Solving Problems by Searching

Problem solving Agents, Example problems, Searching for solutions, Uninformed search Strategies, Informed search Strategies, Heuristic Functions, Beyond Classical Search: Local Search Algorithms and Optimization Problems, Local Search in Continues Spaces, Searching with Nondeterministic Actions, Searching with Partial Observations, Online search Agents and Unknown Environments.

---

1. Problem Solving Agents:-

In AI, search techniques are universal problem solving methods.

Rational Agents or Problem solving agents in AI mostly uses these search strategies or algorithms to solve a specific problem and provide the best result.

Problem solving agents are the goal-based agents and use atomic representation.

Search Algorithm Terminology:-

Search:- It is a step by step procedure to solve a search problem in a given search space.

Search Space:- Search space represents a set of possible solutions, which a system may have.

Initial State or Start State:- It is a state from where agent begins the search.

Goal State:-
It is a function which observe the current state and returns whether the goal state is achieved or not.

Search Tree:- A tree representation of search problem is called search Tree.
The root node which corresponding to the initial state.

Actions:- It gives the description of all the available actions to the agent.

Transition Model or Production Rules:-
A description of what each action do, can be represented as a transition model.

Path Cost:-
It is a function which assigns a numeric cost to each path.

Solution:-
It is an action sequence from the start node to the goal node.

Optimal Solution:-
The solution has the lowest cost among all solutions.

---

2 Properties of Search Algorithms:-
Following are the 4 essential properties of search algorithms to compare the efficiency.

Completeness:- A search algorithm is said to be complete, if it guaranteed to return a solution

Optimality:-
The solution for an algorithm is the best solution (lowest path cost) among all other solutions, then such solution is said to be an optimal solution.

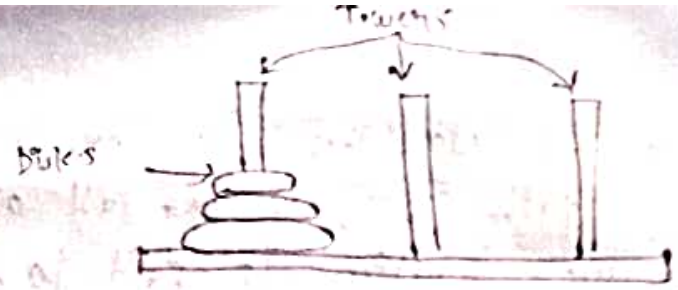
Time Complexity:-
It is a measure of time of an algorithm to complete its task.

Space Complexity:-
It is the maximum storage space required at any point during the search.

Towers of Hanoi:-
It is a mathematical game puzzle, where we have 3 pile (pillars) and 'n' number of disks.

Game Rules:-
→ Only one disk move at a time.
→ The larger disk should be always on the bottom and the smaller disk on the top.
→ Move only the upper most disk.
→ All disks move to destination pile from source pile.

Here we are trying to solve that how many moves are required to solve a problem. It depends on the number of disks.



⇒ 2 disks and 3 pillars.

When we have 2 disks, we require 3 moves to reach the destination pile.

if we have 3 disks we require 4 moves,

if 4 disks we require 8 moves.

Towers of Hanoi problem is an example of recursion and backtracking.

When a function calls itself, its called Recursion. Backtracking is breaking the problem into smaller problems.

There must be a termination condition in the recursion problems.

Source Code:-

```
def TOH (n, source, destination, helper):
    if n==1:
        print ("move disk 1 from peg",
               source, "to peg", destination)
        return
    TOH (n-1, source, helper, destination)
    print ("Move disk", n, "from peg",
           source, "to peg", destination)
    TOH (n-1, helper, destination, source)
n= int(input("Enter the number of disks:"))
TOH (n, 'A', 'B', 'C')
```

Water Jug Problem:-

Statement:- There are 2 jugs with 4L and 3L capacity. You can fill the water in jugs with pump, No marking indicating the levels of jugs. We need to fill the 4L jug with exactly with 2L water capacity.

→ There is a pump, we can fill the jugs with it.

→ You can transfer water from one jug to another and vice-versa.

→ You can pour water from any of the jug on the ground.

→ No marking on jugs indicating the levels.

$x \rightarrow$ 4L jug

$y \rightarrow$ 3L jug

Initial State : (0,0)

Goal State : (2,n)

There are nine production Rules are there,

(i) Fill 3L jug

$(x,y) \rightarrow (x,3)$ if $y \leq 3$

(ii) Fill 4L jug

$(x,y) \rightarrow (4,y)$ if $x \leq 4$

(iii) Empty 3L jug

$(x,y) \rightarrow (x,0)$ if $y \geq 0$

(iv) Empty 4L jug

$(x,y) \rightarrow (0,y)$ if $x \geq 0$

(v) Move all from 3L to 4L

$(x,y) \rightarrow (x+y, 0)$ if $y \geq 0$ & $x+y \leq 4$

(vi) Move all from 4L to 3L

$(x,y) \rightarrow (0, x+y)$ if $x \geq 0$ & $x+y \leq 3$.

(vii) Transfer from 3L to 4L until 4L is full

$(x,y) \rightarrow (4, y-(4-x))$ if $y \geq 0$ & $x+y \leq 4$

(viii) Transfer from 4L to 3L until 3L is full

$(x,y) \rightarrow (x-(3-y), 3)$ if $x \geq 0$ & $x+y \leq 4$

(ix) Transfer 2L water from 3L jug to 4L jug

$(x,y) \rightarrow (0,2) \rightarrow (2,0)$.

4L, 3L

(0,0) → Initial state

(0,3) → rule 2

(3,0) → rule 3

(3,3) → rule 2

(4,2) → rule 7

(0,2) → rule 4

(2,0) → rule 9 → Goal State.

---

3. 8 puzzle Problem:-

The 8 puzzle problem consists of eight numbered, movable tiles in 3x3 frame. One cell of the frame is always empty. Thus making it possible to move an adjacent numbered tile into the empty cell.

The eight puzzle problem also known as N-puzzle problem (or) sliding puzzle problem.

N-puzzle that consists N+1 tiles with an empty tile.

where, N = 8, 15, 24, 35, ...

$\sqrt{N+1}$ rows and $\sqrt{N+1}$ columns.

if N = 15 then $\sqrt{15+1} = \sqrt{16} = 4$ rows X 4 columns
= 4 X 4 frame.

In this we have initial state (or) initial configuration (start state) and the goal state (or) goal configuration.

Time complexity in BFS is $O(b^d)$.

where, b = branch factor
d = depth factor

Eg: $O(b^d) = O(3^{20})$.

To solve the problem with Heuristic search (or) informed search we have to calculate Heuristic values of each node to calculate cost function.

Cost Function $\Rightarrow c(x) = g(x) + f(x)$

where $g(x)$ = cost of reaching from current node to root node.

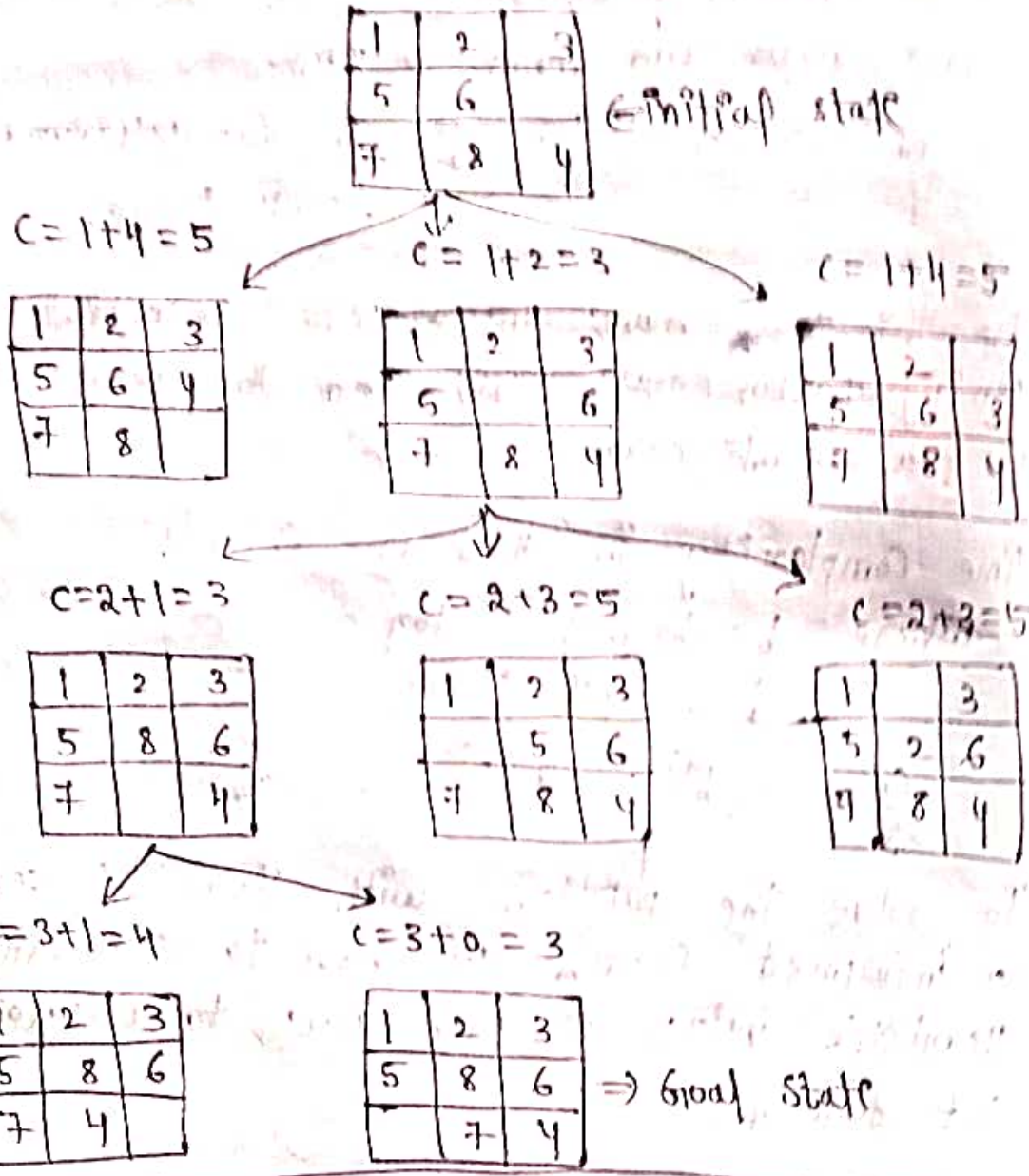
$f(x)$ = approximate cost of reaching answer node from current node.

Initial state

1	2	3
5	6	
7	8	4

Goal state

1	2	3
5	8	6
7	1	4



4. Types of Search Algorithms :-

There are 2 types of search algorithms.

→ Uninformed search (Blind search)

→ Informed search (Heuristic search)

Uninformed (or) Blind search:-

Uninformed search applies a way in which search tree is searched without any information about the search space. It is also called Blind search.

It examines each node of the tree until it achieves the goal node.

- → BFS = Breadth First search
- → Uniform Cost search
- → DFS = Depth First search
- → Depth limited search
- → Bidirectional search.
- → Iterative Deepening search

Informed search (or) Heuristic search:-

In an informed search, problem information is available which can guide the search.

Informed search strategies can find a solution more efficiently than an uninformed search strategy. Informed search is also called as a Heuristic search.

A Heuristic is a way to find a good solution in reasonable time.

Informed search can solve much complex problem which could not be solved in another way.

- → Travelling Salesman Problem
- → Greedy search (Best first search)
- → A* search

BFS :-

BFS - Breadth First Search

It is a recursive algorithm for searching all the vertices of a graph.

Consider two lists visited and not visited.

BFS algorithm is given by

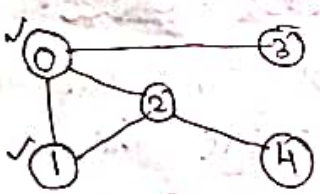
(i) Start by putting any one of the graph vertices at the back of the queue.

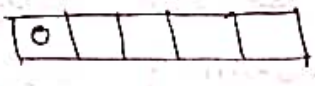
(ii) Take the front item of the queue and add it to the visited list.

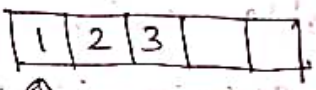
(iii) Create a list of that vertex adjacent nodes. Add nodes ones which aren't in the visited list to the back of the queue.

(iv) Keep repeating steps 2. and 3. until the queue is empty.

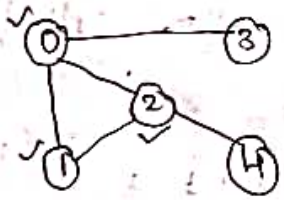
Ex:-

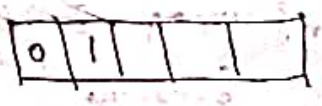


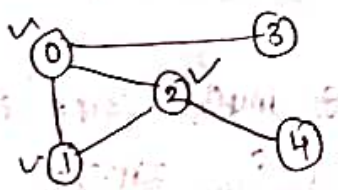


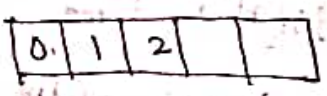


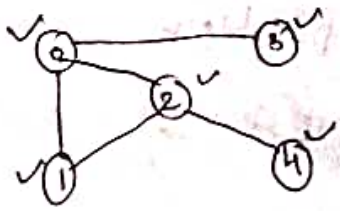
front

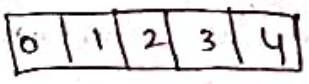












Empty

Time complexity = $O(V+E)$

where, V = vertices (or nodes)

E = Edges

Space Complexity = $O(V)$

Applications of BFS:-

(i) To build index by search index

(ii) For GPS navigation.

(iii) Path finding algorithms

(iv) In Ford-Fulkerson algorithm, to find maximum flow of a network.

(v) Cycle detection in a undirected graph.

(vi) In minimum spanning tree.

DFS:-

DFS - Depth First Search.

It is a recursive algorithm for searching all the vertices of a graph.

Consider two lists visited and not visited.

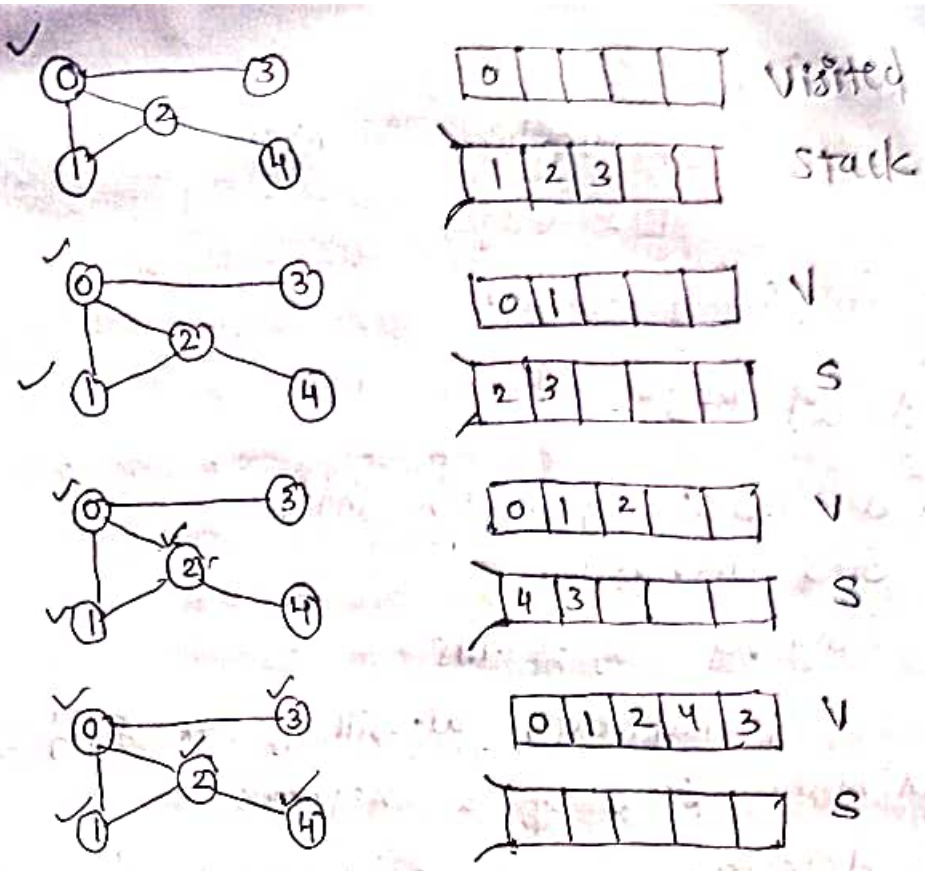
DFS algorithm is given by

(i) Start by putting any one of the graph vertices on top of stack.

(ii) Take the top item of the stack and add it to the visited list.

(iii) Create a list of that vertex adjacent nodes. Add nodes, ones which aren't in the visited list to the top of the stack.

(iv) Keep repeating steps 2 and 3 until the stack is empty.



Time complexity is $O(V+E)$

Space complexity is $O(V)$

where, V = Vertices (or nodes

E = Edges.

DFS applications :-

(i) For finding the path.

(ii) To test the graph is bipartite or not.

(iii) For finding the strongly connected components of a graph.

(iv) For detecting cycles in a graph.

The biggest disadvantage of BFS is that it requires a lot of memory space, it is a memory bounded strategy.

DFS may get trapped in an infinite loop.

DFS uses the concept of backtracking to explore each node in a search tree.

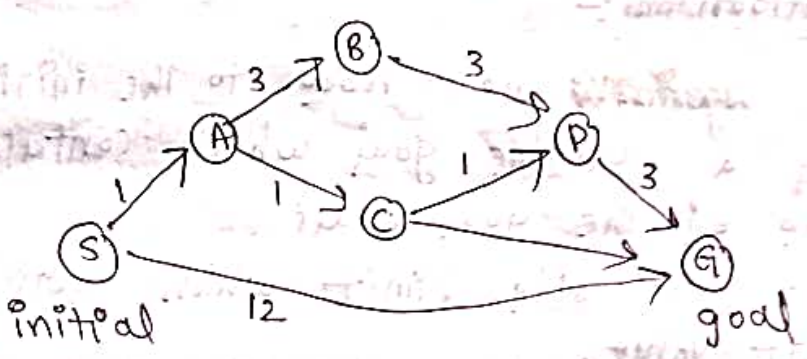
Uniform Cost Search:-

This algorithm is mainly used when the step costs are not the same, but we need the optimal solution to the goal state.

In such cases, we use UCS to find the goal and the path, including the cumulative cost to expand each node from the root node to the goal node.

It searches for the next node with the lowest cost, and in case of the same path cost, let's consider lexicographical order in our case.

Ex:-



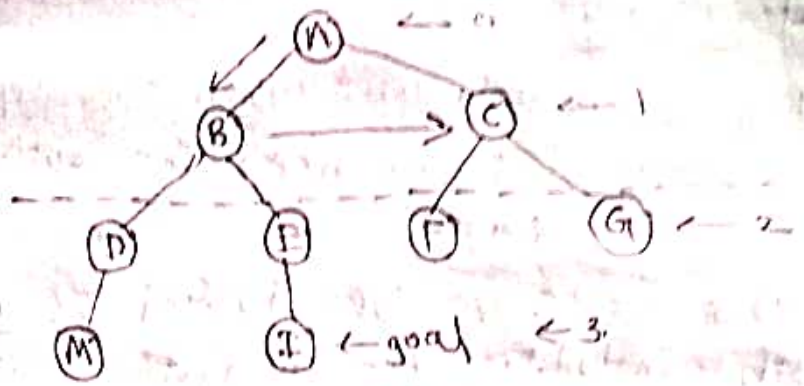
→ UCS is an optimal search method because at every state, the path with the last cost is chosen.

→ This algorithm may be stuck in an infinite loop.

Depth Limited Search (DLS):-

It is similar to DFS, the difference is that in the DLS, we limit the search by imposing a depth limit to the depth of the search tree.

The DFS is a special case of DLS, when the limit - ∞ is infinite.



DLS on a binary tree

In the above figure the depth limit is 1, only level-0 and 1 get expanded in A → B → C DFS sequence, starting from the root node A to B.

DLS Algorithm :-

(i) Set root node to the initial state.

(ii). Set a variable goal which contains the value of the goal state.

(iii) Set a variable limit which carries depth limit value.

(iv) Loop each node by traversing in DFS manner till the depth limit value.

(v) While performing the looping, start removing the elements from the stack in LIFO order.

(vi) If the goal state is found, return goal state Else terminate the search.

Depth limited search does not guarantee to reach the goal node.

It does not give an optimal solution as it expands the nodes till the depth limit.

Time and Space complexities are $O(b^l)$

Iterative Deepening Search:

This search is a combination of BFS and DFS, as BFS guarantees to reach the goal node and DFS occupies less memory space Iterative deepening search combines these two advantages of BFS and DFS to reach the goal node. It gradually increases the depth-limit from 0, 1, 2 and so on and reach the goal node.

Algorithm for IDS:-

(i) Explore the nodes in DFS order.
(ii) Set a variable limit with a limit value.
(iii) Loop each node up to the limit value and further increase the limit value accordingly.
(iv) Terminate the search when the goal state is found.

IDS may (or may not reach the goal state. It does not give an optimal solution always.

Space Complexity is $O(bd)$
Time complexity is $O(d)$

Bidirectional Search:-

The bidirectional search is to run two searches simultaneously. one forward search from the initial state and other from the backside of the goal hoping that both searches will meet in the middle.
The bidirectional search terminates with the goal node. It requires lot of memory space.
Bidirectional search is complete.
It gives an optimal solution. Time & Space : $O(b^{d/2})$

Best First Search (Greedy search):-

A best first search is a general approach of informed search.

Here a node is selected for expansion based on an evaluation function $f(n)$, where $f(n)$ interprets the cost estimate value.

The evaluation function expands the node first, which has the lowest cost.

A component of $f(n)$ is $h(n)$ which carries the additional information required for the search algorithm.

$h(n)$ = estimated cost of the cheapest path from the current node n to the goal node.

Best first search is known as a greedy search because it always tries to explore the node which is nearest to the goal node and selects that path, which gives a quick solution.

Algorithm :-

(i) Set an open list and a close list where the open list contains visited but unexpanded nodes and the close list contains visited as well as expanded nodes.

(ii) Initially traverse the root node and visit its next successor nodes and place them in the open list in ascending order of their heuristic value.

(iii) Select the first successor node from the open list with the lowest heuristic value and expand further.

(iv) Now, rearrange all the remaining unexpanded nodes in the open list and repeat above two steps.

(v) If the goal node is reached, terminate the search, else expand further.

Best first search is incomplete even in finite state space.

It does not provide an optimal solution.

Time and Space Complexity is $O(b^m)$

---

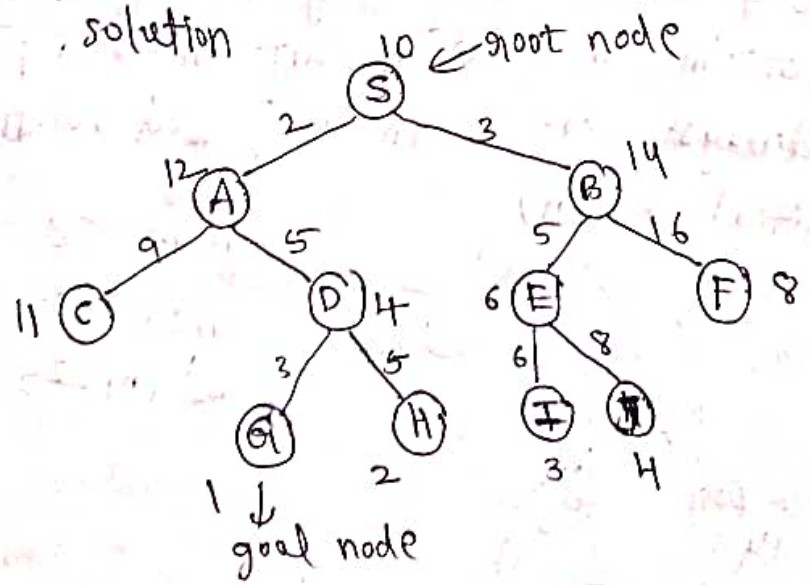
5. A* Search Algorithm :-

A* search is the most widely used informed search algorithm where a node n is evaluated by combining values of the functions g(n) & h(n).

The function g(n) is the path cost from the initial node to a node n and the function h(n) is the estimated cost of the cheapest path from node n to the goal node.

$\therefore \quad f(n) = g(n) + h(n).$

where, f(n) is the estimated cost of the cheapest solution



Calculation of $f(n)$

$f(S)$ = (distance from S to S) + $h(S)$

$f(S) = 0 + 10$

$f(S) = 10.$

$f(A)$ = (distance from S to A) + $h(A)$

$f(A) = 2 + 12 = 14.$

$f(B)$ = (distance from S to B) + $h(B)$

$f(B) = 3 + 14 = 17.$

$S \rightarrow A \rightarrow D \rightarrow G$

A heuristic function can either underestimate (or) overestimate the cost required to reach the goal node.

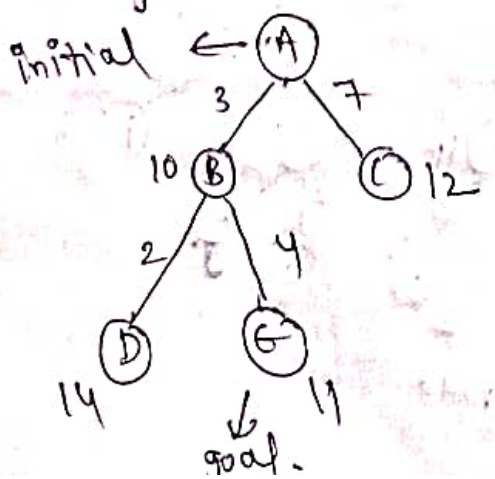
Underestimating the cost value means the cost we assumed in our mind is less than the actual cost.

Overestimating the cost value means the cost we assumed is greater than the actual cost.

$h'(n) <= h(n)$ is underestimation

$h'(n) => h(n)$ is overestimation

An overestimated cost may (or) may not lead to an optimized solution, but an underestimated cost always lead to an optimized solution.



$h(n) = 18$

$h'_1(n) = 12$, underestim

$h'_2(n) = 25$, overestim

The A* search guarantees to reach the goal node.
An understimated cost will always give an optimal solution.

Time and Space complexities are O(bd)

A* mostly runs out of space for a long period

6. Heuristic Functions:-

A heuristic technique helps in solving problems, even though there is no guarantee that it will never lead in the wrong direction.

There are two major ways in which domain-specific, heuristic information can be incorporated into rule-based search procedure.

As a heuristic function that evaluates individual problem states and determines how desired they are.

A heuristic function is a function that maps from problem state description to measures desirability, usually represented as number weights

"The sum of the distances travelled so far", is a simple heuristic function in the TSP.

The purpose of a heuristic function is to guide the search process in the most profitable directions.

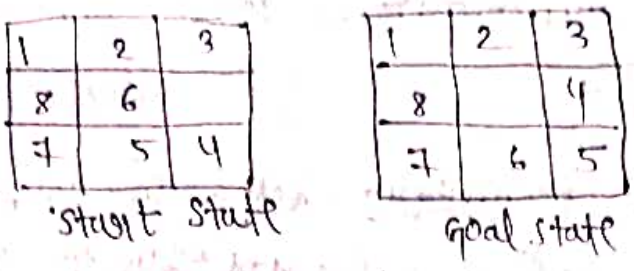
Informed search make use of heuristic functions in order to reach the goal node in a more prominent way.

A good heuristic function is determined by its efficiency.

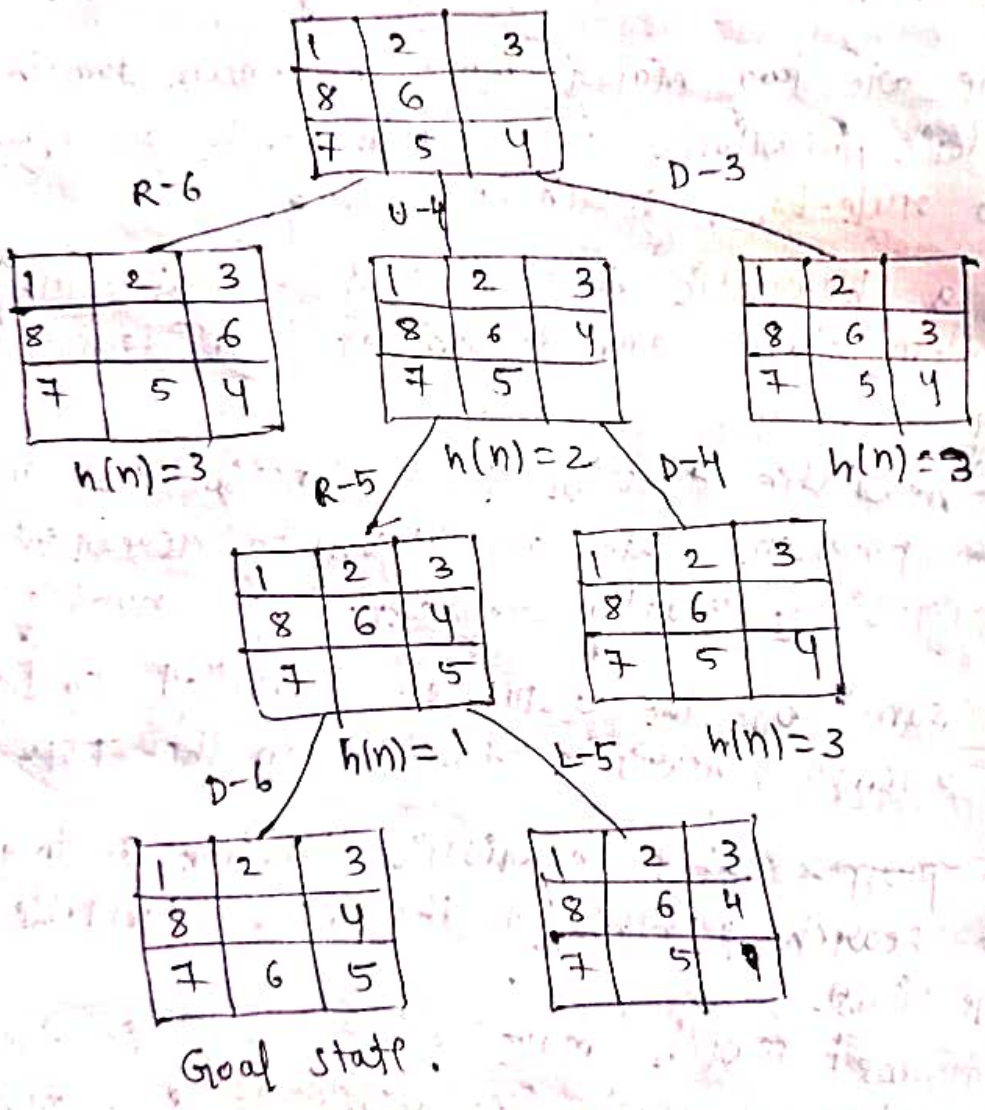
Some problems (such as 8-puzzle, 8 queen, tic-tac-toe etc.) can be solved more efficiently with the help of a heuristic function.

Ex: 8-puzzle problem.

In 8-puzzle, there can be 4 moves; Left, Right, Up, Down.



h(n) = No. of tiles out of position



Goal state.

Properties of Heuristic Functions:-

(i) Admissible Condition: If an algorithm is said to be admissible, it returns an optimal solution.

(ii) Completeness: An algorithm is said to be complete, if it terminates with a solution.

(iii) Dominance Property: If there are 2 admissible heuristic algorithms

(iv) Optimality Property: If an algorithm is complete, admissible, and dominating other algorithms, it will be the best optimal solution.

---

7. Local search Algorithms and Optimization Problems:

A local search algorithm completes its task by traversing on a single current node rather than multiple paths and following the neighbours of that node.

Local search algorithm use a very little (or) constant amount of memory.

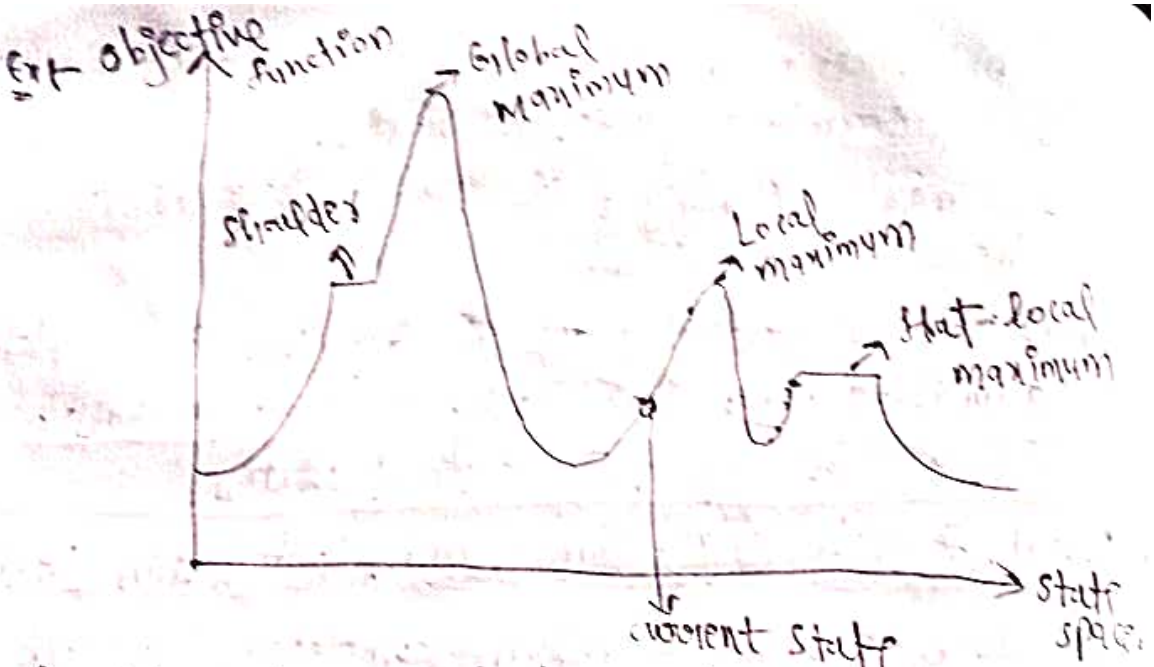
They find a reasonable solution in large (or) infinite state spaces where the classical (or) systematic algorithms do not work.

The local search algorithm work for a pure optimized problem.

The pure optimization problem fails to find high-quality solutions to reach the goal state from the current state.

An objective function whose value is either minimized (or) maximized in different contexts of the optimization problems.

In case of search algorithms, an objective function can be the path cost for reaching the goal node.



A 1-D state space landscape in which elevation corresponds to the objective function.

The different types of local searching algorithms are

→ Hill climbing Algorithm
→ Simulated annealing
→ Local Beam search

Hill climbing algorithm:-

Hill climbing algorithm is a local search algorithm. The purpose of the hill climbing search is to climb a hill and reach the topmost peak/point of that hill.

It is based on the heuristic search technique.

To understand the concept of hill climbing algorithm, consider the above landscape representation

Global Maximum:- It is the highest point on the hill which is the goal state.

Local Maximum:- It is the peak higher than all other peaks but lower than the global maximum

Flat local Maximum:-
It is the flat area over the hill, where it has no uphill (or downhill). It is the saturated point of the hill.
Shoulder:- It is also a flat area where the summit is possible.
Current State:- It is the current position of the pers

Types of Hill Climbing Search Algorithm:-
(i) Simple Hill climbing
(ii) Steepest-ascent hill climbing
(iii) Stochastic hill climbing
(iv) Random-restart hill climbing.

In simple hill climbing algorithm, it finds the next step better than the previous one.

In steepest ascent hill climbing, it considers all the successive nodes, compares them, and choose the node which is closest to the solution. The steepest ascent is similar to the best first search because it focuses on each node instead of one node.

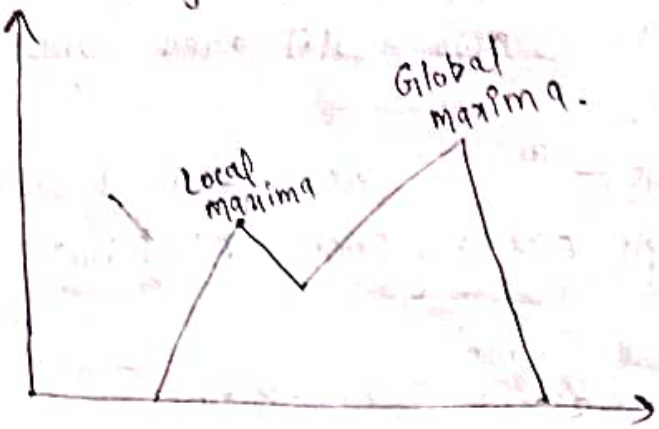
The Stochastic hill climbing does not focus on all the nodes. It selects one node at random and decides whether it should be expanded or search for a better one.

Random-restart hill climbing algorithm is based on try and try strategy. It iteratively searches the node and selects the best one.

Limitations of Hill climbing Algorithm:-
Hill climbing algorithm is fast and furious approach.

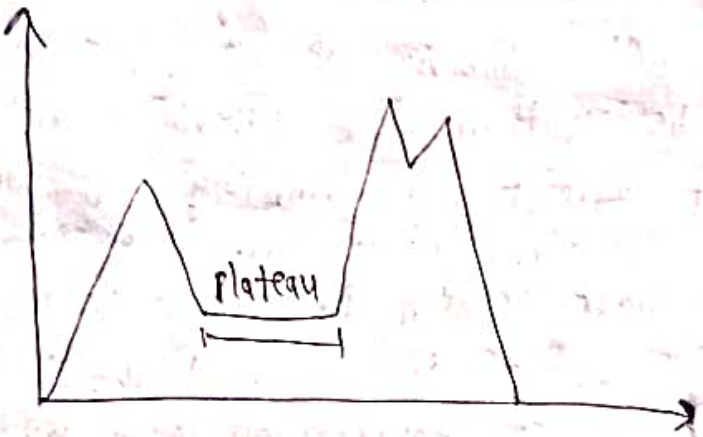
(i) Local Maxima.

It is the peak of the mountain which is higher than all its neighboring nodes but lower than the global maxima.



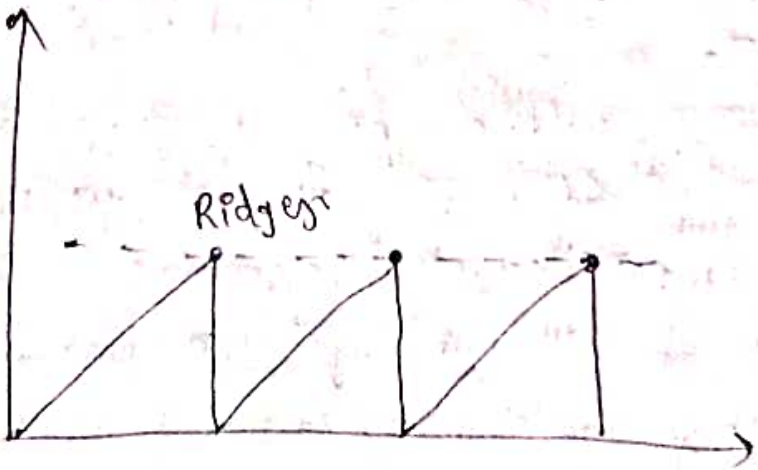
(ii) Plateau.

It is flat surface area where no uphill exi..



(iii) Ridges.

It is difficult to find solution for the problem where two (or) more local maxima of the same height. These are called "ridges".



Simulated Annealing:-

Simulated Annealing is similar to hill climbing algorithm. It works on the current situation. It picks a random move instead of picking the best move.

This search technique was first used in 1980 to solve VLSI layout problems.

---

8. Searching with nondeterministic actions:-

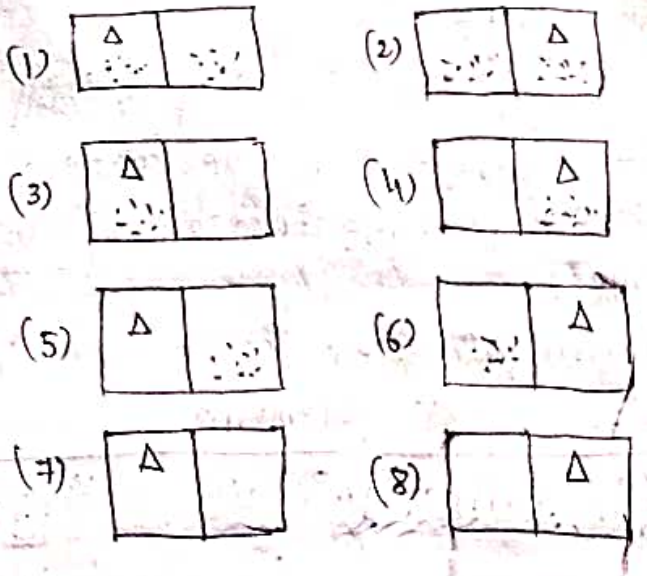
If the environment is fully observable & deterministic and the agent perceptes the information from the enviroment, it calculates exactly which state results from any sequence of actions and always knows which state it is.

If the environment is either partially observable & nondeterministic & both, percepts become useful for the agent, it is easier for the agent to achieve its goals.

In above both cases, the future percepts cannot be determined in advance and the agents future actions will depend on those future percepts. The contingency plan (also known as strategy) that specifies what to do depending on what percepts are received.

Ex:- We use the vaccum world (erratic).

In erratic vaccum world there are 3 actions: Left, Right, and Suck. The state space has 8 states. Its goal is to clean up all the dirt on the floor.



Here, (7) and (8) are goal states.

Many problems in the real, physical world are contingency problems because exact prediction is impossible

The next question is how to find contingent solutions to • nondeterministic problems.

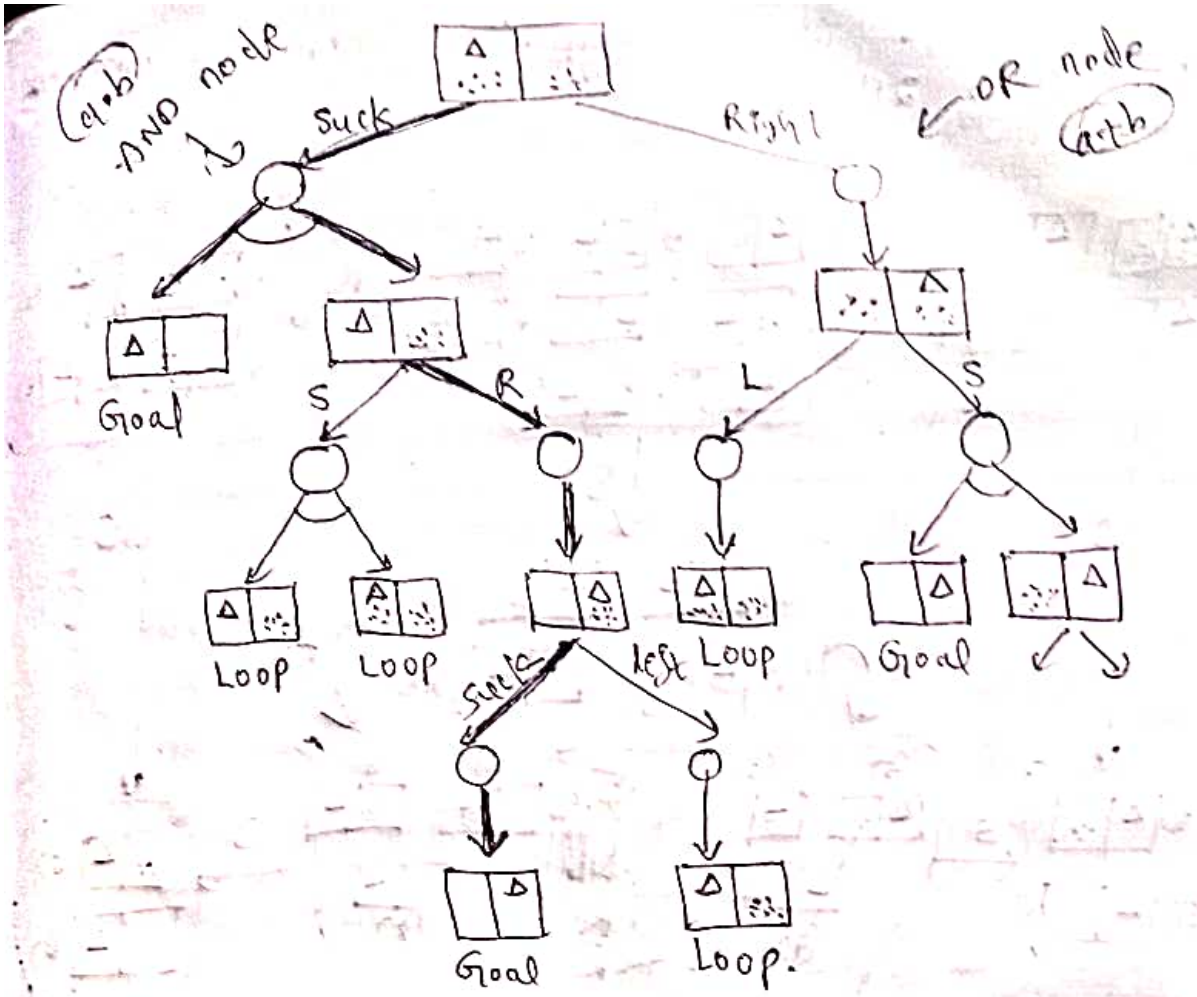
OR node in the vaccum world, the agent chooses Left or Right or Suck.

AND node in the vaccum world, the suck action in state 1 leads to a state in the set {5,7}.

These 2 kinds of nodes AND-OR search tree is illustrated in the given figure.

A solution for an and-or search problem is a subtree that has a goal node at every leaf, specifies one action at each of its OR nodes and includes every outcome branch at each of its AND nodes.

AND nodes shown as circles. The solution found is shown in bold lines.



Searching with Partial Observations:-

The key concept required for solving partially observable problems is the belief state.

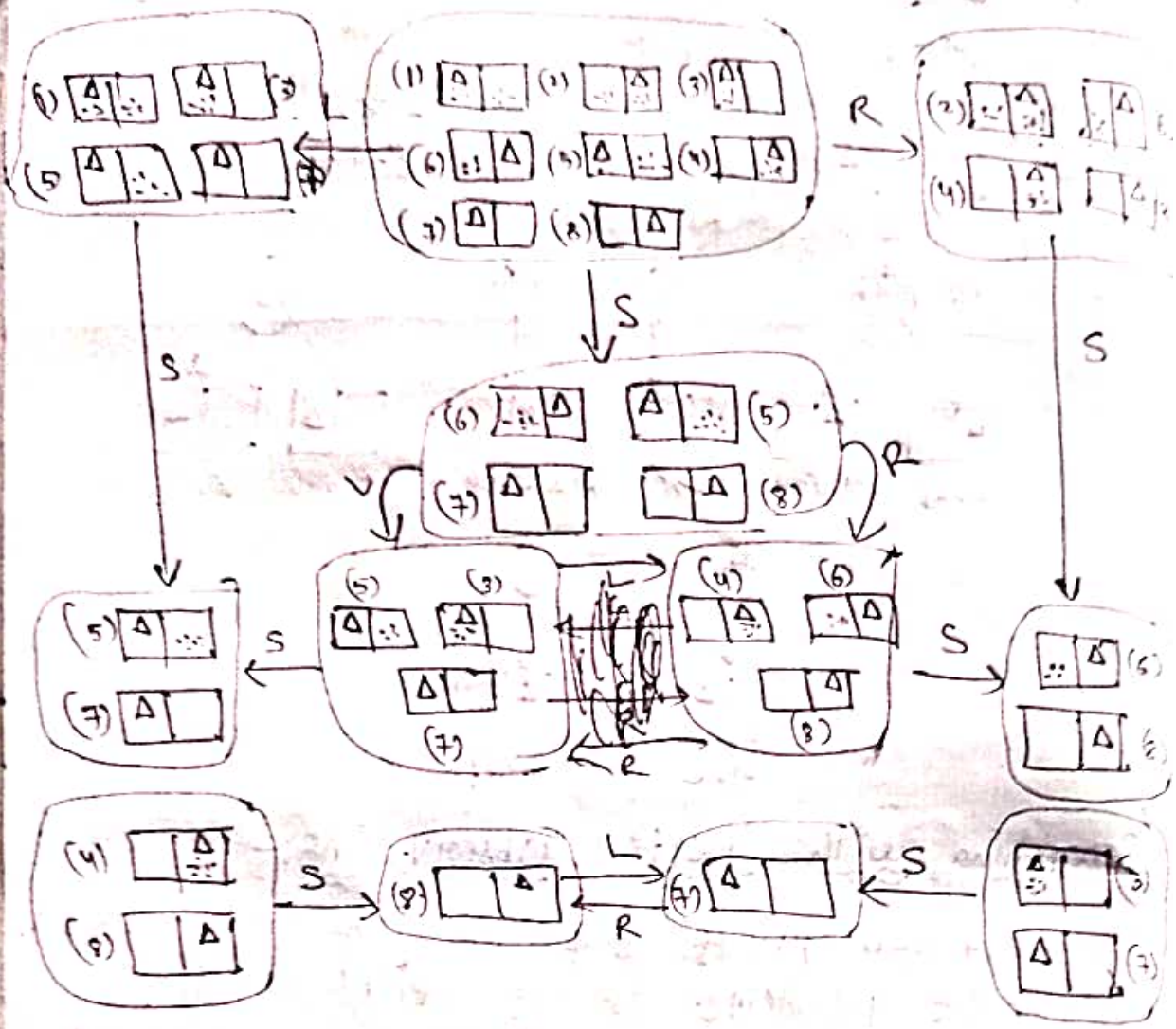
Searching with no observation:-

When the agent's percepts provide no information at all, it is called a sensorless problem (or sometimes conformant problem.

To solve sensorless problems, we search in the space of belief states rather than physical states. Notice that in belief-state space, the problem is fully observable because the agent always knows its own belief state.

Belief states: The entire belief state space contains every possible set of physical states.

If 'p' has 'N' states, then the sensorless problem has up to 2N states.



Online Search Agents & Unknown Environments: