V23CST42 - Machine learning

UNIT - I

Problem Solving.

Topics to be Covered :-

=> Introduction to AI

=> AI Applications

=> Problem Solving agents

=> Search Algorithms

=> Uninformed Search Strategies

=> Heuristic Search Strategies

=> Local Search and optimization problems

=> Adversarial Search.

=> Constraint Satisfaction Problem (CSP)

Lecture 1 :-

Introduction to AI :-

* Act like a human
* Think like a human.
* Thinking rationally - the "laws of thought approach".
* Acting Rationally.

Foundations of AI :-

* Philosophy Provides base to AI by providing theories of relationship between physical brain and mental mind.

* Mathematics gives strong base to AI to develop concrete and formal rules for draw conclusions.
* Economics supports AI to make decisions
* Neuroscience gives information which is related to brain processing.
* Phychology provides strong concepts of how humans and animals think and act.

Strong and Weak AI :-

* Strong form AI provides theories for developing some form of computer based AI that can truly reason and solve problems.
* A strong form of AI is said to be sentient or self aware.
* Weak AI deals with the creation of some form of computer based AI that cannot truly reason and solve problems.
* They can reason and solve problems only in a limited domain.

Lecture 2 :-

AI applications :-

1. Autonomous planning and scheduling :-

* NASA's Remote Agent Program became the first on-board autonomous planning program to control the scheduling of operations for spacecraft.
* Such remote agents can do task of detecting, diagonosing and recovering from problems as they occured.

2. Game playing :-

* A Computer chess program by IBM named as Deep Blue defeated world chess champion Garry kasparov in exhibition match in 1997.
* Such type of gaming programs can be developed using AI techniques.

3. Autonomous Control :-

* The ALVINN Computer vision system was trained to steer car to keep it following a lane.
* It was made to travel 2850 miles in which 98% of the time control was with the system and only 2% of the time human took over.
* AI can give more theories to develop such systems.

4. Diagonosis :-

* Heckerman describes a case where a leading expert on lymph node pathology scoffs at a program's diagonosis of an difficult case.
* The machine can explain the diagnosis. The machine points out the major factors influencing its decision and explain interaction of several of the symptoms in this case.
* If such diagnostic programs are developed using AI then highly accurate diagonosis can be made.

5. Logistic Planning :-

* In 1991 during the Persion Gulf Crisis U.S forces deployed a dynamic analysis and replanning tool name DART for automated logistics planning and scheduling for transportation.
* AI can provide techniques for making fast and acurate plans.

6. Robotics :-

* For doing complex and critical tasks systems can be developed using AI techniques.
* For e.g. Surgeons can use robot assistants in microsurgery which can generate 3D vision of patients internal anatomy.

Lecture 3 :-

Problem Solving Agents :-

Agent :-

* An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators.
* Agent : Software Agent.

  Sensors : keystrokes, file contents and network packets.

  Actuators : Screen, writing files, network packet.

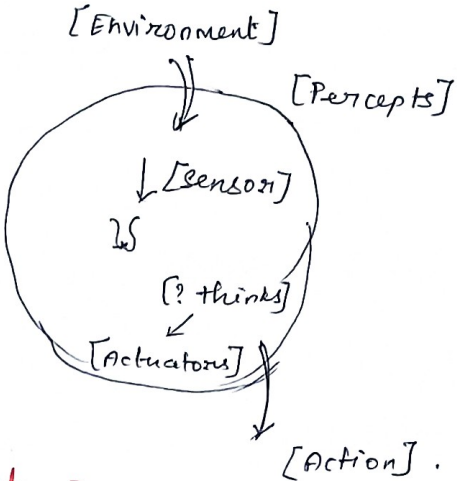
The AI Terminology :-

* Percept
* Percept Sequence
* Agent Function.
* Agent Program.

Architecture of Agent :-

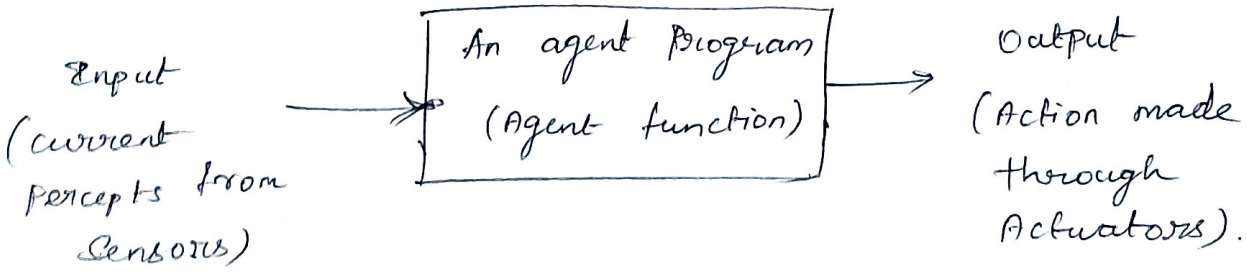
* The agent program runs on some sort of Computing device, which is called the architecture.

Agent = Architecture + Program.



Role of an Agent Program :-

* An agent program is internally implemented as agent function.
* An agent program takes input as the current percept from the sensor and return an action to the effectors [Actuators]



What is Problem Solving :-

* For solving any type problem (task) in real world one needs formal description of the problem. One should have clear understanding of following aspects of the problem.

1. What is the Explicit Goal of the Program?
2. What is Implicit Criteria for Success?
3. What is Initial Situation?
4. Ability to Perform.

Well defined problems :-

* Problem formulation is the process of deciding what actions and states to consider given a goal.
* A problem can be defined formally by four components

1. Initial State that the agent starts in
2. A description of the possible actions available to the agent.
3. The goal test
4. A path cost function.

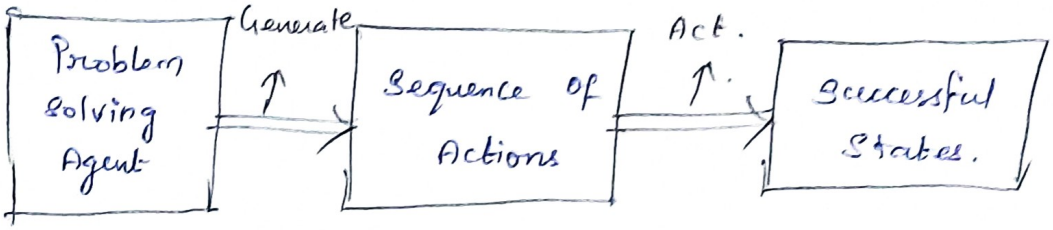
Problem Formulation Types :

1. Incremental formulation.
2. Complete State Formulation.

Solving the problem :-

1. Problem definition.
2. Problem Analysis
3. Knowledge Representation
4. Problem Solving.

Problem Solving agents:-



* Goal based agents are also called as problem solving agent.
* Problem Solving agent adapt to the task environment understand goal and achieve success.
* It determines sequence of agents actions which generate successful state.
* It can be aimed at maximizing performance measure there by developing intelligent problem solving agent.

Steps in problem solving :-

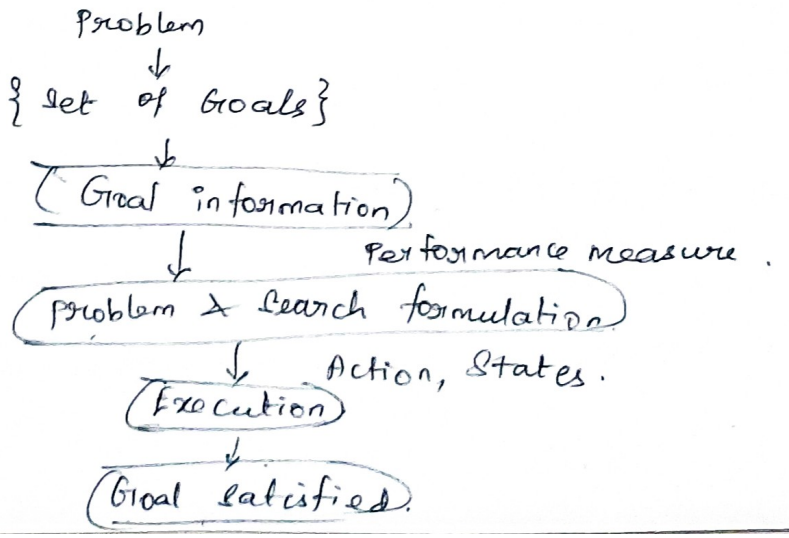
Step 1 : Goal Setting.

Step 2 : Goal Formulation.

Step 3 : Find sequence of actions.

Step 4 : Search in unknown environment.

Step 5 : Execution phase.



Algorithm :-

Procedure or method : Problem solving agent (unknown space, percept).

Result : An action.

Input : P → Percept (Environment perception).

Static :

1) A → An action sequence, initially with null value.
2) S → State - current state.
3) Gr → Goal - A goal initially null.
4) Pr → Problem - A real world situation.

State - update state (state, percept)

If (s) is empty then do

g ← Formulate goal (s)

P ← Formulate problem (s, g)

S ← Search (P)

Gr ← First (s)

S ← Rest (s)

Return a

Procedure.

Example :-

Open - loop system.

1. Static environment
2. Observable environment
3. Discrete environment.
4. Deterministic Environment.

Lecture 4:-

Search algorithms :-

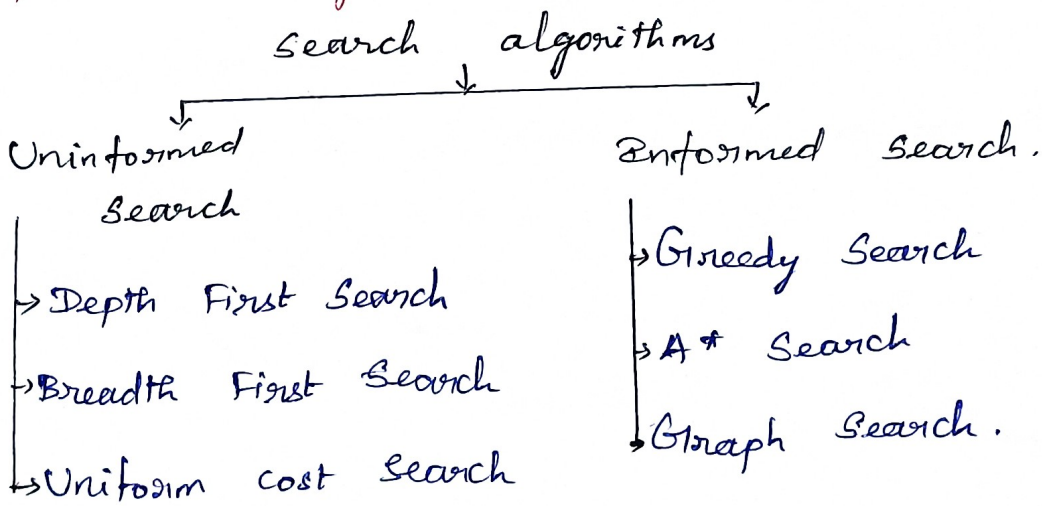
* AI is the study of building agents that act rationally.
* Most of the time these agents perform some kind of search algorithm in the background in order to achieve their tasks.
* A search problem consists of :

  State space :- Set of all possible states where you can be.

  Start State :- The state from where the search begins.

  Goal State :- A function that looks at the current state returns whether or not it is the goal state.
* The solution to a search problem is a sequence of actions, called the plan that transforms the start state to the goal state.
* This plan is achieved through search algorithms.

Types of search algorithms :-



Lecture 5 :-

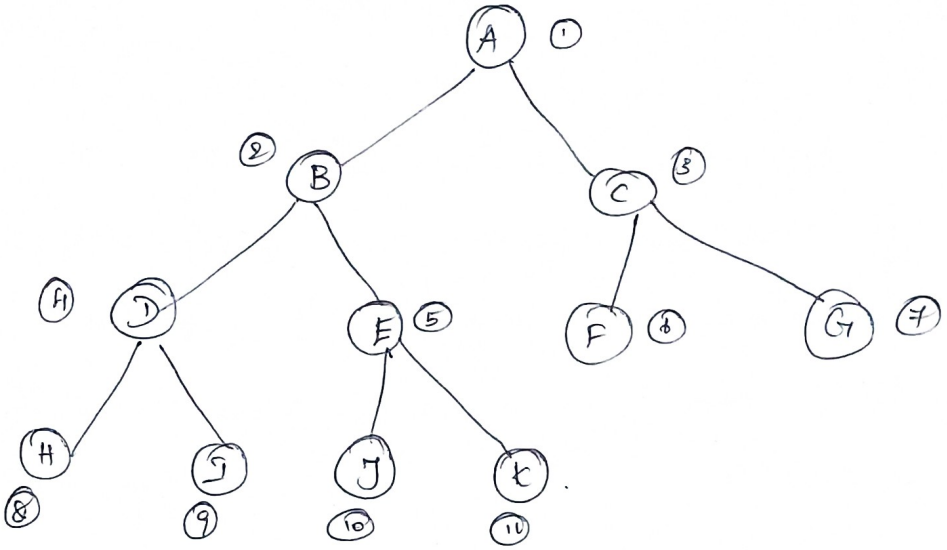
Uninformed Search Strategies :-

* They have no additional information about states other than provided in the problem definition.
* They can only generate successors and distinguish between goal state and non-goal state.
* 1. B.F.S
  2. D.F.S
  3. Depth Limited Search
  4. Iterative Depening DFS.
  5. Bidirectional Search.
  6. Uniform Cost Search.

1. Breadth First Search :-

* In BFS root node is expanded first, then all the successor of root node are expanded and then their successor and so on.
* That is the nodes are expanded levelwise starting at root level.
* BFS using can be implemented using first, in first out queue data structure where fringe will be stored and processed.
* As soon as the node is visited it is added to queue.
* All newly generated nodes are added to the end of the queue, which means that shallow nodes are expanded before deeper nodes.

Working :-



Algorithm :-

Breadth first search of G

```
{
 for i := 1 to n do
    visited [i] := 0;
 for i := 1 to n do
    if (visited [i] = 0) then BFS (i);
}
```

Examples :-

(i) BFS can help in Robot path finding.

(ii) Maze solving algorithm.

Performance Evaluation :-

1. Completeness : BFS is complete.
2. Optimality : BFS will yield optimal solution only when all actions have the same cost, let it be at any depth.
3. Time complexity - $O(b^{d+1})$
4. Space complexity - $O(b^{d+1})$.

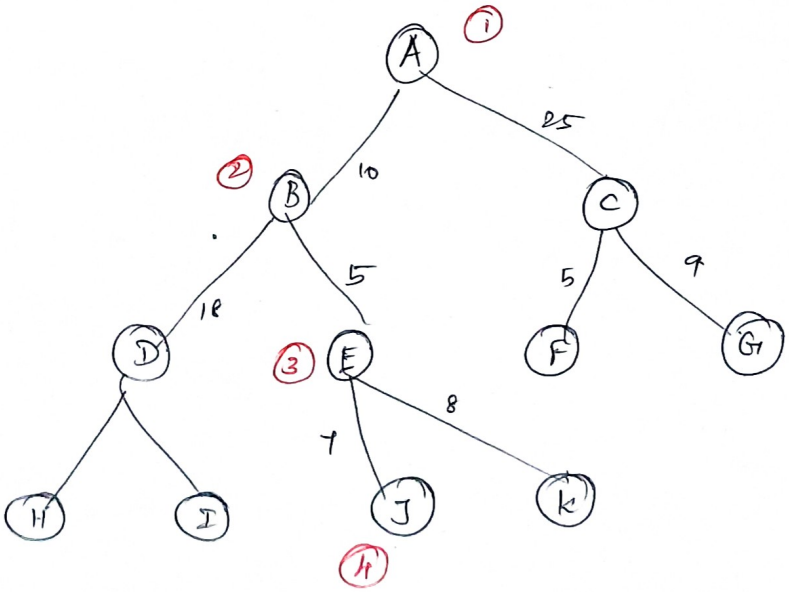
2. Uniform Cost Search :-

* UCS is a pathfinding algorithm that expands the least-cost node first, ensuring that the path to the goal node has the minimum cost.
* UCS takes into account the cost of each path, making it suitable for weighted graphs where each edge has a different cost.

Working :-

1. Initialization :- Starts with root node.
2. Node Expansion : The node with lowest cost is removed from the priority queue. This node is then expanded and its neighbours are explored.
3. Exploring neighbours.
4. Goal check.
5. Repetition.

Example :-



Applications of UCS in AI :-

1. Path finding in Maps.
2. Networking Routing.
3. Puzzle solving.
4. Resource Allocation.

3. Depth First Search :- (DFS)

* It always expands the deepest node in the current unexplored node set of the search tree.
* The search goes in to depth until there is no more successor node.
* As these nodes are expanded, they are dropped from the fringe, so then the search "backs up" to the next shallowest node that still has unexplored successors.

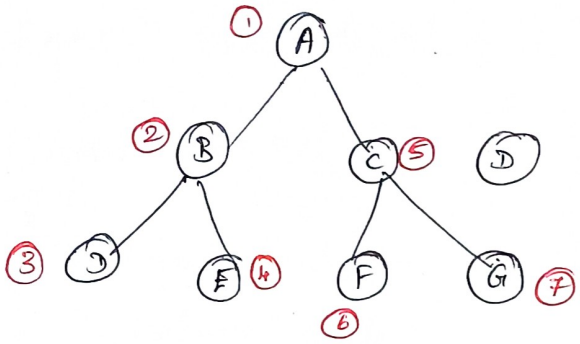
Algorithm :-

```
Algorithm DFS(v)
{
  visited [v] = 1;
  for each vertex W adjacent from v do
   {
      if (visited [w] = 0 then DFS(w);
   }
}
```

Time Complexity - $O(b^d)$

Space Complexity - $O(b^d + 1)$.

Example :-



Applications :-

1. Maze generation.
2. Puzzle - Solving.
3. Pathfinding in Robotics.

4. Depth Limited Search :- (DLS)

* DLS is a modified version of DFS that imposes a limit on the depth of the search.
* This means that the algorithm will only explore nodes up to a certain depth, effectively preventing it from going down excessively deep paths that are unlikely to lead to the goal.
* By setting a maximum depth limit, DLS aims to improve efficiency and ensure more manageable search times.

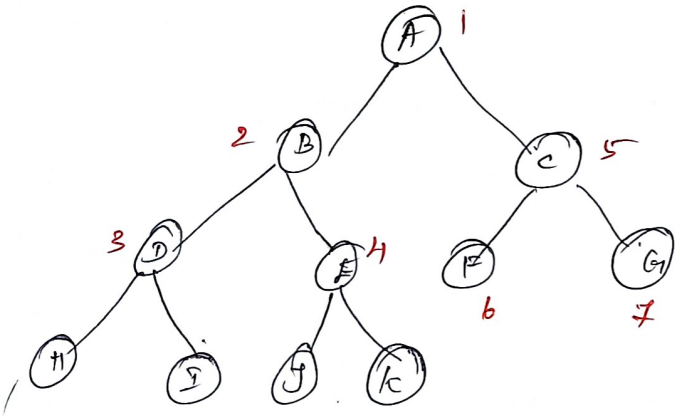
Working :-

1. Initialization
2. Exploration.
3. Depth check.
4. Goal check.
5. Backtracking.

Algorithm :-

```
DLS (node, goal, depth)
{
  if (depth >= 0)
  {
    if (node == goal)
      return node
    for each child expand (node)
      DLS (child, goal, depth-1)
  }
}
```

Example :-



Depth limit = 2.

Applications :-

1. Pathfinding in Robotics.
2. Network Routing Algorithms.
3. Puzzle solving in AI systems.
4. Game playing.
5. Iterative Deepening Depth First Search (IDDFS)

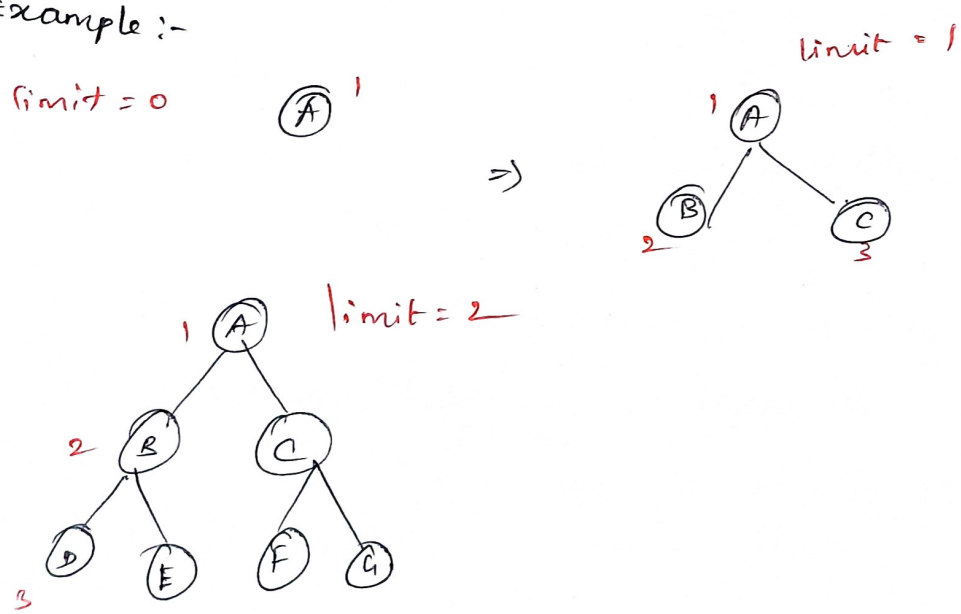
* It combines DFS's space efficiency and BFS's fast search.
* IDDFS calls DFS for different depths starting from an initial value.

* In every call, DFS is restricted from going beyond given depth.

Algorithm :-

```
IDDFS
{
  depth = 0
  while (no solution)
  {
    solution = DLS(root, goal, depth)
    depth = depth + 1.
  }
  return solution
}
```

Example :-



6. Bidirectional search :-

* It is a graph search algorithm which find smallest path from source to goal vertex.
* It runs two simultaneous search

1. Forward search from source / initial vertex toward goal vertex.
2. Backward search from goal / target vertex toward source vertex.

* The search terminates when two graphs intersect.

Lecture 6 :-

Heuristic Search & Strategies :-

* Heuristics operates on the search space of a problem to find the best or closest to optimal solution via the use of systematic algorithms.
* In contrast to a brute - force approach, which checks all posible solutions exhaustively, a heuristic search method uses heuristic information to define a route that seems more plausible than the rest.
* It refers to a set of criteria or rules of thumb that offer an estimate of a firm's profitability.
* Utilizing heuristic guiding, the algorithms determine the balance between exploration and exploitation, and thus they can successfully tackle demanding issues.
* Therefore they enable an efficient solution finding process.

Components :-

1. State space.
2. Initial State
3. Goal Test
4. Successor Function.
5. Heuristic Function.

1. A* Search algorithm :-

* A* search algorithm is one of the best and popular technique used in path-finding and graph traversals.
* Many games and web-based maps use this algorithm to find the shortest path very efficiently.

algorithm :-

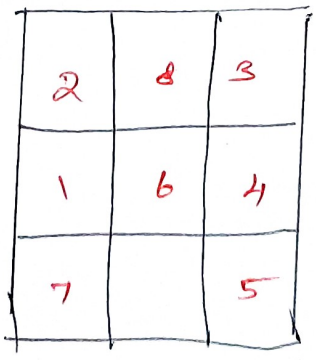
1. A* is most popular form of best first search.
2. A* evaluates node based on two functions namely
   - $g(n)$ – The cost to reach the node 'n' (or) depth.
   - $h(n)$ – The cost to reach the goal node from node 'n'.
3. These two functions' cost are combined into one, to evaluate a node. New function $f(n)$ is deviced as,

$$f(n) = g(n) + h(n)$$ that is,

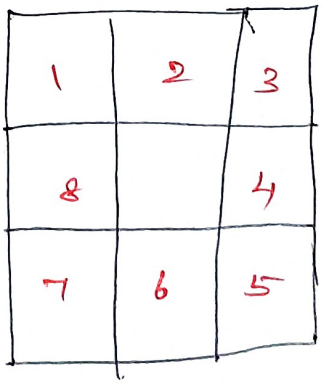
$f(n)$ = Estimated cost of the cheapest solution through 'n'.

Example :-

Given an initial state of a 8-puzzle problem and final state to be reached -



Initial state



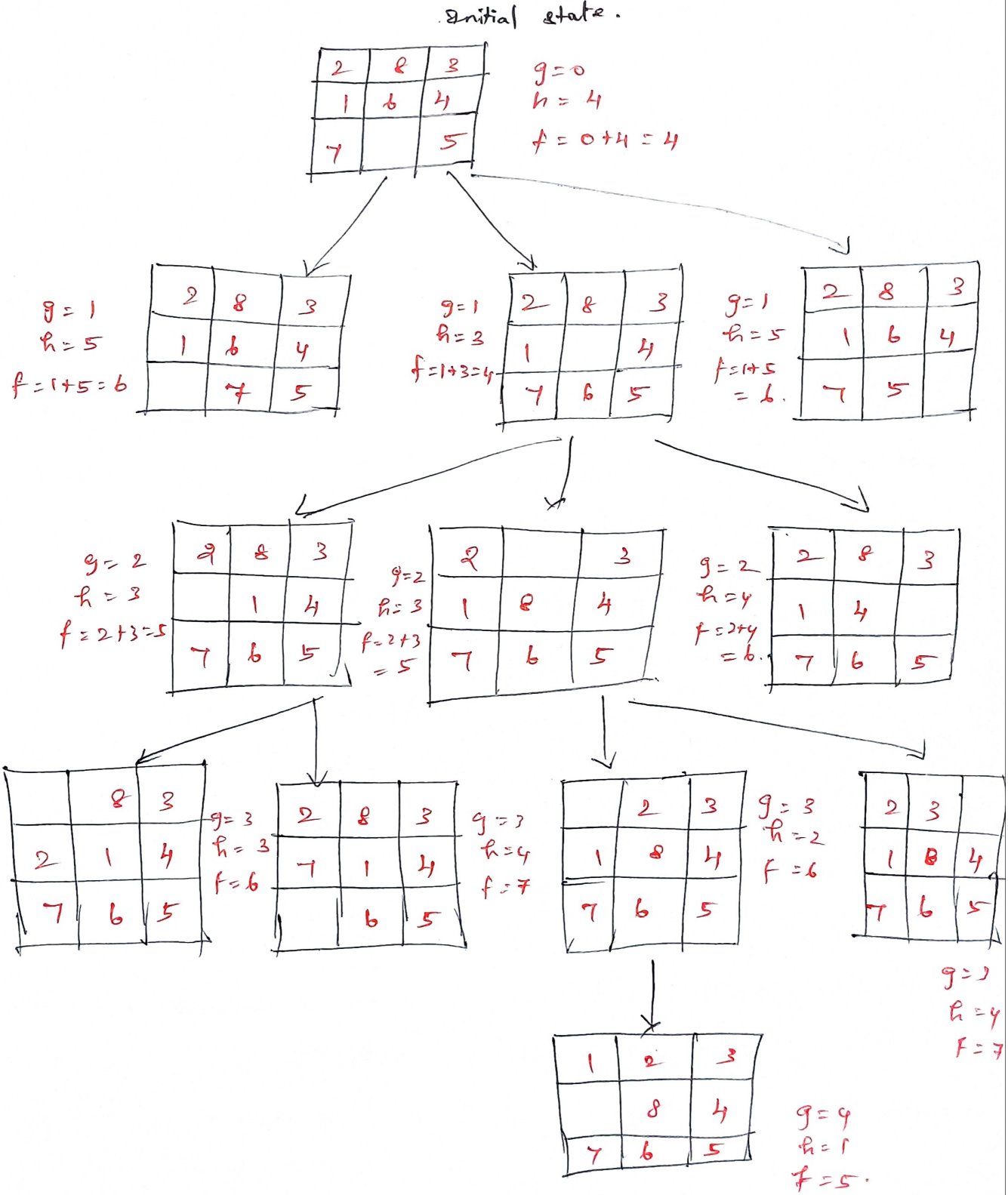
Final state.

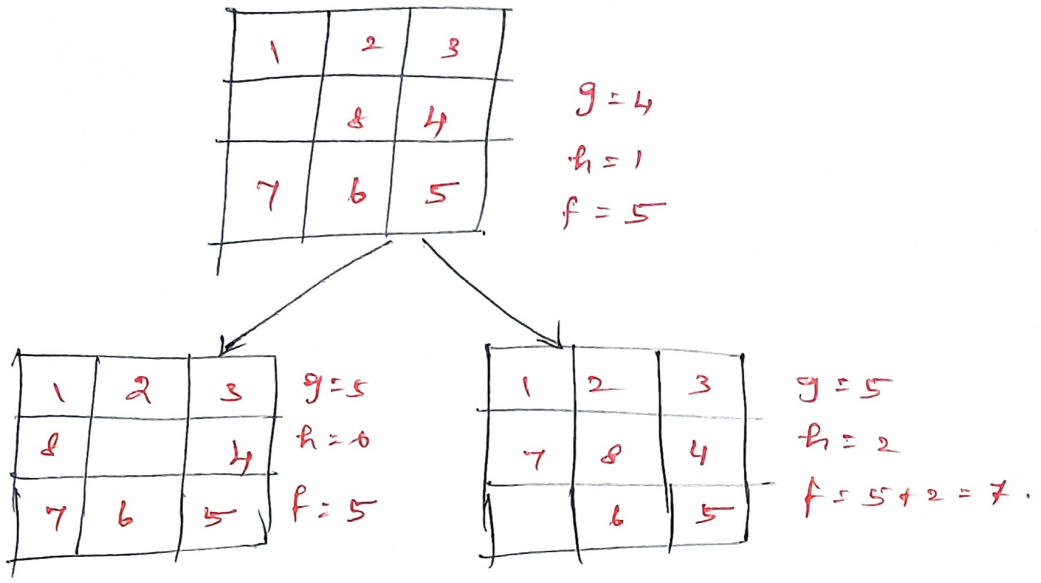
★ Find the most cost effective path to reach the final state from initial state using A* algorithm.

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

★ Consider $g(n)$ = Depth of node and

$h(n)$ = Number of misplaced tiles.





Goal state.

Lecture 6 :-

Local Search and optimization problem :-

* Local search algorithms are essential tools in artificial intelligence and optimization, employed to find high quality solutions in large and complex problem spaces.
* Key algorithms include
  - ⇒ Hill climbing search
  - ⇒ Simulated Annealing.
  - ⇒ Local beam search.
  - ⇒ Genetic algorithms.
  - ⇒ Tabu search.

Hill climbing Search algorithm :-

* It is a heuristic search algorithm used primarily for mathematical optimization problems in AI.
* It is a form of local search, which means it focuses on finding the optimal solution by making incremental changes to an existing solution and then.

evaluating whether the new solution is better than the current one.

* The process is analogous to climbing a hill where you continually seek to improve your position until you reach the top, or local maximum, from where no further improvement can be made.

How algorithm works?

* The process begins with an initial solution, which is then iteratively improved by making small, incremental changes.
* These changes are evaluated by a heuristic function to determine the quality of a solution.
* The algorithm continues to make these adjustments until reaches a local maximum – a point where no further improvement can be made with an current set of moves.

Initial state :-

Start with an arbitrary or random solution.

Neighboring States :-

Identify neighboring states of the current solution by making small adjustments.

Move to Neighbour :-

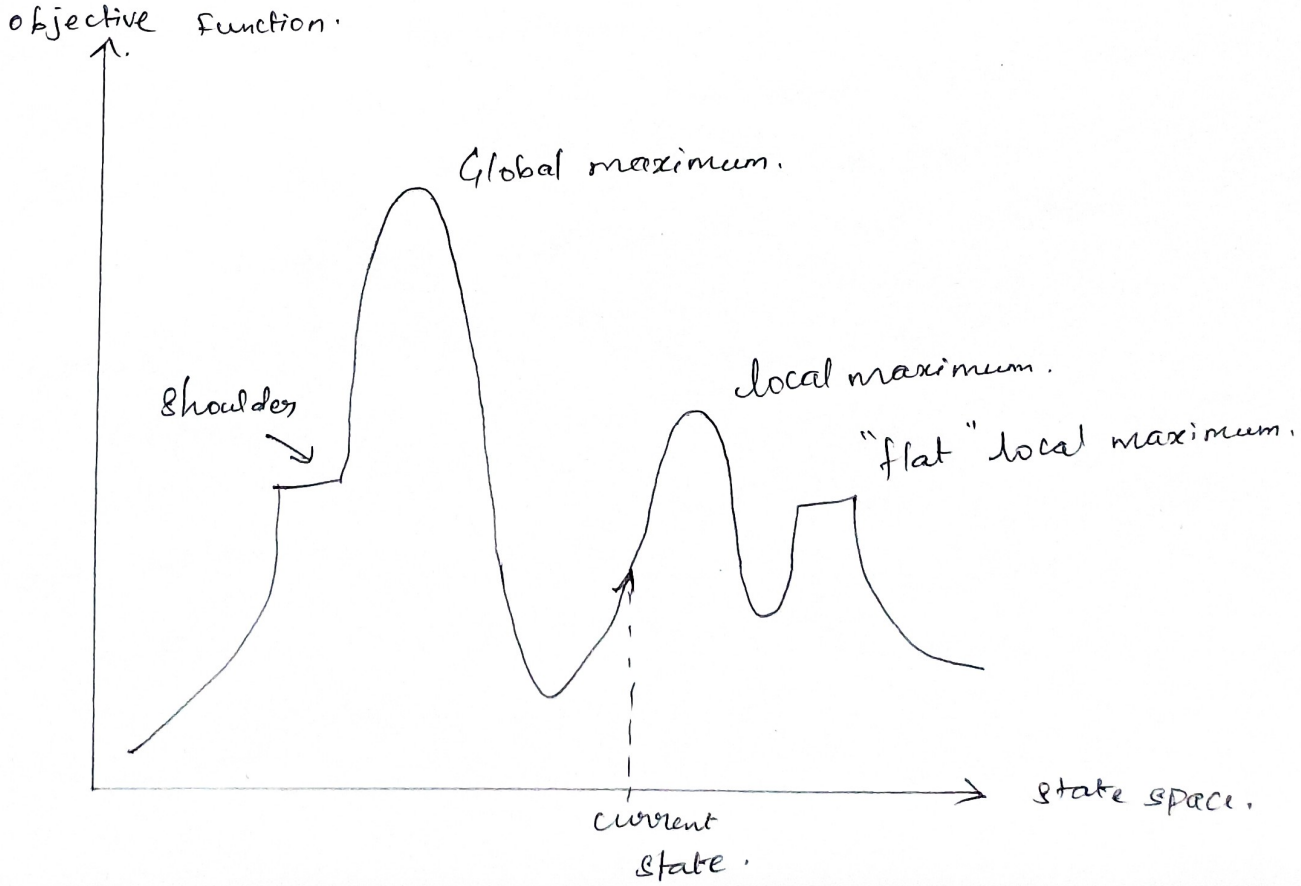
If one of the neighboring states offers a better solution move to this new state.

Termination :-

Repeat this process until no neighboring state is better than the current one.

Algorithm.

1. Evaluate the initial state. If it is a goal state, return success.
2. Make the initial state the current state.
3. Loop until a solution is found or no operators can be applied:
   (i) Select a new state that has not yet been applied to the current state.
   (ii) Evaluate the new state.
   (iii) If the new state is the goal, return success.
   (iv) If the new state improves upon the current state, make it the current state and continue.
   (v) If it doesn't improve, continue searching neighboring states.
4. Exit the function if no better state is found.



Simulated Annealing :-

* It is an optimization algorithm designed to search for an optimal or near optimal solution in a large solution space.
* The algorithm starts with an initial solution and a high "temperature", which gradually decreases over time.

Initialization :-

Begin with an initial solution S_0 and an initial temperature T_0. The temperature controls how likely the algorithm is to accept worse solutions as it explores the search space.

Neighborhood search :-

At each step, a new solution S' is generated by making a small change to the current solution S.

Objective Function Evaluation :-

The new solution S' is evaluated using the objective function. If S' provides a better solutions than S, it is accepted as the new solution.

Acceptance probability :-

$$P(\text{accept}) = e^{-\frac{\Delta E}{T}}$$

Cooling schedule :

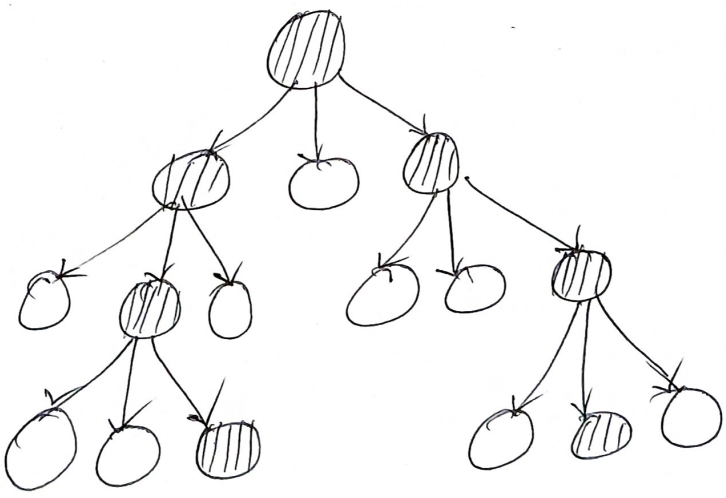
Termination :

Beam Search algorithm:-

* It is a heuristic search algorithm that navigates a graph by systematically expanding the most promising nodes within a constrained set.
* This approach combines elements of BFS to construct its search tree by generating all successors at each level.
* It only evaluates and expands a set number, W of the best nodes at each level, based on their heuristic values.
* This selection process is repeated at each level of the tree.

Example :-

Consider a search tree where W = 2 and B = 3. Only two nodes are selected based on their heuristic values for further expansion at each level.



Lecture 8 :-

Adversarial Search :-

* Adversarial search is a well - suited approach in a competitive environment, where two or more agents have conflicting goals.
* It can be employed in two player zero sum games which means what is good for one player will be the misfortune for the other.
* It plays a vital role in decision making, particularly in competitive environments associated with games and strategic interactions.
* By employing adversarial search, AI agents can make optimal decisions while anticipating the actions of an opponent with their opposing objectives.
* It aims to establish an effective decision for a player by considering the possible moves and the counter moves of the opponents.

Role of adversarial search in AI :-

Game playing :-

* It finds a significant application in game playing scenarios, including renowned games like chess, Go and Poker.
* It offers the simplified nature of these games that represents the state of a game in a straightforward approach

Decision making :-

* It plays a central role in adversarial search algorithms, where the goal is to find the best possible move or strategy for a player in a competitive environment against one or more components.
* This requires strategic thinking, evaluation of potential outcomes, and adaptive decision-making throughout the game.

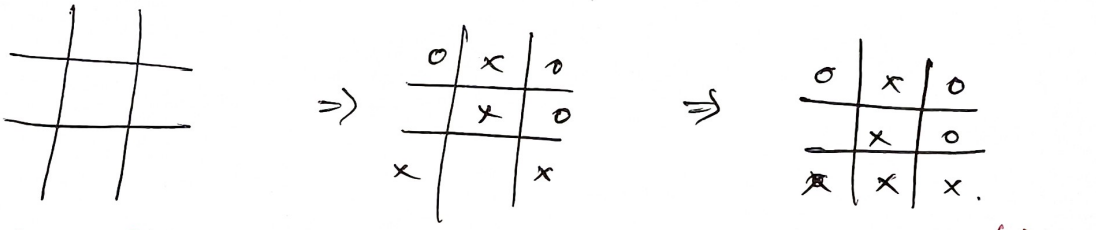
Min-Max algorithm :-

1. The start node is MAX (player 1) node with current board configuration.
2. Expand nodes down (play) to some depth of look ahead in the game.
3. Apply evaluation function at each of the leaf nodes.
4. "Back up" values for each non-leaf nodes until computed for the root node.
5. At MIN (player 2) nodes, the backed up value is the minimum of the values associated with its children.
6. At MAX nodes, the backed up value is the maximum of the values associated with its children.

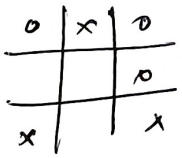
Example :-

Tic - Tac - Toe.

* Each player marks a 3*3 grid by 'x' and 'o' in turn.
* The player who puts respective mark in a horizondal, vertical or diagonal line wins the game.
* If both players fail to reach above criteria and all boxes in the grid are filled, then the game is draw.



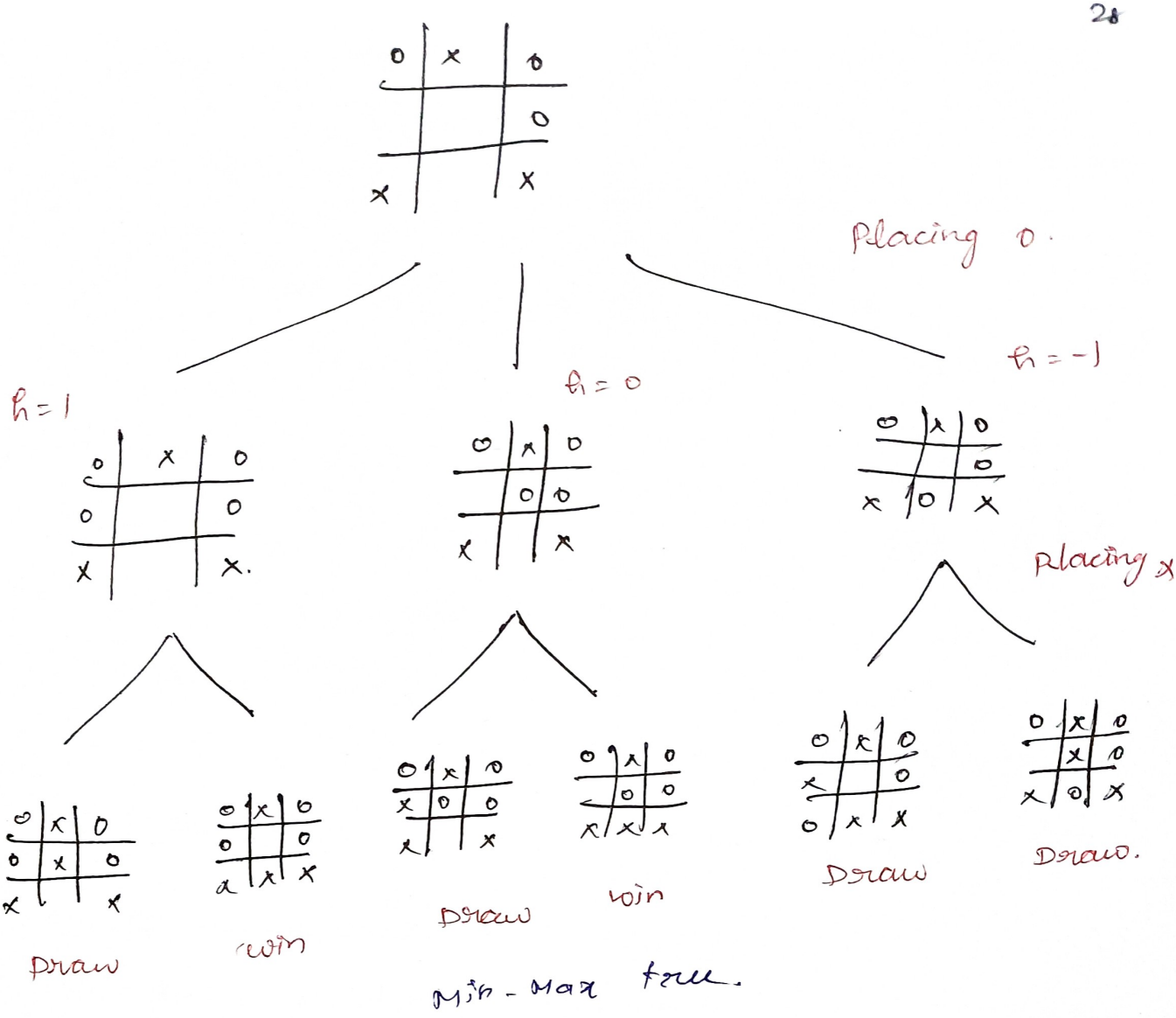
Initial State | Intermediate state. | Final / Goal State.



Heuristic Function :

H = x's probability to win - o's probability to win.

If H = 0 → draw
H = 1 → x wins
H = -1 → o wins.



Min - Max tree.

Alpha - Beta Pruning :-

* It is a optimization technique for the minmax algorithm.
* It reduces the number of nodes evaluated in the game tree by eliminating branches that cannot influence the final decision.
* This is achieved by maintaining two values alpha and beta which represents the minimum score that are the maximizing player is assured of and the maximum score that the minimizing

Player is assumed of, respectively.

Working :-

1. Initialization: start with alpha set to negative infinity and beta set to positive infinity.

2. Max Node Evaluation :-

For each child of max node

=> Evaluate child node using the Min max algorithm with alpha - beta pruning.

=> update alpha : α = max (α, child value).

=> If alpha is greater than or equal to beta, prune the remaining children.

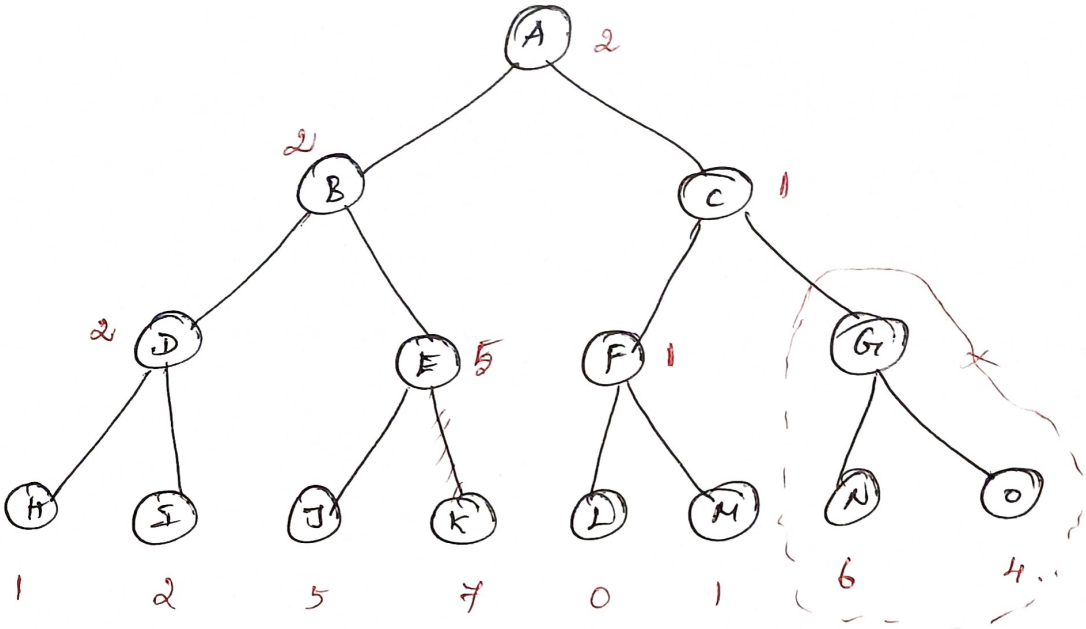
3. Min Node Evaluation :-

For each child of a min node

=> Evaluate the child node using the min max algorithm with alpha - beta pruning.

=> update Beta = β = min (β, child value.)

=> If beta is less than or equal to alpha, prune the remaining children.



Lecture 9 :-

Constraint Satisfaction Problem :-

* A CSP is a mathematical problem where the solution must meet a number of constraints.
* In a CSP, the objective is to assign values to variables such that all the constraints are satisfied.
* CSPs are used extensively in artificial intelligence for decision making problems where resources must be managed or arranged within strict guidelines.

Common applications of CSP :-

1. Scheduling :- Assigning resources like employees or equipment while respecting time and availability constraints.
2. Planning :- Organising tasks with specific deadlines or sequences.
3. Resource Allocation :- Distributing resources efficiently without overuse.

Components of CSP:-

1. Variables :- The things that need to be determined are variables. Variables in a CSP are the objects that must have values assigned to them in order to satisfy a particular set of constraints.

   Eg: Boolean, integer.

2. Domain :- The range of potential values that a variable can have is represented by domains. In Sudoku, the set of numbers from 1 to 9 can serve as the domain of a variable.

3. Constraints :- The guidelines that control how variables relate to one another are known as Constraints.

Types of CSP :-

1. Binary CSP :- In these problems, each constraint involves only two variables. In Scheduling Problem, the constraint could specify that task A must be completed before task B.

2. Non-Binary CSP :- These problems have constraints that involve more than two variables.

   Eg :- Seating arrangement.

3. Hard and Soft constraints :- Hard Constraints must be strictly satisfied, while Soft constraints can be violated but at a certain cost.

CSP Algorithms :-

1. Back Tracking algorithm :-

* It is a DFS method used to systematically explore possible solutions in CSPs.
* It operates by assigning values to variables and backtracks if any assignment violates a Constraint.
  (i) The algorithm selects a variable and assigns it a value.
  (ii) It recursively assigns value to subsequent variables.
  (iii) If a conflict arises, the algorithm backtracks to the previous variable and tries a different value.
  (iv) The process continues until either a valid solution is found or all possibilities have been exhausted.
* This method is widely used due to its simplicity but can be inefficient for large problems with many variables.

2. Forward checking algorithm :-

* The forward checking algorithm is an enhancement of the backtracking algorithm that aims to reduce the search space by applying local consistency checks.

(i) For each unassigned variable, the algorithm keeps track of remaining valid values.

(ii) Once a variable is assigned value, local

Constraints are applied to neighboring variables, eliminating inconsistent values from their domains.

(iii) If a neighbour has no valid values left after forward checking, the algorithm backtracks.

* This method is more efficient than pure backtracking because it prevents some conflicts before they happen, reducing unnecessary computations.

3. Constraint Propagation algorithms :-

* Constraint propagation algorithms further reduce the search space by enforcing local consistency across all variables.

(i) Constraints are propagated between related variables

(ii) Inconsistent values are eliminated from variable domains by leveraging information gained from other variables.

(iii) These algorithms refine the search space by making inferences, removing values that could lead to conflicts.

* It is used to increase efficiency by narrowing down the solution space early in the search process.

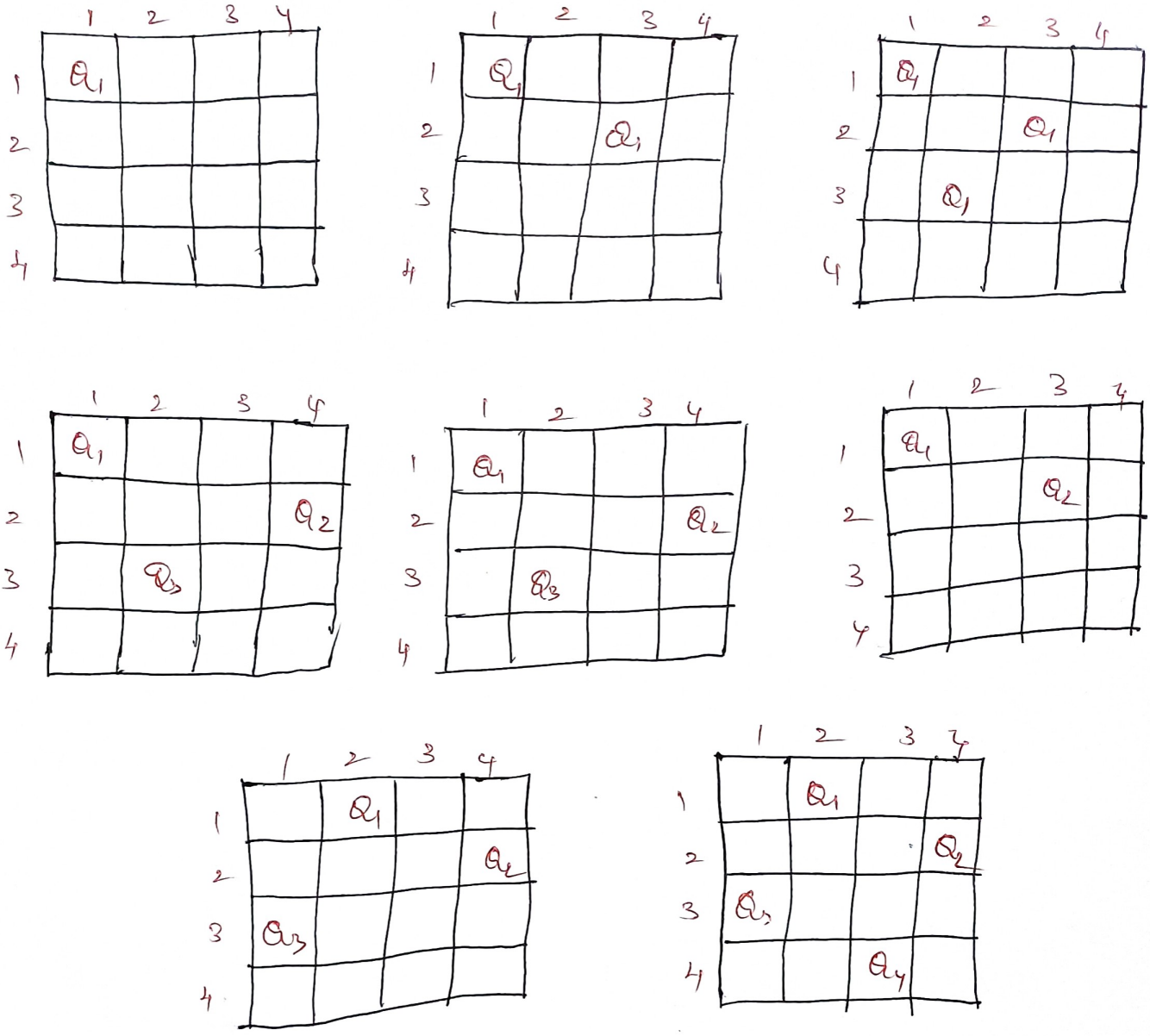
Example :-

(i) N Queen problem:-

* The N Queens is the problem of placing N chess queens on an N x N chessboard so that no two queens attack each other.

Steps :-

1. Start in the leftmost column.
2. If all queens are placed return true.
3. Try all rows in the current column. Do the following for every row.
   - => If the queen can be placed safely in the row,
     - * Then mark this [row, column] as part of the solution and recursively check if placing queen here leads to a solution.
     - * If placing the queen in [row, column] leads to a solution then return true.
     - * If placing queen doesn't lead to a solution then unmark this [row, column] then backtrack and try other rows.
   - => If all rows have been tried and valid solution is not found return false to trigger backtracking.



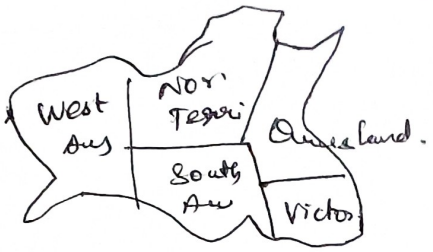
The Solutions are (2, 4, 1, 3)

(3, 1, 4, 2)

(ii) Map Colouring :-

* Two adjacent regions cannot have the same color no matter whatever color we choose.
* The goal is to assign colors to each region so that no neighboring regions have the same color.

* color the following map using red, green, blue such that adjacent regions have different colors.



Variables :- {WA, NT, Q, NSW, V, SA, T}

Domains :- {red, green, blue}.

Constraints :- adjacent regions must have different colors. e.g :- WA ≠ NT.

Constraint graph.