"Syllabus of Artificial Intelligence"

1) *** Approach to AI (Heuristic Search, Game playing) A*, AO*, Best first Const. Satis. DFS, BFS

2) Knowledge Representation (Approaches, Predicate logic, Reasoning)

3) Planning (Overview, Hierarchical, goal stack)

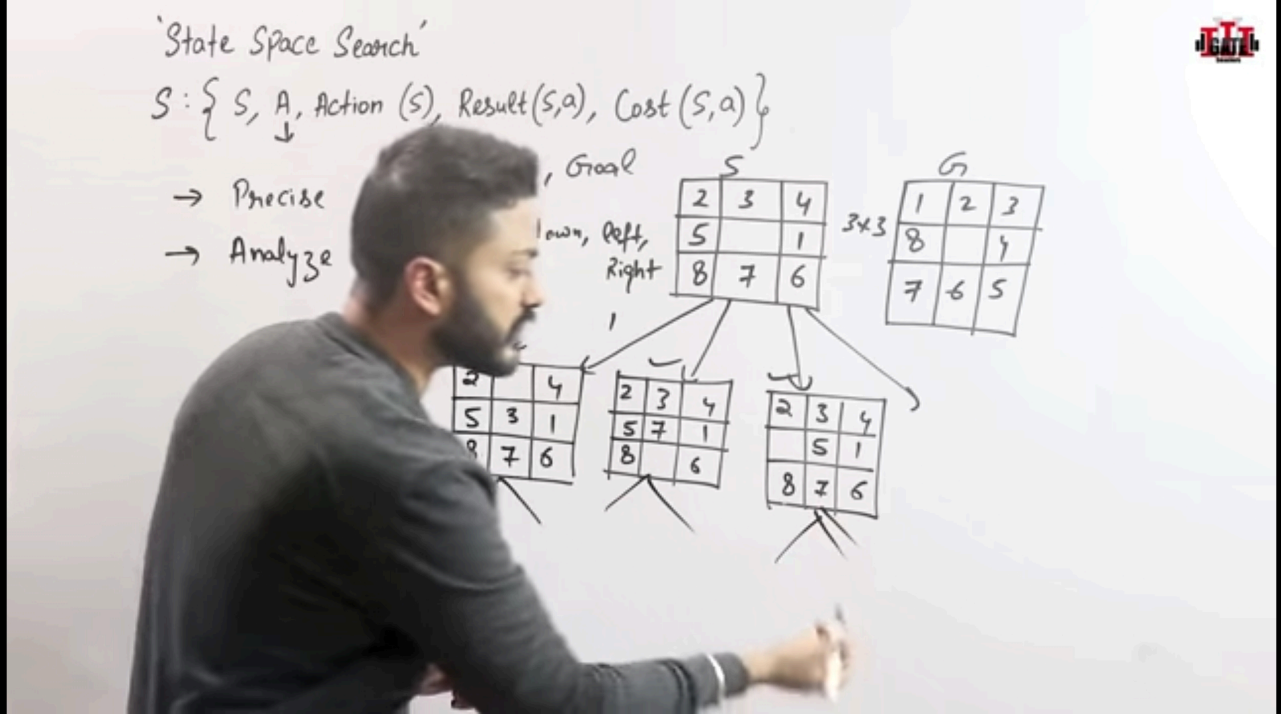
4) NLP (Syntactic, Semantic)

Rich and Knight

5) Multiagent System (types, properties)

6) *** Fuzzy Sets (Crisp, fuzzy set, α-cut, operations)

7) ANN and Genetic Algo. (Single, Multilayer feed forward, Recurrent, Machine Learning)


'State Space Search'
S : { S, A, Action (s), Result (s,a), Cost (s,a) }
→ Precise
→ Analyze
, Goal
own, Left,
Right
S
G
3x3
2 3 4
5 1
8 7 6
1 2 3
8 1
7 6 5
2 3 4
5 7 1
8 6
2 3 4
5 1
8 7 6
SUBSCRIBE

'State Space Search'

$S : \{ S, A, Action(s), Result(s,a), Cost(s,a) \}$

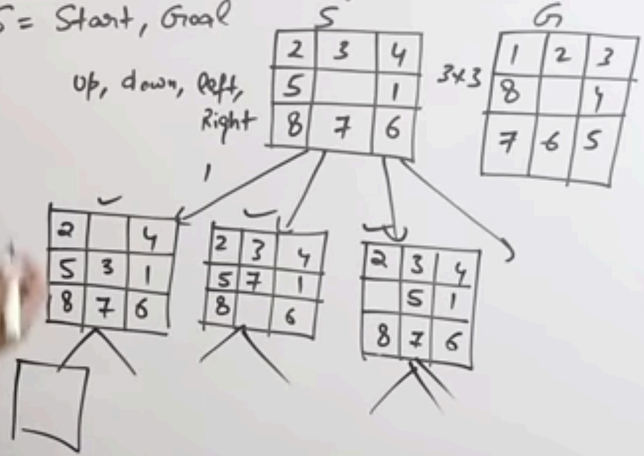
S = Start, Goal

→ Precise

→ Analyze

Uninformed

informed



Uninformed Searching

1) Search without Information
2) No knowledge
3) Time Consuming
4) More Complexity (Time, Space)
5) DFS, BFS etc.

Informed Searching

1) Search with information
2) Use knowledge to find steps to solution
3) Quick solution
4) Less Complexity (Time, Space)
5) A*, Heuristic DFS, Best first Search

BFS(Breadth First Search) → Uninformed Search technique

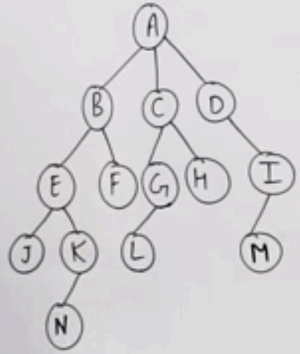
→ FIFO(Queue)

→ Shallowest Node

→ Complete

→ optimal

→ Time Complexity



~~A~~

B C D

~~B~~ D E F

~~B~~ E F G H

E F G H I

BFS(Breadth First Search) → Uninformed Search technique

→ FIFO(Queue)

↝ Shallowest Node

↝ Complete

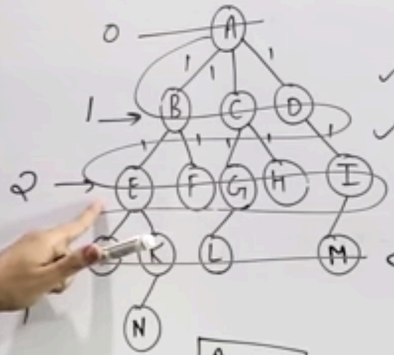
✓ optimal

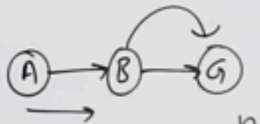
→ Time Complexity

$O(V+E)$

$O(b^d)$

b → branch factor





many tree

A

B C D

C D E F

D E F G H

E F G H I

F G H I J K

G H I J K

H I J K L

I J K L

J K L M

K L M N

L M N

M N

N

A C G

b = 3

d =

Depth First Search (DFS)

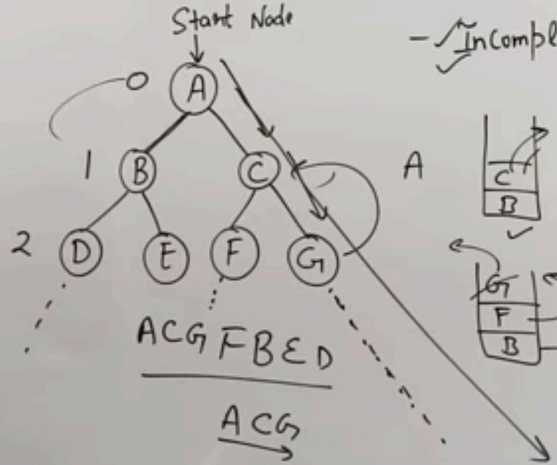
- Uninformed
- Stack (LIFO)
- Deepest Node
- Incomplete
- Time Complexity

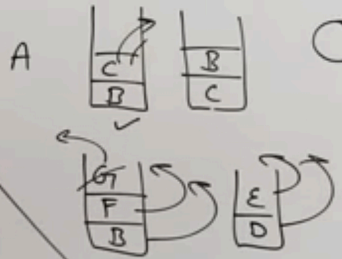
$O(V+E)$

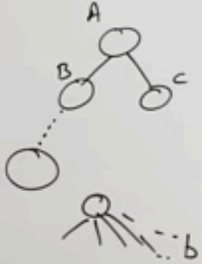
$O(b^d)$

b = branching factor

d = depth







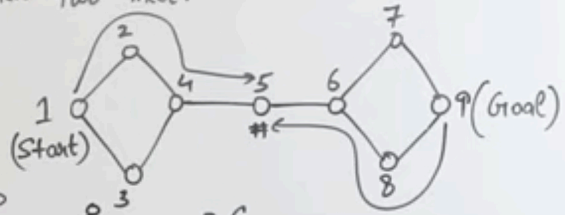
'Bidirectional Search'

→ Two simultaneous search from an initial node to goal and backward from goal to initial, stopping when two meet.

Time Complexity : $2\left(b^{d/2}\right)$

Complete in breadth first search

Not in depth first search



$O(b^d)$ $O(b^d)$ $O(2b^{d/2} \; b^{d/2} + b^{d/2})$

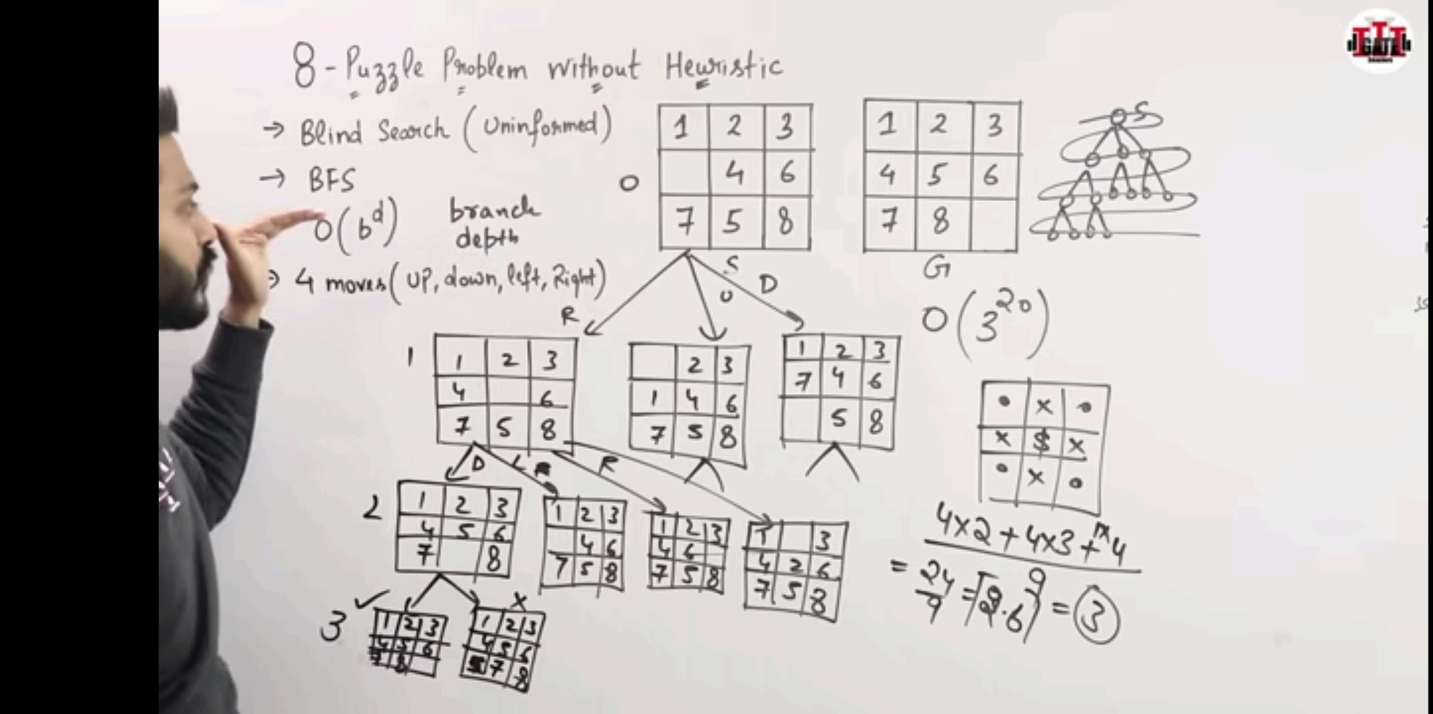
8-Puzzle Problem without Heuristic

→ Blind Search (Uninformed)

→ BFS

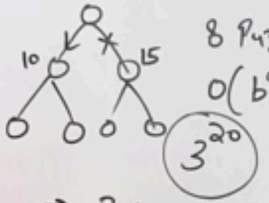
→ $O(b^d)$ branch depth

→ 4 moves (UP, down, left, Right)



`Heuristic in AI' (Rule of thumb) [What, Why, How]

→ It is a technique designed to solve a problem quickly.



8 Puzzle $O(b^d)$

15 Puzzle 10^{13}

24 Puzzle 10^{24}

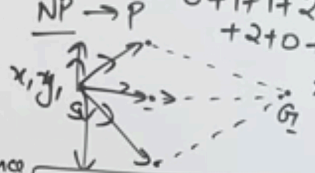
→ Reduce.

Euclidean distance

→ good

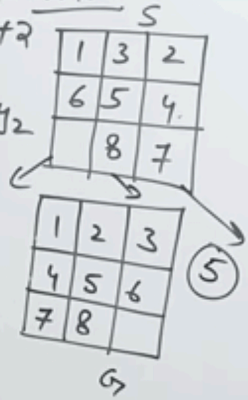
→ optimal

NP → P



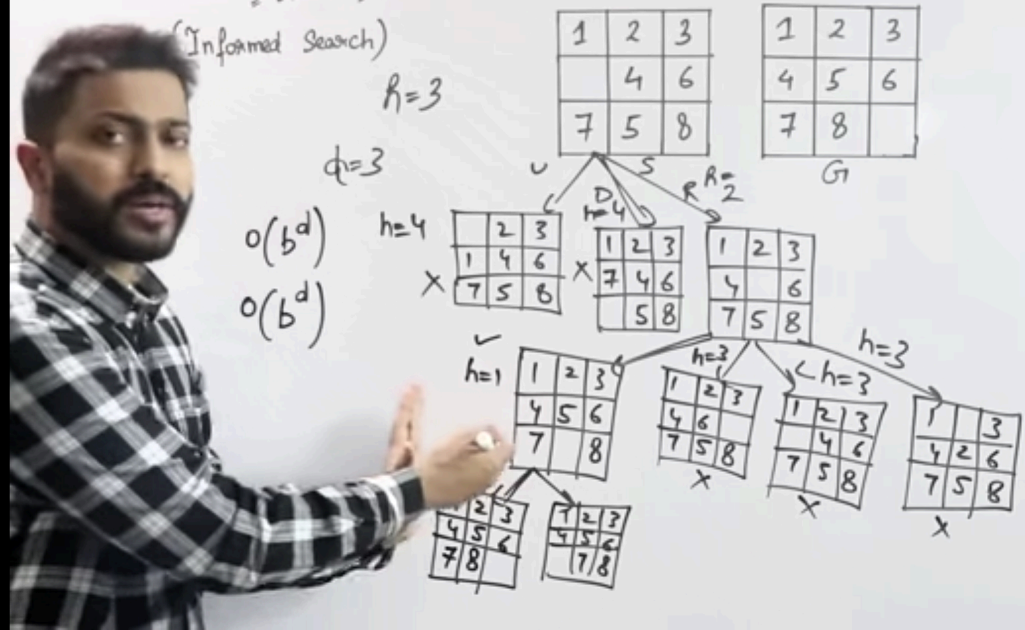
0+1+1+2+0+2
+2+0+

Manhatten



8 - Puzzle Problem with Heuristic

(Informed Search)



Generate and Test (Heuristic technique, DFS with backtracking)

1) Generate a possible solution
2) Test to see if this is a actual solution
3) If a solution is found, quit. Otherwise Go to step 1.

Properties of Good Generators

1) Complete
2) Non Redundant
3) Informed

0-99

00 00 00
00 00 01
00 02
25 25 03
25:

$(25)^3 \approx 15000$

$(100)^3 = 1M = 1$ Million

$1M = 5 \times 60 = 300$ M = Minute

10W

Best First Search (Informed, Heuristic)

Algorithm:

→ Let 'OPEN' be a priority queue containing initial state.

Loop

if OPEN is empty return failure

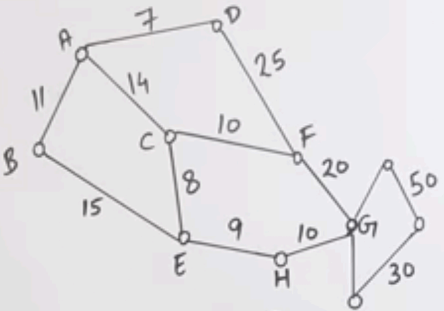
Node ← Remove-First(OPEN)

if Node is a Goal

then return the Path from initial to Node

else generate all successors of Node and Put the newly generated Node into OPEN according to their f values

END Loop



Straight line dist.

A → G1 = 40
B → G1 = 32
C → G1 = 25
D → G1 = 35
E → G1 = 19
F → G1 = 17
H → G1 = 10
G1 → G1 = 0

Best First Search (Informed, Heuristic)

Algorithm:

→ Let 'OPEN' be a priority queue containing initial state.

Loop

if OPEN is empty return failure

Node ← Remove-First(OPEN)

if Node is a Goal

then return the Path from initial to Node

else generate all successors of Node and

Put the newly generated Node into OPEN

according to their f values

END Loop

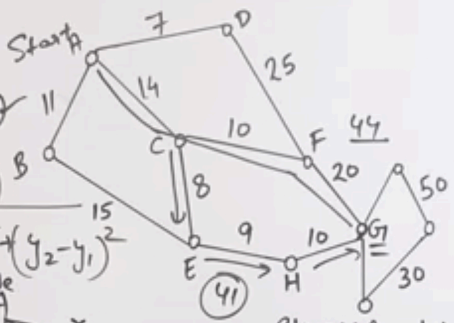
good, optimal — $O(b^m)$ — (b^d) better

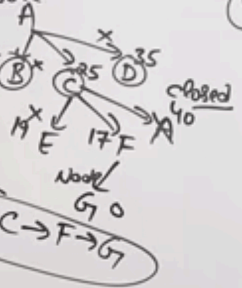
Heuristic

(Straight line dist.) $\sqrt{(x_2-x_1)^2+(y_2-y_1)^2}$

A, C, B, D Node

F, E





A → C → F → G1

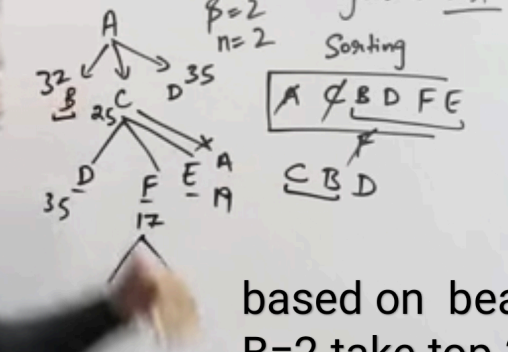
Straight line dist.

A → G1 = 40
B → G1 = 32
C → G1 = 25
D → G1 = 35
E → G1 = 19
F → G1 = 17
H → G1 = 10
G1 → G1 = 0

'Beam Search Algo'

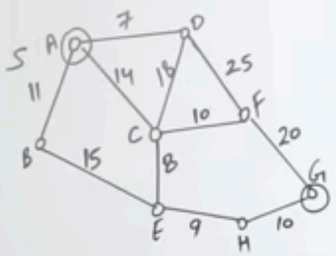
→ Take care of space complexity (constant)

→ Beam width (β) is given



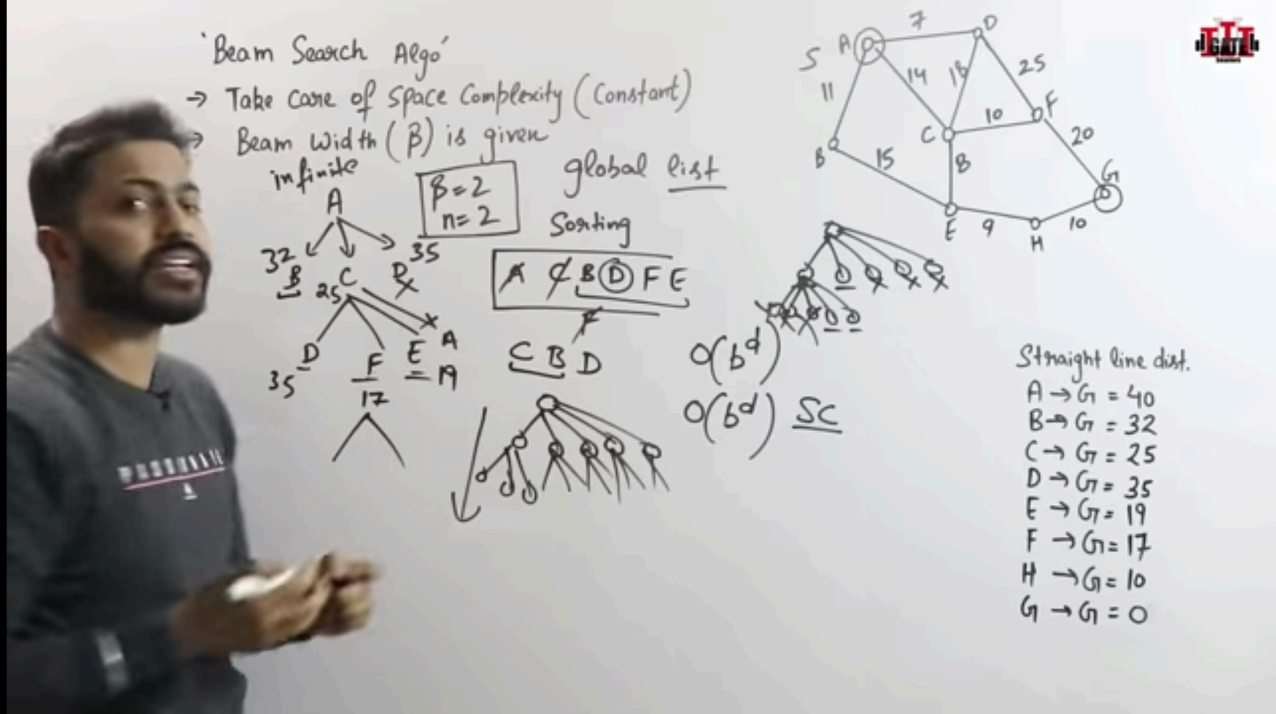
based on bean value

B=2 take top 2 of pq



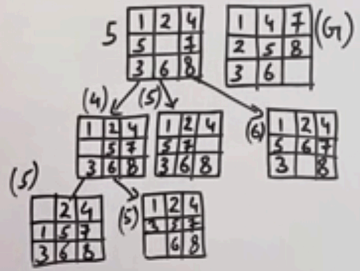
Straight line dist.

A → G = 40
B → G = 32
C → G = 25
D → G = 35
E → G = 19
F → G = 17
H → G = 10
G → G = 0


'Beam Search Algo'
→ Take care of Space Complexity (Constant)
→ Beam Width (β) is given
infinite
β=2
n=2
global list
Sorting
A
32
35
B
C
D
25
D
F
E
A
35
17
A B D F E
C B D
O(b^d)
O(b^d) SC
S
A
D
7
11
14
18
25
10
C
F
20
B
15
B
G
E
9
H
10
Straight line dist.
A → G = 40
B → G = 32
C → G = 25
D → G = 35
E → G = 19
F → G = 17
H → G = 10
G → G = 0

'Simple Hill Climbing Algorithm' (Local search Algo, greedy approach, No backtrack)

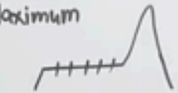
1. Evaluate the initial state
2. Loop Until a solution is found or there are no operators left
   - Select and Apply a new operator
   - Evaluate the new state:
     if goal then quit
     if better than current state then it is new current state.



Problems in Hill climbing

1) Local Maximum

2) Plateau / Flat Maximum

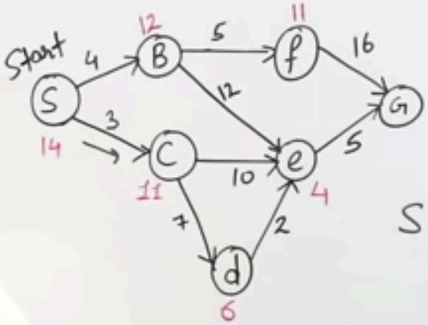


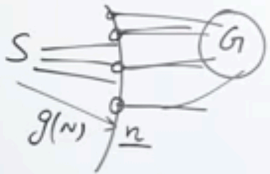
3) Ridge

A* algorithm → Informed Searching

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

- $g(N)$: Actual Cost from Start node to n
- $h(N)$: Estimation Cost from n to Goal node





$f(S) = 0 + 14 = (14)$

$S \rightarrow B$

$4 + 12 = (16)$

$S \rightarrow C$

$3 + 11 = (14)$ ✓

$SC \rightarrow e$

$SC \rightarrow d$

$3 + 10 + 4 = 17$ ✓

$SC \rightarrow d$

$3 + 7 + 6 = (16)$ ✓

Like & Subscribe

SUBSCRIBE

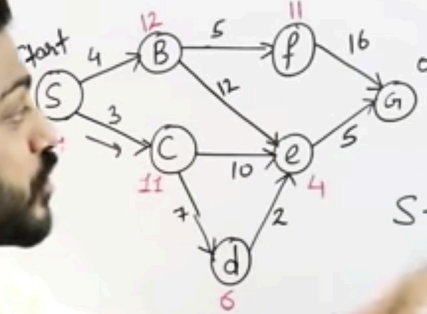
A* algorithm → Informed Searching

branch factor, depth

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

$g(N)$: Actual Cost from Start node to n

$h(N)$: Estimation Cost from n to Goal node



TC = O(V+E)
= O(b^d)
SC = O(b^d)

f(S) = 0 + 14 = (14)

S → B
+ 12
= (16) ✓

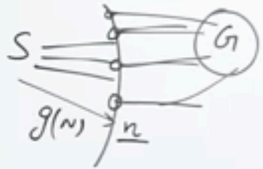
S → C
3 + 11 = (14) ✓

C → d
3 + 10 + 4
= 17 ✗

SC → d
3 + 7 + 6 = (16) ✓ ✗

SC d → e
3 + 7 + 2 + 4 = (16) ✗

Scde → G
12 + 5 + 0
= (17)

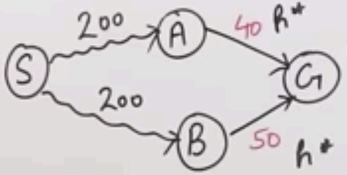


SB → f
5 + 4 + 11 = (20)

SB → e
4 + 12 + 4 = (20)

SBf → G
9 + 16 + 0
= (25)

"How to make A* admissible":



$g(G_1)$ =

$g(A) = 200$

$g(B) = 200$

$h(n) \leq h^*(n)$ - underestimation (h - estimated, h^* - Actual)

$h(n) \geq h^*(n)$ - overestimation

optimal

$S \xrightarrow{g(n)} n \xrightarrow{h} G_1 \qquad f(n) = g(n) + h(n)$

Case I overestimation

$h(A) = 80$

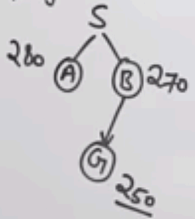
$h(B) = 70$ $\Big]$ $> h^*$

$f(A) = 200 + 80 = 280$

$f(B) = 200 + 70 = 270$

$f(G_1) = g(G_1) + h(G_1)$

$250 + 0 = \underline{250}$

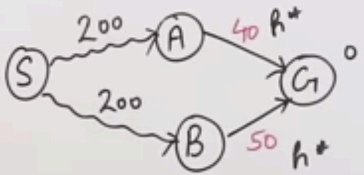


Like & Subscribe

How to make A* admissible:

$h(n) \leq h^*(n)$ - underestimation (h = estimated, h^* = Actual)

$h(n) \geq h^*(n)$ - overestimation



$g(G_1)$ =

$g(A) = 200$

$g(B) = 200$

Case I overestimation

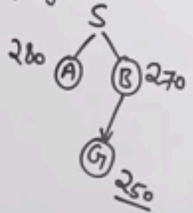
$h(A) = 80$, $h(B) = 70$ $\Big] > h^*$

$f(A) = 200 + 80 = 280$

$f(B) = 200 + 70 = 270$

$f(G_1) = g(G_1) + h(G_1)$

$= 250 + 0 = 250$



$f(G_1) = g(G_1) + h(G_1)$

$= 240 + 0$

$S \xrightarrow{g(n)} n \xrightarrow{h} G_1$ $\quad f(n) = g(n) + h(n) = 240$

Case II underestimation

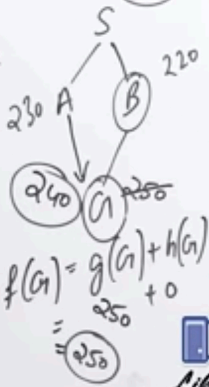
$h(A) = 30$

$h(S) = 20$

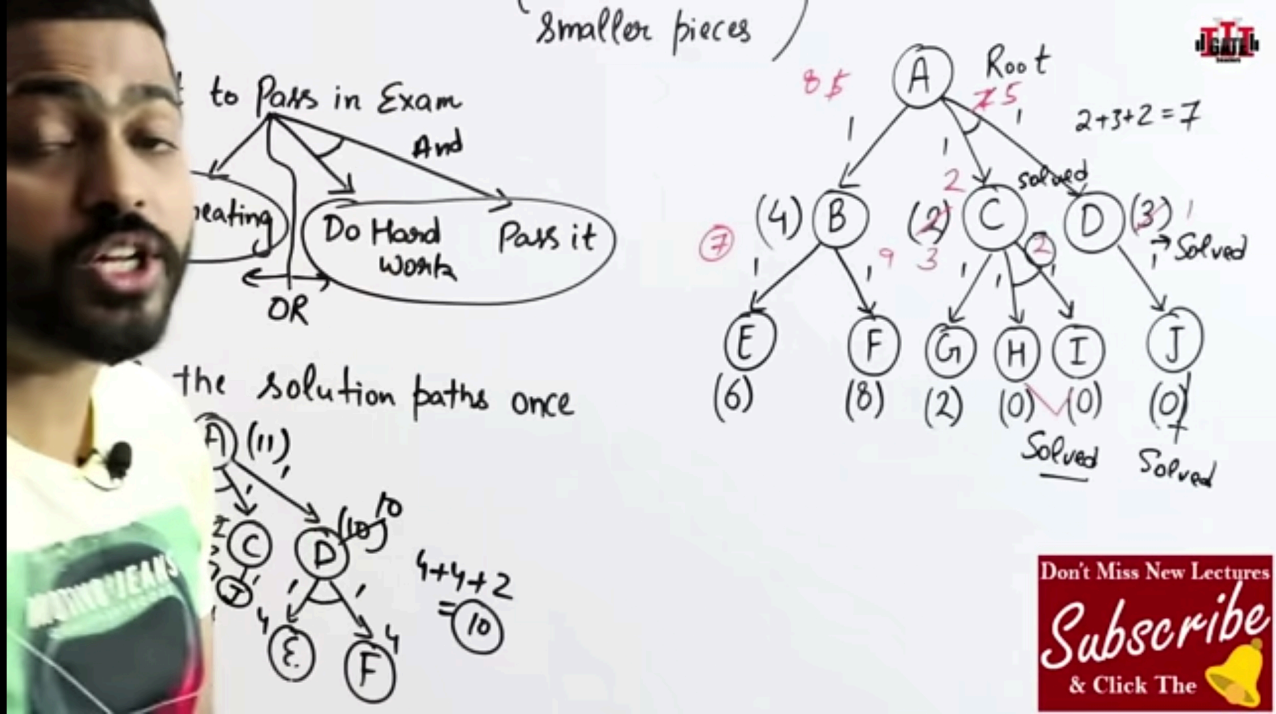
$f(A) = g(A) + h(A)$

$= 200 + 30 = 230$

$f(B) = 200 + 20 = 220$



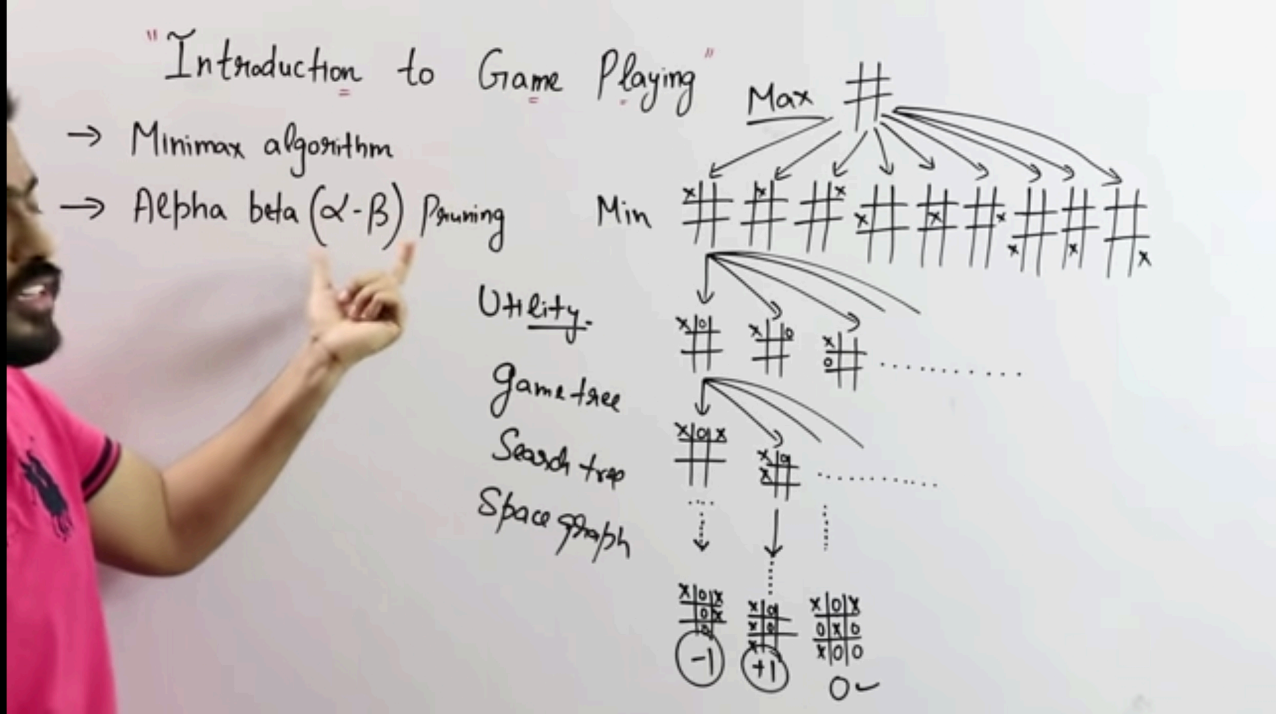
Like & Subscribe


Smaller pieces
to Pass in Exam
And
eating
Do Hard Work
Pass it
OR
the solution paths once
A (11)
C
D (10)
E
F
4+4+2 = 10
Root
8
5
2+3+2 = 7
solved
(4) B
(2) C
D (3)
Solved
(7)
E
F
G
H
I
J
(6)
(8)
(2)
(0)
(0)
(0)
Solved
Solved
Don't Miss New Lectures
Subscribe
& Click The

"Introduction to Game Playing"

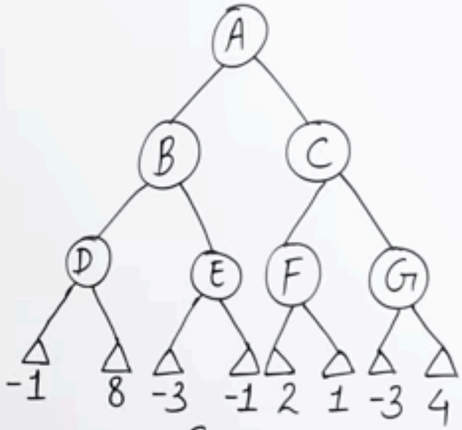
→ Minimax algorithm

→ Alpha beta (α-β) Pruning



"Minimax Algorithm"

- ⇒ Backtracking algorithm
- ⇒ Best move strategy used
- ⇒ Max will try to maximize its utility (Best Move)
- ⇒ Min will try to minimize Utility (Worst move)



Game Tree

Like & Subscribe

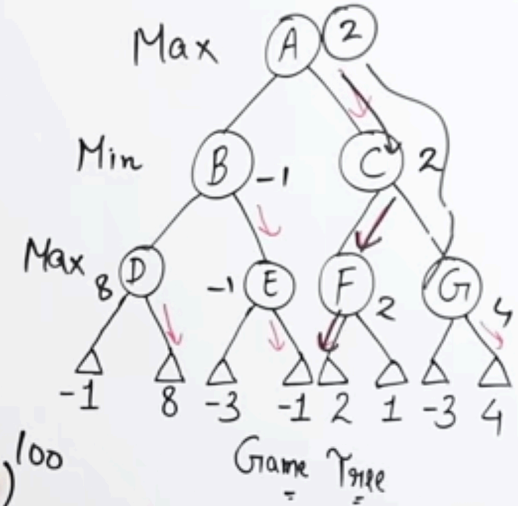
"Minimax Algorithm"

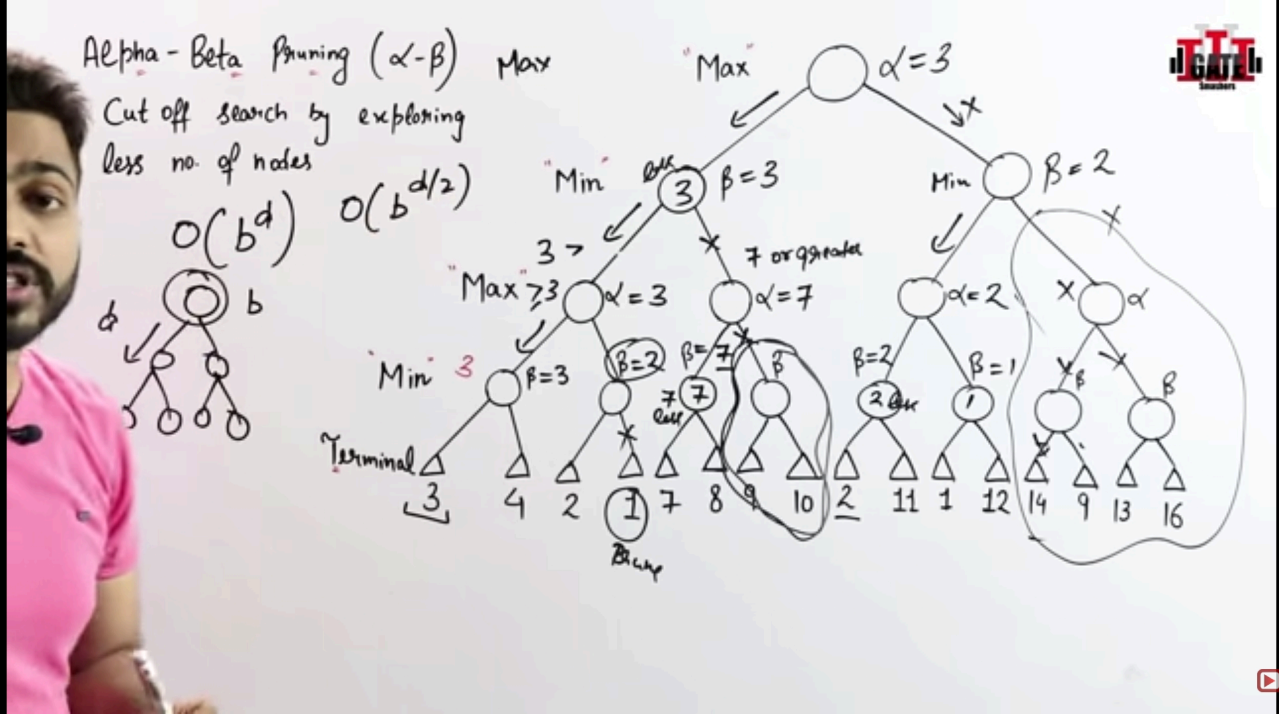
- → Backtracking algorithm
- → Best move strategy used
- → Max will try to maximize its utility (Best Move)
- → Min will try to minimize Utility (Worst move)

$= 2^2 = 9$

$O(b^d)$

$)^{100}$

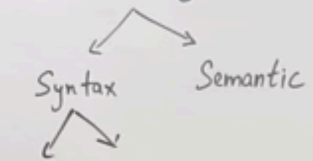



Alpha - Beta Pruning (α-β)
Cut off search by exploring less no. of nodes
O(b^d)
O(b^d/2)
d
b
Max
Min
Terminal
3
4
2
1
7
8
9
10
2
11
1
12
14
9
13
16
α=3
β=3
β=2
α=7
α=2
β=1
7 or greater
Cut
SUBSCRIBE

"Knowledge Representation and Reasoning"

- Logic → Propositional logic, Predicate logic — Syntax
- Rules → if then — Semantic
- Semantic Net → GoogleGraph
  Meaning graph
- Frame → Slots and fillers
  Object Attribute
- Script

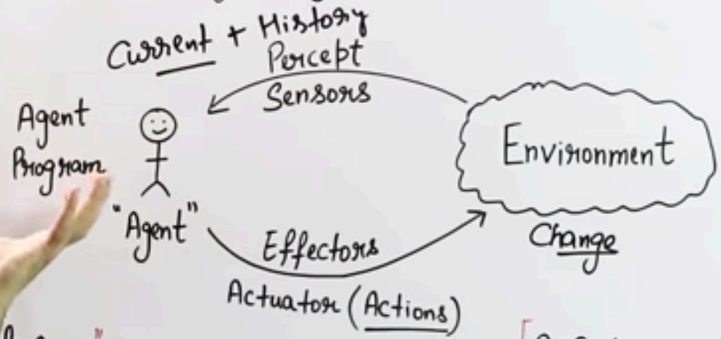
Propositional Logic (Either True or False, not Both)



- $\neg$ Negation (Today is Not Friday)
- $\vee$ Disjuction (You should Eat or Watch TV at a time)
- $\wedge$ Conjuction (Please like my video And Subscribe my channel)
- $\rightarrow$ if then (if there is Rain then the roads are Wet)
- $\leftrightarrow$ iff (I will go to Mall iff I have to do shopping)

* You can access the internet from Campus only if you are CSE student or you are not freshman.

Agents / Intelligent Agents



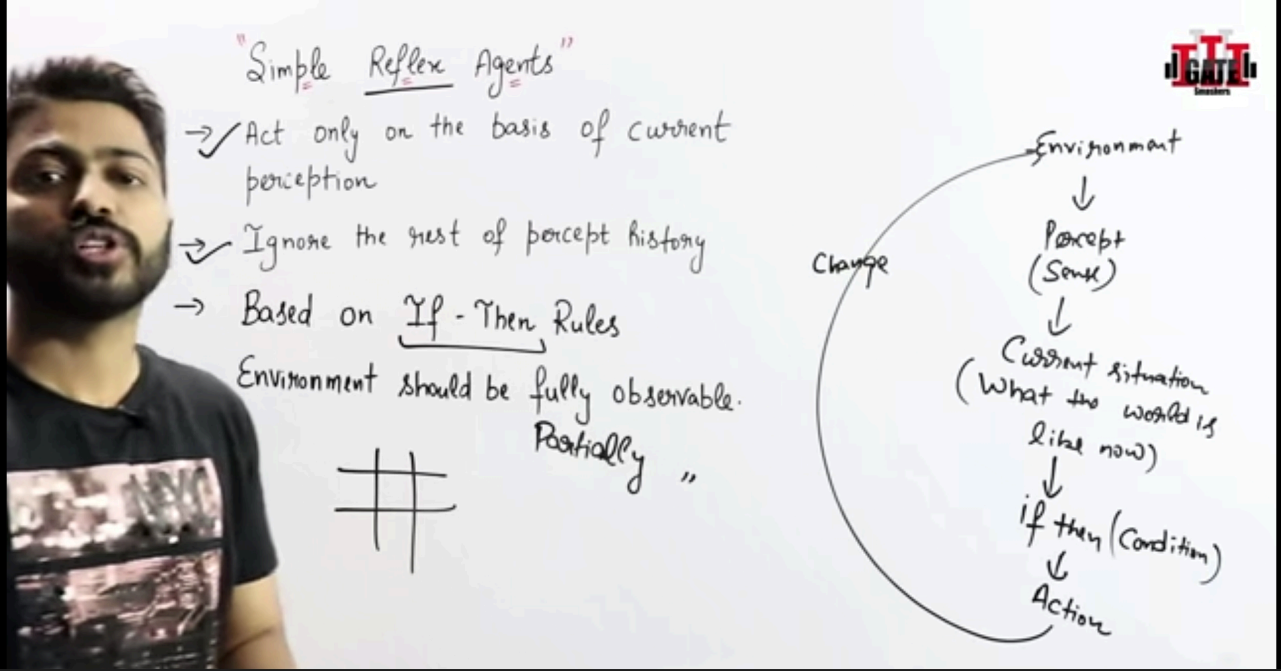
"Types"

1) Simple Reflex Agents
2) Model Based Reflex
3) Goal Based Agents
4) Utility-Based Agents
5) Learning Agents

"Goal of Agent" → High Performance
Optimized Result
Rational Action

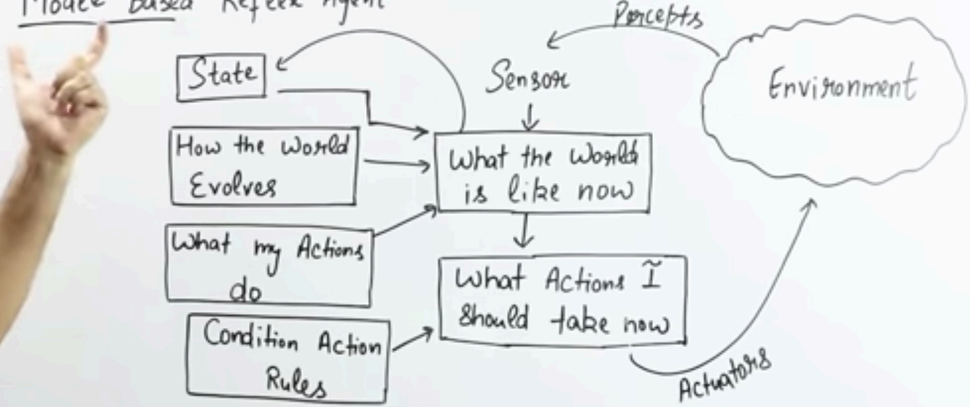
P: Performance
E: Environment
A: Actions
S: Sensors

Agent → Percept → Decision → Actions

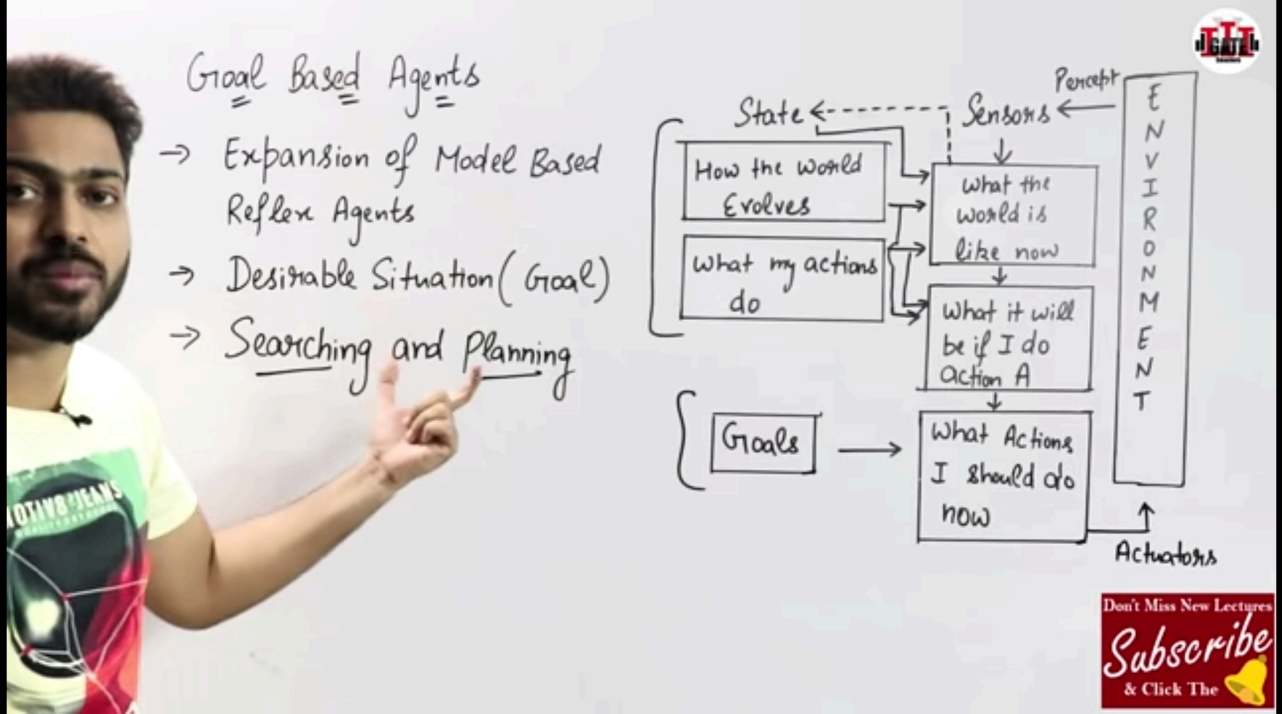


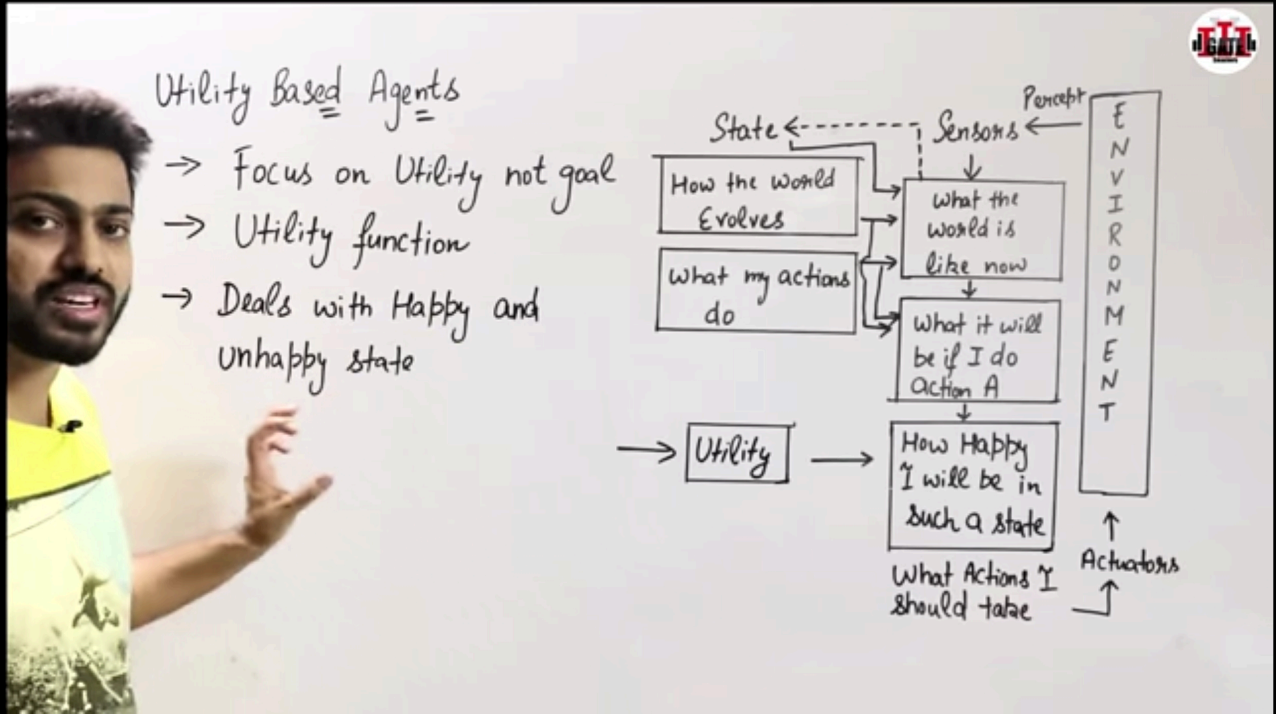
No connection

"Model Based Reflex Agent"



→ Partially observable Environment
→ Store Percept History (Internal Model)


Goal Based Agents
→ Expansion of Model Based Reflex Agents
→ Desirable Situation (Goal)
→ Searching and Planning
Percept
State
Sensors
ENVIRONMENT
How the world Evolves
What my actions do
What the world is like now
What it will be if I do action A
Goals
What Actions I should do now
Actuators
Don't Miss New Lectures
Subscribe
& Click The


Utility Based Agents
→ Focus on Utility not goal
→ Utility function
→ Deals with Happy and unhappy state
Percept
State
Sensors
ENVIRONMENT
How the world Evolves
What my actions do
What the world is like now
What it will be if I do action A
Utility
How Happy I will be in such a state
What Actions I should take
Actuators

Adversarial Search

- It is a **game-playing** technique where agents are surrounded by a competitive environment. A conflicting goal is given to the agents.
- Game-playing means discussing those games where **human intelligence** & **logic factor** is used. **Tic-tac-toe, chess, checkers.**
- This search is based on the concept of **'Game Theory.'**
- To complete game, one has to win the game & other looses automatically.

We are opponents- I win, you loose.

Techniques required to get the best optimal solution

There is always a need to choose those algorithms which provide the best optimal solution in a limited time. So, we use the following techniques which could fulfill our requirements:

- **Pruning:** A technique which allows ignoring the unwanted portions of a search tree which make no difference in its final result.
- **Heuristic Evaluation Function:** It allows to approximate the cost value at each level of the search tree, before reaching the goal node.

Types of algorithms in Adversarial search

- In a **normal search**, we follow a sequence of actions to reach the goal or to finish the game optimally. But in an **adversarial search**, the result depends on the players which will decide the result of the game.
- It is also obvious that the solution for the goal state will be an optimal solution because the player will try to win the game with the shortest path and under limited time.
- There are following types of adversarial search:
  - **Min-max Algorithm**
  - **Alpha-beta Pruning**

Optimal Decision Making in Games

Humans' intellectual capacities have been engaged by games for as long as civilization has existed, sometimes to an alarming degree. Games are an intriguing subject for AI researchers because of their abstract character. A game's state is simple to depict, and actors are usually limited to a small number of actions with predetermined results. Physical games, such as croquet and ice hockey, contain significantly more intricate descriptions, a much wider variety of possible actions, and rather ambiguous regulations defining the legality of activities. With the exception of robot soccer, these physical games have not piqued the AI community's interest.

Games are usually intriguing because they are difficult to solve. Chess, for example, has an average branching factor of around 35, and games frequently stretch to 50 moves per player, therefore the search tree has roughly 35100 or 10154 nodes (despite the search graph having "only" about 1040 unique nodes). As a result, games, like the real world, necessitate the ability to make some sort of decision even when calculating the best option is impossible.

Inefficiency is also heavily punished in games. Whereas a half-efficient implementation of A search will merely take twice as long to complete, a chess software that is half as efficient in utilizing its available time will almost certainly be beaten to death, all other factors being equal. As a result of this research, a number of intriguing suggestions for making the most use of time have emerged.

Optimal Decision Making in Games

Let us start with games with two players, whom we'll refer to as MAX and MIN for obvious reasons. MAX is the first to move, and then they take turns until the game is finished. At the conclusion of the game, the victorious player receives points, while the loser receives penalties. A game can be formalized as a type of search problem that has the following elements:

- **S_0:** The initial state of the game, which describes how it is set up at the start.
- **Player (s):** Defines which player in a state has the move.
- **Actions (s):** Returns a state's set of legal moves.
- **Result (s, a):** A transition model that defines a move's outcome.
- **Terminal-Test (s):** A terminal test that returns true if the game is over but false otherwise. Terminal states are those in which the game has come to a conclusion.
- **Utility (s, p):** A utility function (also known as a payout function or objective function) determines the final numeric value for a game that concludes in the terminal state ***s*** for player ***p***. The result in chess is a win, a loss, or a draw, with values of +1, 0, or 1/2. Backgammon's payoffs range from 0 to +192, but certain games have a greater range of possible outcomes. A zero-sum game is defined (confusingly) as one in which the total reward to all players is the same for each game instance. Chess is a zero-sum game because each game has a payoff of 0 + 1, 1 + 0, or 1/2 + 1/2. "Constant-sum" would have been a preferable name, 22 but zero-sum is the usual term and makes sense if each participant is charged 1.

The game tree for the game is defined by the beginning state, ACTIONS function, and RESULT function—a tree in which the nodes are game states and the edges represent movements. The figure below depicts a portion of the tic-tac-toe game tree (noughts and crosses). MAX may make nine different maneuvers from his starting position. The game alternates between MAXs setting an X and MINs placing an O until we reach leaf nodes corresponding to terminal states, such as one player having three in a row or all of the squares being filled. The utility value of the terminal state from the perspective of MAX is shown by the number on each leaf node; high values are thought to be beneficial for MAX and bad for MIN

The game tree for tic-tac-toe is relatively short, with just 9! = 362,880 terminal nodes. However, because there are over 1040 nodes in chess, the game tree is better viewed as a theoretical construct that cannot be realized in the actual world. But, no matter how big the game tree is, MAX's goal is to find a solid move. A tree that is superimposed on the whole game tree and

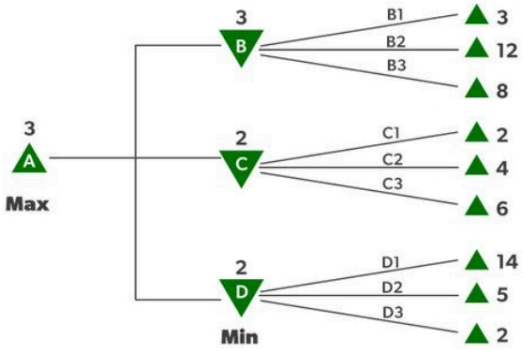
A sequence of actions leading to a goal state—a terminal state that is a win—would be the best solution in a typical search problem. MIN has something to say about it in an adversarial search. MAX must therefore devise a contingent strategy that specifies M A X's initial state move, then MAX's movements in the states resulting from every conceivable MIN response, then MAX's moves in the states resulting from every possible MIN reaction to those moves, and so on. This is quite similar to the AND-OR search method, with MAX acting as OR and MIN acting as AND. When playing an infallible opponent, an optimal strategy produces results that are as least as excellent as any other plan. We'll start by demonstrating how to find the best plan.

We'll move to the trivial game in the ***figure below*** since even a simple game like tic-tac-toe is too complex for us to draw the full game tree on one page. MAX's root node moves are designated by the letters a1, a2, and a3. MIN's probable answers to a1 are b1, b2, b3, and so on. This game is over after MAX and MIN each make one move. (In game terms, this tree consists of two half-moves and is one move deep, each of which is referred to as a ply.) The terminal states in this game have utility values ranging from 2 to 14.

Game's Utility Function

The optimal strategy can be found from the minimax value of each node, which we express as MINIMAX, given a game tree (n). Assuming that both players play optimally from there through the finish of the game, the utility (for MAX) of being in the corresponding state is the node's minimax value. The usefulness of a terminal state is obviously its minimax value. Furthermore, if given the option, MAX prefers to shift to a maximum value state, whereas MIN wants to move to a minimum value state. So here's what we've got:

$$\text{MINIMAX}\ (s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}\ (s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$



Optimal Decision Making in Multiplayer Games

Let's use these definitions to analyze the game tree shown in ***the figure above***. The game's UTILITY function provides utility values to the terminal nodes on the bottom level. Because the first MIN node, B, has three successor states with values of 3, 12, and 8, its minimax value is 3. Minimax value 2 is also used by the other two MIN nodes. The root node is a MAX node, with minimax values of 3, 2, and 2, resulting in a minimax value of 3. We can also find the root of the minimax decision: action a1 is the best option for MAX since it leads to the highest minimax value.

This concept of optimal MAX play requires that MIN plays optimally as well—it maximizes MAX's worst-case outcome. What happens if MIN isn't performing at its best? Then it's a simple matter of demonstrating that MAX can perform even better. Other strategies may outperform the minimax method against suboptimal opponents, but they will always outperform optimal opponents.

Article Tags : Machine Learning Artificial Intelligence

IMPERFECT REAL-TIME DECISIONS

- Claude Shannon's paper *Programming a Computer for Playing Chess (1950) proposed* instead that programs should cut off the search earlier and apply a heuristic **evaluation function** to states in the search,effectiely turning nonterminal nodes into terminal leaves.
- In other words, the suggestion is to alter minimax or alpha–beta in two ways: replace the utility function by a heuristic evaluation function EVAL, which estimates the position's utility, and replace the terminal test by a **cutoff test that decides when to apply EVAL.**

Stochastic Games

- Many games are **unpredictable** in nature, such as those involving **dice throw**. These games are called as **Stochastic Games**. The outcome of the game depends on skills as well as luck.
- In the Stochastic Games, the winner of the game is not only decided by the skill but also by luck.
- Examples are

❖ Gambling game

❖ Golf ball game

❖ Backgammon etc…

Stochastic Search Algorithms(Methods)

- Stochastic search algorithms are **designed for problems** with inherent random noise or deterministic problems solved by injected randomness.
- Desired properties of search methods are
  - ❖ High probability of finding near-optimal solutions (Effectiveness)
  - ❖ Short processing time (Efficiency)
- They are usually **conflicting**; a compromise is offered by stochastic techniques where certain steps are based on random choice.

Why Stochastic Search?

- Stochastic search is the method of choice for solving many hard combinatorial problems.
- Ability of solving hard combinatorial problems has increased significantly
- Solution of large propositional satisfiability problems
- Solution of large travelling salesman problems Good results in new application areas

A **partially observable system** is one in which the entire state of the system is not fully visible to an external sensor. In a partially observable system the observer may utilise a memory system in order to add information to the observer's understanding of the system.[1]

An example of a partially observable system would be a card game in which some of the cards are discarded into a pile face down. In this case the observer is only able to view their own cards and potentially those of the dealer. They are not able to view the face-down (used) cards, nor the cards that will be dealt at some stage in the future. A memory system can be used to remember the previously dealt cards that are now on the used pile. This adds to the total sum of knowledge that the observer can use to make decisions.

In contrast, a fully observable system would be that of chess. In chess (apart from the 'who is moving next' state, and minor subtleties such as whether a side has castled, which may not be clear) the full state of the system is observable at any point in time.

Partially observable is a term used in a variety of mathematical settings, including that of artificial intelligence and partially observable Markov decision processes.

players to keep a ball in the air.

9. Simulation: A game with some real world implication/element so that the player's skills in the real world domain will be improved after playing the game. The game can also relate the player to real world issues.	195	12.56
10. Sports: A game that has an explicit sporting event focus, such as Olympics or NBA etc. The player does NOT have to be physically active to play the	31	2.00

'Constraint Satisfaction Problem (CSP)'

→ CSP consists of three components V, D, C

→ V is set of variables $\{V_1\ V_2 \ldots V_n\}$

→ D is set of domains $\{D_1\ D_2\ D_3 \ldots D_n\}$ one for each variable

→ C is set of constraints that specify allowable combination of values.

$C_i = (\text{Scope}, \text{rel})$

→ Where Scope is set of variables that participate in constraint

→ Rel is relation that define the values that variable can take

$\{C_1\ C_2\ C_3\}$

$V_1\ V_2$

A B

(1,2) (2,4)

$C_1 = \{(V_1, V_2), (V_1 \neq V_2)\}$

$C_1 = (V_1, V_2), (A, B)$

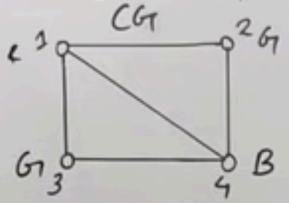
$C_1 = (V_1, V_2), (1,2)(1,4)(2,4)$

9×9 (81) (1,9)

(3×3) (81)

(1 to 9)

'Constraint Satisfaction Problem (CSP)' Backtracking



V = { 1, 2, 3, 4 }

D = { Red, Green, Blue }

C = { 1≠2, 1≠3, 1≠4, 2≠4, 3≠4 }

Initial.

	1	2	3	4
Initial Dom.	R, G, B	RGB	RGB	RGB
1=R	R	GB	GB	GB
2=G	R	G	GB	B
3=G	R	G	G	B

Empty

Ques:- Explain Min & Max theorem. [Game Playing]

It is a specialized Search Algo that returns optimal sequence of moves for a player in Zero sum Game.

↳ Recursive/Backtracking algo which is used in decision making and Game theory. → Two Player.

↳ uses recursion to search through Game Tree.

↳ Algo Computes minimax decision for current state.

↳ Two Players → MAX [Selects maximum Value]
→ MIN [Selects minimum Value]

↳ Depth-first Search Algo is used for exploration of complete Game Tree.

↳ CHESS, checkers, Tic-tac-toe ...

[max = $-\infty$] Worst Values.
[min = ∞] [Initial].

Properties:- → COMPLETE

① Definitely found soln (if exists)

② optimal

③ Time Complexity = $O(b^m)$ (m = maximum depth.)

④ Space Complexity = $O(b^m)$

↳ b = branching factor of game tree

Limitations:-

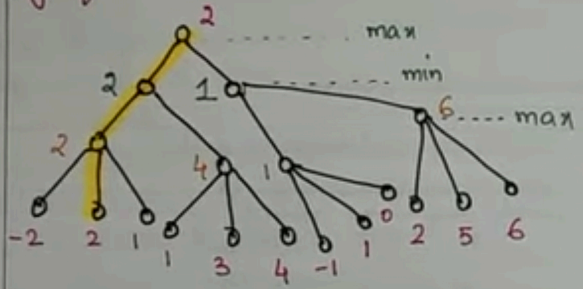
↳ Slow for complex Games such as chess.

↳ 35 choices/moves.

$(35)^{100}$ d = 100 (for both players)

↳ BIG.

Eg:- of MiniMax theorem:-



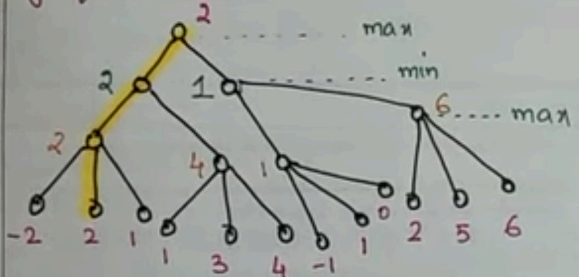
Initial Values $\begin{bmatrix} max = -\infty \\ min = \infty \end{bmatrix}$

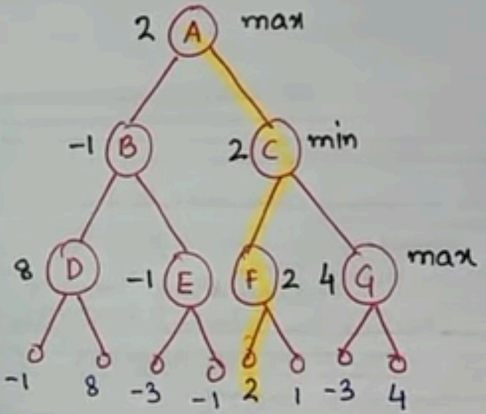
$(-2, -\infty)$

$\hookrightarrow (-2, 2)$

$\hookrightarrow (2, 1) \hookrightarrow (2)$

Eg:- of MiniMax theorem:-





Initial Values [max = $-\infty$, min = ∞]

$(-2, -\infty)$
→ $(-2, 2)$
→ $(2, 1)$ → (2)

Ques:- What is the Significance of ~~Interface~~ Inference Engine in Expert Systems? Discuss forward chaining and backward chaining Strategies.

Inference Engine acquires and manipulates Knowledge from the Knowledge Base to arrive at particular soln.

Rule Based Expert System:

↳ i) Applies Rules repeatedly to facts (→ rules generated from earlier appln.)
ii) Add new Knowledge to Knowledge Base.
iii) Resolve Rule Conflicts.

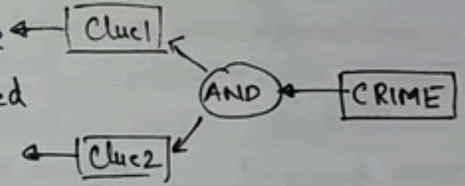
Forward chaining:

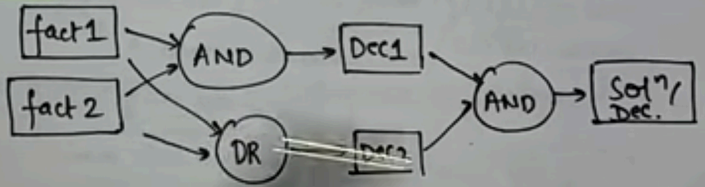
↳ "what can happen next?"

Backward chaining

"Why this happened?"

↳ Finding out Cause/reason.





Ques:- Differentiate between Forward and Backward ~~chaining~~ Reasoning.

Forward Reasoning:- It moves forward from start to goal state. Also called as Data Driven Reasoning.

↳ Search Tree has Initial Configuration(s) at root of tree.

↳ Next Level of Tree is generated by finding all the rules whose left side matches the root node. [IF-THEN RULES]

[Medical Diagonsis]

if S1 → C1

↳ If C1 → C2

↳ If C2 → D1

① Initially System is provided with one or more constraint.

② Rules are Searches for each constraint.

③ Satisfying rules (R.H.S) becomes L.H.S

Backward Reasoning:- It moves backward from goal to initial state. Also called as Goal Driven Reasoning.

↳ Search Tree has Goal Configuration(s) at the root of the tree.

↳ Next Level of Tree is generated by finding all the rules whose right side matches root nodes.

D1 → S1

① Goal State and rules are selected where Goal State resides in THEN part.

② from If part of Selected rules subgoals are made

yptArithmetic Problem:-

Type of Constraint Satisfaction Problem (CSP)

→ Constraints →
- No two letters have Same Value. =
- Sum of digits must be as shown in problem.
- There should be only one Carry forward.

Digits that can be assigned to a word/alphabet (0–9) Range.

0 1 2 X

```
  T O
+ G O
-------
O U T
```

Letter	Digit
T	2
O	1
G	8
U	0

Left most digit = 1

0–9 ≠0

9+8

17

Carry.

2+G=U+10

2+9=11 X

2+8=10

```
  2   1
+ 8   1
-------
  1   0   2
```

```
  SEND
+ MORE
------
 MONEY
```

S+M, if m=1, S+m ≥ 10., S=9.

E+0=N → zero. } if $C_2 = 0$ ✗ → $C_2 = 1$.

Let E=5

E+0+C_2=N

N=6.

N+R=E

6+R=5 → C_1=1

Let R=9 } S=9. ⊗

6+9=15 → E

D+E=Y.

D+5=Y, 7+5=Y

Y=12 ≠ Carry

D≠5

D ≠ 5 − E

D ≠ 6 − N

7 ≠

8 − R

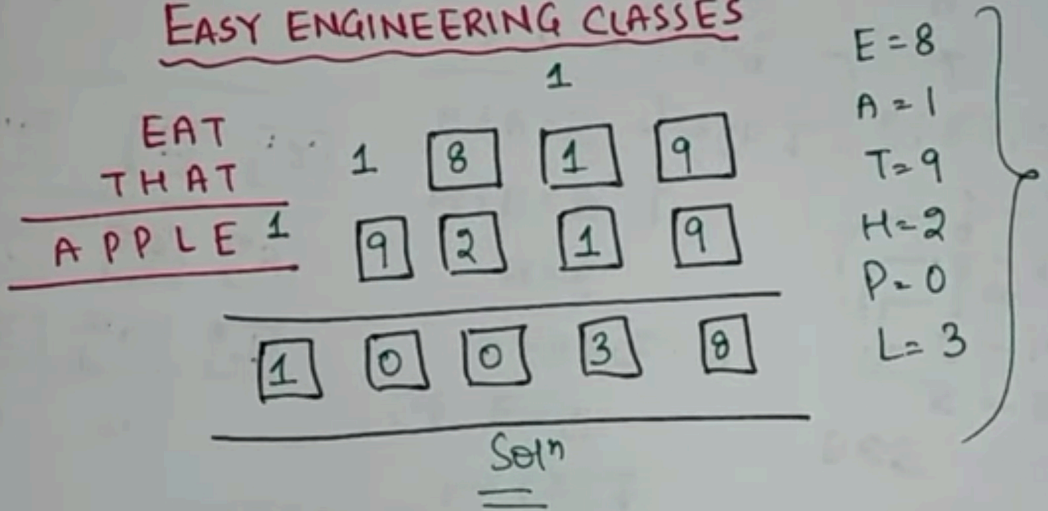
9 − S

6+8+1=15 → R.

C_4	C_3	C_2	C_1
S 9	E 5	N 6	D 7
+ M 1	O 0	R 8	E 5
M 1	O 0	N 6	E 5	Y 2

So

EASY ENGINEERING CLASSES



Cryptarithmetic Problem:

```
  S O M E
  T I M E
---------
S P E N T
---------
    1
```

C

	S 1	O 9	M 3	E 4
+	T 8	I 5	M 3	E 4
S 1	P 0	E 4	N 6	T 8

(T, N) Even = 0, 2, 4, 6, 8 → [T=8, E=4; N=6, M=3]

0+1 = 14 → E

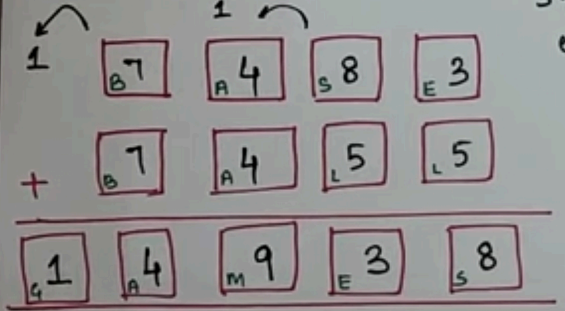
Carry.

14 → (7,7) x
14 → (8,6) x
14 → (9,5)

Solution.

Cryptarithmetic Problem:-

```
  B A S E
+ B A L L
---------
G A M E S
```



$E+L=S-(a)$ [no Carry] $\quad E+L=S$

$E+L=S+10-(b)$ [Carry] $\quad$ or $S-E=L$

$\hookrightarrow \boxed{E=S-L+10}$ $\quad$ or $S-E=5$

$S+L=E$

$S+L=S-L+10$

or $2L=10$

$L=5$

(S,E) — $(5,0)$ X, $(6,1)$ X, $(7,2)$, $(8,3)$, $(9,4)$ — X

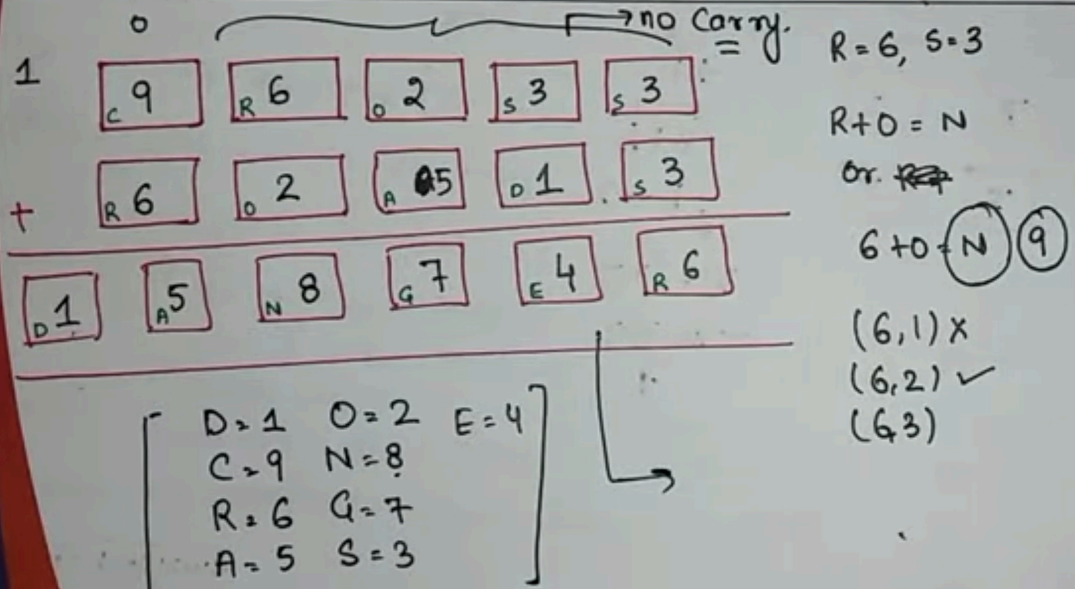
$B+B=A+\text{Carry}$.

B — 5 X, 6, 7, 8, 9

Let $(S,E)=(7,2)$ and $B=6$.

$(S,E)=(8,3)$ $\quad 6+6=12$ (A, C)

$B=7$



Constraint propagation

Last Updated: 2021-11-25

Constraint propagation is the process of communicating the domain reduction of a decision variable to all of the constraints that are stated over this variable.

Constraint propagation is the process of communicating the domain reduction of a decision variable to all of the constraints that are stated over this variable. This process can result in more domain reductions. These domain reductions, in turn, are communicated to the appropriate constraints. This process continues until no more variable domains can be reduced or when a domain becomes empty and a failure occurs. An empty domain during the initial constraint propagation means that the model has no solution.

For example, consider the decision variables *y* with an initial domain [0..10], *z*

For example, consider the decision variables *y* with an initial domain [0..10], *z* with an initial domain [0..10] and *t* with an initial domain [0..1], and the constraints

```
y + 5*z <= 4
t != z
t != y
```

over these three variables.

The domain reduction of the constraint *y + 5*z <= 4* reduces the domain of *y* to [0..4] and *z* to [0]. The variable z is thus fixed to a single value. Constraint propagation invokes domain reduction of every constraint involving *z*. Domain reduction is invoked again for the constraint *y + 5*z <= 4,* but the variable domains cannot be reduced further. Domain reduction of the constraint *t != z* is invoked again, and because *z* is fixed to 0, the constraint removes the value 0 from the domain of *t*. The variable *t* is now fixed to the value 1, and constraint propagation invokes domain reduction of every constraint involving *t*.

the domains are larger and the constraints are more complex. Filtering on the global constraint level can also be more effective than unit clause propagation in pruning the search space.

Constraint propagation then means the process of applying filtering to the constraints in the CSP instance at hand. As filtering one constraint can reduce the domains of its variables, it can trigger further reduction when filtering another constraint that also involves the reduced-domain variables. Thus a constraint can be filtered multiple times during the constraint propagation process. Usually the process is run to the end, meaning until no more reduction happens in filtering any of the constraints. Or if this takes too long, then until some resource usage limit is exceeded.

Let's define **constraint filtering** more mathematically. As input, it takes

- a constraint $C \subseteq D(y_1) \times \ldots \times D(y_k)$, and
- the current domains $D_{\text{before}}(y_i) \subseteq D(y_i)$ for $i \in [1..k]$.

It then produces new domains $D_{\text{after}}(y_i) \subseteq D_{\text{before}}(y_i)$ for all $i \in [1..k]$ such that

$$C \cap (D_{\text{before}}(y_1) \times \ldots \times D_{\text{before}}(y_k)) = C \cap (D_{\text{after}}(y_1$$

Example

Propositional Calculus:-

↳ It is a system that deals with the method used for manipulation of the symbols according to some rules.

ALPHABET SET:

↳ i) Set of Variables or Propositional Symbols P, Q, R
ii) Logical Constants → True (T), false (F)
iii) Two Parentheses "(" and ")" (X, Y)
iv) Set of logical operators

IMP:- ↳ i)

	word	symbol.	example
i)	not	¬	¬X
ii)	and	∧	X ∧ Y
iii)	or	∨	X ∨ Y
iv)	implies	→	(X → Y) if X then Y
v)	if and only if	↔	(X ↔ Y)

Propositional Calculus:-

↳ It is a system that deals with the method used for manipulation of the symbols according to some rules.

X: It is Hot
Y: It is Humid
Z: It is raining

① if it is humid ('Y') then it is hot. (X)
↳ (Y → X)

② if it is hot (X) and humid (Y) then it is not raining (Z).
(X ∧ Y) → ¬Z

ALPHABET SET:

↳ i) Set of Variables or Propositional Symbols P, Q, R
ii) Logical Constants → True (T), false (F)
iii) Two Parentheses "(" and ")" (X, Y)
iv) Set of logical operators

IMP:- ↳

	word	symbol.	example
i)	not	¬	¬X
ii)	and	∧	X ∧ Y
iii)	or	∨	X ∨ Y
iv)	implies	→	(X → Y) if X then Y
v)	if and only if	↔	(X ↔ Y)

Propositional Calculus:-

↳ It is a system that deals with the method used for manipulation of the symbols according to some rules.

X: It is Hot
Y: It is Humid
Z: It is raining

{All, Some → (PL) x}

(1) if it is humid ('Y') then it is hot. (X)

↳ (Y → X)

ALPHABET SET:

↳ i) Set of Variables or Propositional Symbols P, Q, R
ii) Logical Constants → True (T), false (F)
iii) Two Parentheses "(" and ")" (X, Y)
iv) Set of logical operators

(2) if it is hot (X) and humid (Y) then it is not raining (Z).

(X ∧ Y) → ¬Z

IMP:-

	↳ i) C word	symbol.	example
	not	¬	¬X
ii)	and	∧	X ∧ Y
iii)	or	∨	X ∨ Y
iv)	implies	→	(X → Y) if X then Y
v)	if and only if	↔	(X ↔ Y)

vi) Set of equivalence relations or laws: (P, Q, R) are Variables.

- Commutative Laws: $P \wedge Q \cong Q \wedge P$, $P \vee Q \cong Q \vee P$
- Associative Laws: $(P \wedge Q) \wedge R \cong P \wedge (Q \wedge R)$, $(P \vee Q) \vee R \cong P \vee (Q \vee R)$
- Double Negation: $\sim(\sim P) \cong P$
- imp De-Morgan's Law: $\neg(P \vee Q) = \neg P \wedge \neg Q$, $\neg(P \wedge Q) \cong \neg P \vee \neg Q$
- Absorption Law: $P \wedge (P \vee Q) \cong P$, $P \vee (P \wedge Q) \cong P$
- Law of Contradiction: $P \wedge \neg P \equiv$ False $\begin{matrix} P=1 \\ \neg P=0 \end{matrix}$ ⓪ $\begin{matrix} P=0 \\ \neg P=1 \end{matrix}$ ⓪
- Law of excluded middle: $P \vee \neg P =$ True $\begin{matrix} P=1 \\ \neg P= \end{matrix}$ $\begin{matrix} P=0 \\ 1 \end{matrix}$ ①
- Law of impotency: $P \wedge P \cong P$ $\begin{bmatrix} P=1 \rightarrow 1 \\ P=0 \rightarrow 0 \end{bmatrix}$

RULES OF INFERENCE:

→ ① MODUS PONENS: If 'P' and 'P → Q' is given to be true, then we can infer that 'Q' is true.

P: It is a holiday ✓ (T)
Q: The school is closed] → we can infer that it is true.
P → Q: If it is a holiday, then school is closed ✓ (T)

② MODUS TOLLENS: If '~Q' and '~P → ~Q' are given to be true, then we can infer that ~P is true.

~Q = School is not closed.
if it is not a holiday, then school is not closed.
(~P → ~Q)
→ ~P] it is not a holiday (True).

Ques:- Define Tautology and truth table.

Truth Table shows how the truth or falsity of a compound statement depends on the truth or falsity of simple statements.

Some of Truth Table Eg:-

① Negation:-

P	~P
T	F
F	T

Simple Statements.

② AND: $(P \wedge Q) \rightarrow$ True when P and Q both are true.

P	Q	$P \wedge Q$
T	T	T
T	F	F
F	T	F
F	F	F

→ Compound Statement

A **Tautology** is a formula which is always true. ↳ opposite is Contradiction (always false)

Eg:- Show that $(P \rightarrow Q) \vee (Q \rightarrow P)$ is tautology.

P	Q	$P \rightarrow Q$	$Q \rightarrow P$	$(P \rightarrow Q) \vee (Q \rightarrow P)$
T	T	T	T	T
T	F	F	T	T
F	T	T	F	T
F	F	T	T	T

all are true