotic agent - substitutes cameras & infrared range finders for the sensors & various motors for effectors.

• Software agent - It can have keystrokes, file contents received n/w packages as sensors & displays on the screen, files, sent n/w packages as actuators.

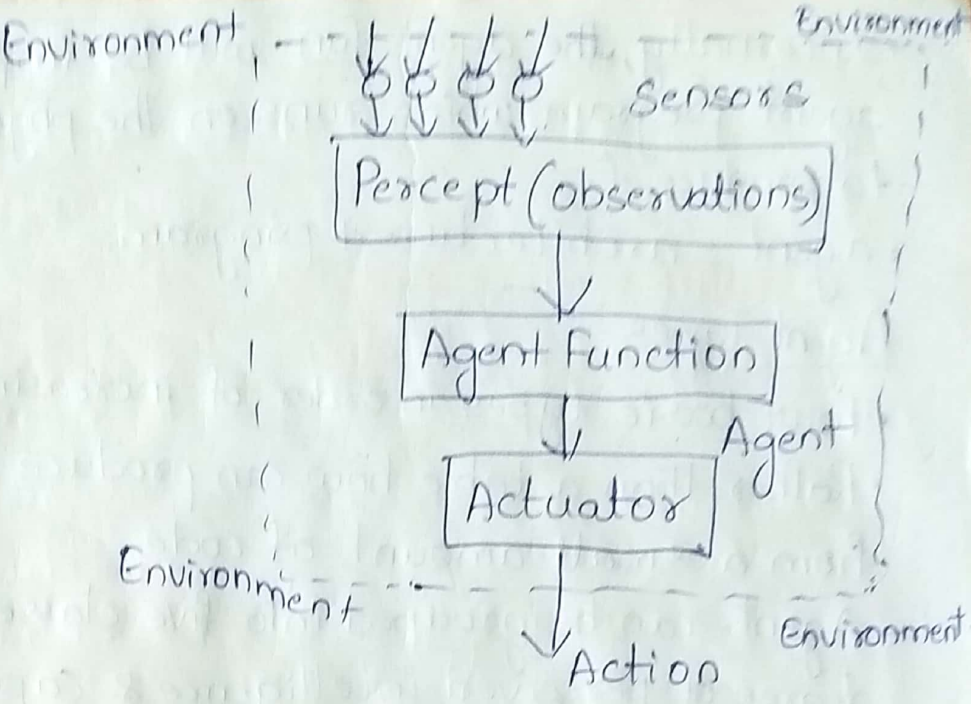
Sensor - It is a device which detects the change in the environment & sends the info to other electronic devices. An agent observes its environment Through sensors
- The complete set of i/ps at a given time is called percept.
- The current percept or a sequence of percepts can influence the actions of an agent.
- It can change the environment through effectors.
- An operation involving an actuator is called an action, which can be grouped into action sequences.

Actuators - These are the component of machines that converts energy into motion. The actuators are only responsible for moving & controlling a system. An actuator can be an electric motor, gears, rails etc

Effectors - These are devices which affect the environment Effectors can be legs, wheels, arms, fingers, wings, fins & display screen.

Agent Function

• A mathematical formula called the agent fun converts a series of observations into actions.
• The fun is implemented as the agent program.
• Actuator refers to the component of an agent that performs an action.
• environment → sensors → agent function → actuators → environment.




- A rational agent is one who can make the best choice in any circumstance.
- A set of standards & testing ground for an agent's behaviour that serves as a performance measure.
- Based on the agent's intended impact on the environment, performance metrics should be developed.

## Agents & Environments

- Percept sequence – complete history of everything the agent has ever perceived.
- Percept – percept is use to the agent's perceptional i/ps at any given instant.
- An agent's choice of action at any given instant can depend on the entire percept sequence observed to date.
- An agent's behaviour is described by the agent fun which maps from percept histories to actions:

  $[f \rightarrow P^* \rightarrow A]$

- We can imagine tabulating the agent fun that describes any given agent (External characterization)

- Internally, the agent fun will be implemented by an agent program which runs on the physical architecture to produce f.
- agent = architecture + program.

## Agent Types

- four basic types in order of increasing generality
- Rather than a table how can produce rational behavior from a small amount of code.

- Agents can be grouped into five classes based on their degree of perceived intelligence & capability. All these agents can improve their performance & generate better action over the time.

Example - Vacuum-cleaner Agent (describes the concept of agent fun, agent program & agent designs)

- The Vacuum cleaner world which consists of a robotic vacuum cleaning agent in a world consisting of squares that can be dirty or clean.
- Fig shows a configuration with just two squares A & B
- Percepts
  - Location & status
    eg [A, Dirty].
- Actions
  Left, Right, Suck, do Nothing (NoOp)
- Consider the simple vacuum cleaner agent that cleans a square if it is dirty & moves to the other if not.
- function Vacuum-Agent([location, status]) returns an action

```
if status = dirty then return Suck
else if location = A then return Right
else if location = B then return Left
```

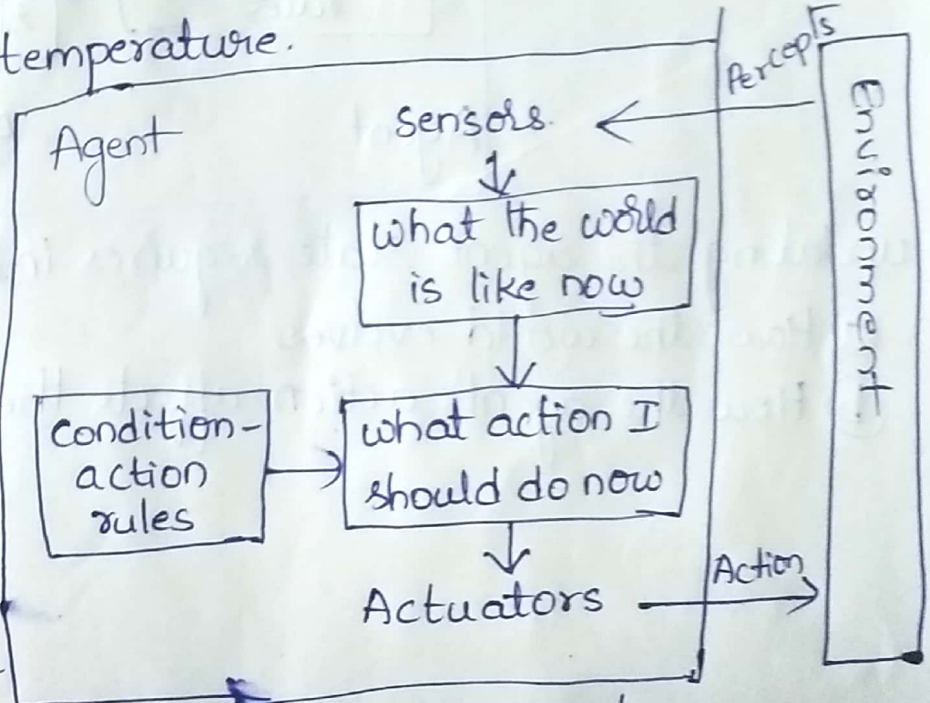
# Types of AI Agents

Agents can be grouped into five classes based on their degree of perceived intelligence & capability. All these agents can improve their performance & generate better action over the time.

## ① Simple Reflex Agent

The simple reflex agents are the simplest agents. These agents take decisions on the basis of the current percepts & ignore the rest of the percept history.

- These agents only succeed in the fully observable environment
- The simple reflex agent does not consider any part of percepts history during their decision & action process.
- These agents works on condition-action rule, which means it maps the current state to action. Eg Room cleaner agent, it works only if there is dirt in the room.
- For eg, a thermostat turns the heater on & off based on the current temperature.

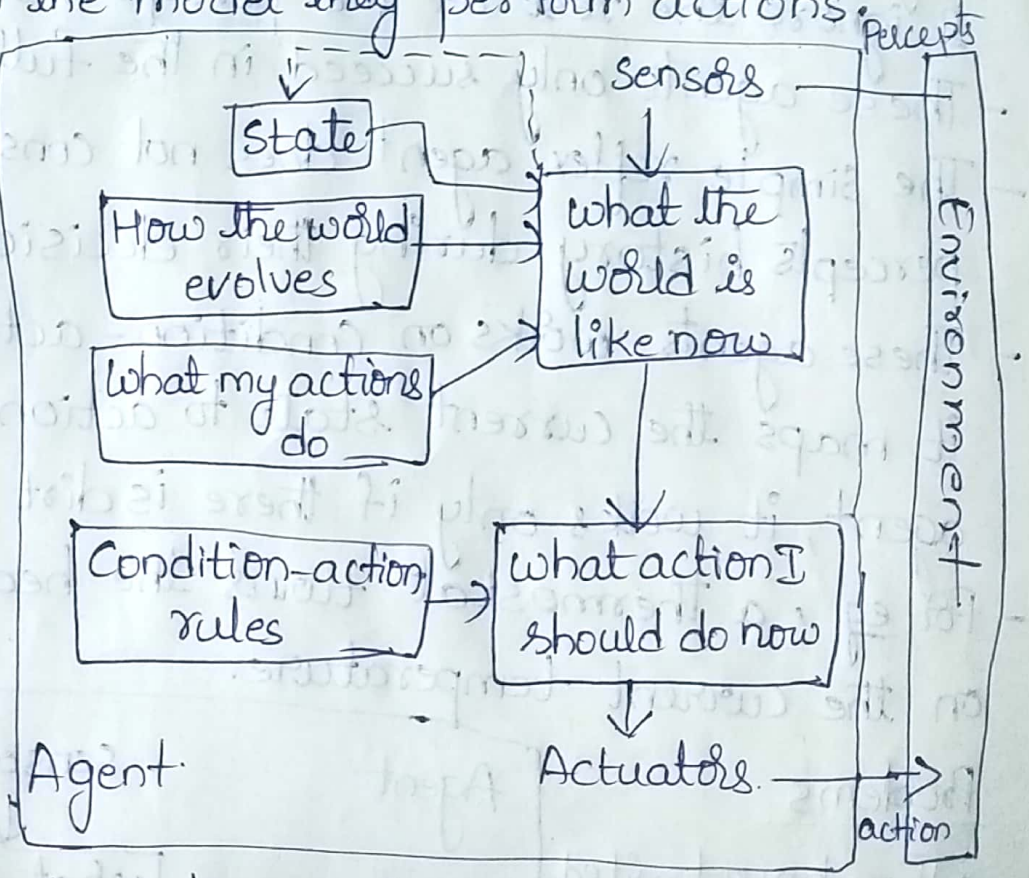



**Problems**

- They have limited intelligence
- They do not have knowledge of non-perceptual parts of the current state
- Mostly too big to generate & to store
- Not adaptive to changes in the environment.

## ② Model-based reflex agent

- This agent can work in a partially observable environment & track the situation.
- A model-based agent has two important factors:
  - Model: It is knowledge about "how things happen in the world", so it is called Model-based agent.
  - 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" & based on the model they perform actions.




updating the agent state requires info about

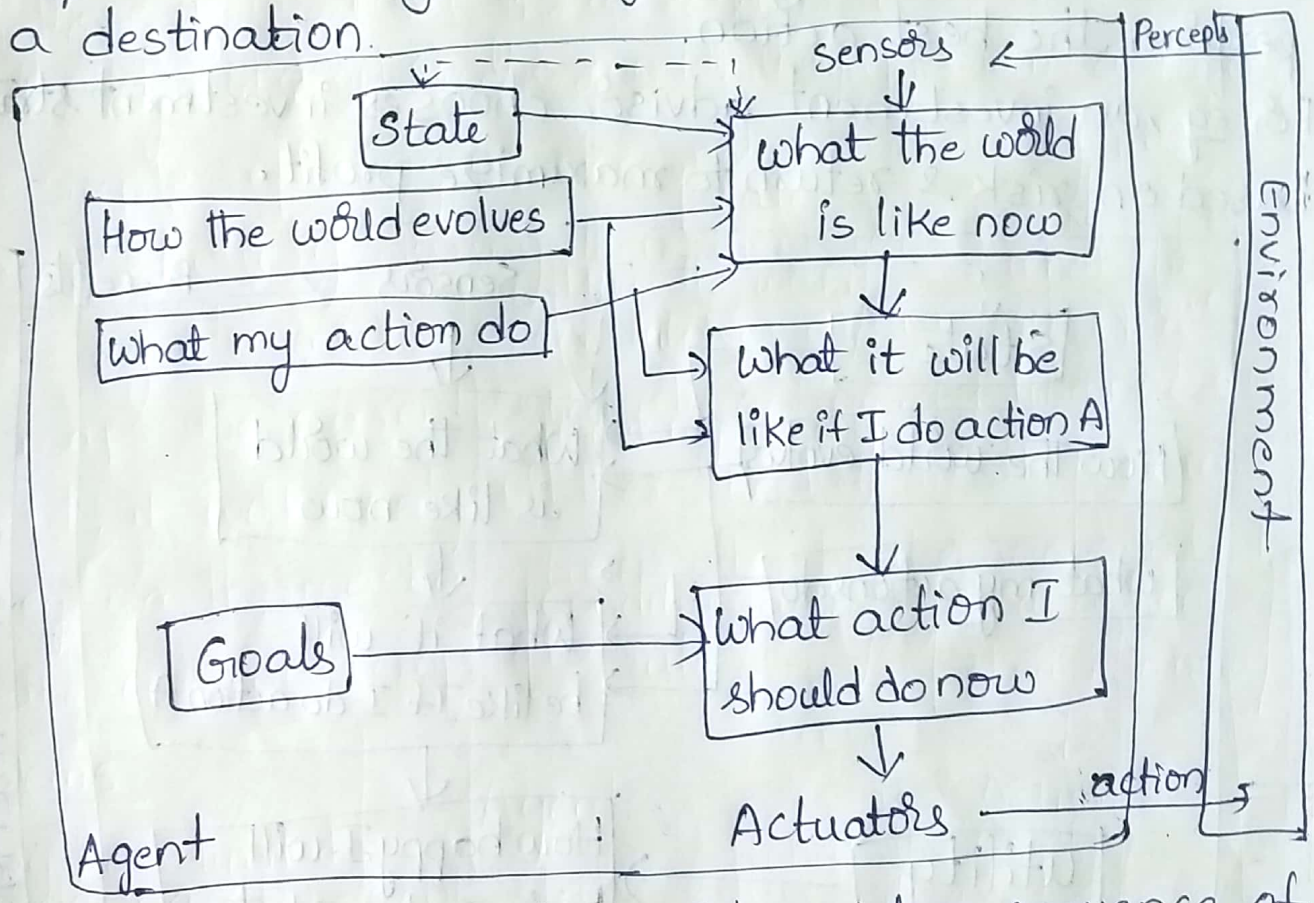
(a) How the world evolves.

(b) How the agent's action affects the world.

## ③ Goal-based agents

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 its goal which describes desirable situations.
- These agents expand the capabilities of the model-based agent by having the "goal" information.
- They choose an action, so that they can achieve the goal.
- For eg, a GPS navigation system plans the best route to a destination.

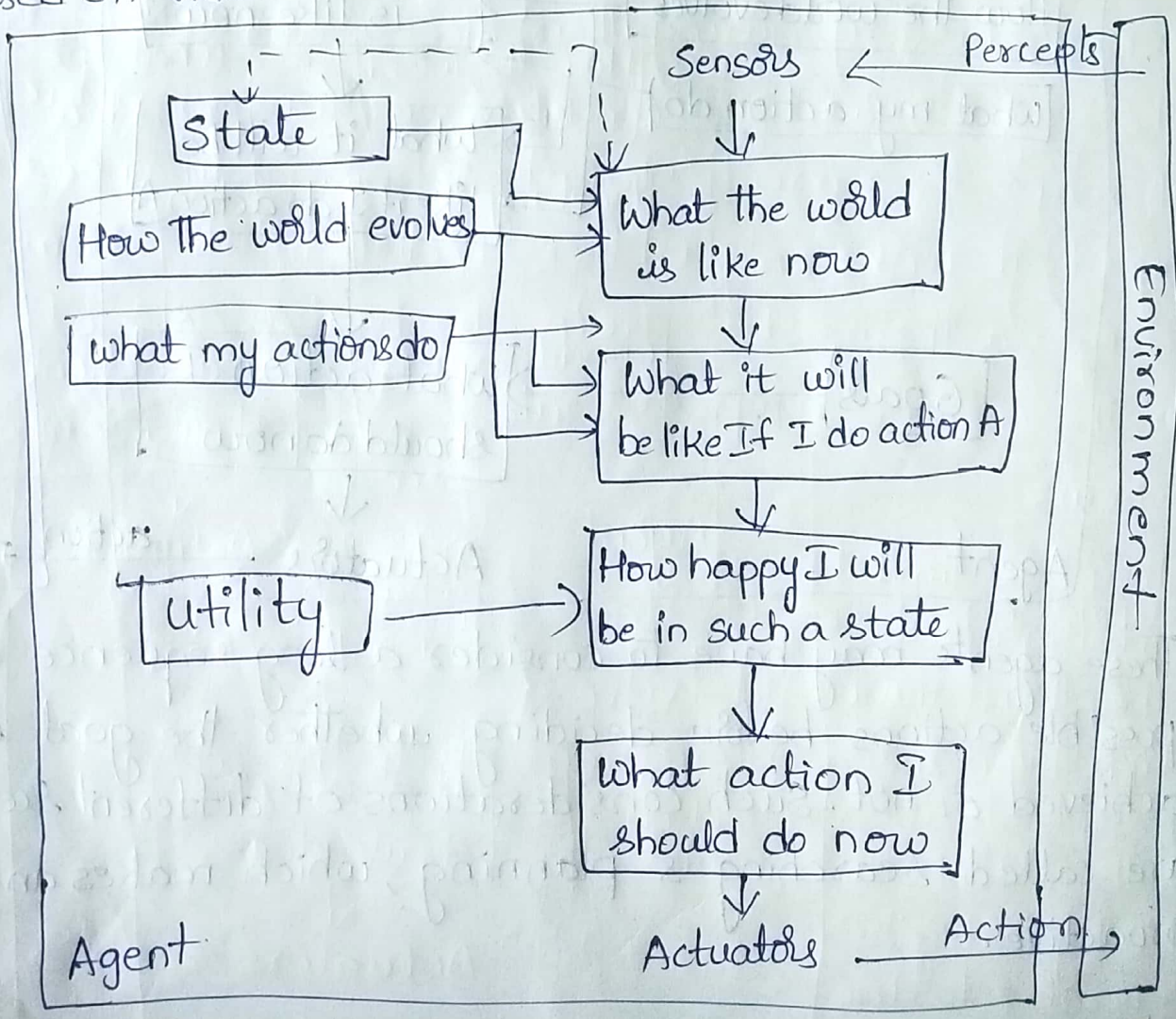



- These agents may have to consider a long sequence of possible actions before deciding whether the goal is achieved or not. Such considerations of different scenario are called searching & planning, which makes an agent proactive.

## ④ Utility-based agents

The agents are similar to the goal based agent but provide an extra component of utility measurement which makes them different by providing a measure of success at a given state.

- These agents act based on not only goals but also the best way to achieve the goal.
- This agent is useful when there are multiple possible alternatives & an agent has to choose in order to perform the best action.
- For eg, an investment advisor chooses investment strategies based on risk & return to maximize profit.



## (5) Learning Agents

- A learning agent in AI is the type of agent which can learn from its past experience or it has learning capabilities.
- It starts to act with basic knowledge & then able to act & adapt automatically through learning.
- Learning agents are able to learn, analyze performance & look for new days to improve the performance.
- For eg, a recommendation system learns user preferences to provide better recommendations over time.

## Problem Solving Through Search

- The reflex agents are known as the simplest agents because they directly map states into actions. Unfortunately, these agents fail to operate in an environment where the mapping is too large to store & learn.
- Goal based agent on the other hand considers future actions & the desired outcomes.
- Problem solving agents in AI mostly used these search strategies & algs to solve a specific problem & provide the best result.
- Problem solving agents use atomic representations, states of the world are considered as wholes, with no internal structure visible to the problem solving algorithms.
- A problem solving agent is a goal driven agent & focuses on satisfying the goal.
- Steps performed by Problem-solving agent.
  - Goal Formulation - It is based on the current situation & the agent's performance measure. It organises steps required to achieve the goal.
  - Problem Formulation - It is process of deciding what actions should be taken to achieve the formulated goal.
  - Components involved in Problem Formulation -
    - Initial state - starting state or initial step of the agent towards its goal.
    - Actions - It is a description of the possible actions available to the agent.
    - Transition model - It describes what each action does
    - Goal state - It determines if the given state is a goal state.
    - Path cost - It assigns a numeric cost to each path that follows the goal.

Note: Initial state, actions & transition model together define the state-space of the problem implicitly.

- State space of a problem is a set of all states which can be reached from the initial state followed by any sequence of actions.
- The state-space forms a directed map or graph where nodes are the states, links b/w the nodes are actions & the path is a sequence of states connected by the sequence of actions.
  - Search - It identifies all the best possible sequence of actions to reach the goal state from the current state. It takes a problem as an i/p & returns sol as its o/p.
  - Solution - It finds the best alg out of various algs, which may be proven as the best optimal sol.
  - Execution - It executes the best optimal sol from the searching algs to reach the goal state from the current state.
- For solving different kinds of problem, an agent makes use of different strategies to reach the goal by searching the best possible algorithms. This process of searching is known as search strategy.

Search Algorithm

- Uninformed / Blind
  - Breadth First Search
  - Depth First Search
  - Depth Limited Search.
  - Iterative Deepening Search
  - Bi-directional Search.
- informed / Heuristic.
  - Best First Search
  - A* algorithm

- Uninformed search is also called brute force search & blind search & exhaustive search. It is called blind search because of the way in which search tree is searched without using any information about the search space. It is called brute-force because it assumes no additional knowledge other than how to traverse the search tree & how to identify the leaf nodes & goal nodes. This search utlimately examines every node in the tree until it finds a goal.
- Informed search is also called as Heuristic & guided search. These are the search techniques where additional info about the problem is provided in order to guide the search in a specific direction. A heuristic is a method that might not always find the best sol but it is guaranteed to find a good sol in reasonable time. By scarifying completeness, it increases efficiency.

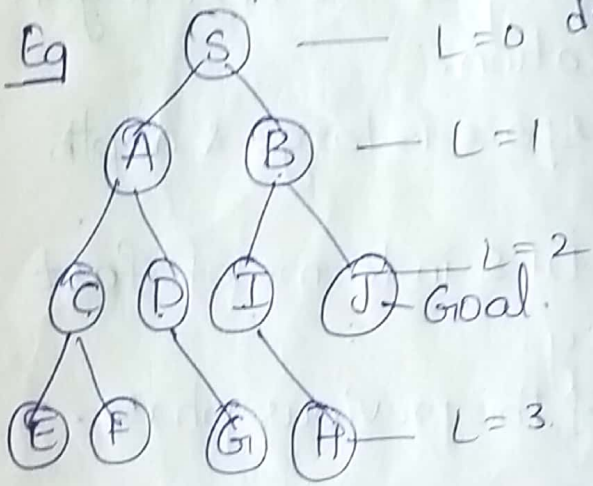
# Depth Limited Search (DLS)

It is similar to DFS with a predetermined limit. DLS can solve the drawback of the infinite path in the DFS. In this alg, the node at the depth limit will treat as it has no successor nodes further.

Adv - Memory efficient

disadv - incompleteness, It may not be optimal if the problem has more than one solution.

Eg




depth limit=2

step1 - S - d=0

step2 - S, A, B - d=1

step3 - S, A, C - d=2

step4 - S, A, C, D

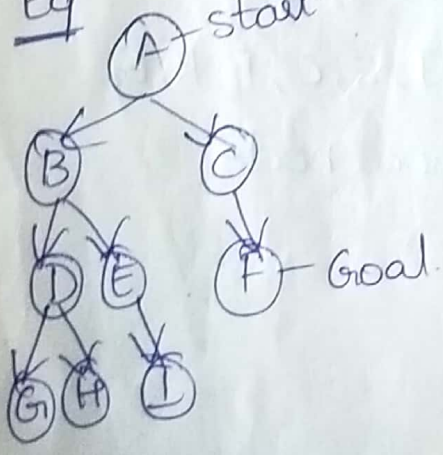
step5 - S, A, C, D, B

step6 - S, A, C, D, B, I

step7 - S, A, C, D, B, I, J

path S - A - C - D - B - I - J.

Eg




step1 - A

Step2 - A, B

Step3 - A, B, D.

Step4 - A, B, D, E

Step5 - A, B, D, E, C

Step6 - A, B, D, E, C, F

Time Complexity - $O(b^l)$ - b - branching factor.
l - limit.

Space Complexity - $O(b \times l)$

# Iterative Deepening DFS.

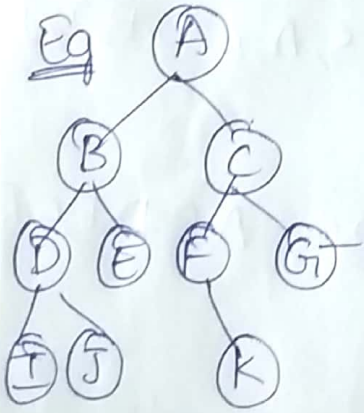
It is a combination of BFS & DFS. This search alg finds out the best depth limit & does it by gradually increasing the limit until a goal is found.

- It begins by performing DFS to a depth of zero, then depth of one & so on until a goal is found & some max depth is reached.
- It is like BFS in that it explores a complete layer of new nodes at each iteration before going to the next layer. It is likes DFS for a single iteration.
- It is useful when the search space is large & depth of goal node is unknown.

Adv - Combines benefits of BFS & DFS in terms of fast search & memory efficiency.

Disadv - It repeats all the work of the previous phase.

Eg




Step 1 - A

Step 2 - A, B, C

Step 3 - A, B, D, E, C, F, G

Step 4 - A, B, D, I, J, E, C, F, K, G.

In 4th iteration, the alg will find the Goal node.

Time complexity - $O(b^d)$

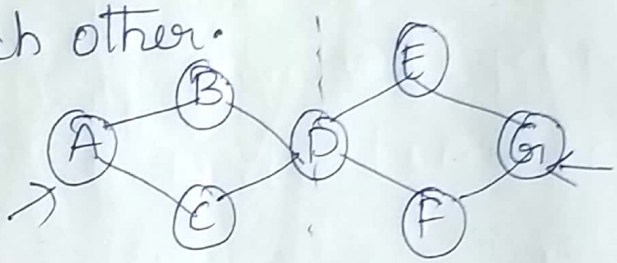
Space complexity - $O(bd)$

## Bidirectional Search.

It runs two simultaneous searches, one from initial state & other from goal node.

- This search replaces one single search graph with two small subgraphs in which one starts the search from an initial vertex & other starts from goal vertex.
- The search stops when these two graphs intersect each other.

Eg



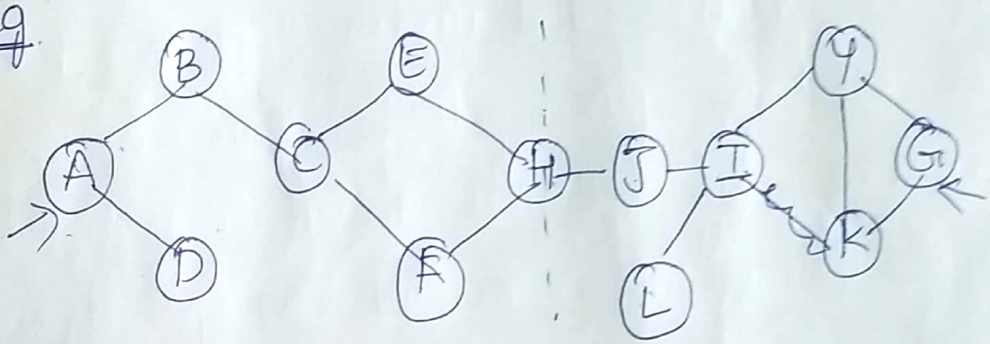

A - B - C

A - B - C - D

G - E - F

G - E - F - D

Eg




A

A B

A B C

A B C E

A B C E H

G

G Y

G Y I

G Y I L

G Y I L J. , G Y I L Y H

# Local Search Algorithms

- Previously to find sol for a given problem, we apply a search alg either uninformed & informed search techniques to explore the search space systematically to search from start state to a goal state & obtain the path from the start to goal as sol of the given problem
- In many problems, the path to goal is irrelevant (8 queen problem).
- If the path to the goal doesn't matter, then we can consider local search algorithms.
- Local search algs are useful for solving pure optimization problems, in which the aim is to find the best state according to an objective fun.
- In optimization problems, the path to goal is irrelevant and the goal state itself is the solution.
- In some optimization problems, the goal is not known & the aim is to find the best state.
- Although local search algs are not systematic, they have two key advantages:
  - use very little memory
  - find reasonal sols in large & infinite (continuous) state spaces.

## Hill climbing.

- It is a heuristic optimization process that iteratively advances towards a better sol at each step in order to find the best sol in a given search space.
- It is a technique which is used for optimizing the mathematical problems. Eg TSP - min distance - Salesman.
- It is also called greedy local search as it only looks to its good immediate neighbor state & not beyond that.
- A node of hill climbing alg has two components which are state & value.
- It is mostly used when a good heuristic is available.

### Features.

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

### Types

- **Simple Hill Climbling** - It only evaluates the neighbor node state at a time & selects the first one which optimizes current cost & set it as a current state.
  - It only checks its one sucessor state & if it finds better than the current state, then move else be in same state.
  - **Features** - less time consuming, less optimal sol & the sol is not guaranteed.

Algorithm

Step 1 - Evaluate the initial state, if it is a goal state then return success & stop.

Step 2 - Loop until a sol is found or there is no new operator left to apply.

Step 3 - Select & apply an operator to the current state

Step 4 - Check new state

a. If it is goal state, then return success & quit.

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

c. else if not better than the current state, then return to step 2.

Step 5 : Exit.

Steepest Ascent Hill Climbing

- It is a variation of simple hill climbing alg. This alg examines all the neighboring nodes of the current state & selects one neighbour node which is closest to the goal state.
- This alg consumes more time as it searches for multiple neighbors

Algorithm

Step 1 - Evaluate the initial state, if it is goal state then return success & stop else make current state as initial state

Step 2 - Loop until a sol is found or the current state does not change.

a. Let SUCC be a state such that any successor of the current state will be better than it.

b. For each operator that applies to the current state:

(i) Apply the operator & generate a new state.

(ii) Evaluate the new state. If it is a goal state then return & quit else compare it to succ. If it is better, than set succ to this state. If it is not better, leave succ alone.

* If the succ is better than the current state, then set the current state to succ.

Stochastic Hill climbing

It does not examine for all its neighbours before moving. Rather this search alg selects one neighbour node at random & decides whether to choose it as a current state or examine another state.

State Space Diagram

The state-space landscape is a graphical representation of the hill-climbing alg which is showing a graph b/w various states of alg & objective fun/cost.

- X-axis - denotes the state space i.e states or configurations our alg may reach.
- Y-axis - denotes the value of objective fun corresponding to a particular state.

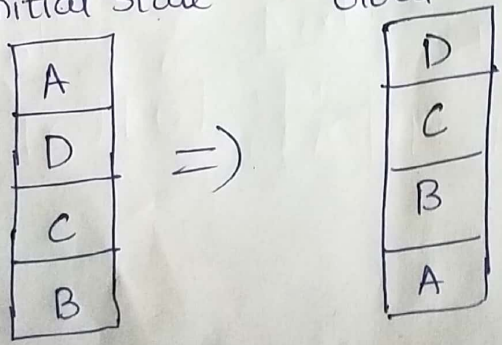
## Different regions in the state space landscape

- Local Maximum – It is a state which is better than its neighbor states, but there is also another state (Global maximum) which is higher than it.
- Global Maximum – It is the best possible state of state space landscape. It has the highest value of the objective fun.
- Current State – It is a state in a landscape diagram where an agent is currently present.
- Flat local Maximum – It is a flat space in the landscape where all the neighbour states of current states have the same value.
- Shoulder – It is a plateau region which has an uphill edge.
- Ridge – It is a location that is higher than its surroundings yet has a slope of its own. This particular local max is unique.

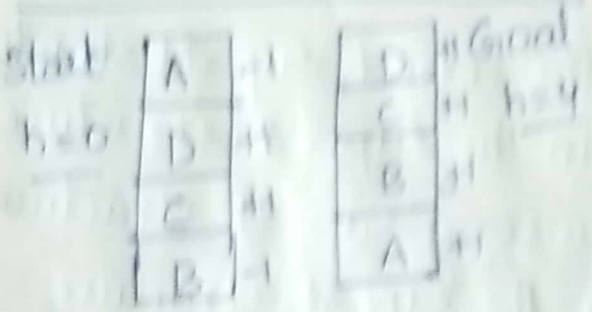
Example – BLOCKS WORLD PROBLEM

- If the block is appropriately positioned,
  $h(x) = +1$ for all blocks in the support structure
  otherwise $h(x) = -1$ for all blocks in the support structure

Initial State ⇒ Goal State



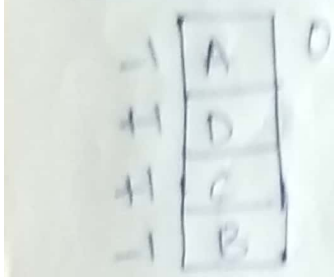
## Local Heuristic function

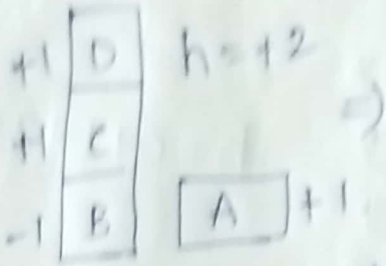



h(x) = +1 - correct position for each block

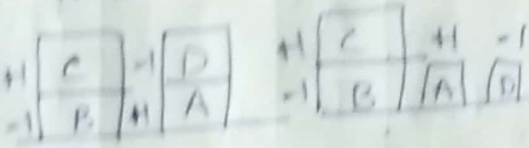
h(x) = -1 - wrong position

### Next State



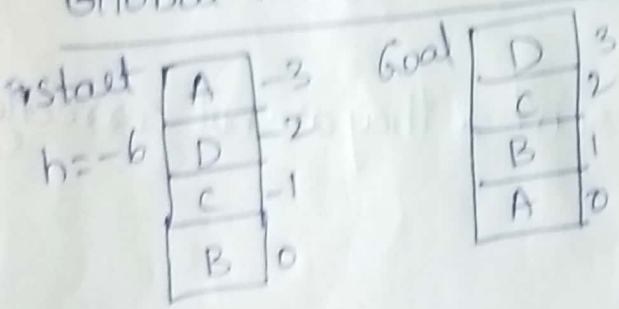




### Next State

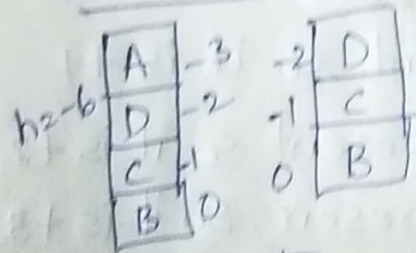



- After this we can't reach to goal state as next state has h=0 which is less than current state h=2
- This is the disadvantage of local heuristic fun...

## Global heuristic function.

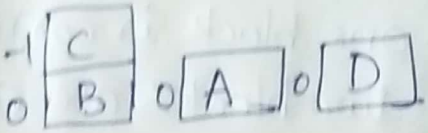



### Next State

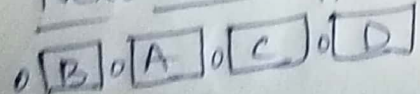



### Next State

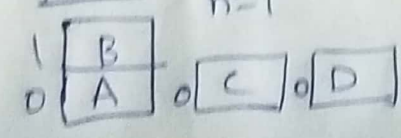
h=-1



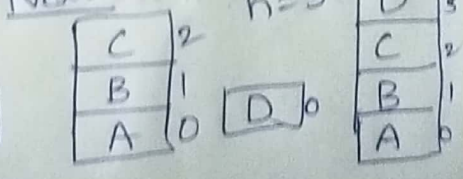

### Next State




### Next




### Next

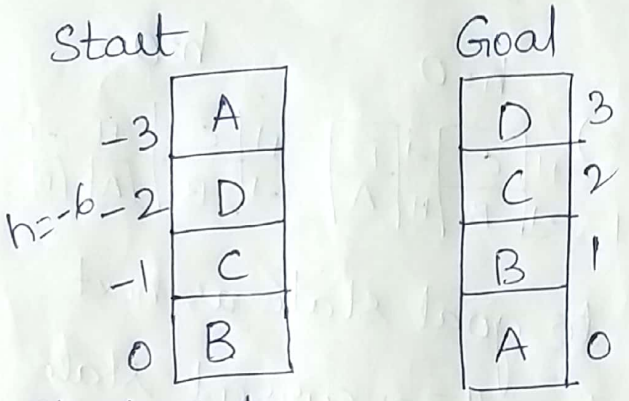



Next

## Global Heuristic function

For each block that has the correct support structure
: +1 to every block in the support structure

For each block that has a wrong support structure
: -1 to every block in the support structure




### Start state

- Below B no block is present so it is Zero.
- Below C one block is present & it is not correctly placed so it is -1.
- Below D two blocks are present & they are not correctly placed so it is -2.
- Below A ~~two~~ three blocks are present & they are not correctly placed so it is -3.

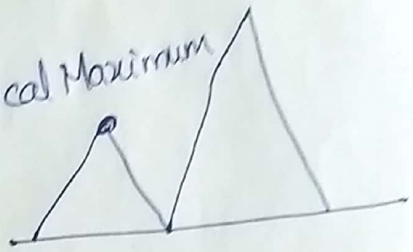
### Goal state

- Below A no block is present so it is zero
- Below B one block is correctly present so it is +1
- Below C two blocks are correctly present so it is +2
- Below D three blocks are correctly present so it is +3.

# Problems in Hill Climbing Algorithm

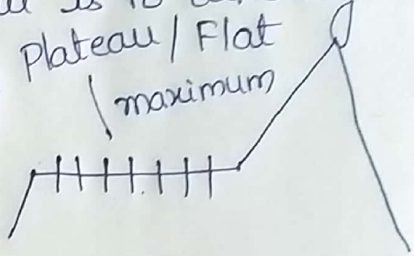
① **Local Maximum** – A local maximum is a peak state in the landscape which is better than each of its neighboring states, but there is another state also present which is higher than Local Maximum.

**Solution** – Backtracking technique can be a sol of the local max in state space landscape. Create a list of promising path so that the alg can backtrack the search space & explore other paths as well.



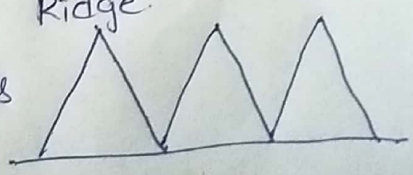

② **Plateau** – It is a flat area of the search space in which all the neighbor states of the current state contains the same value, because of this alg does not find any best direction to move. A hill-climbing search might be lost in the plateau area.

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




③ **Ridges** – It is a special form of the local maximum. It has an area which is higher than its surrounding area, but itself has a slope & cannot be reached in a single move.

**Solution** – With the use of bidirectional search & by moving in different directions we can improve this problem.



# Adversarial Search

- It is a search, where we examine the problem which arises when we try to plan ahead of the world & other agents are planning against us.
- In previous topics, we have studied the search strategies which are only associated with a single agent that aims to find the sol which often expressed in the form of a sequence of actions.
- But there might be some situations where more than one agent is searching for the sol in the same search space & this situation usually occurs in game playing.
- The environment with more than one agent is termed as multi-agent environment, in which each agent is an opponent of other agent & playing against each other. Each agent needs to consider the action of other agent & effect of that action on their performance.
- So, Searches in which two or more players with conflicting goals are trying to explore the same search space for the sol, are called adversarial search often known as Games.
- Games are modeled as a Search problem & heuristic evaluation fun & these are two main factors which help to model & solve games in AI.

# Types of Games in AI

| | Deterministic | Chance Moves |
|---|---|---|
| perfect info | chess, checkers, go, othello | Blackgammon, monopoly. |
| Imperfect info | Battleships, blind tic-tac-toe | Bridge, poker, scrabble, nuclear war |

- Perfect Information

A game with perfect info is that in which agents can look into the complex board. Agents have all the info about the game & they can see each other moves also Eg chess, checkers, Go etc.

- Imperfect information

If in a game agents do not have all info about the game & not aware with what's going on, such types of games are called the game with imperfect info such as tic-tac toe, Battleship, blind, Bridge etc

Non Deterministic games

Those are games which have various unpredicatable events & has a factor of % chance or luck. This factor of chance or luck is introduced by either dice & cards These are random & each action response is not fixed. Such games are also called as stochastic games.

Eg Backgammon, Monopoly, Poker etc.

- Deterministic games
These are the games which follows a strict pattern & set of rules for the games & there is no randomness associated with them. Eg chess, checkers, tic-tac-toe etc

## Zero-Sum Game

- These are adversarial search which involves pure competition.
- In zero-sum game each agents gain & loss of utility is exactly balanced by the losses & gains of utility of another agent.
- One player of the game try to maximize one single value, while other player tries to minimize it.
- Each move by one player in the game is called as ply.
- Chess & tic-tac-toe are examples of a zero-sum game.

## Game tree

It is a tree where nodes are the game states & edges are the moves by players. Game tree involves initial state, action fun & result fun

Example Tic-Tac-Toe game tree

The following fig is showing part of the game-tree for the tic-tac-toe game. Following are some key points of the game:

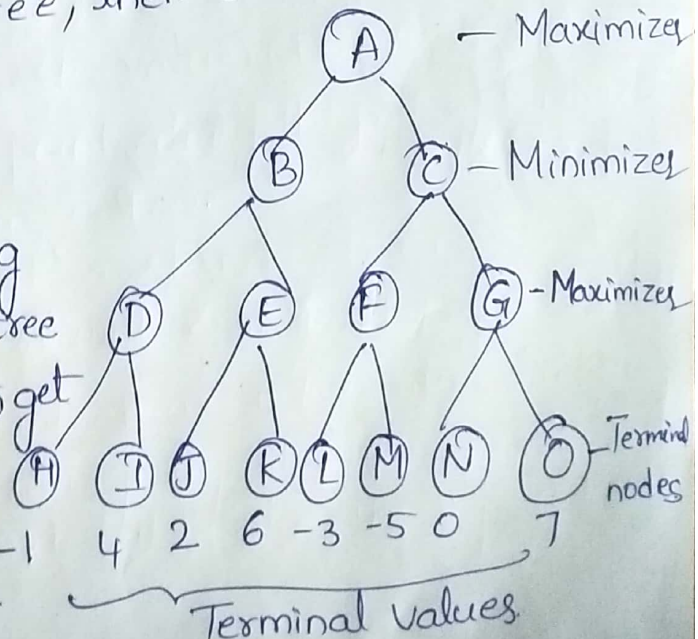
- There are two players MAX & MIN
- Players have alternate turn & start with MAX.
- MAX maximizes the result of the game tree.
- MIN minimizes the result.

# Mini-Max Algorithm

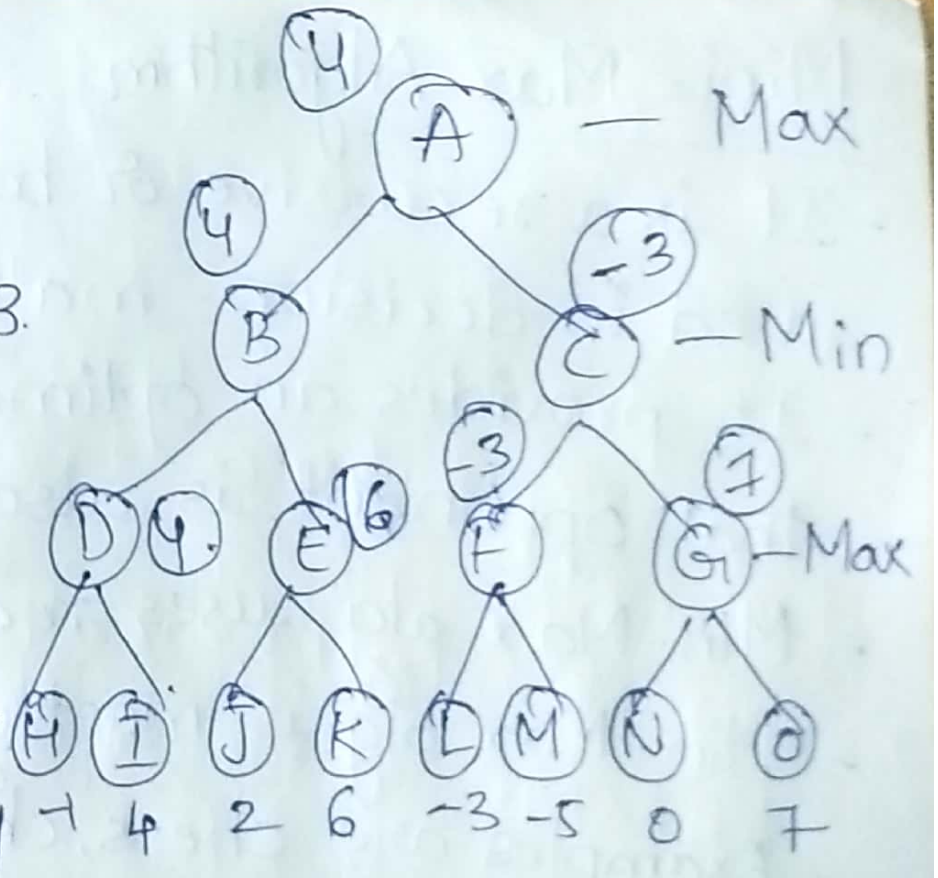
- It is a recursive & backtracking algorithm which is used in decision-making & game theory.
- It provides an optimal move for the player assuming that opponent is also playing optimally.
- Min-Max alg uses recursion to search through game-tree
- Min-Max alg is mostly used for game playing in AI. Examples are chess, checkers, tic-tac-toe, go & various two-players game.
- In this alg, two players play the game, one is called MAX & the other is called MIN.
- Both the players fight it as the opponent player.
- Both players of the game are opponent of each other, where MAX will select the maximized value & MIN will select the minimized value.
- The minimax alg performs a depth-first search alg for the exploration of the complete game tree.
- The minimax alg proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

## Example

- In the first step, the alg generates the entire game-tree & applies the utility fun to get the utility values for the terminal states.



- For node D $\max(-1, 4) = 4$
- For node E $\max(2, 6) = 6$
- For node F $\max(-3, -5) = -3$.
- For node G $\max(0, 7) = 7$.
- For node B $\min(4, 6) = 4$
- For node C $\min(-3, 7) = -3$.
- For node A $\max(4, -3) = 4$



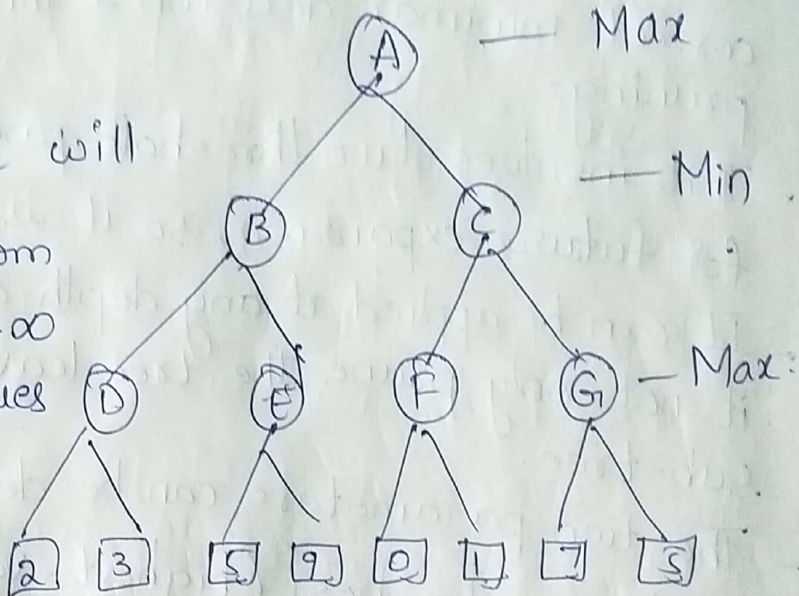

∴ Sol to this problem is 4

# Alpha-Beta Pruning

- It is a modified version of the minimax alg. It is an optimization technique for the minimax alg.
- As we have seen in the minimax search alg that the no of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half.
- Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision & this technique is called pruning.
- This involves two threshold parameter alpha & beta for future expansion, so it is called alpha-beta pruning.
- It can be applied at any depth of a tree & sometimes it not only prune the tree leaves but also entire sub-tree.
- The two parameters can be defined as
  - alpha - The best (highest-value) choice we have found so far at any point along the path of Maximizer. The inital value of alpha is $-\infty$.
  - Beta - The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The inital value of beta is $+\infty$.
- The alpha-beta pruning to a standard minimax alg returns the same move as the standard alg does, but it removes all the nodes which are not really affecting the final decision but making alg slow.
- Hence by pruning these nodes, it makes the alg fast.

- Condition for Apha-beata pruning is $\alpha >= \beta$.
- Key points
  - The MAX player will update only the value of $\alpha$
  - The MIN player will update only the value of $\beta$.
  - While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha & beta.
  - We will only pass the alpha, beta values to the child nodes.

Example




Step1 - Max player will start first move from node A where $\alpha = -\infty$ & $\beta = +\infty$, These values will be passed down to node B & node B passes the same value to its child Node D. $\alpha$

Step2
At Node D, the value of $\alpha$ will be calculated as its turn for Max.
Node D Max$(-\infty, 2) = 2 \Rightarrow \alpha = 2, \beta = +\infty, \alpha \not\geq \beta$
Node D Max$(2, 3) = 3 \Rightarrow \alpha = 3, \beta = +\infty, \alpha \not\geq \beta$

Step3
Now the alg backtracks to node B, where the value of $\beta$ will change as this is the turn of Min.
At Node B Min$(\infty, 3) = 3, \beta = 3, \alpha = -\infty, \alpha \not\geq \beta$
Node B value passed to Node E with $\alpha = -\infty, \beta = 3$

Step 4
At node E, Max will take its turn & value of alpha will change.
At Node E $\max(-\infty, 5) = 5$, $\beta = 3$, $\alpha \geq \beta$ ✓
At Node so right successor of E will be pruned & alg will not traverse it & the value at Node E will be 5.

Step 5
At next step, alg again backtrack the tree from node B to node A.
At node A, Max will take its turn & value of alpha will change.
At node A $\max(-\infty, 3) = 3$, $\alpha = 3$, $\beta = +\infty$, $\alpha \not\geq \beta$
& these two values now passes to right successor of A which is node C.
At Node C, $\alpha = 3$ & $\beta = +\infty$ & the same values will be passed on to node F.

Step 6
At Node F, Max will take its turn & value of alpha will change.
Node F $\max(3, 0) = 3$, $\alpha = 3$, $\beta = +\infty$, $\alpha \not\geq \beta$
Node F $\max(3, 1) = 3$, $\alpha = 3$, $\beta = +\infty$, $\alpha \not\geq \beta$
but Node F value updated to 1

Step 7
Node F returns the node value 1 to Node C.
At Node C, Min will take its turn & value of beta gets updated to 1. $\alpha = 3$, $\beta = 1$ $\alpha \geq \beta$ ✓

- So the next child of C which is G will be pruned & the alg will not compute the entire sub-tree G.

Step 8

Node C now returns the value of 1 to Node A.

At Node A, Max will make its turn so max (3, 1) = 3.

- Following is the final game tree which is showing the nodes which are computed & nodes which has never computed.
- Hence the optimal value for the Maximizer is 3 for this example.