Fundamentals of AI & ML.

Artificial Intelligence
↓ man-made
↳ thinking power

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

→ Why Artificial Intelligence ?
- Built robot to make them work harsh in environment.
- create devices which can solve real world problems easily and with accuracy.
- personal virtual assistant like Cortana & Siri.

- Main goals of AI.
- Replicate human intelligence
- Solve knowledge intensive tasks
- Build a machine which can perform tasks that require human intelligence
  ↳ playing chess etc.

→ To create AI, we should know what is intelligence and what is it comprised of ?

- Reasoning
- learning
- problem solving
- perception
- language understanding.

- To achieve all above, it should have knowledge of some basic disciplines, comprising of :

- Mathematics
- Biology
- Psychology
- Sociology
- Computer Science
- Neurons study
- Statistics.

→ Advantages of Artificial Intelligence

- High accuracy problem solving capability with less or minimal errors.
- High speed, due to their huge computing power.
- High Reliability : - Useful for risky areas
- Digital Assistant.
- Public Utility : self-driving cars, facial recognition for security.
  NLP (Natural Language Processing) to communicate with humans in human language.

→ Disadvantages of AI.

- High Cost & Maintenance.
- No feeling or emotions.
- No original creativity → DALL-E

Chat GPT.

- Cannot think out of the box.
  ↳ can only do thing it is programmed to do.

Type of AI, based on Capabilities.

◦ Weak AI or Narrow AI.

- Can only perform a dedicated task with intelligence.
- Cannot do more than it is programmed to do.
- it has limitations.

- eg - Apple's Siri, Microsoft's Cortana, Amazon Alexa.

- It also includes chess playing AIs like IBM Deep Blue & ~~AlphaZero~~. Google Alphazero. (Deepmind).

→ LLMs

◦ General AI (AGI)

- Can perform any intellectual task with efficiency of a human
- idea is to make a system which could be smarter and think like a human by its own.

◦ Super AI.

- Intelligence systems which could surpass human intelligence and can perform any task better than humans.

AI Types : Based on functionality.

→ Reactive Machines : Most basic
these systems do not store memories or past experiences.
Can only focus on current scenarios and react to them.
Eg IBM's Deep Blue.

→ Limited Memory : Can store past experiences or data for limited time period
Eg - Self-Driving Cars
↳ They tend to store speed of nearby cars, speed limit, navigation etc.

→ Theory of Mind : AI which can understand human emotions people, beliefs and be able interact socially like humans.

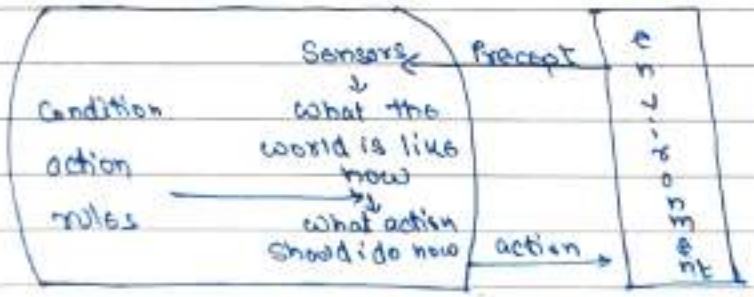
→ Self-Awareness AI : Super intelligent machines which have their own consciousness, sentiments and self awareness.

AI Agents

In Artificial Intelligence, an agent is a computer program or system that is designed to perceive its environment and then make decisions and take actions to achieve a specific goal or set of goals.

→ Simple Reflex Agents

- These agents take decisions on the basis of current percepts / scenario and forget past scenarios.
- It only succeds in a fully observable environment.
- It only works on condition action rule. Eg - Room cleaner only works if there is dirt in the room.
  - They have very limited intelligence
  - Non-adaptive to changes in environment.



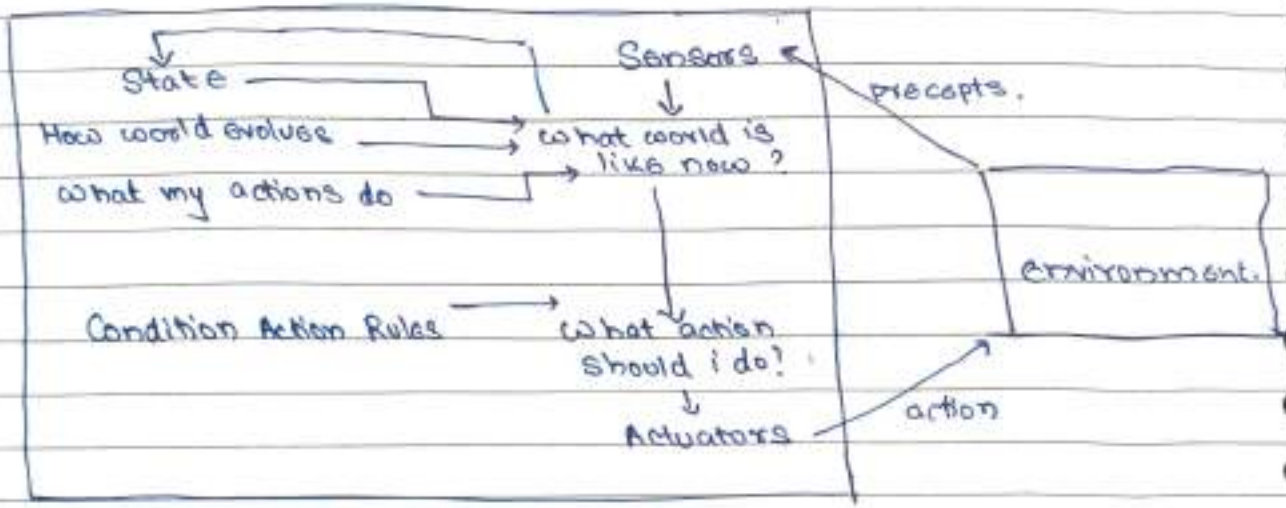
→ Model Based Reflex Agent.

- Works in partially observable environment and track the situation

Model : Knowledge about "how things happen in the world"
Internal State : representation of current state based on percept history.

- These agents perform actions based upon "their knowledge of the world".

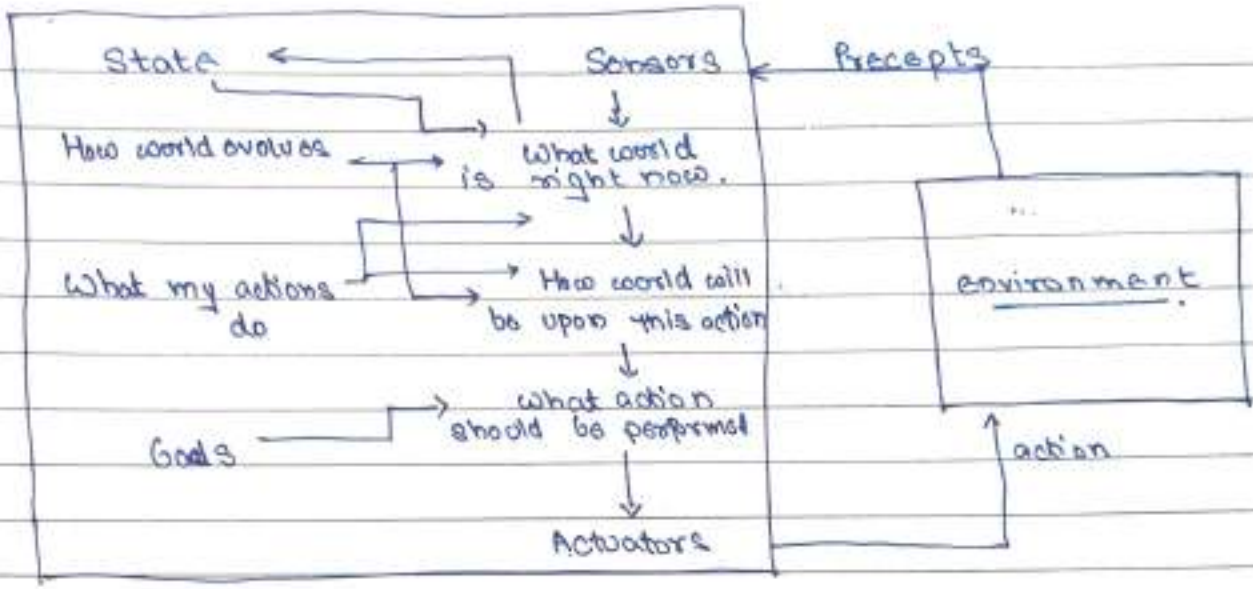
- Updating the agent stat : - How the world evolves
  requires - How agent actions effects the world.



→ Goal based agents. → proactive in nature

- The agent knows the goals it needs to achieve
- It extends on reflex based model by having goals instead of condition action rules.
- They choose an action based upon goal.

→ - These agents may have to consider a long sequence of possible actions before deciding whether goal is achieved or not.
  This long sequence of possible actions under consideration of different scenarios is called searching & planning.
  ↳ proactive agent

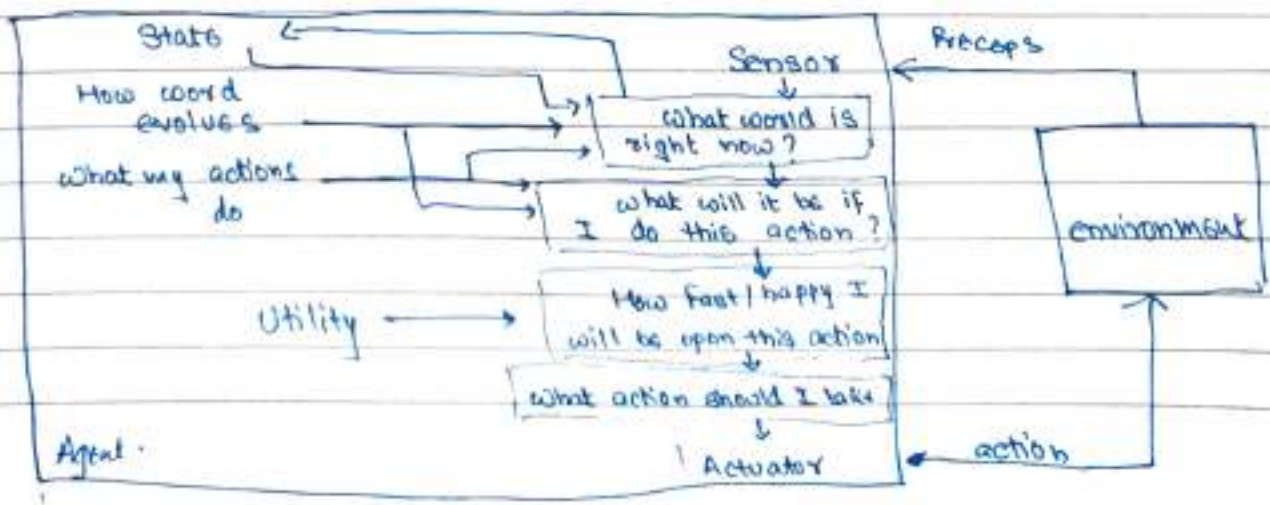


→ Utility Based Agents

- These agents are similar to goal based agent but also provides an extra component of utility measurement which makes them different by providing a measure of success at a given state.

- They are not only goal based, but also provide best way to achieve that goal.

- Best useful in scenrios where there are multiple possible alternatives.
- Utility function maps each state and checks how efficiently each action achieves goal.



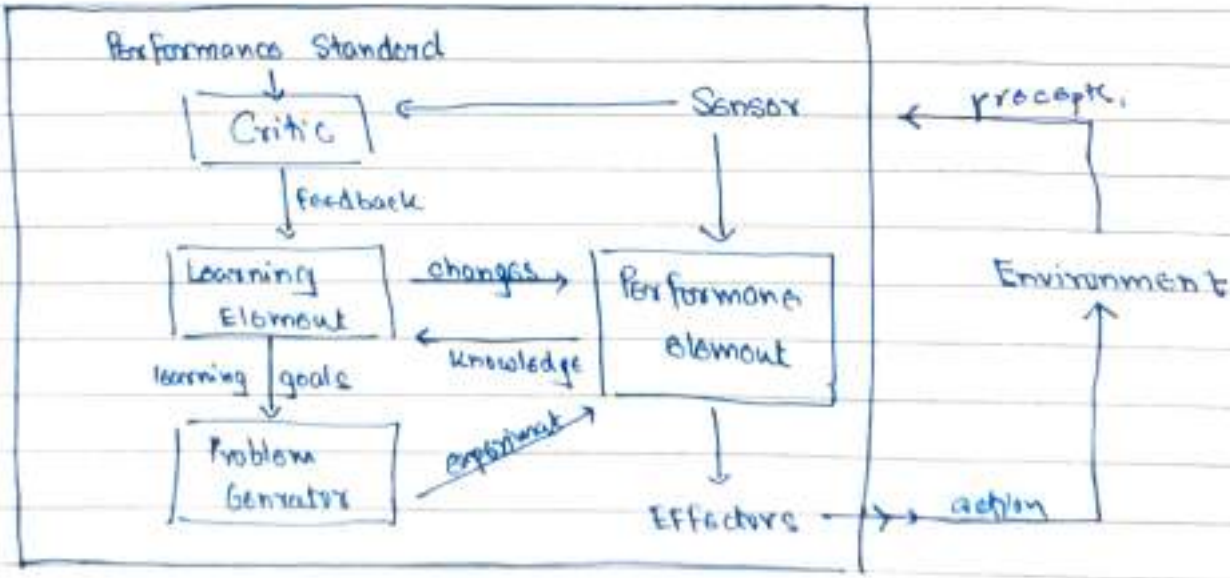
→ Learning Agent

- It is a type of agent who can learn from past experiences.
- It starts with a basic knowledge then it is able to act and adapt automatically through learning

↳ components of learning agent include:

- Learning element : responsible for learning from environment to make improvements
- Critic : learning element takes feedback from here and tell how well the agent is doing.
- Performance element : responsible for selecting external action
- Problem generator : responsible for suggesting actions that will lead better experiences

→ Thus learning agents are able to learn, analyse performance and look for new ways to improve the performance.



Intelligent Agents

→ anything that takes input through a sensor and act through actuators.

→ An agent runs in cycle of perceiving, thinking and acting.

- Human Agent for eg.

  Eyes, Ears ⟶ Sensors
  Hands, Legs ⟶ Actuators

- Robotic Agent Camera, NLP → Sensors Various motors → Actuators

- Software Agent Keyboards, File content, .eventListner → Sensory input → Sensor Display on screen → Output. code → actuator.

→ Sensor : It is a device that detects change in environment. An agent observes its environment through sensors.

→ Actuators : Component of machines that convert energy into motion for eg motor, gears etc.

→ Effectors : This are devices which affect

⇒ 4 main rules for an AI agent.

- It should be able to perceive environment
- The observation must be used to make decisions
- Decision should result in an action
- Decision must be rational

→ Rational Agent

- it acts in a way to maximize its performance with all possible actions
- Rational action is one of the most important part of AI.

- Rationality
  of an agent is measured by its performance
  - performance measure which defines the success criterion
  - Agent ~~has~~ prior knowledge of its environment
  - best possible actions agent can perform
  - The sequence of percepts.

→ Structures of AI Agent.

Agent = Architecture + Agent program
(architecture: machinery that AI agent executes on.)
(Agent program: implementation of an agent function)

Agent Function: used to map ~~percept~~ percept to an action

⇒ PEAS Representation.

P : Performance measure
E : Environment
A : Actuator
S : Sensor

Eg Self Driving Car.

- Performance Measure : Safety, time, legal drive, comfort
- Environment : Roads, other vehicles, road signs, pedestrians
- Actuators : Steering, accelerator, brake, signal, horn
- Sensors : Camera, GPS, speedometer, odometer, sonar.

It can be described as situation in which an agent is present

Agent Environment.

The environment is where an agent lives, operate and provide the agent with something to act upon.

⟹ Features of Environment. (Types)

→ Fully observable vs Partially observable.

- If agent can sense or access the complete state of an environment at each point of time then it is a fully observable environment, else it is partially observable.

- In fully observable environment there is almost no need to keep memory / internal state to keep track of environment.

→ Deterministic vs Stochastic.

- When current state of environment, determines its next state it is called deterministic.

- The stochastic environment is random in nature which is not unique and cannot be predicted by agent.

- Eg Chess : current moves in game would determine future state.
  Self-Driving Cars : it varies from time to time, and there are no patterns.

→ Competitive vs Collaborative.

- An agent is said to be on competitive environment when it competes against another agent to optimise the output. eg- Chess.
- An agent is said to be in collaborative env. when multiple agent cooperate to get the desired output.

multiple cars are found on track, the work together to avoid collision (Self-driving cars)

→ Single agent vs Multi-agent

- An environment consisting of only one agent → single agent. (eg- person left alone in a maze)
- An environment involving more than one agent → multiagent
  ↳ eg - Football game. (11)

→ Dynamic vs Static

- An environment that constantly keeps changing → dynamic (Humidity in air.)
- An idle environment with no change → static
  ↳ empty room

→ Discrete vs Continous.

- If in an environment, the number of actions that can be performed to get an output are finite → discrete (eg - Chess)
- The environment, where choices are infinite to get results → continous, eg- driving

→ Episodic vs Sequential

- In episodic environment, agent's actions are broken down, in each incident, agent receives an input from environment and then takes action. No dependency blw current & previous incidents.
- In Sequential environment, previous decisions can affect future decisions. The next action by agent depends on actions he previously took.
  eg- Checkers

→ Known vs Unknown.

- In known environment, the output for all probable actions is given.
- In Unknown environment, for an agent to make decision it has to gain knowledge and hence the environment works.

IS this works?

15

agents : it can understand its environment and act upon it.
↳ with actions
can perform action.

State : configuration b/w agent and environment that changes with time and action.
↳ there is an initial state → goal state

Actions (s) is a function that returns set of actions that can be executed in state S.
(s ← state)

Transition model : Result (s, a) is function that returns state when action A is performed.
(s ← state, a ← action)

Goal test : way to determine weather given state is goal state.

Search Algorithms

⇒ Terminologies

- Search : Searching is a step by step procedure to find something in given space. It has 3 factors
  - Search Space : it represents set of possible solutions, which system may have.
  - Start State : it is the state from where agent begins the search.
  - Goal State : it is a function which observe the current state and returns whether goal state is achieved or not.
- Search Tree : it is a tree like representation of search problem.
  The root of such tree is the root node which is the initial state.
- Actions : description of all available action to the agent.
- Transition model : description of what each action does via a model.
- Path Cost : function which assigns numeric cost to each path.
- Solution : Action sequence which leads from start node to the goal node.
- Optimal Solution : solution that has lowest cost among all solutions.

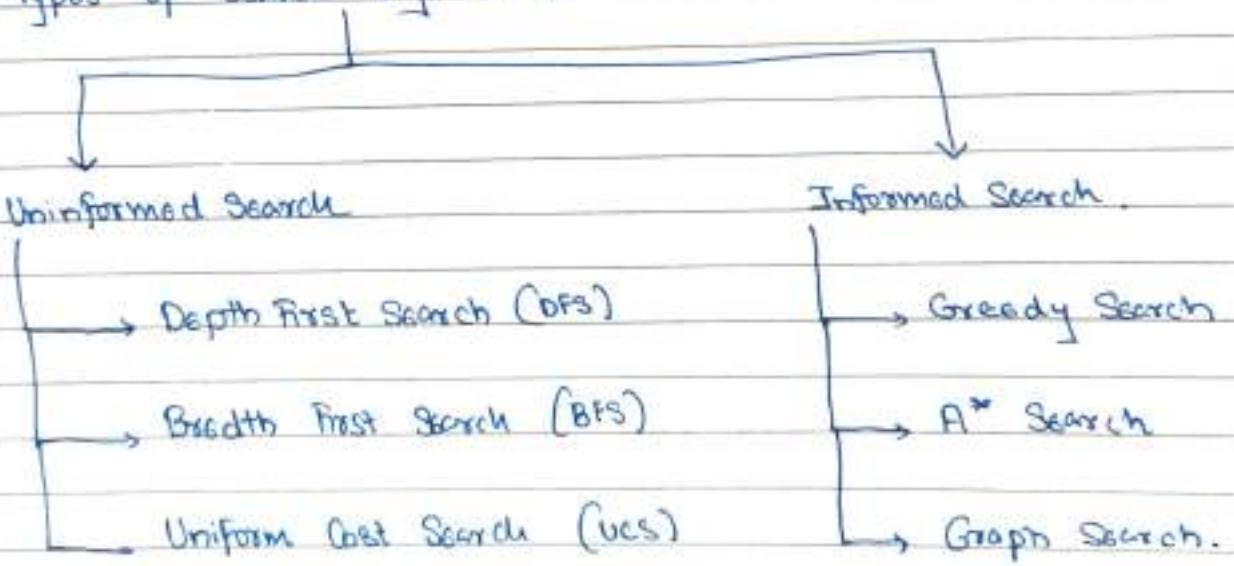
Stack: last-in-first-out data type. (DFS)

Queue: First-in-First Out (BFS)

⇒ Properties of Search Algorithms.

- Completeness: A search algorith is said to be complete if it guarantees to return a solution.
- Optimality: solution which is the best solution i.e has lowest cost path is optimal solution.
- Time Complexity: Measure of time for an algorithm to complete its task.
- Space Complexity: maximum storage space required at any point during a search, as complexity of problem.

⇒ Types of Search Algorithms.



Does not contain any domain knowledge such as closeness / location of goal.

class of general-purpose search algorithms which operates in brute force way.

Uninformed Search Algorithms

- They have no additional information about the goal node other than one provided in problem defination.
- The plans to reach the goal state from the start state differ only by the order / length of actions.
- They are also known as blind search algorithms.

→ Each Algorithm will have :

- A problem graph, containing start node S and goal node G.
- A strategy, describing the manner in which graph will travel from S → G
- A fringe, which is a data structure used to store all the possible states (nodes) that you can go from current state.
- A tree, results while traveling to the goal node.
- A Solution plan, sequence nodes from S to G (S → A → B → D → G)

→ A Brute force algorithm solves a problem through exhaustion. it goes through all possible choices until a solution is found.

Time Complexity ∝ Size of input.

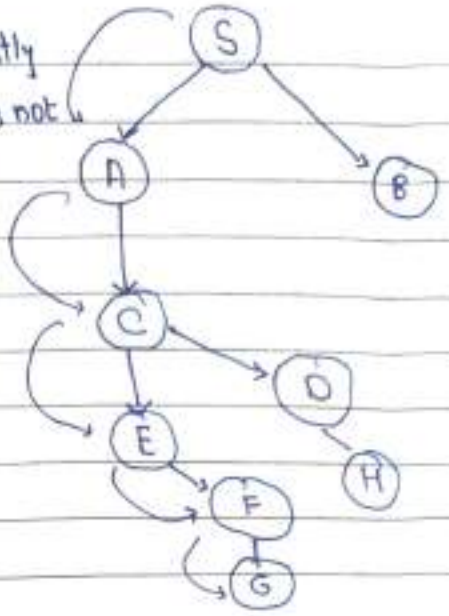
- they are simple, consistent but slow.

⇒ Depth First Search (DFS)

- DFS starts with a initial node of graph then goes deeper until it finds the goal nodes or nodes having no children.
- then, it backtracks from the dead end towards more recent node that is yet to be explored completely.

- Stack Data structure is used in DFS
- DFS works in LIFO (Last in, first out) manner.

As DFS go through the tree, it firstly goes to the deepest node first if it is not the goal node, then go back track to the parent node to check for another child node.



→ Algorithm

1) Enter root node on stack.
2) Do this until stack not empty.
   a) Remove node (POP)
      i) If node = goal node, stop
      ii) Push all children of current node into stack.

- Path : $S \rightarrow A \rightarrow C \rightarrow E \rightarrow F \rightarrow G$

→ Properties.

- d = the depth of search tree = no. of levels of search tree.
- n^i = no. of nodes in level i
- Time Complexity : Equivalent to the no. of nodes traversed in DFS.

$$T(n) = 1 + n^2 + n^3 + n^4 + \ldots + n^d = O(n^d)$$

- Space Complexity : Equivalent to how large can fringe get

$$S(n) = O(n \times d)$$

- Completeness : DFS is complete if search tree is finite, meaning for a given finite search tree, DFS will come up with a solution if it exists.

Optimality: DFS is not optimal, so no. of steps in reaching the solution or cost spent in reaching goal is high.

• Breadth First Search (BFS)

→ Advantages

- It requires less memory
- It requires less time to reach goal, if transversal is on right path

→ Disadvantages

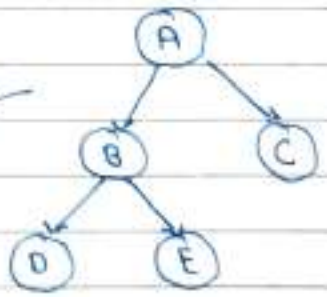
- No guarantee of finding a solution
- It can go in infinite loop.

⇒ Breadth First Search (BFS)

- BFS explores / traverses a graph in a breadth ward motion and uses a queue to remember to get the next vertex to start a search, when dead end occurs in any iteration.

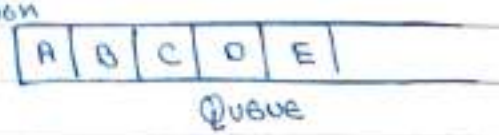
- It explores all the nodes a given depth before proceeding to the next level.
- It uses Queue data structure to implement FIFO (first in first out) manner.
- It gives optimal solution.

- BFS traverses the tree as "shallowest node first"



→ Algorithm.

- Enter starting node in queue
- If Queue is empty - Stop
- If first element on queue is goal node, then return success and stop
  ELSE
  Remove and expand first element from queue and place its children in queue.
- Go to Step 2.



Queue

→ Properties.

s = depth of shallowest solution
n^i = number of nodes in level i

Time Complexity : Equivalent to number of nodes traversed in BFS until the shallowest solution.

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

Space Complexity : Equivalent to how large can fringe get

$$S(n) = O(n^s)$$

Completeness : BFS is complete, for a given search tree. BFS will come up with solution if it exists.

Optimality : BFS is optimal as long as cost of all edges are equal.

⇒ ~~Uniform Cost Search~~.

→ Advantages.

- Find a solution if it exists
- It will try to find the minimal solution in least no. of steps.

→ Disadvantages.

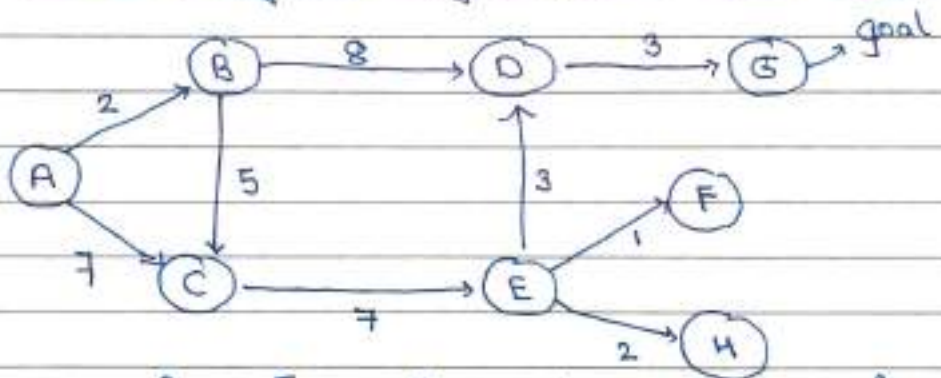
- It requires less memory
- It needs lots of time if solution is far from root node.

Uniform Cost Search Algorithm

- It is used for weighted tree / Graph traversal.
- Goal is to find lowest cost path from start node to goal node.
- It uses backtracking and priority queue for implementation.

→ Advantage : It gives truely optimal solution at every state / path with least cost is choosen.

→ Disadvantage : It does not care about no. of steps involved in searching and only concerned about the cost path. Due to which algorithm maybe stuck in infinite loop.



Path : $A \xrightarrow{2} B \xrightarrow{5} C \xrightarrow{7} E \xrightarrow{1} F \xrightarrow{1} E \xrightarrow{2} H \xrightarrow{2} E \xrightarrow{3} D \xrightarrow{3} G$
(backtrack)

Total Path Cost : 26
↳ Not totally optimal as Path: $A \xrightarrow{2} B \xrightarrow{8} D \xrightarrow{3} G$
Total cost : 13

let C = cost of solution
ε = it is each step to get closers to the goal node.

Number of steps = $C/\varepsilon + 1$

Time Complexity = $O(b^{1 + C/\varepsilon})$

Space Complexity = $O(b^{1 + C/\varepsilon})$

$f(n)$ → ~~cost~~ to next best node and ~~so on~~ it is

$g(n)$ → sum, commulative cost from start to n.

$h(n)$ → sum of costs from n to goal.

Informed Search Algorithms.

- The algorithms have information on goal state, which helps in more efficient searching. This information is obtained by Heuristics.
- It includes following: Greedy Search
  A* Tree Search
  A* Graph Search.

→ Search Heuristics: In an informed search, a heuristic is a function that estimates how close a state is to goal state. Different Heuristics are used in different algorithms.

eg- Manhattan Distance in

⇒ Greedy Search: it is any algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage.

A greedy algorithm is an approach for solving a problem by selecting the best available option at the moment. It doesn't worry whether the current best result will bring overall optimal result.

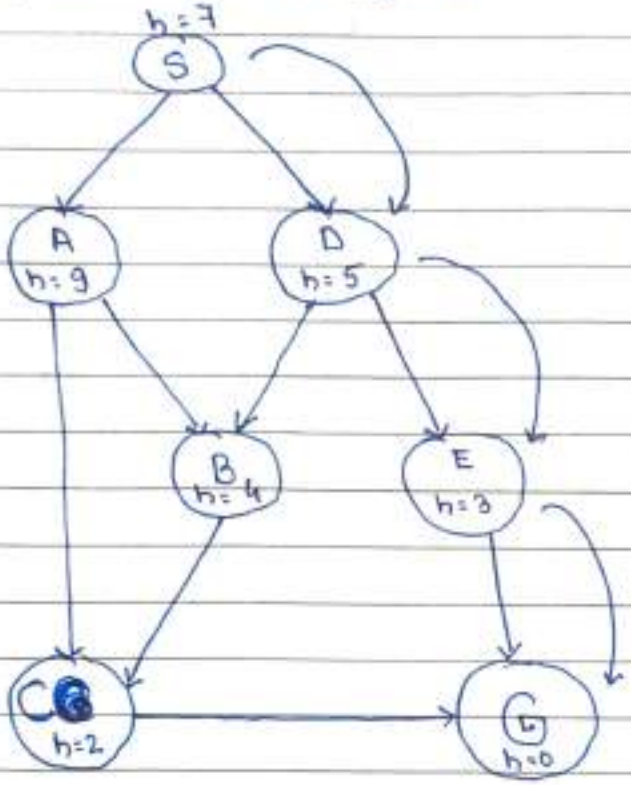
Here we expand the node closest to the goal node. The closeness is estimated by a heuristic $h(x)$.

Heuristic: A heuristic h is defined as -

$h(x)$ = Estimate of distance of node x from goal node.

low $h(x)$ → closer we are to goal node.

Q Find path from S to G using greedy search. Heuristic values of each node are given.



Path : S → D → E → G

We always jump on the node with a lower h(x) value.

→ Advantage : Works well with informed search problems, with few steps to reach goal. It gives a feasible solution.

→ Disadvantage : Can turn into a unguided DFS in worst case.

- The problems the requires either minimum or maximum results is known as optimization problem.

- Greedy method is one of the strategies used for solving optimisation problems.

A greedy algorithm makes good local choices in hope that the solution should be either feasible or optimal.

→ Applications of Greedy Algorithm.

- It is used in finding the shortest path
- It is used to find the minimum spanning tree using the prim's algorithm or the Kruskal's algorithm.
- It is used in job sequencing with a deadline
- It is used to solve fractional knapsack proble

→ Pseudocode:

```
Algorithm Greedy (a, n)
{
  Solution := 0
  for i = 0 to n do
  {
    x := select (a)
    if feasible (solution, x)
    {
      Solution := union (solution, x)
    }
    return solution
}}
```

defined as :
→ given a set of items having some weight and value / profit associated with it.

→ Knapsack Problem.

- In knapsack problem, we find set of items having less/equal weight than limit and having as much value / profit as possible.

→ Two techniques to solve problem :

- Brute-Force Approach : it tries all the possible solutions with all the different fractions but it is time-consuming approach.

- Greed Approach : here we select ratio of profit / weight and accordingly we will select the item. The item with the highest ratio would be selected.

→ Job Scheduling Problem : (optimization problem) problem of scheduling jobs out of a set of N jobs on a single processor which maximises profit as much as possible.
Consider N jobs, each taking unit time for execution. Each job is having some profit and deadline associated with it.

The greedy algorithm always gives optimal solution to this problem :

→ Step 1 : Sort all the given jobs in decreasing order of their profit

→ Step 2 : Check the value of maximum deadline
Draw a Gantt chart based on it.

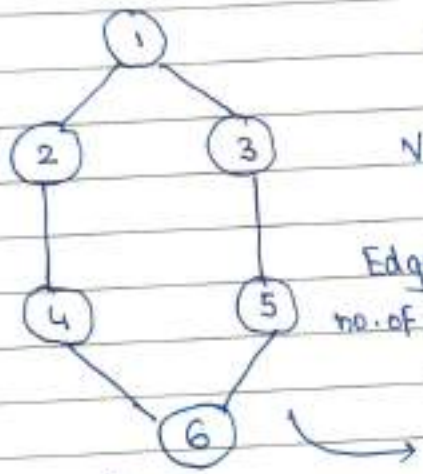
→ Step 3 : Pick up jobs one by one
Put the jobs on Gantt chart as far as possible from 0
Ensuring jobs get completed before the deadline.

Minimum Cost Spanning Tree → Prim Algorithm
→ Kruskal Algorithm.

→ Prim's Algorithm.

- It is a greedy algorithm that is used to find minimum spanning tree from a graph.

→ Converting a graph into a tree with minimum cost path.



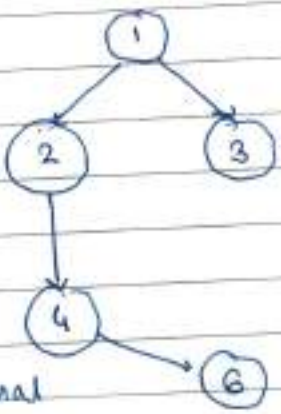
Graph

Graph = (Vertices, Edges)

$V = \{1, 2, 3, 4, 5, 6\} = 6$

Edges = {(1,2), (2,3), (3,1) .... }

no. of edge = $V - 1 = 5$



Tree.

by removing one of the edges.

To get minimum cost spanning tree, we must remove edge costing us highest and get to the minimal cost.

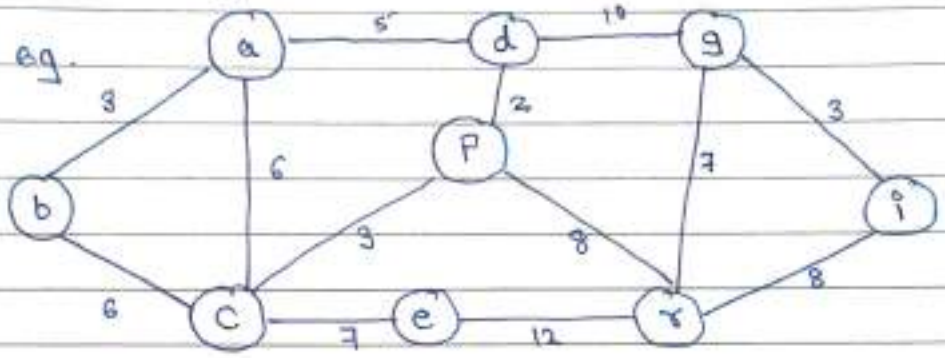
Thus, this is an optimization problem.

→ Spanning Tree : It is a subgraph of an undirected connected graph

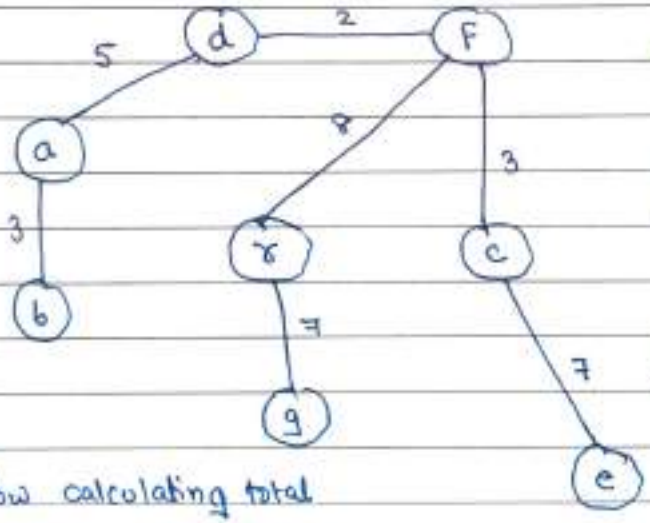
→ Minimum Spanning Tree : It can be defined as the spanning tree with sum its cost path being the minimum.

- In Prim's algorithm, we start with an edge with lowest cost, select its 2 nodes and always go with the lowest cost path adjacent.

For eg.



Minimum costing edge is , now at node d & F, find minimum edge.
it is F-C at 3.
now at all nodes, find minimum edge.
it is a-d at 5.
now a-b at 3.
go on like that.
cannot go a-C or b-C as it is forming a cycle.



Now calculating total path cost = 5+3+2+8+7+3+7
= 35.

→ Kruskal's Algorithm : It does same thing as Prim's Algorithm but takes a different approach.

It treats graph like a forest, it always select the edge with least path, and thinks they will get connected eventually.

If $h(n)$ is admissible then tree search is optimal.
If $h(n)$ is consistent then graph search is optimal.

A* Search Algorithm

if consistent, then always admissible. 29

- A* is a graph traversal and path search algorithm, which often used in many fields of computer science due to its completeness, optimality and optimal efficiency.

- From start of the node, it aims to find a path to the given goal node having the smallest cost.
- It does this by maintaining paths of originating from start-node and extending those paths one edge at a time until we reach goal node.

- A* needs to determine which of its path to extend. It does so based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal. A* selects path which has minimal

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

$g(n)$: actual cost from start node to n

$h(n)$: Estimation cost from n to goal node.

$h(n)$ is heuristic function which estimates the cost of the cheapest path from n to goal node.

⟹ Algorithm:

1. Enter the starting node in OPEN list
2. If OPEN list is empty return fail
3. Select node from OPEN list which has the smallest $(g+h)$ value
   if node == goal, return success.
4. Expand node 'n', generate all its successors
   compute $(g+h)$ for each successor node
5. If node 'n' is already in OPEN, attach back pointer
6. go to (3).

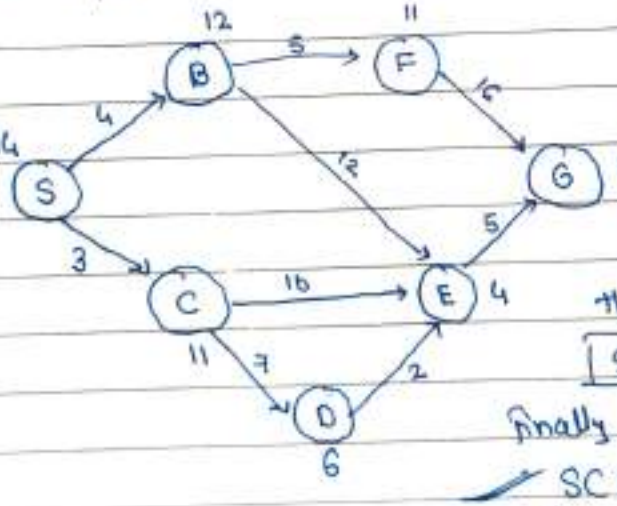
→ Advantages

- Best searching algorithm
- Optimal and complete
- Solving complex problems

→ Disadvantages.

- Doesnt always give optimal solution
- Some complexity issues
- It requires memory.

→ Example:



So, S → B : 4 + 12 = 16

S → C : 3 + 11 = 14

then

SC → E : (3 + 10) + 4 : 17

SC → D : (3 + 7) + 6 : 16

then

SCD → E : (3 + 7 + 2) + 4 : 16

finally

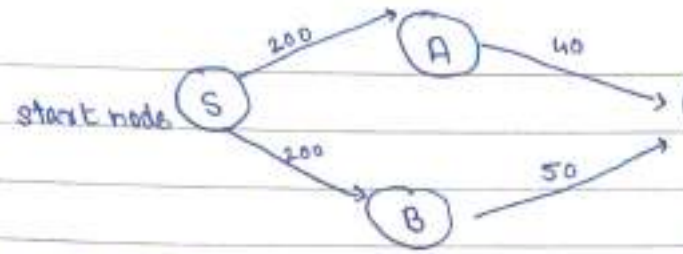
SCDE → G : 3 + 7 + 2 + 5 : 17

Pathcost : 17

→ How to make A* Admissible.

1. Admissible : In this Heuristic function, never underestimate the cost reaching the goal
   eg $H(n) <= H^*(n)$

2. Non-Admissible : In this Heuristic function, never overestimate the cost of reaching goal.



$g(A) + 200 = g(B)$

A to G = 40 } h^*

B to G = 50. }

Case 1: Overestimation.

let us say $h(A) = 80 > h^*$

$h(B) = 70$

(250)

So, here $200 + 70 = 270 < 280$, so we could choose $S \to B \to G$ and wouldn't consider going $S \to A \to G$ (240).

Case 2: Underestimation

let us say $h(A) = 30 < h^*$, here first we would

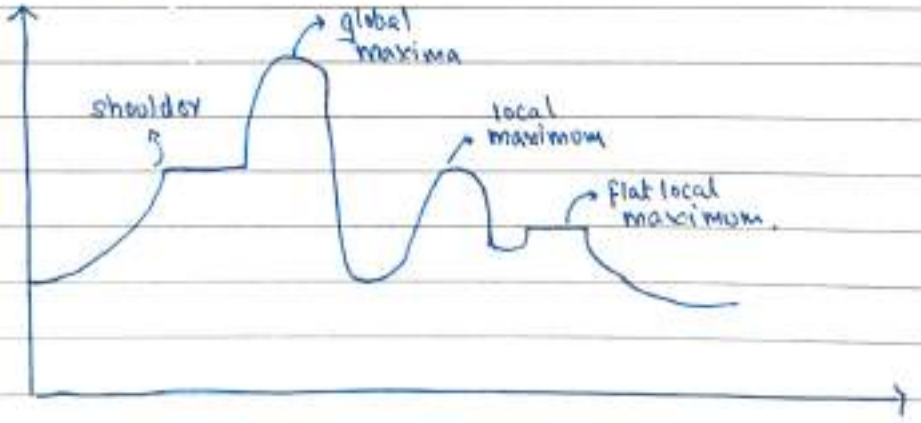
$h(B) = 20$ go to $S \to B \to G$, but after 50 path cost from $B \to G$ which is way higher than our expectations, we would backtrack to $S \to A \to G$ to check if their path cost is lesser, thus Underestimation > Overestimation.

Local Search Algorithms

- It operates using a single current node and generally move only to neighbours of that node.
- It only keeps small number of nodes in a memory. They are suitable for problems where solution is the goal state itself not the path.
- They are helpful in optimization problems in which objective is to find best state.
- Examples include: Hill Climbing and Simulated Annealing

→ Hill Climbing Algorithm.

- It is a local search algorithm which continously moves in the direction of increasing elevation / value to find best solution to the problem or peak of the mountain.
- It terminates when it reaches a peak value where no neighbour has a higher value.



- Limitations: Hill climbing cannot reach the optimal / best state (global maximum) if it enters
  - Local maxima: higher than neighbours, but lower than global maxima
  - Flat local maxima.

→ Simulated Annealing

- It is pretty similar to hill climbing, instead of picking the best move however, it picks a random move. If the move improves the situation is always accepted.

→ Constraint Satisfaction Problem : In AI and operations research, constrait satisfaction is the process of finding a solution to a set of constraints that impose conditions that variables must satisfy.

- Constraint propogation methods are used along with search to make a given problem simpler to solve.
- Examples of different problems include :
  - map coloring problem
  - crosswords, sudoku and many logical puzzles.

Constraint satisfaction depends upon 3 components :

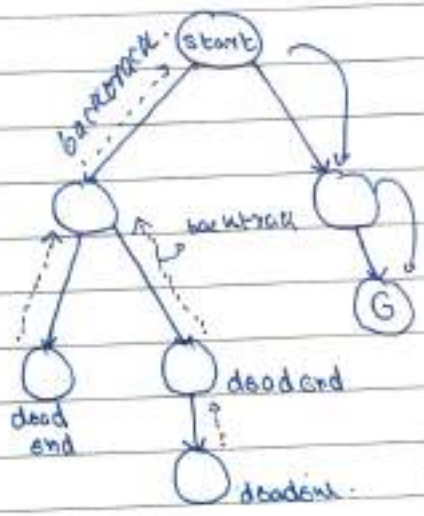
X : set of variables

D : set of domains where variables reside.
↳ there is specific domain for each variable

C : It is a set of constraints followed by a set of variables.

→ Backtracking Search

- It is depth-first approach, it chooses for one variable at a time and backtracks when that variable has no legal values left to assign
- Examples where backtracking is used: Queens puzzle, crosswords, verbal arithmetic, Sudoku and Peg Solitaire.



→ Types of constraint in backtracking

- Implicit Constraint
- Explicit Constraint.

- In backtracking technique we can backtrack to the last valid path as soon we hit dead end.
- Backtracking reduces the search space since we no longer have to follow down any path we know are valid.

→ Constraint Propogation : Inference.

- A method of influence that assigns values to variables characterising a problem in such a way that some conditions (constraints) are satisfied.
- The process of using the constraints to reduce the no. of legal values for a variable, which in turn can reduce the legal values for another variable and so on.

Game Playing

- It is the design of artificial intelligence programs to be able to play more than one game successfully.

- Game playing is a search problem defined by:

a) Initial State b) Successor function c) Goal test d) Path Cost

- It is designed and engineered in such a way that player should not feel difference between computer and a human player.

- A game must feel natural when played by an Artificial Intelligence like
  - it must obey laws of game
  - Character aware
  - Path finding (A^* algorithm)
  - Decision making
  - Planning.

- The game AI is all about the illusion of human behavior.
  - Smart to a certain extent
  - Non-repeating behaviour
  - Emotional Influences
  - Being integrated in the environment

- Game AI needs various computer science departments to gather
  a) Knowledge based systems
  b) Machine learning
  c) Multi-agent systems
  d) Computer graphics & animation
  e) Data structures

MAX → maximizing my chances of winning

MIN → minimizing my chances of winning (opponent)

⟹ Optimal Decisions in Game : here we examine problem before it arises and try to plan ahead.

- Optimal Solution : In adversial search, the optimal solution is contigent strategy. for example :

In the game chess our move in the initial state (MAX) results in change of state and now MIN's move (opponent), we will check every possible move MIN can make, then MAX's moves in that state, then MIN's all possible move and so on. In short, we will think in future to plan ahead and make optimal choices.

→ One move deep : If a game ends after one move by MAX & min Each, we can say the tree is one move deep.

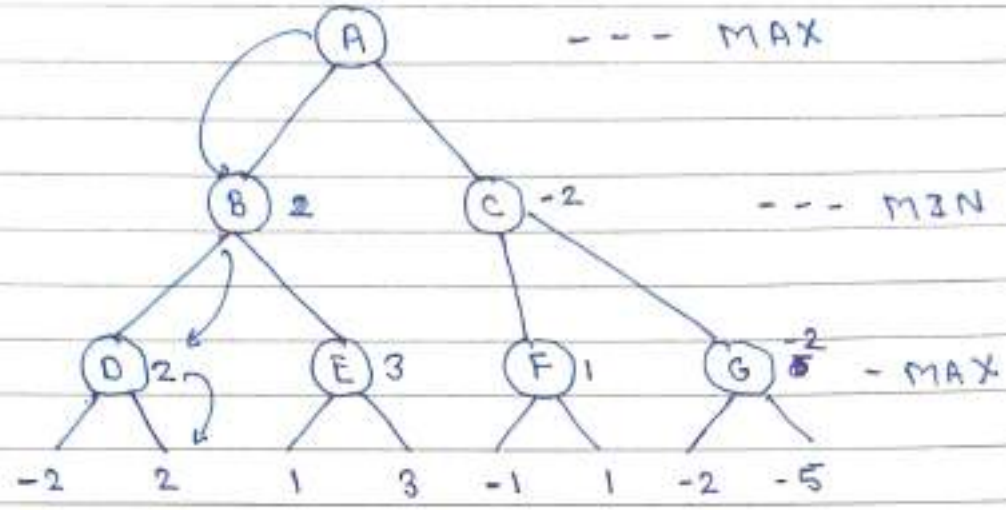
⟶ Min-Max theory : It is a specialised search algorithm that returns optimal sequence of moves for a player in zero-sum game. → (One wins, other has to loose).

- Recursive / Backtracking algorithm which is used in decision making and game theory.
- It uses recursion to search through game tree.
- Algorithm computes minmax decision for current state.

- Depth First search algorithm is used for operation of complete game tree.

- Minmax value : The minmax value of a node is the utility of being in the corresponding state, assuming that both players play optimally from there to the end of the game

→ Example of min-max algorithm.



At MAX's first turn at A, will choose B as the minimum it goes is 2. MIN will go with D at it is lower than E. MAX's will at last choose the winning move to 2.

Its Time complexity is $O(b^d)$

Alpha-Beta Pruning

- Alpha-beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the min-max algorithm in its search tree.
- It is a derivation of minmax search algorithm + optimization technique.
- In this technique we prune some tree leaves, sometimes also the entire sub-tree.

→ It has two parameters and they can be defined as:

- Alpha : The best (highest) value we have founded so far at any point along the path. Its initial value is $-\infty$.
- Beta : It chooses the lowest value we have founded along the path. Its initial value is $+\infty$.

- Main condition required is : $\alpha >= \beta$

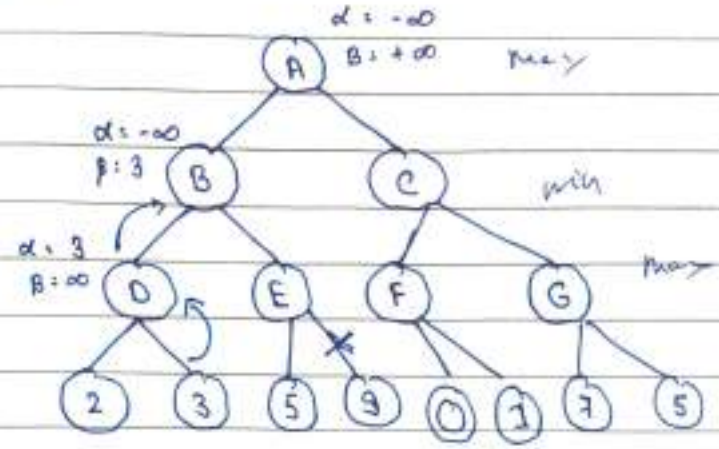
→ Key Points
- while backtracking the tree, the node values will be passed to upper nodes instead of alpha and beta.
- The max player will only update the value of alpha
- The min player will only update the value of beta.
- We will pass the alpha & beta values to the child node.

--- Working of $\alpha\beta$ Pruning :- —

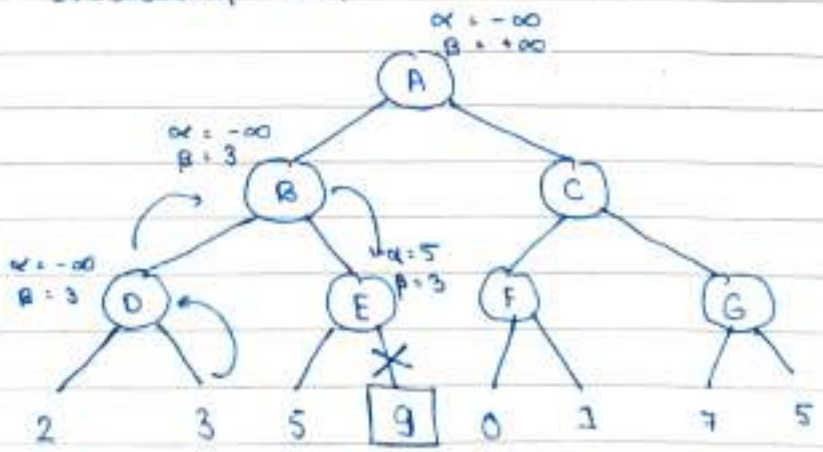
Step 1 : The max player will begin from node A where $\alpha = -\infty$, $\beta = +\infty$ and will pass values to its children, at first node B, and node B will pass this value to node D.

Step 2: Now at Node D: its max's turn so Max (2,3) will be picked, which is 3. So node value is 3, and $\alpha = 3$ $\beta = +\infty$ at node D.

Step 3: Algorithm now backtracks to node B, at B, value of β will change to 3. now that value will be passed to node E.

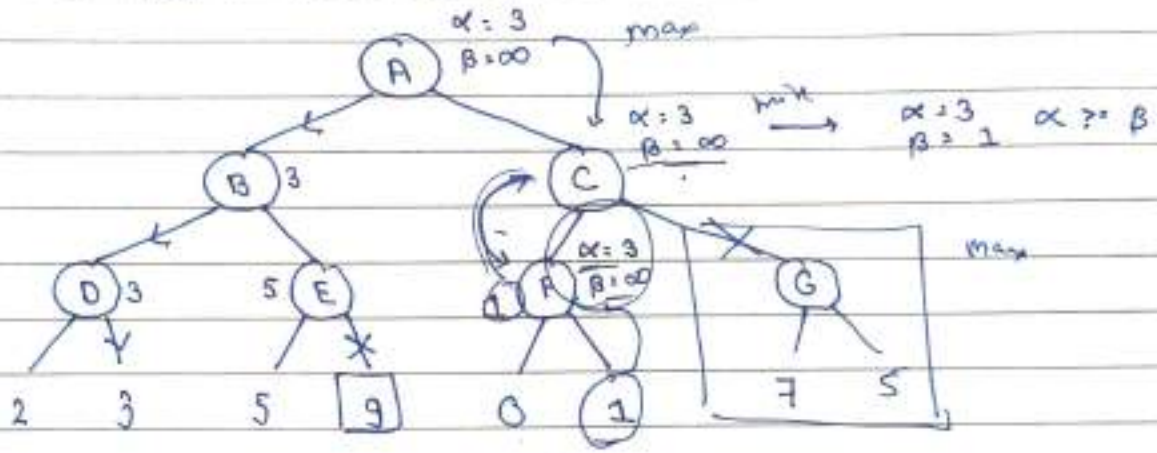


Step 4: Now at Node E, $\alpha = -\infty$ $\beta = 3$. Max will take its turn to change value of α to 5, as $-\infty < 5$. But here, $\alpha >= \beta$, now will cut prune the right successor of E.



Step 5: Now we will go backwards from node B to node A and change value of $\alpha = 3$ as $3 > -\infty$. Now this value will transfer to A's right successor, Node C.

Step 6: At Node C, we will pass its value to child Node F which will have a node value of 1.



Step 7.: We backtrack to Node C with value of 1 (Node F), thus β will get updated to 1 as $1 < \infty$ (smaller the better) but here $\alpha >= \beta$. So its another child G will be pruned.

Now returning to A with $\alpha: 3$ & $\beta: 1$ (updated), showing that Maximum MAX can achieve is 3 & minimum is 1.

The best is $A \xrightarrow{max} B \xrightarrow{min} D \xrightarrow{max} 3$.

Time Complexity.

→ Worst Ordering : At some instances, alpha-beta does not prune anything and performs similar to minimax algorithm. Sometime the optimal move is on the right side of the tree. The temporal complexity is $O(b^m)$.

→ Ideal Ordering : When there is a lot of pruning in the tree, and the best movements are made on the left side of the tree, the optimal placement for alpha-beta pruning takes place. We use DFS which initially searches left and goes deep.
Complexity for ideal ordering is $O(b^{m/2})$

Stochastic Games.

- introduced by Lloyd Shapley in early 1950s.

- It is a dynamic game with probabilistic transitions played by one or more players. The game is played in a sequence of stages. At the beginning of each stage, the game is in a certain stage.

→ Applications : Economics, evolutionary biology, and computer networks.

o Games which are unpredictable in nature are called as Stochastic games such as ~~chance~~ throwing a dice. The outcome of the game depends on skills as well as luck.