U-2 (1)

Problem Solving [Problem, Problem Spaces & search]

The most important application of AI. is to develop intelligent system to solve real world problems.

Problem solving - means the agent is in some situation and wants to be in some desired situation

Given state ⟶ Desired state

The task of the agent is to make a series of decision or a series of moves which will transformed it to given situation to desired situation.

To solve these problems efficiently AI uses following processes —

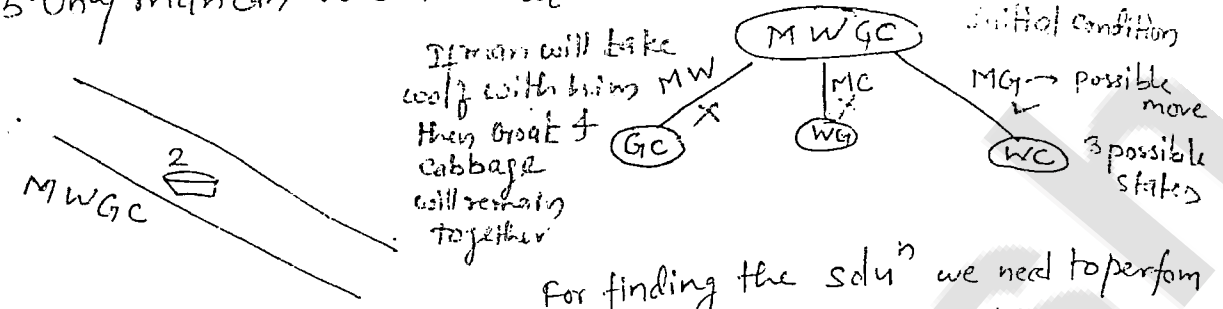
1. Defining a search space -
2. Deciding → start state
   → Goal state
3. Finding the path from start state → Goal state through search space.
4. Production Rules -
   To decide the movements from start state to goal state some set of rules should be designed which is called as production rules.
   So, production rules are nothing but valid moves described by the problems.

For example :- [River Crossing Puzzle]

1. Man, Wolf, Goat and cabbage are on one side of the river
2. There is a boat and only two things can go at one time.
3. The whole set of entities are on one side of the river and they have to be transported to the other side of the river
4. There are som constraints about what can go together and

and what can not go together
If man keeps goat with cabbage then goat will eat cabbage so they could not be kept together. like wise if man keeps wolf & goat together then wolf will eat goat.

5. Only man can row the boat.



For finding the soln we need to perform search if the steps are not known before hand

→ For searching process 3 things are required.

1. Initial state description of the problem ---
Here 4 M, W, G and C at the bank
2. Boat with capacity of 2

2. Set of legal operators (actions) →
We can not move M, W, G & C all together so can not be a legal move

3. final / goat state →
Here, final state is M, W, G & C are at another bank of river.

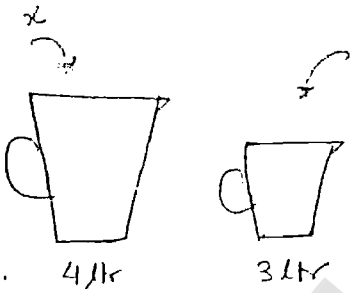
Example-2

Water-jug

Problem is defined as:
① We are given 2 water jugs having no measuring ② marks on these. The capacities of jugs are 3 liter and 4 liter. It is required to fill the bigger jug with exactly 2 liter of water. The water ③ can be filled in a jug from a tap.

④ We are assuming that we can pour water from any of the jug on to ground.

Steps to solve AI problems



Step 1 → State Description
let 2 integer (x, y) denote 2 jugs.
x jug can have 0, 1, 2, 3, 4 ltrs.
y jug can have 0, 1, 2, 3 ltrs.

Step 2 → Describe the Initial state & Goal state

Initial state would be (0, 0) &
Goal state would be (2, n) → 2 ltr could have any [illegible]. it is [illegible]
according to problem definition.
4 ltr jug should have 2 ltr.

Step 3 → List all the actions in production [illegible]

Solution

	water in 4 ltr jug	water in 3 ltr jug	
Initial state →	0	0	Initially both are empty
	0	3	Fill 3 ltr jug.
	3	0	Pour this 3 ltr in 4 ltr jug
	3	3	Again fill 3 ltr jug
	4	2	fill 4 ltr jug full so 2 ltr will remain in 3 ltr jug
	0	2	Empty 4 ltr jug.
Goal state →	2	0	pour 2 ltr from 3 ltr jug to 4 ltr jug.

Solution

	water in 4 ltr	water in 3 ltr.	
Initial state →	0	0	Initially both are empty.
	4	0	Fill 4 ltr jug
	1	3	Pour 3 ltr from 4 ltr to 3 ltr jug.
	1	0	Empty 3 ltr jug
	0	1	Pour 1 ltr from 4 ltr jug → 3 ltr jug
	4	1	fill 4 ltr jug.
Goal state →	2	3	Pour in 3 ltr jug so 2 ltr will remain in 4 ltr jug.

Assignment: The capacity of jugs are 5ltrs and 3 Ltrs. It required to fill the bigger jug with exactly 4 ltrs of water.

Soln →

1. State space → It is the representation of all state i.e. it contains all the necessary info about states.

So, all possible states can be-

1.	(0,0)
2.	(4,0)
3.	(0,3)
→ 3a	(3,0)
4.	(4,1)
5.	(1,3)
6.	(4,2)
7.	(2,3)
8.	(2,0)
9.	(0,2)
10.	(1,0)
11	(0,1)
12	(3,3)

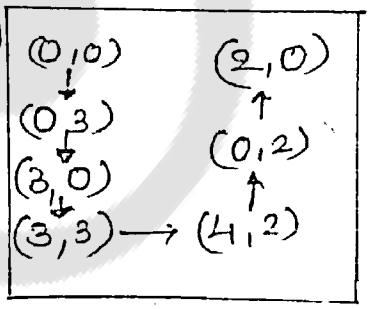
Rule applied	water in 4 ltr jug	water in 3 ltr jug
starting	0	0
3	0	3
3a	3	0
12	3	3
6	4	2
9	0	2
8	2	0

2. Initial State →

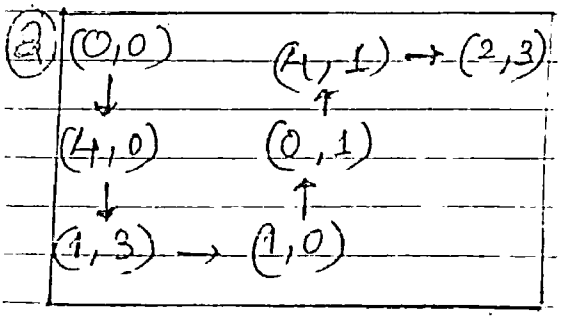
0 ltr in 3 ltr jug &
0 ltr in 4 ltr jug.

3. Possible Soln →

(1)



(2)



4. State Sequence →

Soln-1 (0,0) → (0,3) → (3,0) → (3,3) → (4,2) → (0,2) → (2,0)

Soln-2 (0,0) → (4,0) → (1,3) → (1,0) → (0,1) → (4,1) → (2,3)

3. Missionaries and Cannibals Problem →

"Three missionaries and three cannibals are present at one side of a river and need to cross the river. There is only one boat available. At any point of time, the number of cannibals should not outnumber the number of missionaries at that bank. It is also known that only two persons can occupy the boat available at a time."

Soln →

1. Rules / Possible states —

M denotes Missionaries
C denotes Cannibals

Rule 1:	(0,M)	One missionary	sailing boat from Bank-1 to Bank 2
2:	(M,0)	"	" " 2 to " 1
3:	(M,M)	Two "	" " " Bank-1 to Bank 1
4:	(M,M)	" "	" " " 2 " 1
5:	(M,C)	One missionary & One Cannibal	sailing 1 to " 2
6:	(C,M)	" "	" 2 " 1
7:	(0,C)	One cannibal	" " 1 to 2
8:	(C,0)	"	" 2 to 1
9:	(C,C)	Two cannibals	" " 1 to 2
10:	(C,C)	"	" 2 to 1

2. Initial State —

Bank 1	Bank 2
M1, M2, M3, C1, C2, C3	0

Goal state —

Bank 1	Bank 2
0	M1, M2, M3, C1, C2, C3

3. Possible Solution —

Rules applied and sequence

After application of rule	Persons in Bank 1	Persons in Bank 2	Boat position
start state	M,M,M,C,C,C	0	Bank - 1
5	M,M,C,C	M,C	Bank - 2
2	M,M,C,C,M	C	Bank - 1
9	M,M,M	C,C,C	Bank - 2
8	M,M,M,C	C,C	Bank - 1
3	M,C	C,C,M,M	Bank - 2
6	M,C,C,M	C,M	Bank - 1
3	C,C	C,M,M,M	Bank - 2
8	C,C,C	M,M,M	Bank - 1
9	C	M,M,M,C,C	Bank - 2
8	C C	M,M M C	B ank - 1
9	0	M,M M,C,C,C	Bank - 2

4. Sequence of possible solun —

5 → 2 → 9 → 8 → 3 → 6 → 3 → 8 — 9 — 8 — 9

4. 8- Puzzle Problem -

This problem belongs to the category of "sliding-block-puzzle" type.

Description:

"It has set of a 3x3 board having 9 work spaces out of which, 8 blocks are having tiles bearing number from 1 to 8. One space is left blank. The tile adjacent to blank space can move into it. We have to arrange the tiles in a sequence."

n →

1: ~~Production~~ / Rules →

Actions [operations]

1. up
2. down
3. right
4. left

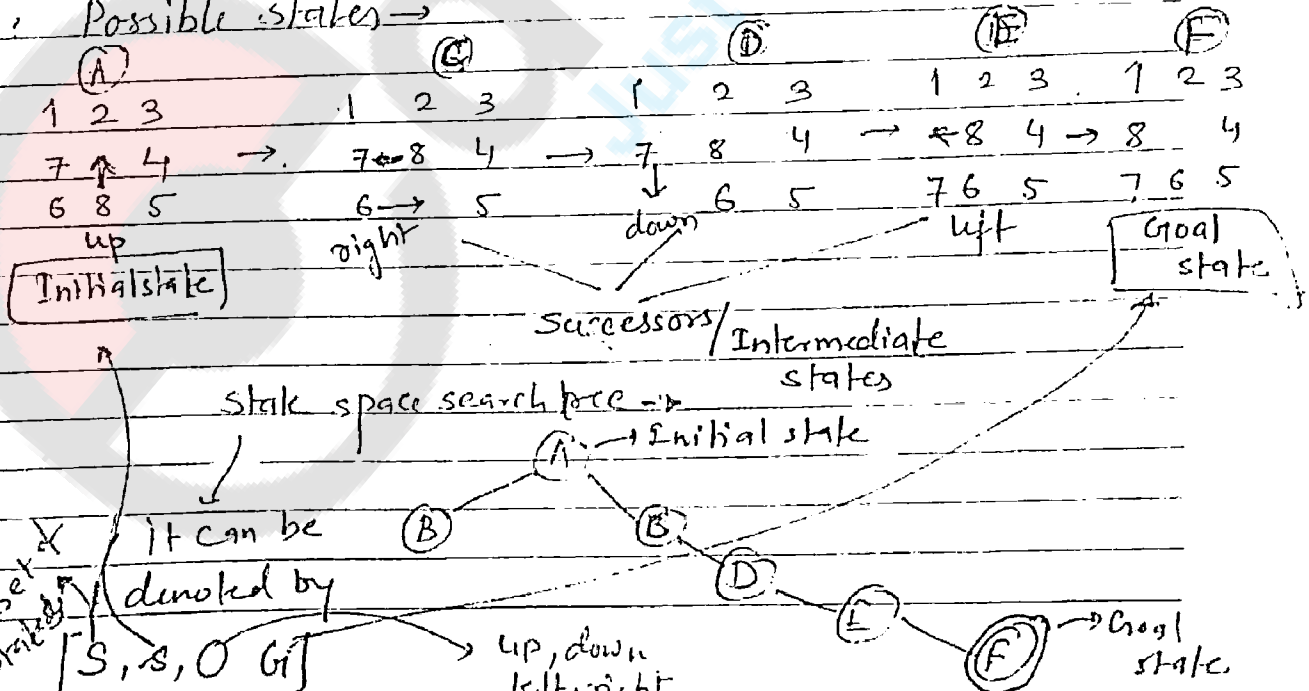
2: Initial state -

1	2	3
7		4
6	8	5

Goal state -

1	2	3
8		4
7	6	5

2: Possible states →



State space search tree →

it can be denoted by

[S, s, O, G]

Scanned by CamScanner

Assignment –

Initial state	Goal state
1 – 4	1 2 3
6 5 8	4 5 6
2 3 7	7 8 –

Find Possible states & note down the sequence of soluⁿ.

5. Tower of Hanoi →

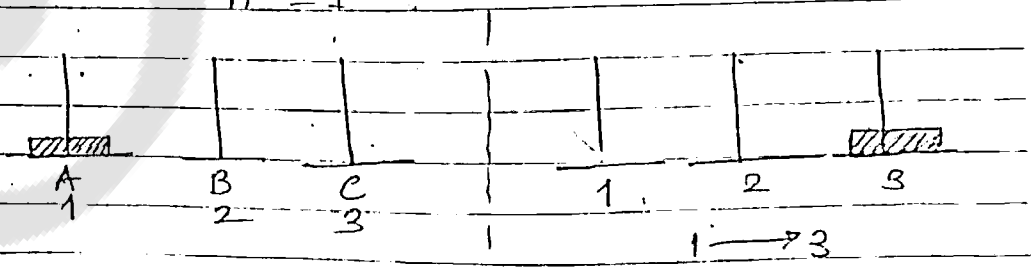
Problem: " There are n rings of tapering size rest on peg A, these rings are to be transfer to peg C in minimum no of moves".

Rules: 1. At a time only one disk can be moved.
2. Large disk must never be put on sma one.
3. Peg B can be used as auxillary (storage temporary) peg.

Aim: Find out minimum no of moves requir

Case 1. no of Disk = 1

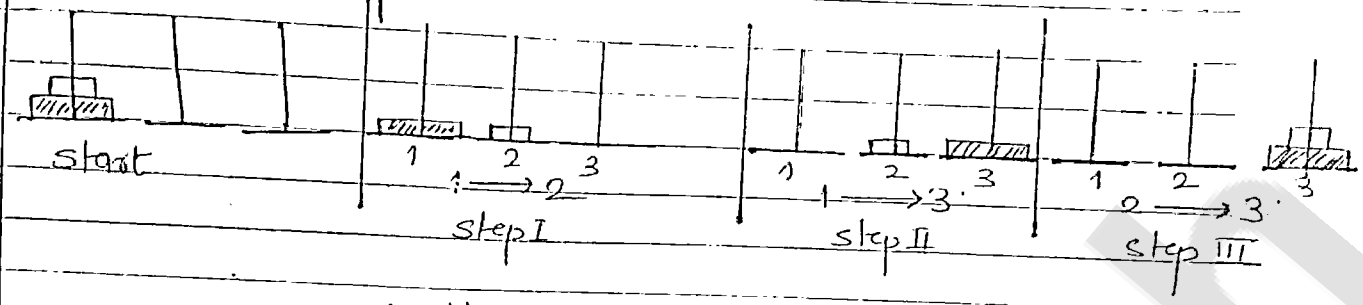
$n = 1$



∴ no of moves $H_n = 1$

i.e. $H_1 = 1$

Case-II no of Disk (n) = 2

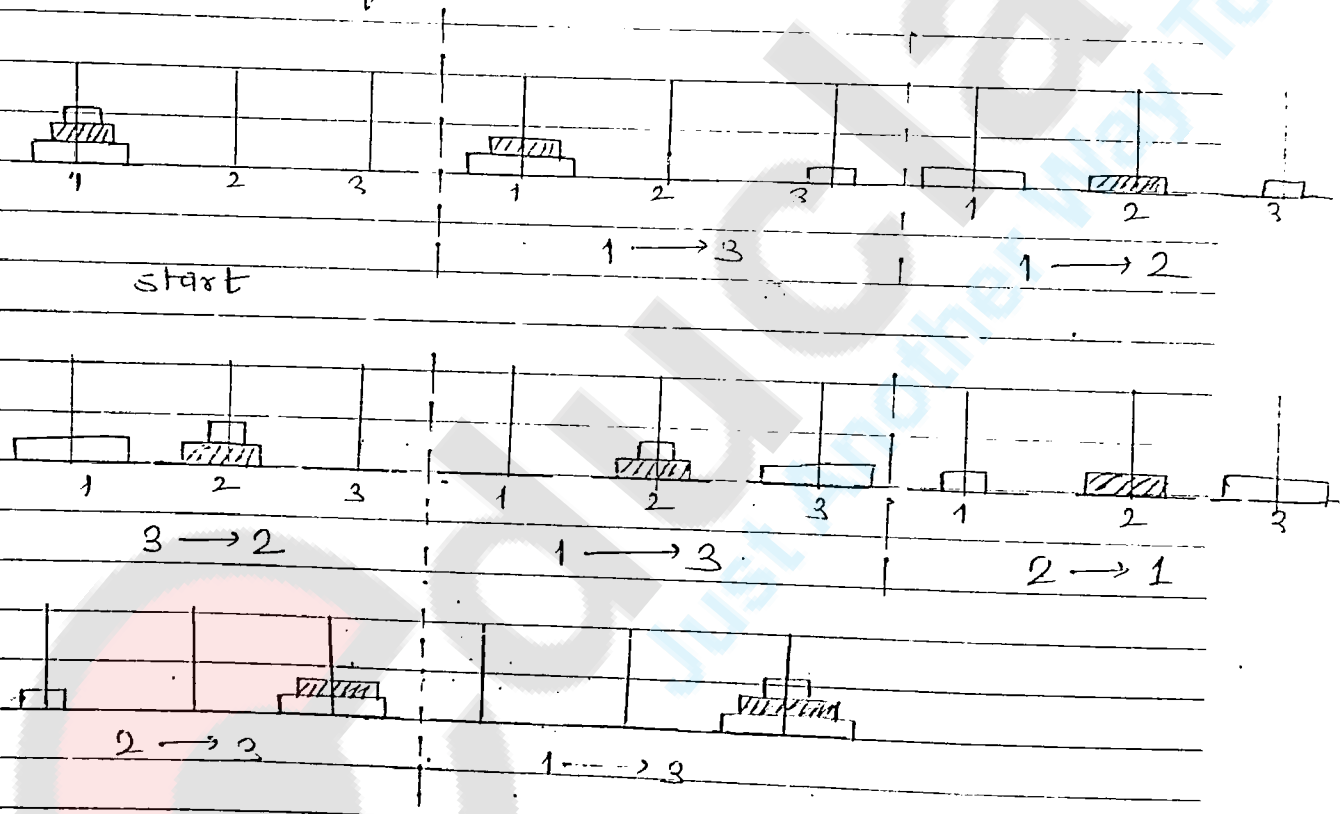


$\therefore H_n = 3$

$\because n = 2$

$\therefore H_2 = 3$

Case III no of disk (r) = 3



$H_n = 7$

$H_3 = 7$

when 1 disk ie $n=1$ $H_n = 1 = 2^1 - 1$

$n=2$ $H_n = 3 = 2^2 - 1$

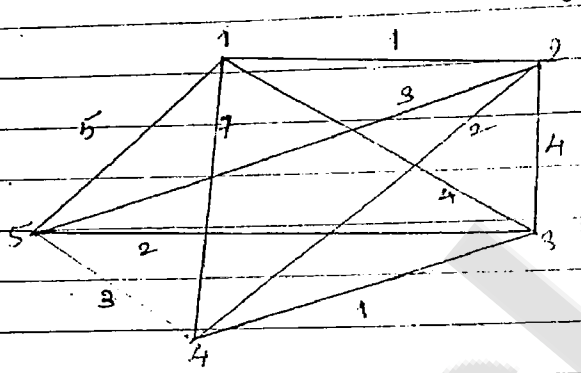
$n=3$ $H_n = 7 = 2^3 - 1$

for n disk the solution will be $\boxed{H_n = 2^n - 1}$

6. Travelling Salesmen Problem:-

Statement:

A salesman has list of cities, each of which he must visit exactly once. (Figure shows) there are direct roads b/w each pair of cities on the list. Find the route the salesman should follow so that he travels the shortest possible on a round trip starting at any one of the cities and then returning there.



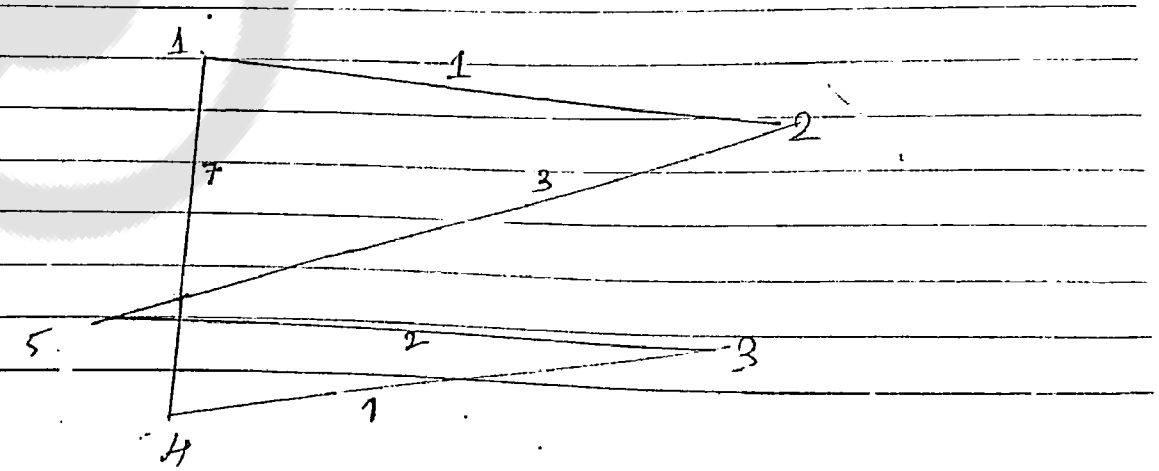
Soln →

Using the heuristic search we may proceed to solve as follows:-

① Arbitrarily select a starting city.

② To select the next city, look at all cities not yet visited ③ Select the one closest to the current city, go to next.

Repeat step 2 untill all cities have been visited.



The soln of this problem is done using Hebbian Network. The NN can solve the 5 city case just as fast as the 30 city case whereas a conventional computer takes much longer time for these.

§ Production system →

b/w AI & E.

Production sys acts as a bridge contains

In Artificial Intelligese production systems

1. Rules in the form of

represents current problem state

Condition ⟶ Action. ← represents generated o/p state

L.H.S. of rules represents Condition &
R.H.S. of rules represents Action

A rule is applicable if its LHS. matches with current problem state

So, in a simple way we can say that Production System contains a number of rules that are defined by LHS. & R.H.S.

Some times it may happen the rule can be of following form

$P_1 \wedge P_2 \wedge \cdots \wedge P_n \rightarrow Q$

In this case all L.H.S terms should be satisfied for applying the rule. After applying the rule, the L.H.S of the rule becomes the current state.

2. One or more knowledge/data bases –

It contains all the appropriate info^n for the particular task. Some part of the db may be permanent while some part may contain the soln of current problem.

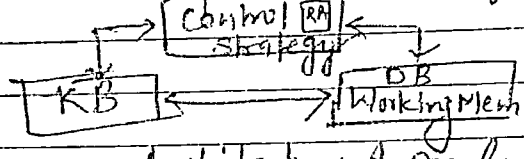
3. A control strategy →

It specifies the order of rules in which rules will be

compared to the DB of rules.
Also ~~[illegible] the~~ specifies a way of resolving the conflicts when several rules matches simultaneously.

4. A Rule Applier → RA

It checks the aplicability of rule by matching the current state with the left hand side of the rule and finds the appropriate rule from the DB of rules.



Architecture of production sys.

Example –

Problem: We have two jugs of capacity 5 ltr and 3 ltr. and a tap with endless supply of water. The objective is to obtain 4 ltr exactly in the 5 ltr jug with the minimum steps possible.

Production system for above problem.

① Fill the 5 ltr jug from tap.
② Empty the 5 ltr jug.
③ Fill the 3 ltr jug from tap
④ Empty " — "
⑤ Empty 3 ltr jug to 5 ltr jug.
⑥ " 5 " 4 3 " "
⑦ Pour water from 3 ltr to 5 ltr jug.
⑧ 4 4 4 5 4 8 4 11

Soln → 1 → 8 → 4 — 6 → 1 → 8

OR

3 → 5 → 3 → 7 → 2 → 5 → 3 → 5

But Seq^{n} 1 will be the optimum since it has min steps

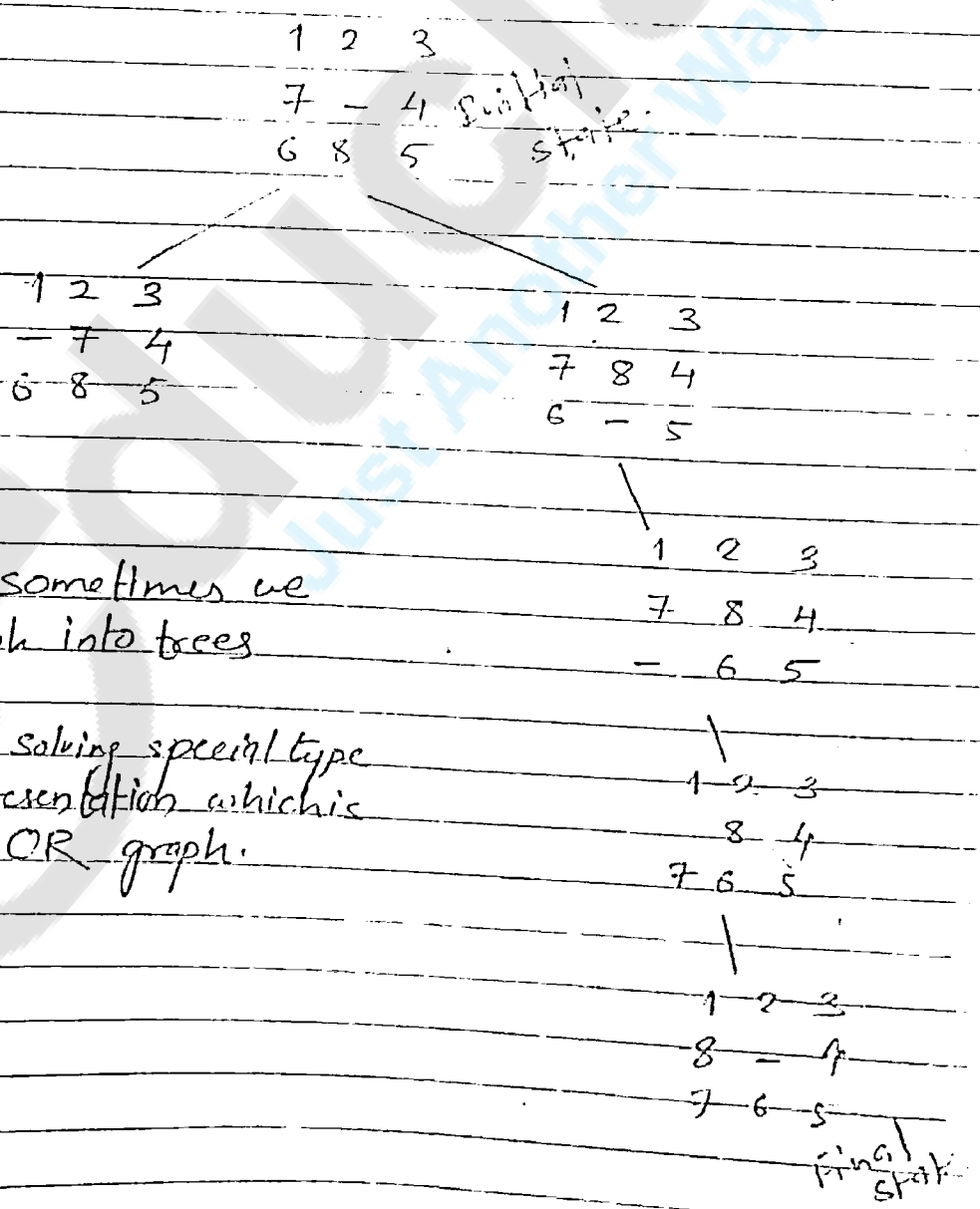
Production Sys Characteristics →

1. Data structure →

Graphs and trees are best suited datastructure for traditional AI problems.

In which, nodes represents the problem state and arcs b/w nodes represents valid transitions.

Ex- In an 8-puzzle problem, various different states derived from a single state will be children of that state.



For searching sometimes we convert graph into trees

In AI problem solving special type of tree representation which is called AND-OR graph.

2. Control strategies →

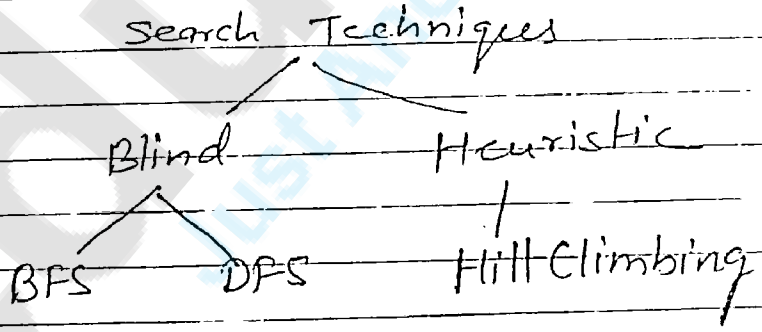
Are used for applying the rules and searching the possible solution in search space. It means, if the wrong control strategy is applied then it may possible that a solution never obtained, even if it exists.

3. Searching ~~Planning~~ Techniques →

As we know AI problems searches the path from start state to goal state. This searching techniques helps to find out viable path to reach the goal state and make the entire process efficient and economical.

Basically there are two types search-

a. Blind / unguided / uninformed search-
b. Heuristic / guided / informed search-



* Blind search-
In such searching methodology the problem definition is given but except that no other addition information is provided.

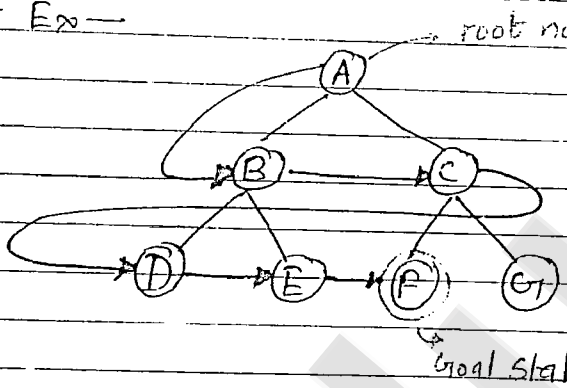
* Heuristic Search-
In such searching techniques additional info^n about the problem is provided along with the problem definition in order to guide the search in a specific direction.

§ 1. Breadth-First Search –

Here the state space is represented in form of a tree.
The soln is obtained by traversing through the tree.
The nodes of tree represent

Start state
Intermediate states
Goal state

While searching for the solution, the root node is expanded first, then all the successors of the root node are expanded and in next step all successors of every node are expanded.
The process continues till goal state is achieved.
The datastructure used is BFS is First-In-First-Out (FIFO)
For Ex –

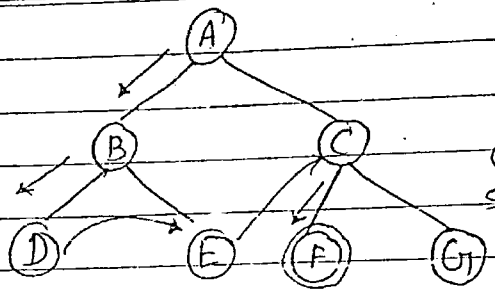


2. Depth-First Search –

In this type of approach, instead of probing the width we can explore one branch of a tree until the soln is found.
The search proceeds immediately to the deepest level of search tree or until the goal is encountered.

DFS has lesser space complexity, because at a time it needs to store only single path from root to leaf node.

The datastructure used for DFS is Last-In-First-Out. (LIFO)



OPEN

If we use Queue as Data structure for variable OPEN → BFS

Stack Data: → DFS

Outline of the Algorithm

start state

1: Initialize: set OPEN = {S}
Datastructure

2: Fail: if OPEN = { }, Terminate with failure

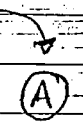
3: Select: Select a state n from OPEN.

4: Terminate: If n ∈ G, terminate with success.
Goal

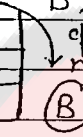
5: Expand: Generate the successor of n and insert in open

6: Loop: Goto step 2

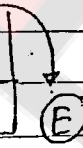
OPEN as Stack (LIFO)

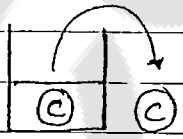
1. A


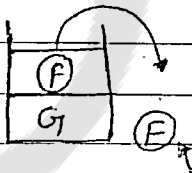
Successor of A are B & C. the order of putting does not matter.

2. B C


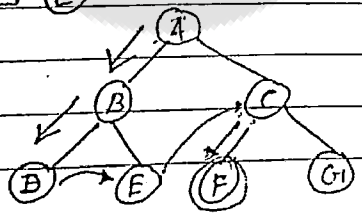
3. D E C


4. E C


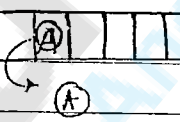
5. C


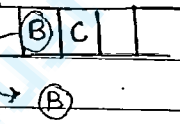
6. F G


Goal state

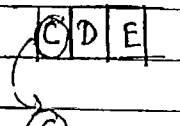


OPEN as Queue (FIFO)

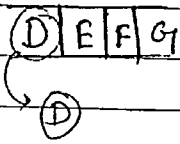
1. A


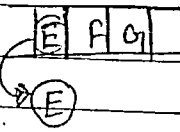
2. B C


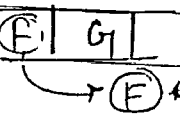
put the successors of C i.e. D & E

3. C D E


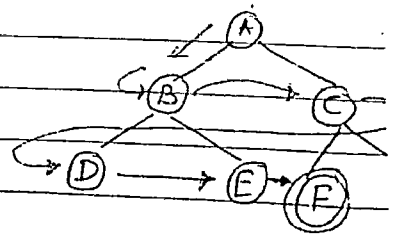
put the successors of C i.e. F & G

4. D E F G


5. E F G


6. F G


F ← Goal state.



Hill Climbing →

Q. What is heuristic func? Explain simple hill climbing algorithm along with a state space diagram of hill climbing.

Heuristic search → (meaning: Enabling a person to discover something from themselves)

If the problems are too difficult to be solve by direct techniques then they have to be solved only by suitable heuristic search techniques.

Though the heuristic techniques can be described independently, they are domain specific. They are called "Weak Methods", since they are vulnerable to (attacked by) combinatorial explosion.

(rapid growth of complexity of a problem due to how the combinatorics of the problem is affected by the input)

Heuristic function →

Heuristic functions usually finds good but not essentially optimum solution.

Heuristic func is a func that will rank all the possible alternative at any branching step in search algorithm based on the available information. It helps the algorithm to select the best route out of possible routes.

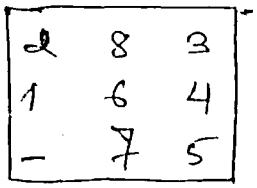
A heuristic function at a node n is an estimate of the optimum cost from the current node to a goal. It is denoted by h(n).

h(n) = estimated cost of the cheapest path from node n to a goal node

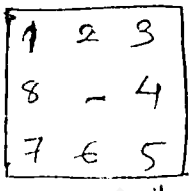
Example 1:
We want a path from Kolkata to Guwahati
Heuristic for Guwahati may be straight-line distance b/w Kolkata and Guwahati

h (Kolkata) = euclideanDistance (Kolkata, Guwahati)

Ex 2 - 8 puzzle: Misplaced Tiles Heuristics is the number of tiles out of place.



Initial state



Goal state

↓ it shows the current state n.

↓ It shows goal state

h(n) = 5
because the tiles 2, 8, 1, 6, 7 are out of place.

Simple Hill Climbing Algo →

This algo starts from single current state and move to neighboring states.

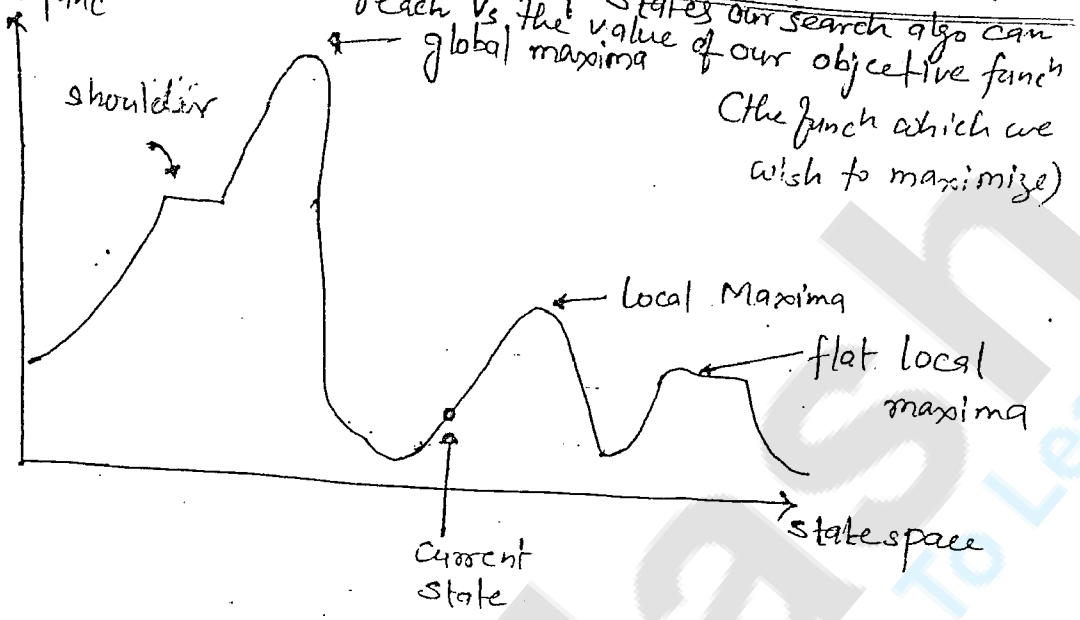
Idea behind this is - Algo starts with an initial guess at a solution and incrementally improve it untill it reaches to goal.

Advatage: - This task can be performed using very little memory.

- we can find reasonable solution in a large state space

02 (19) Statespace diagram is a graphical repres^{n} of the set of states our search algo can reach Vs the value of our objective func^{n} (the func^{n} which we wish to maximize)

statespace Diagram →



Hill climbing does not look ahead of the immediate neighbors of the current state.

1. local maxima → It is a state that is better than all its neighbors but is not better than some other state farther away.
2. Plateau – An area of the state space where the evaluation func^{n} is flat. [flat local maxima]
3. Ridge – It is a region which is higher than the neighbors but itself has a slope. It is a special kind of local maxima.

4. Current state – The region of state space diagram where we are currently present during the search.
5. shoulder – It is a plateau that has an uphill edge
6. Global Maxima – It is the best possible state in the state space diagram. At this state, objective func^{n} has highest value.

Algorithm →

Step 1: Evaluate the initial state. If it is goal state then exit else make the current state as initial state.

Step 2: Repeat these steps until a solution is found or current state does not change.

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

ii. for each operator that applies to the current state.

a. apply the new operator and create a new state.

b. evaluate the new state.

c. if this state is goal state then quit

else

compare with 'target'

d. If this state is better than 'target', then

set this state as 'target'.

e. If target is better than current state then

set current state to 'Target'.

Step3: Exit