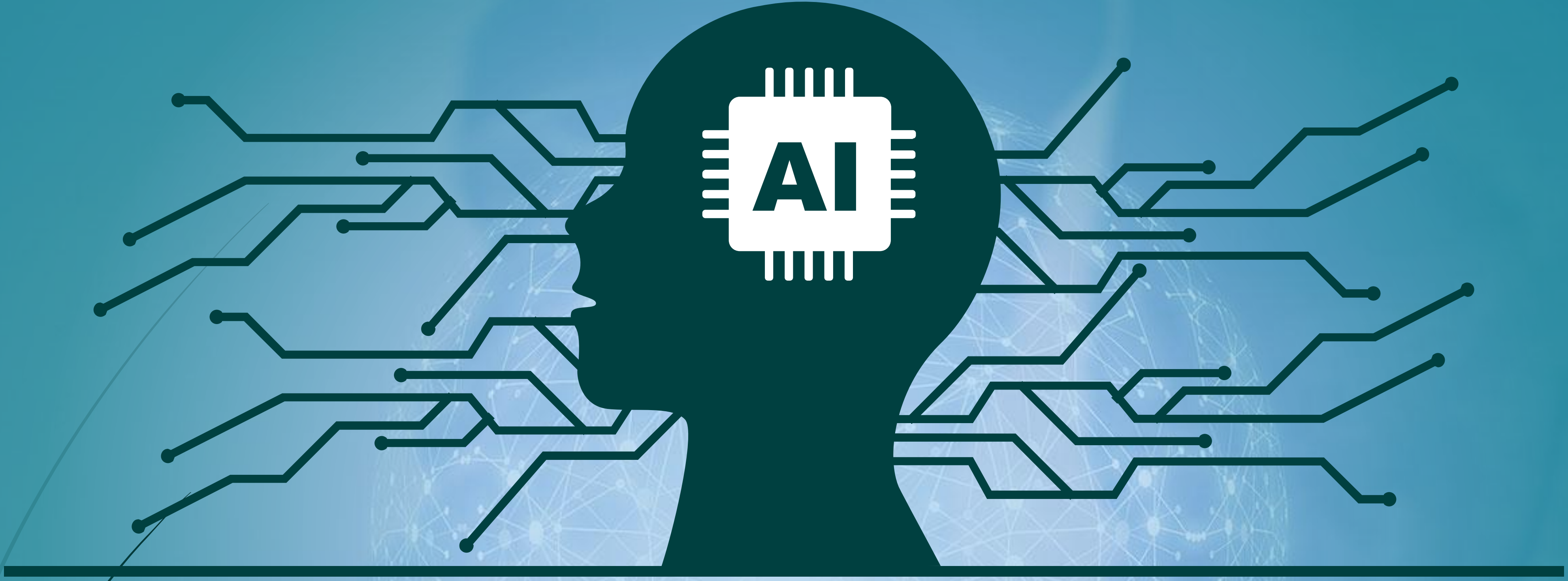


Advanced Artificial Intelligence

Week #5

Dr. Qurat Ul Ain
Assistant Professor
Dept. of AI & DS
FAST NUCES, Islamabad
Email: quratul.ain@isb.nu.edu.pk

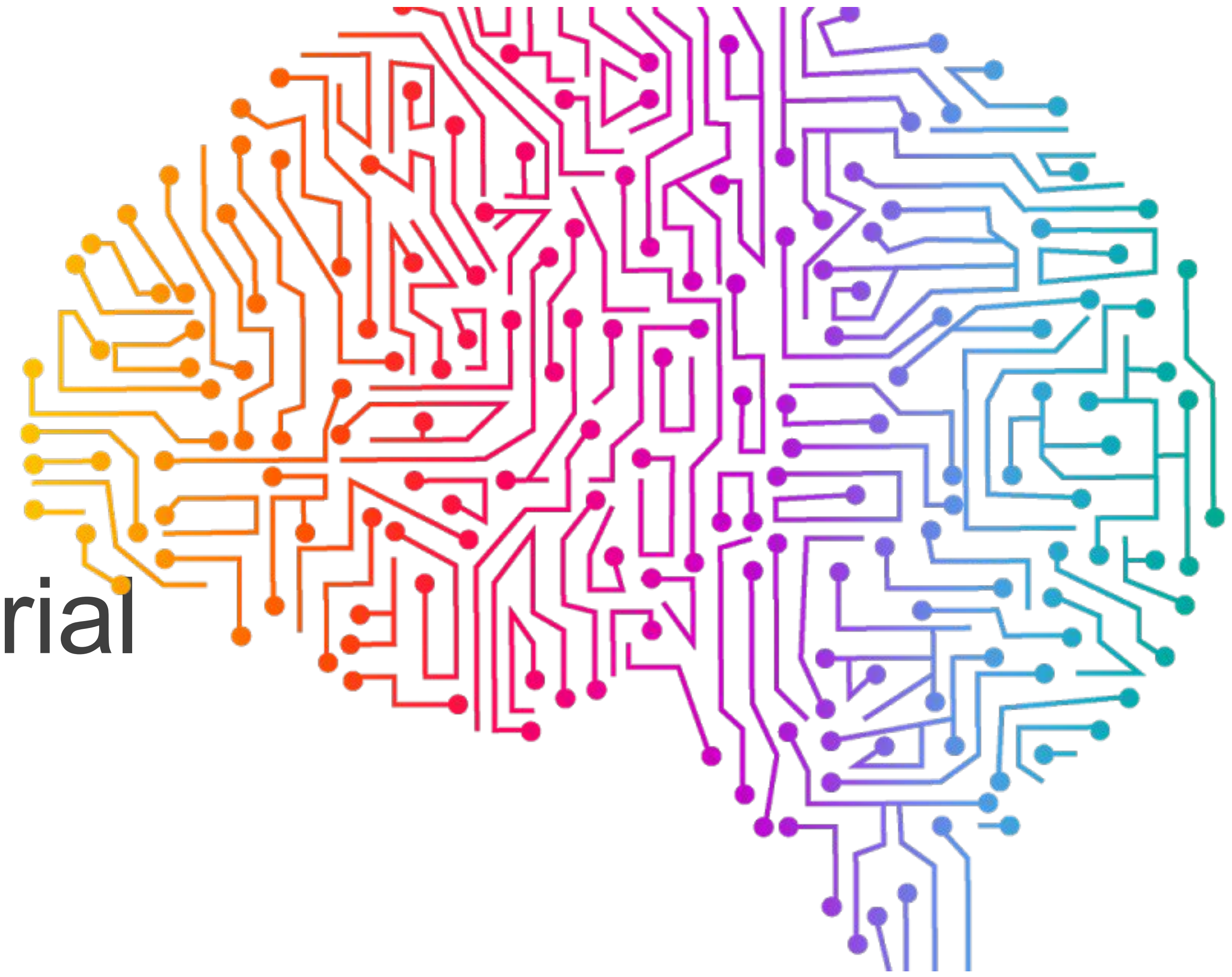
Adversarial Search in Artificial Intelligence

7311

Learning Objective of this Topic

- What is Adversarial Search (Game Playing)?
- MinMax Algorithm
  - Limitations of MinMax Algorithm
- Alpha-Beta Pruning Algorithm

What is Adversarial Search?

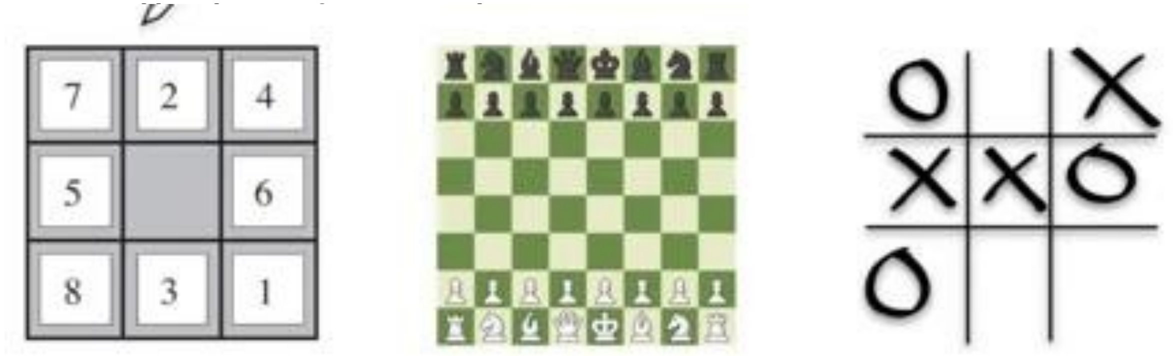


Adversarial Search

- In previous topics, the search strategies were only associated with a single agent that aims to find the solution, often expressed in a sequence of actions.
- But, there might be some situations where **more than one agent is searching for the solution** in the same search space, and this situation usually occurs in **game playing.**
- The environment with more than one agent is termed a multi-agent environment, in which each agent is an opponent of other agents and playing against each other. Each agent needs to consider the action of the other agent and the effect of that action on their performance.
- So, Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games.

Adversarial Search

- Search in competitive environment
  - Where agent’s goal are in conflict
- Games are good examples of adversarial search
  - States are easy to represent (unlike many other real-world problems)
  - Agents are restricted to perform small number of actions
  - Outcome of agent actions defined by precise rules
  - [illegible]



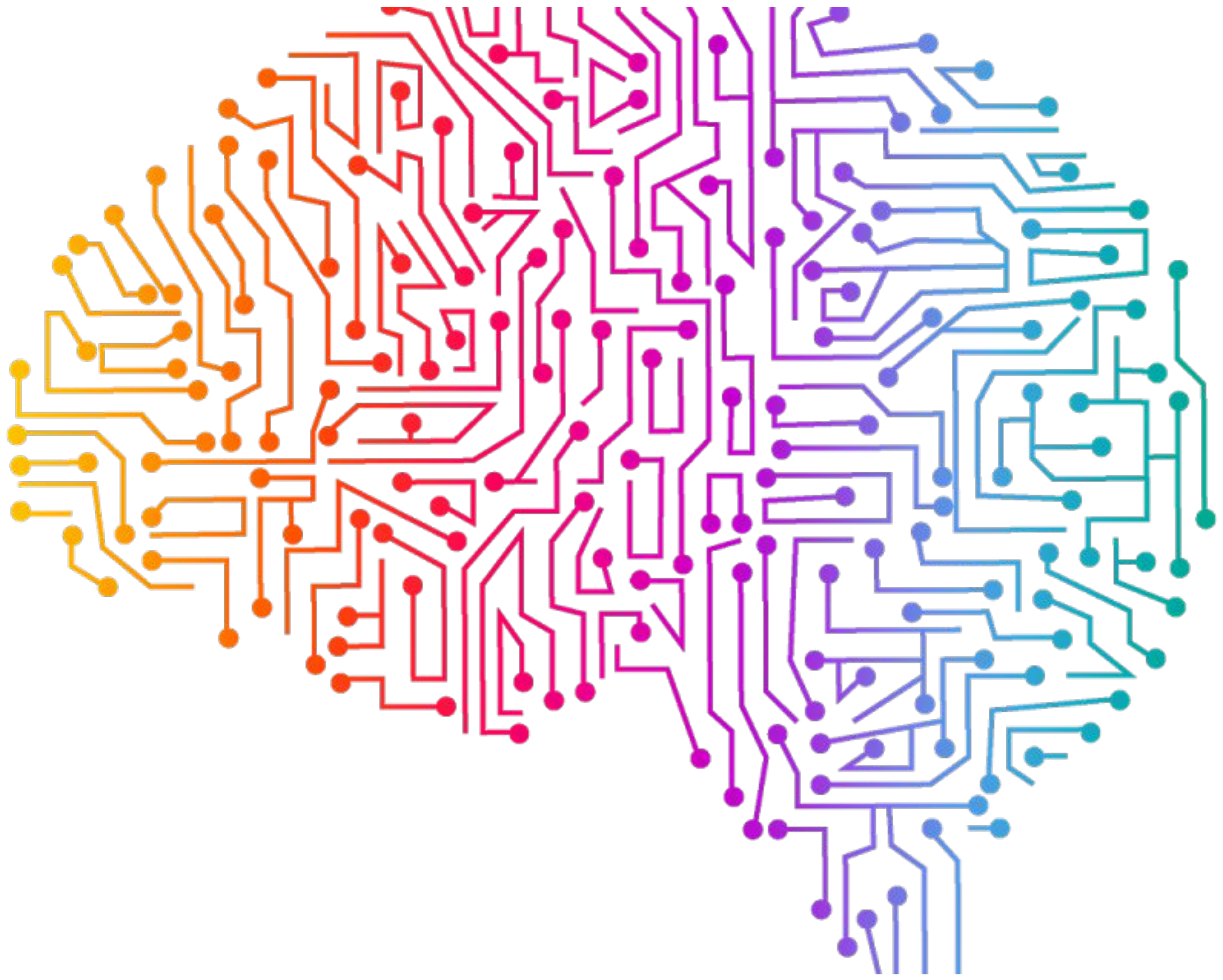
Which one/s are adversarial search?

How to formally define a game

- Consider a game with two players MAX and MIN
    - MAX moves first (places X) followed by MIN
- A game can be formally defined as search problem with following elements:
    - **States:** S (start at s_0)
    - **Players (S) :** defines which player has the move in a state (S)
    - **Action (S):** returns set of possible moves in a state
    - **Transition Function (S, A) :** Maps current state and action to successor state
    - **Terminal Test:** true when game is over, otherwise false
    - **Utility (S, P):** A utility function (objective or payoff function)
        - gives a numeric value for the terminal states (win, loss, draw)
        - defines the final numeric value for the game that ends in a terminal state S for player P
        - For example, in tic-tac-toe, the outcome values can be -1, 0 or +1.
- The initial state, action function and transition function define a game tree for a particular game
    - **Game tree:** a tree where nodes are game states and edges are moves

MinMax

Algorithm



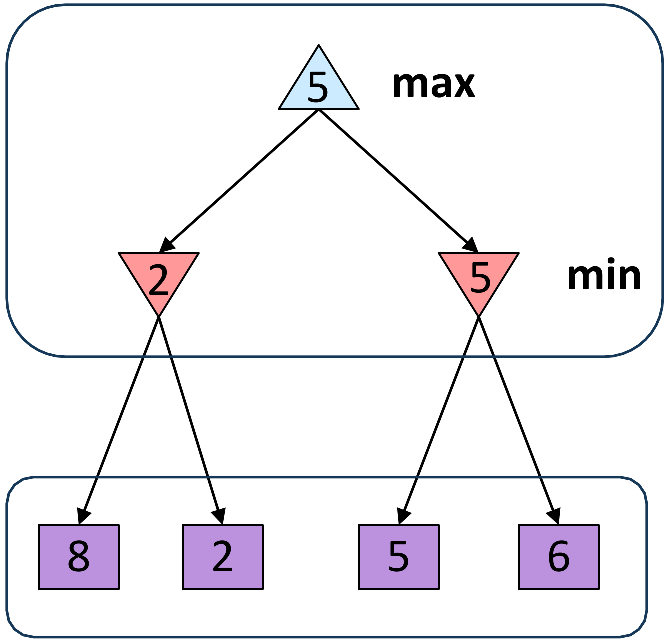
MinMax Algorithm

- Mini-max algorithm is a backtracking algorithm which is used in decision-making and game theory. The Min-Max algorithm uses recursion to search through the game tree.
- The Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, and various two-player games.
- **In this algorithm two players play the game, one is called MAX, and the other is called MIN.**
- Both players fight it as the opposing player gets the minimum benefit while they get the maximum benefit.
- Both game players are opponents of each other, **where MAX will select the maximized value and MIN will select the minimized value.**
- The minimax algorithm performs a **depth-first search algorithm for the exploration of the complete game tree.**
- The minimax algorithm proceeds down to the terminal node of the tree, then backtracks the tree as the recursion.

Adversarial Search (Minimax)

- Deterministic, zero-sum games:
  - Tic-tac-toe, chess, checkers
  - One player maximizes result
  - The other minimizes result
- Minimax search:
  - A state-space search tree
  - Players alternate turns
  - Compute each node's minimax value: the best achievable utility against a rational (optimal) adversary

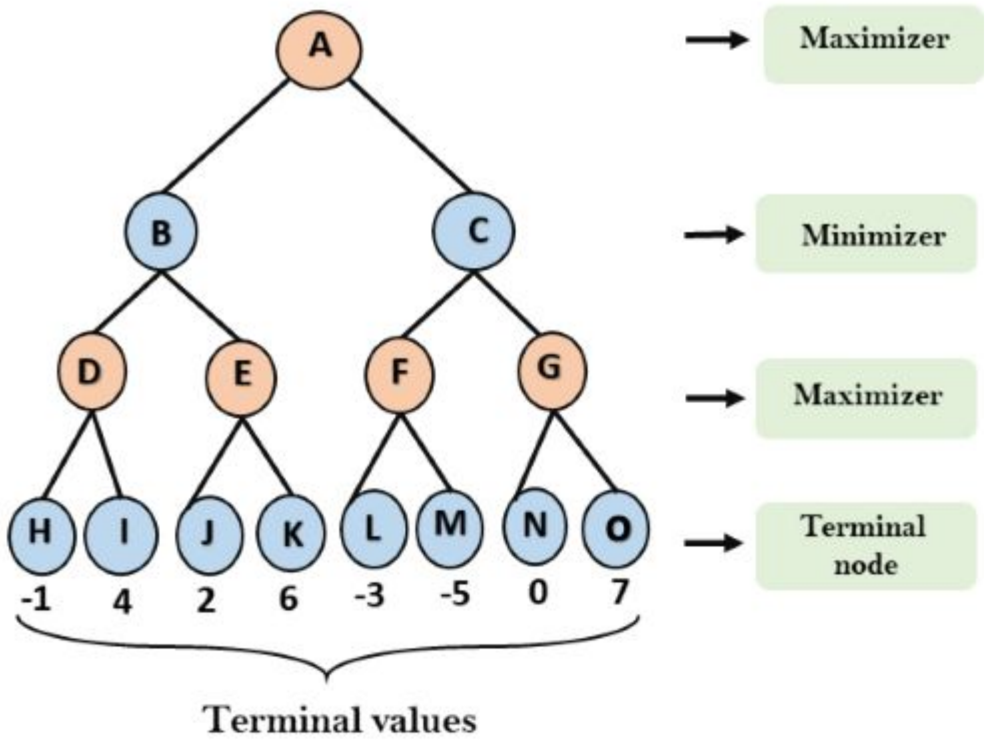
Minimax values: computed recursively



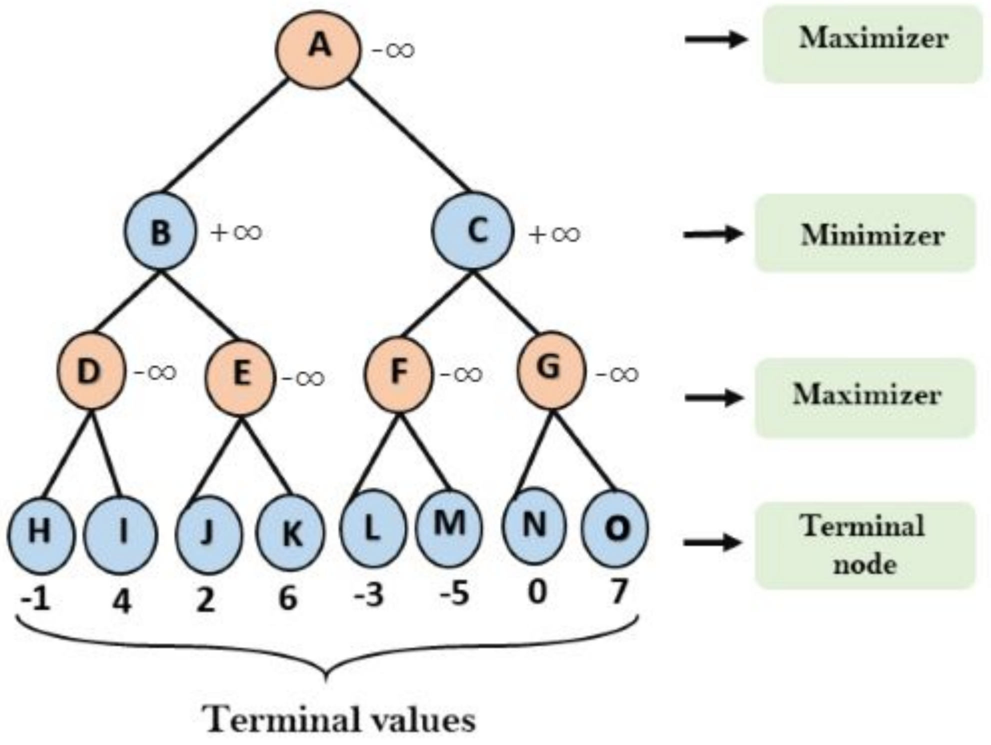
Terminal values: part of the game

Working of Min-Max Algorithm

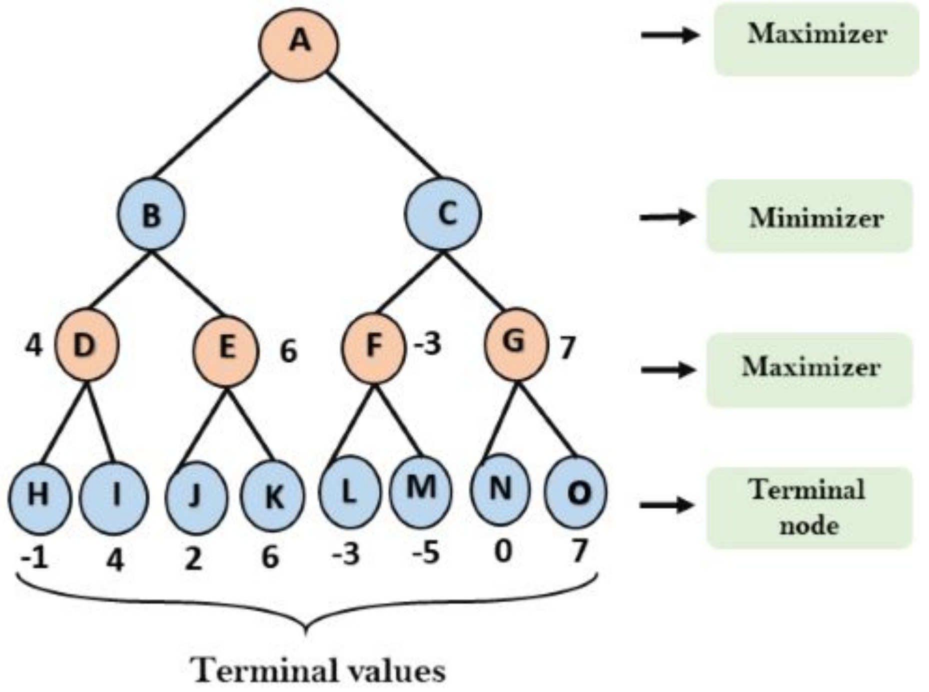
- The working of the minimax algorithm can be easily described using an example of game tree which represents the two-player game.
- In this example, there are two players one is called Maximizer and the other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs. Following are the main steps involved in solving the two-player game tree:



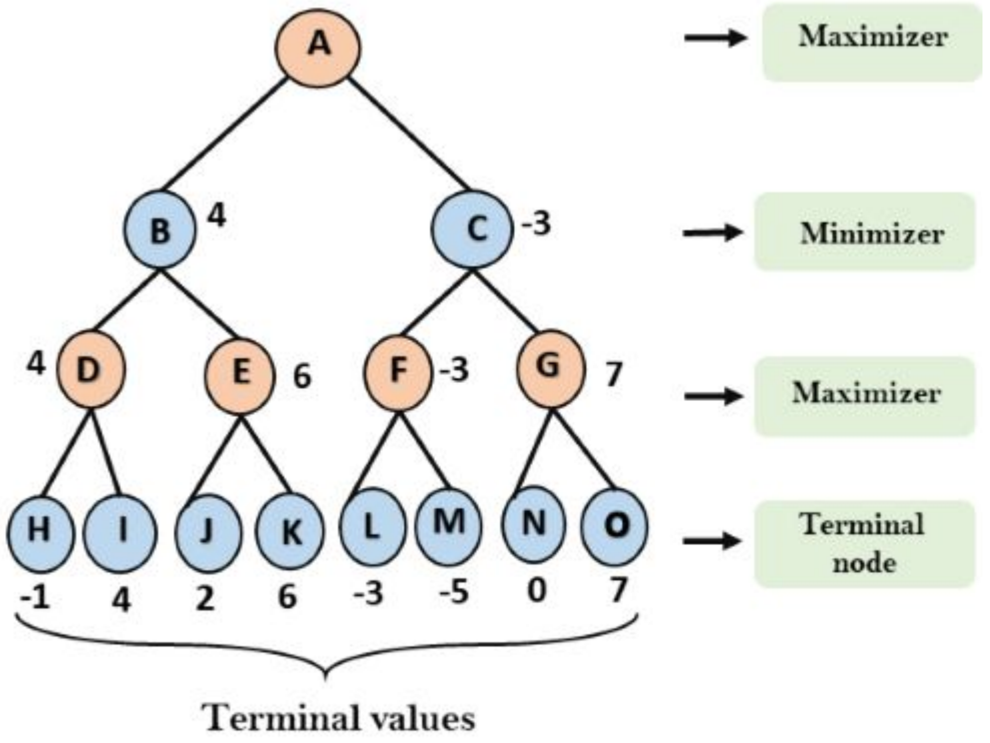
- **Step-1:** In the first step, the algorithm generates the entire game tree and applies the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes the first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



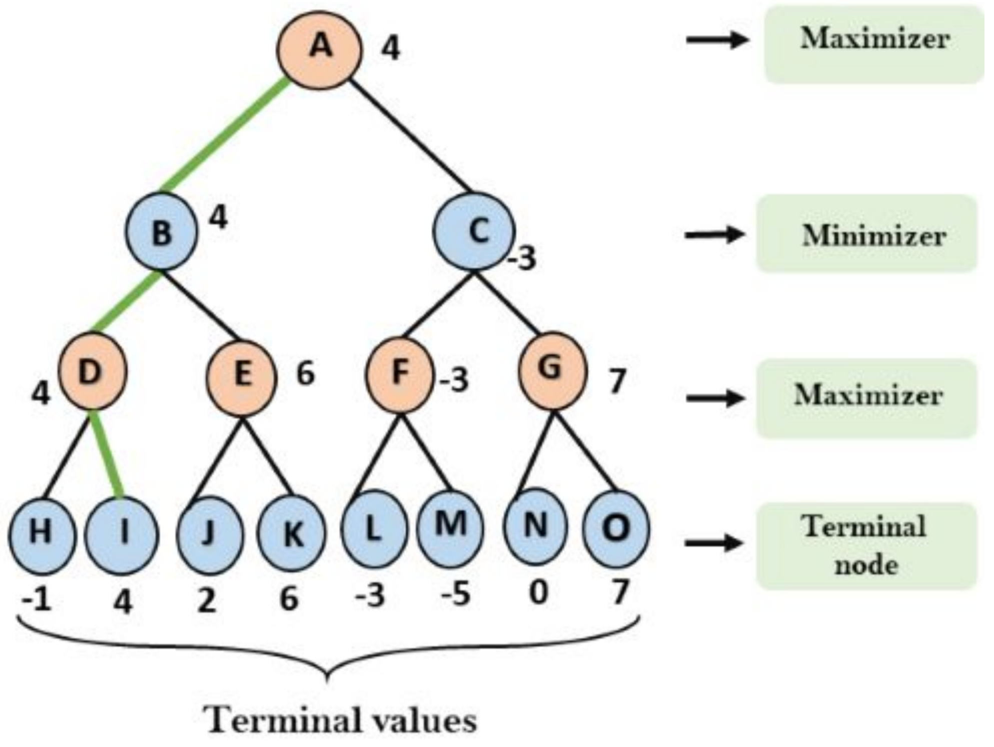
- Step 2: Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.
- For node D max(-1, -∞) => max(-1,4)= 4
- For Node E max(2, -∞) => max(2, 6)= 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) => max(0, 7) = 7



- **Step 3:** In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.
- For node B min(4, +∞) => min(4,6) = 4
- For node C min(-3, +∞) => min (-3, 7) = -3



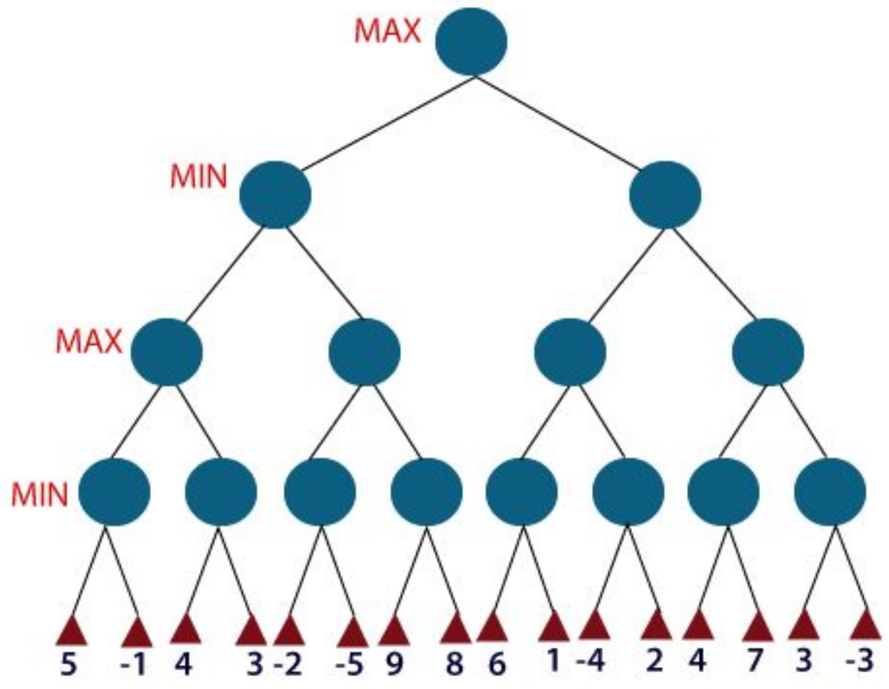
- **Step 4:** Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.
- For node A max(4, -∞) => max(4, -3)= 4



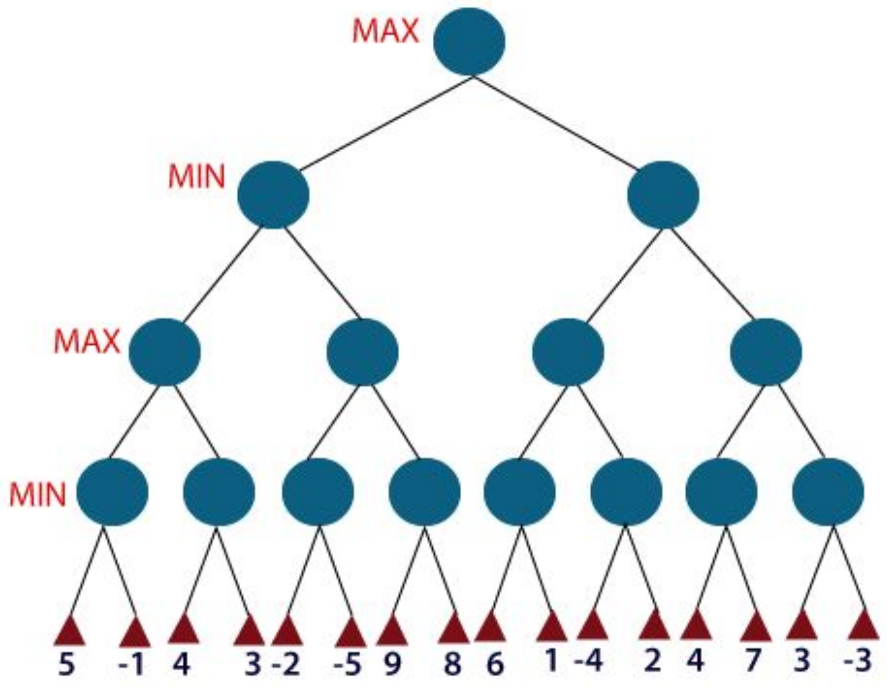
Properties of Mini-Max algorithm

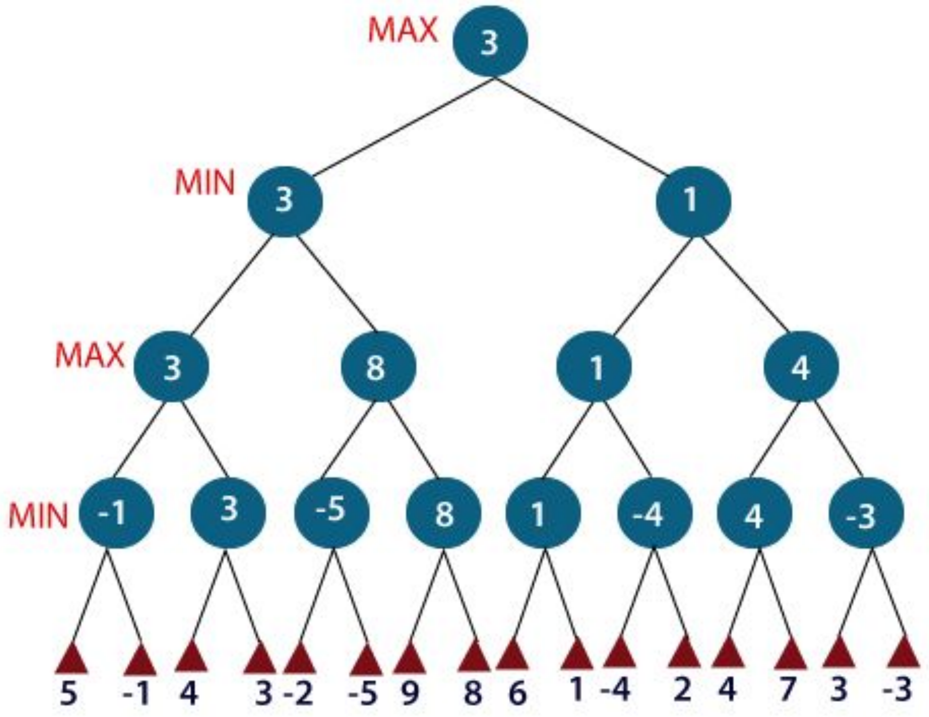
- **Complete-** Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- **Optimal-** Min-Max algorithm is optimal if both opponents are playing optimally.
- **Time complexity-** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is **$O(b^m)$**, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- **Space Complexity-** Space complexity of Mini-max algorithm is also similar to DFS which is **$O(b^m)$**.

Activity



Activity

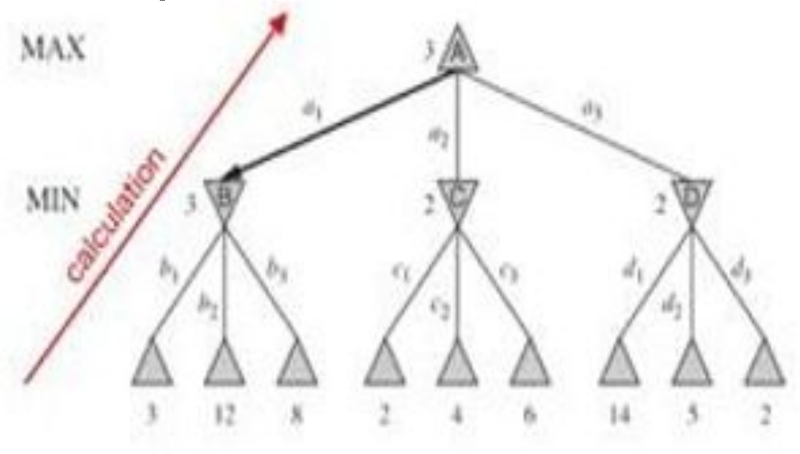

MAX
MIN
MAX
MIN
5 -1 4 3 -2 -5 9 8 6 1 -4 2 4 7 3 -3


MAX
3
MIN
3
1
MAX
3
8
1
4
MIN
-1
3
-5
8
1
-4
4
-3
5 -1 4 3 -2 -5 9 8 6 1 -4 2 4 7 3 -3

How to calculate minimax value?

- Consider reduced tic tac toe game (as game tree for full is too big)
  - The possible moves for MAX at the root are a1, a2, a3
  - Possible replies to a1 for MIN are b1,b2,b3 and so on
- Optimal strategy can be defined using minimax value of each node MINIMAX(n)
  - MINIMAX(n) for the user MAX is the utility of being in corresponding state
  - So, MAX will always move to a state of maxi

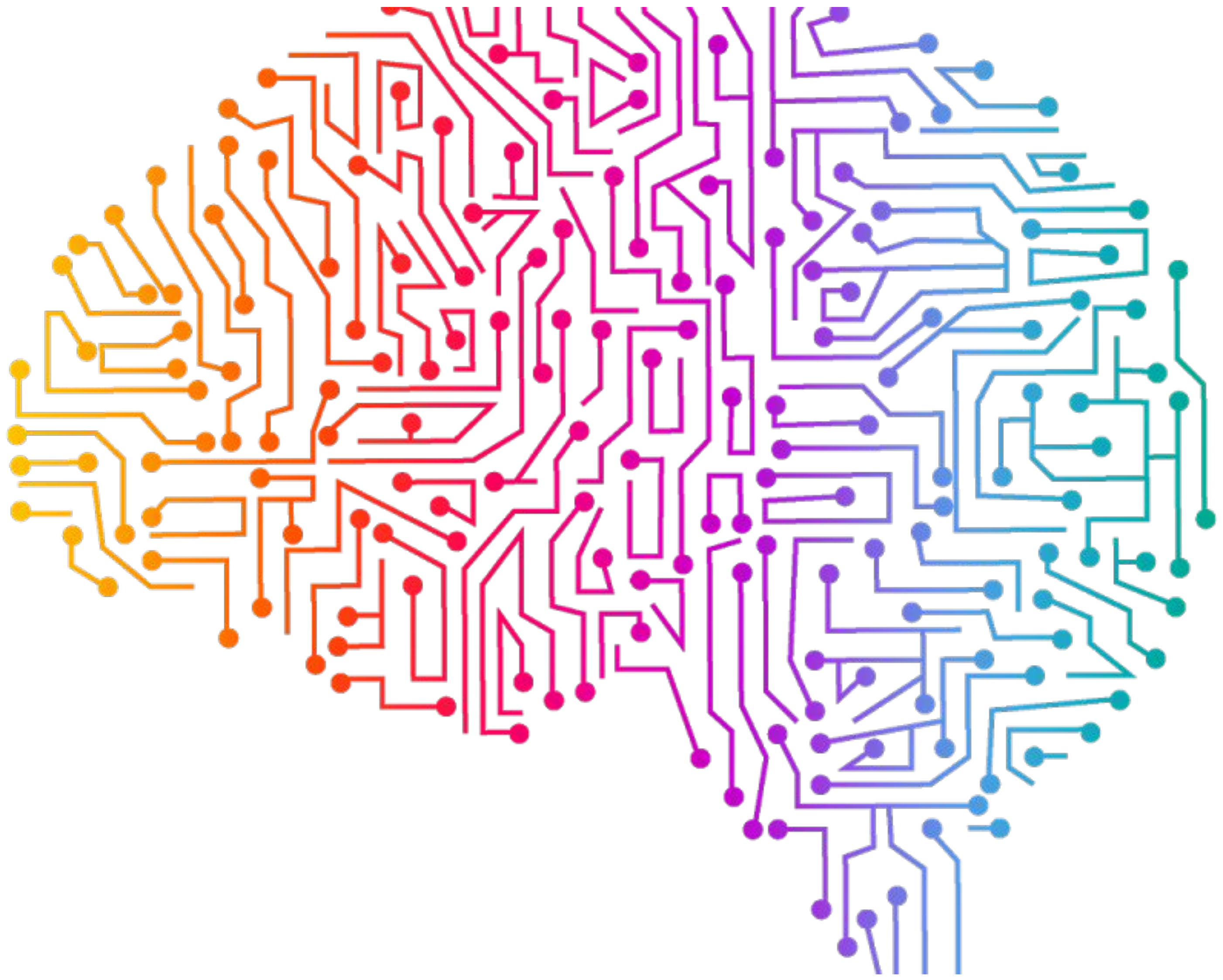
$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$



Limitation of the Minimax Algorithm

- The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc. This type of games has a huge branching factor, and the player has lots of choices to decide. This limitation of the minimax algorithm can be improved from **alpha-beta pruning**.

Alpha-Beta Pruning



Alpha-Beta Pruning

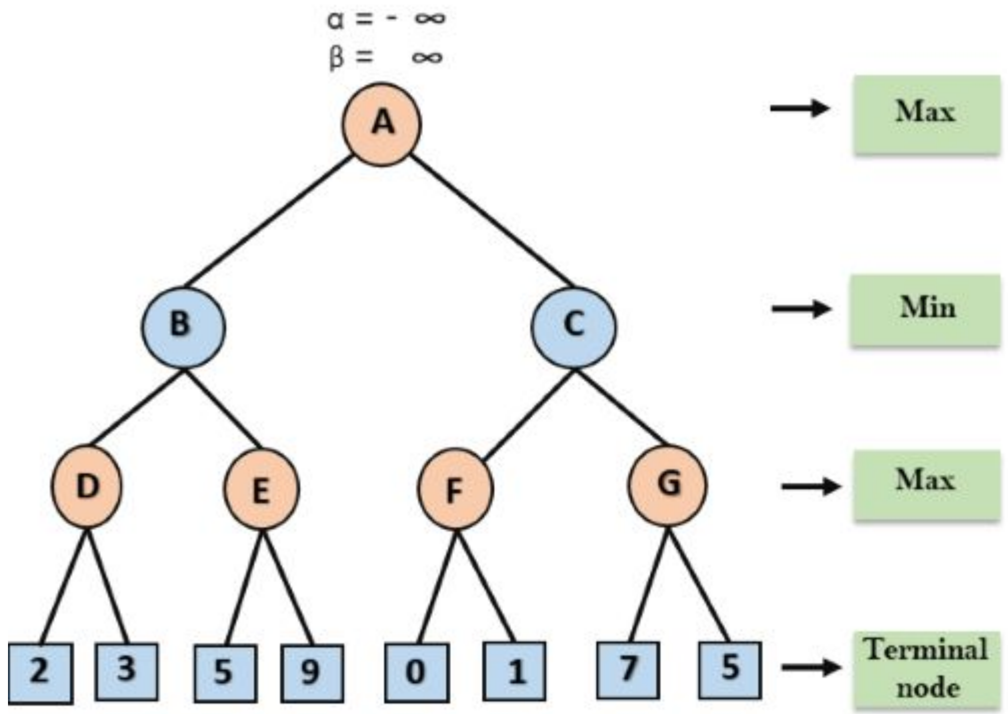
- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision, and this technique is called **pruning**.
- This involves two threshold parameter Alpha and beta for future expansion, so it is called **alpha-beta pruning**. It is also called as **Alpha-Beta Algorithm**.
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.

- The two-parameter can be defined as:
  - **Alpha** is the best value that the maximizer currently can guarantee at that level or above. **The initial value of alpha is -∞.**
  - **Beta** is the best value that the minimizer currently can guarantee at that level or below. **The initial value of beta is +∞.**

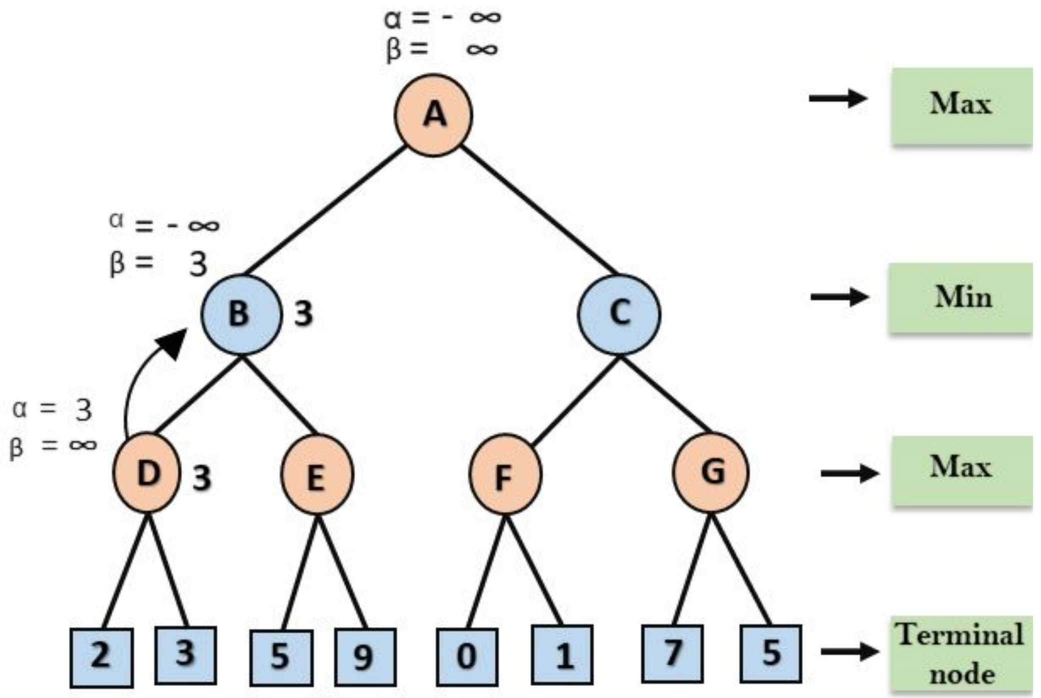
- The Alpha-beta pruning to a standard minimax algorithm returns the same move as the standard algorithm does, but it removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence pruning these nodes makes the algorithm fast.

Working of Alpha-Beta Pruning

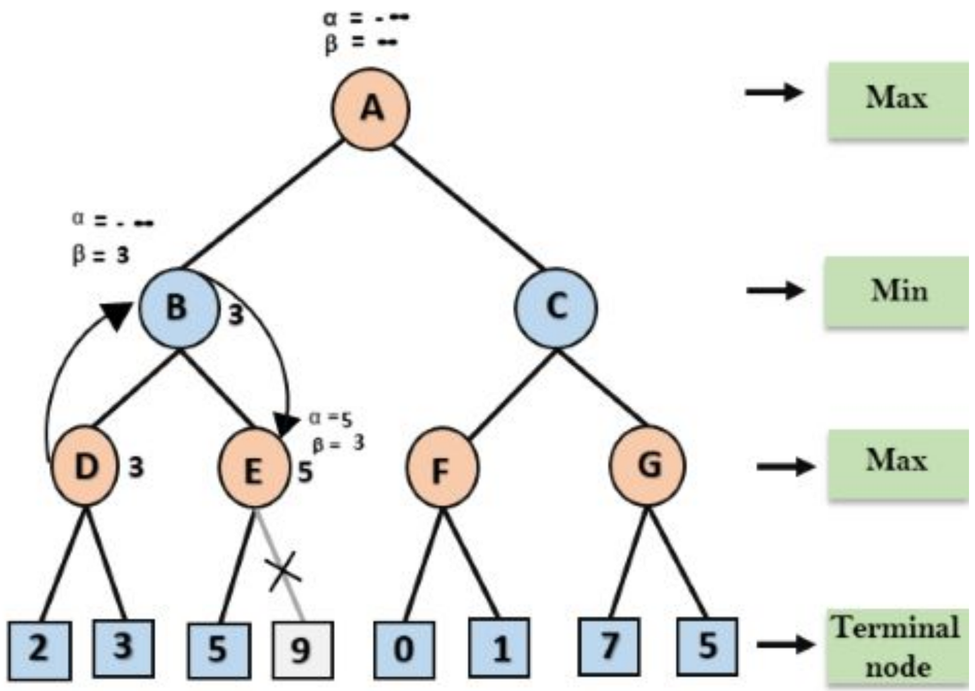
- **Step 1:** At the first step, the Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



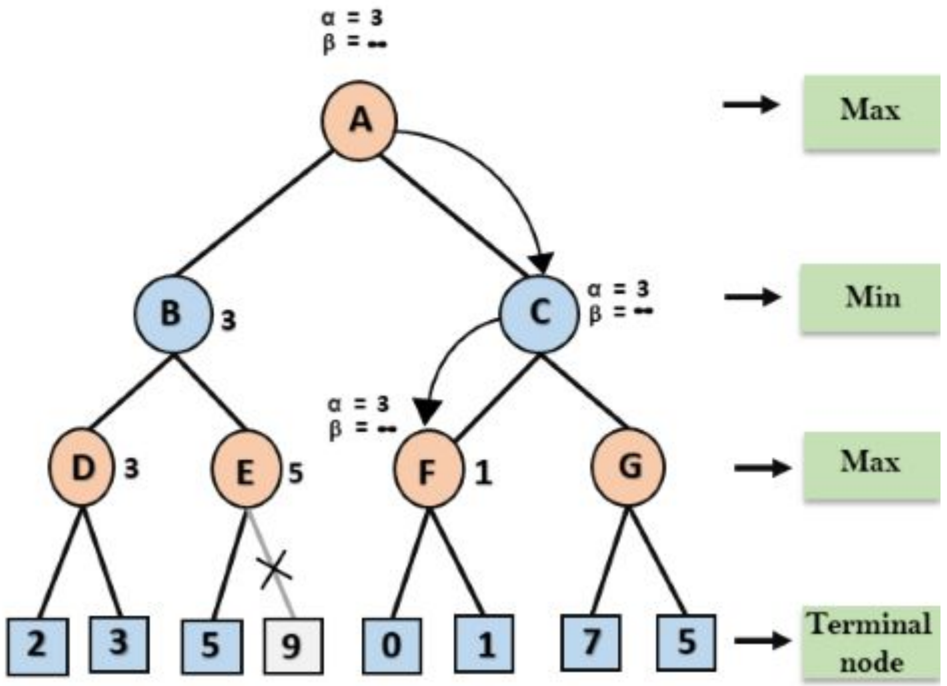
- **Step 2:** At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.
- **Step 3:** Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



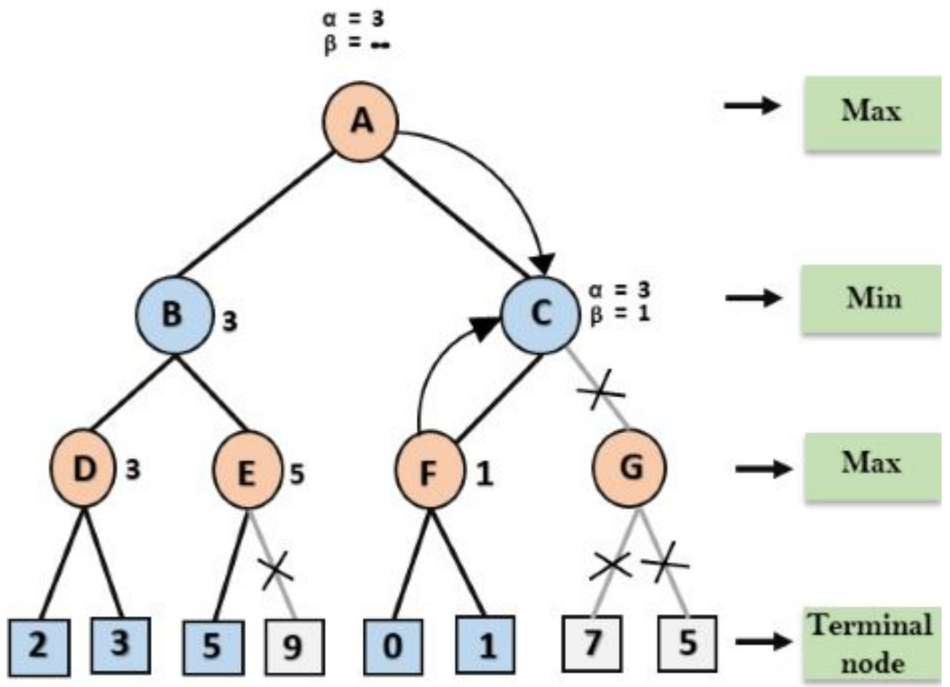
- **Step 4:** In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.
- **Step 5:** At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



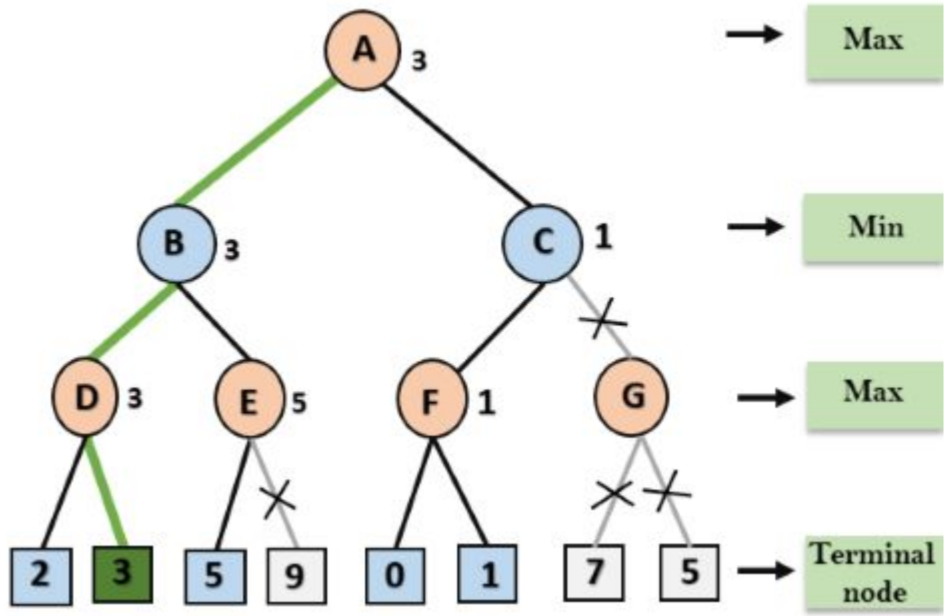
- **Step 6:** At next step, algorithm again backtracks the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.
- At node C, α=3 and β= +∞, and the same values will be passed on to node F.
- **Step 7:** At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



- **Step 8:** Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.

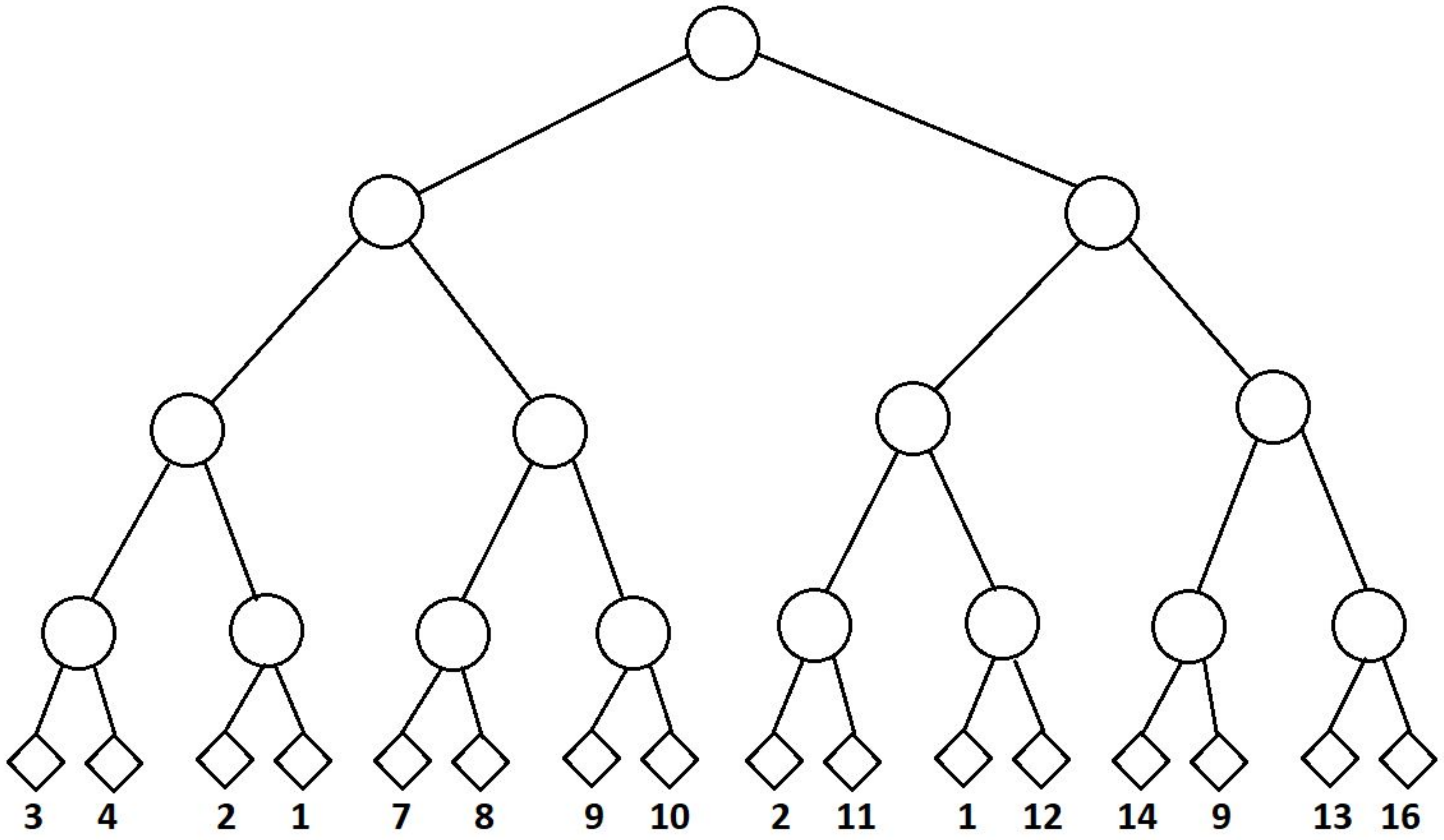


- **Step 9:** C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.

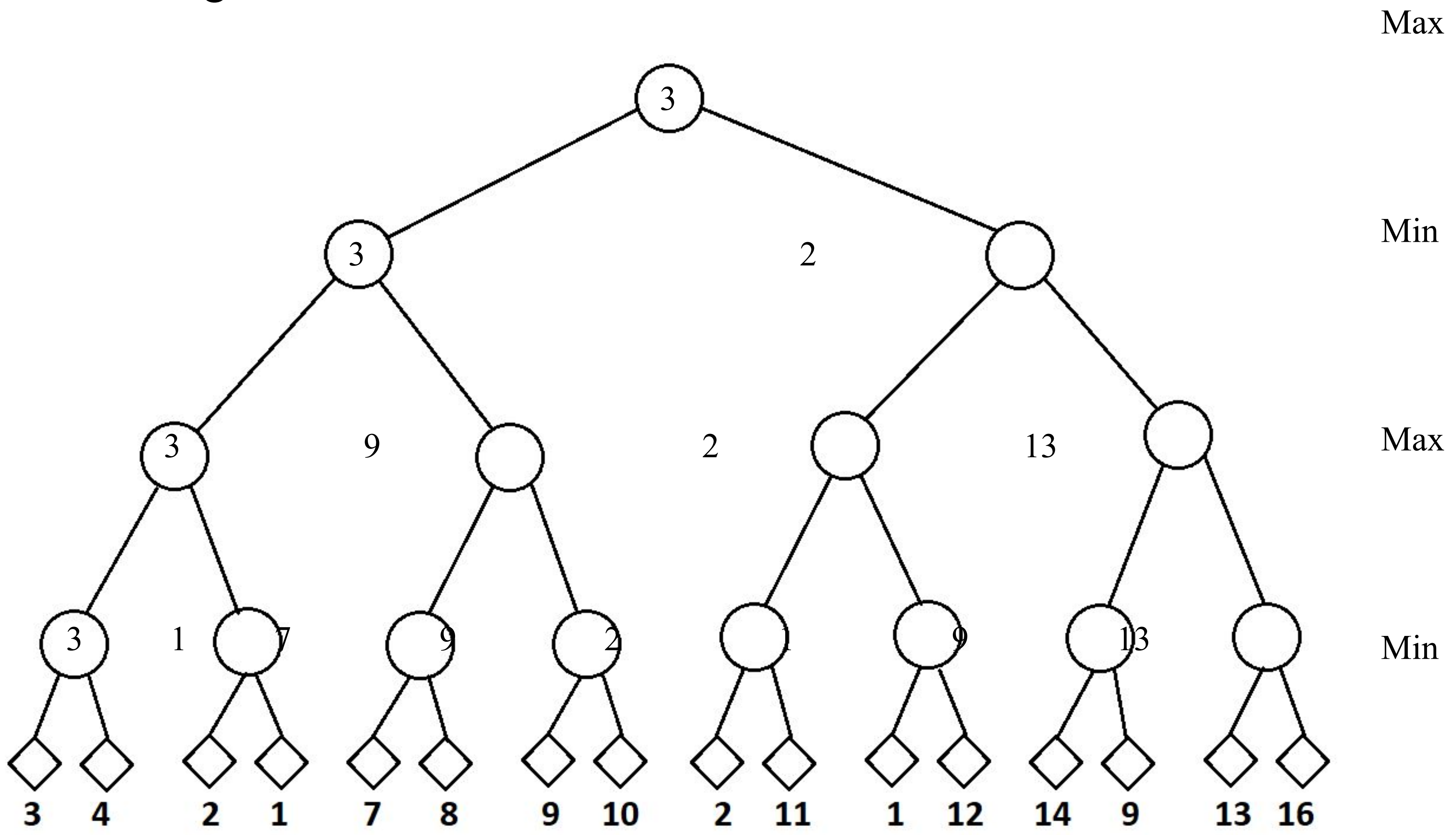


CLASS TASK

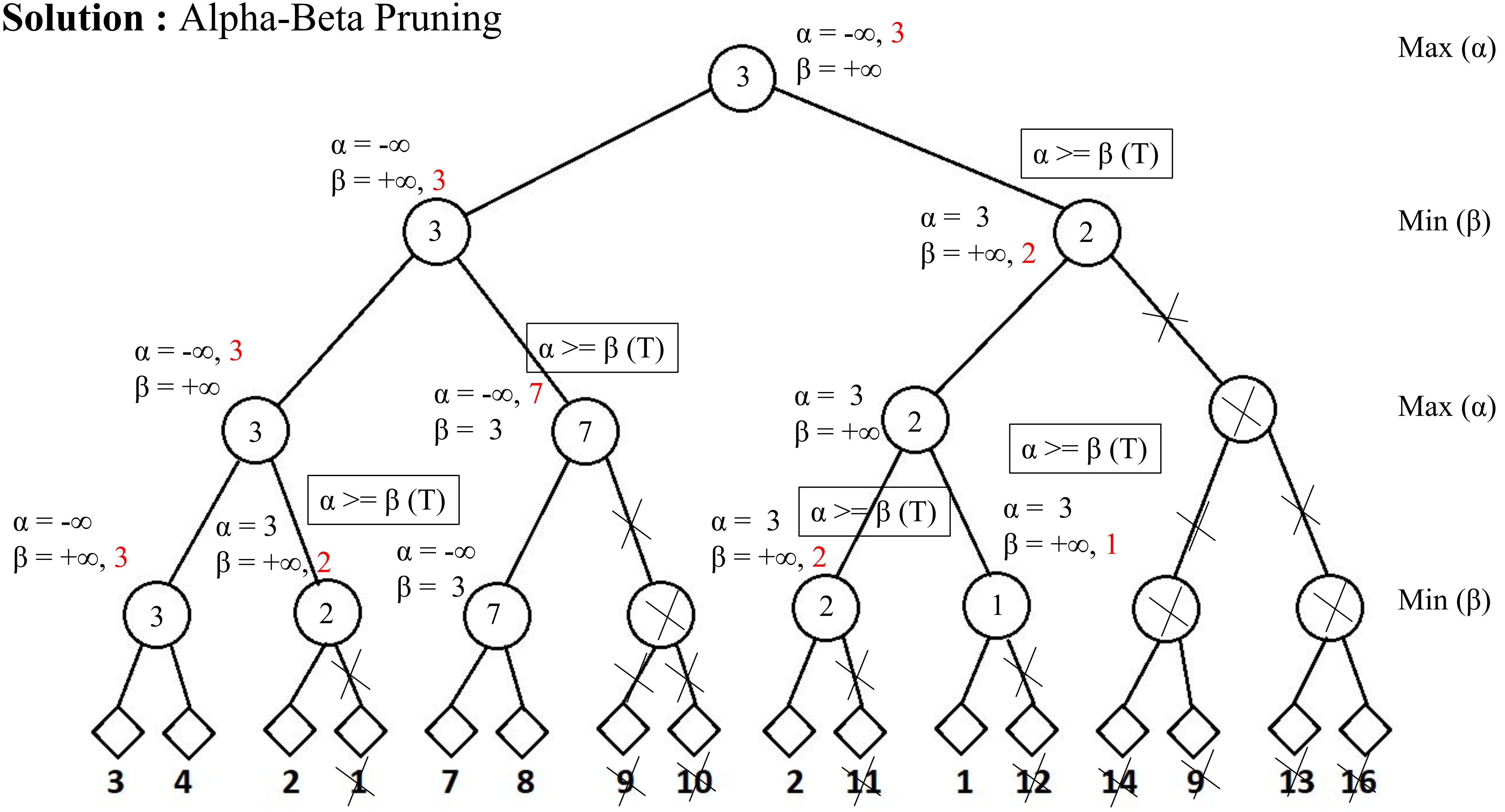
Given the following game tree for a two-player game (Player A is the **Max** and Player B is the **Min**). **(i)** Use the minimax algorithm to determine how the values are propagated up the tree. **(ii)** Using alpha-beta pruning, explain how the algorithm would traverse the tree and identify at which points the pruning occurs. Illustrate your explanation with a diagram of the tree, showing the values of alpha and beta at each node and where the pruning cuts off further exploration of branches."



Solution : MinMax Algorithm



Solution : Alpha-Beta Pruning

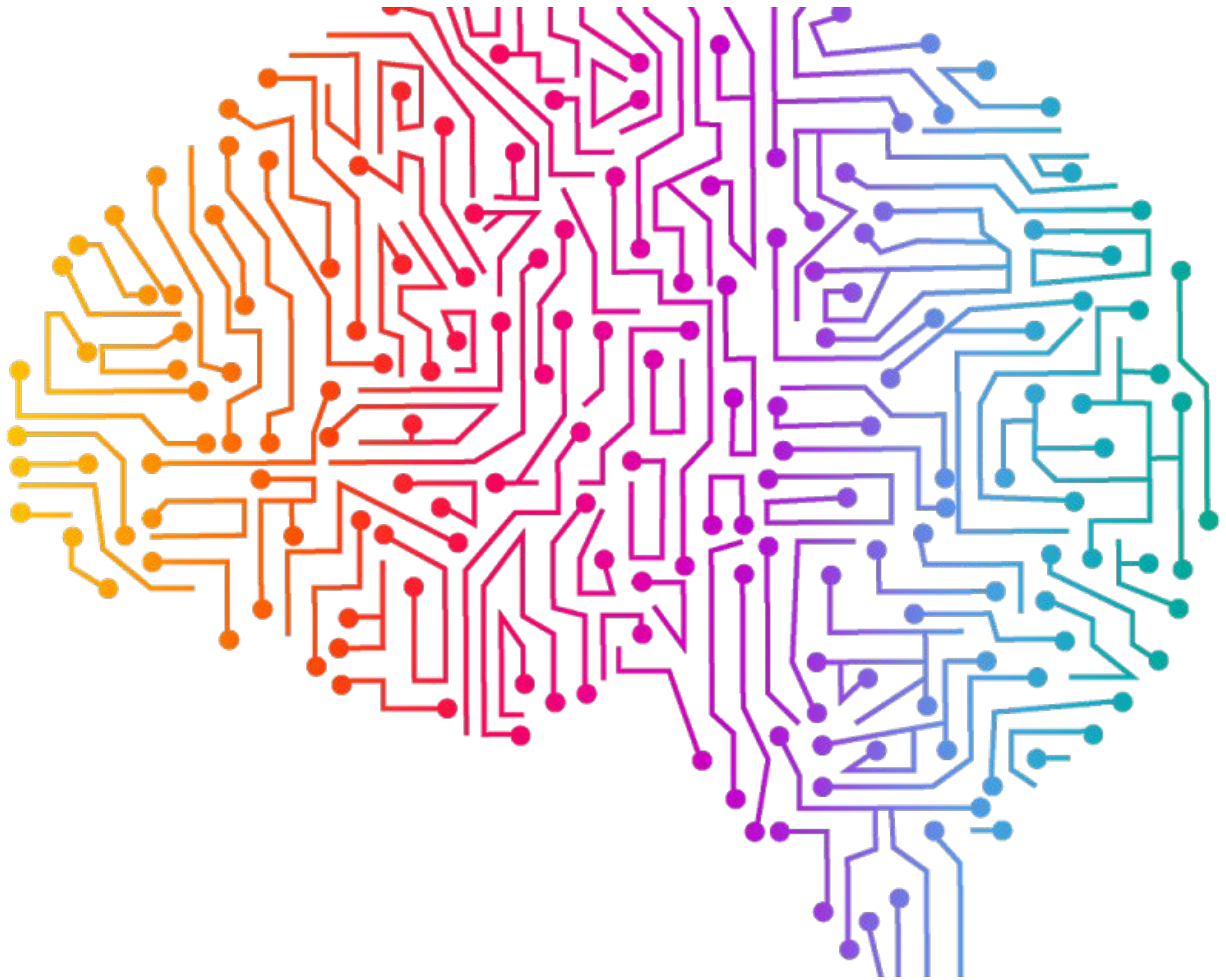


Constraint Satisfaction Problem in AI

Learning Objective of this Topic

- What is the Constraint Satisfaction Problem (CSP)?
- Basic Components of CSP
- Map Example CSP
- Classic CSP Problem: Cryptarithmetic
  - Examples
- CSP Applications

CONSTRAINT SATISFACTION PROBLEM



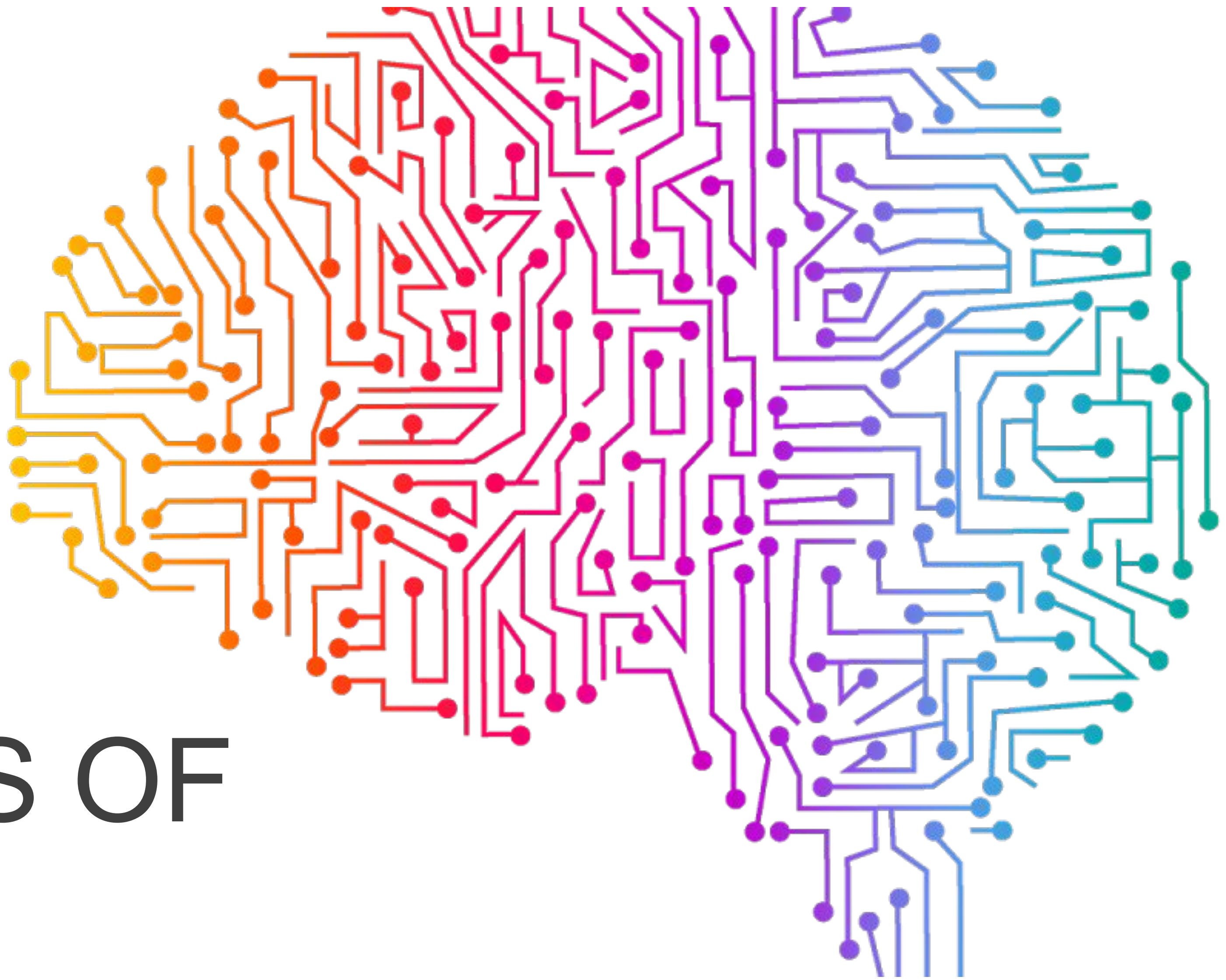
Constraint Satisfaction Problem

In the fields of artificial intelligence (AI) and computer science, an issue has been designated as a constraint satisfaction problem (CSP).

It is described by a set of variables, a domain for each variable, and a set of constraints that outline the possible combinations of values for these variables.

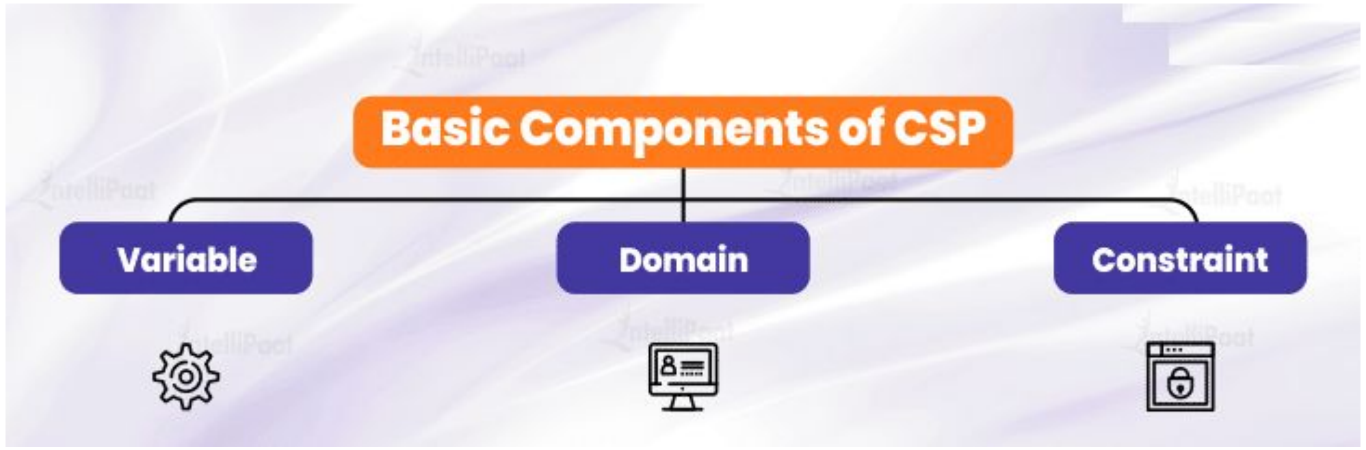
Finding a variable assignment that meets all of the criteria is the main objective of solving a CSP.

The goal of constraint satisfaction problems is to identify values for a collection of variables that satisfy a set of limitations or guidelines.



COMPONENTS OF CSP

Basic Components of CSP

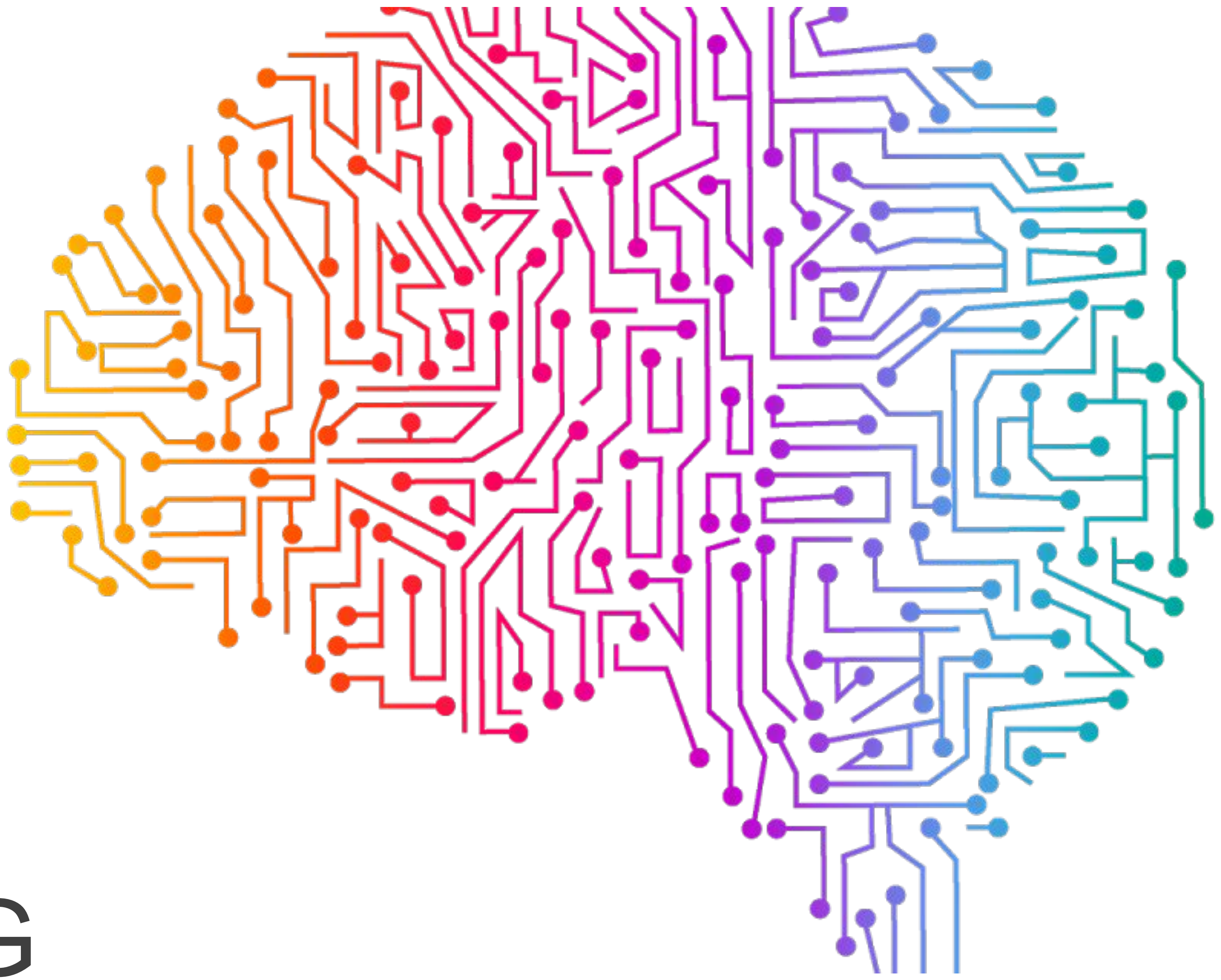


- Variable: Variables are the items that need to be determined. The objects in a CSP that must have values given to them in order to meet a specific set of constraints are known as variables. The set of variables is denoted as {X1,X2,…..,Xn}.

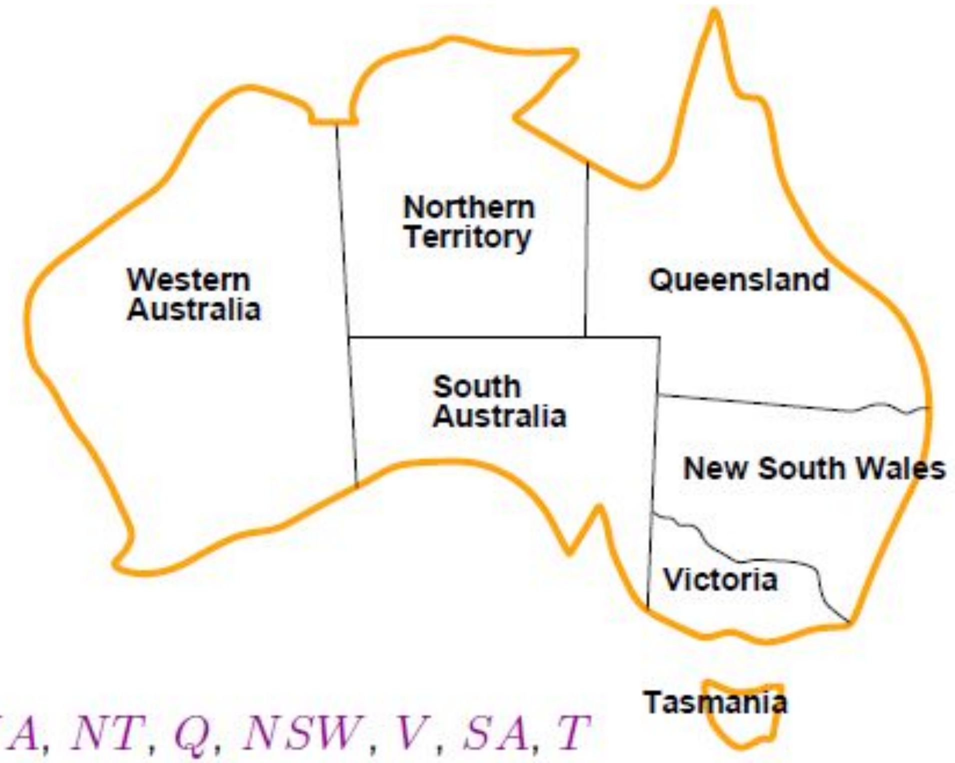
- Domain: Domains describe the variety of possible values that a variable might have. A domain may be finite or limitless, depending on the problem. For example, 12 colors.

- Constraints: Constraints are the rules that control how variables interact with one another.

EXAMPLE:
MAP COLORING



Example: Map-Coloring

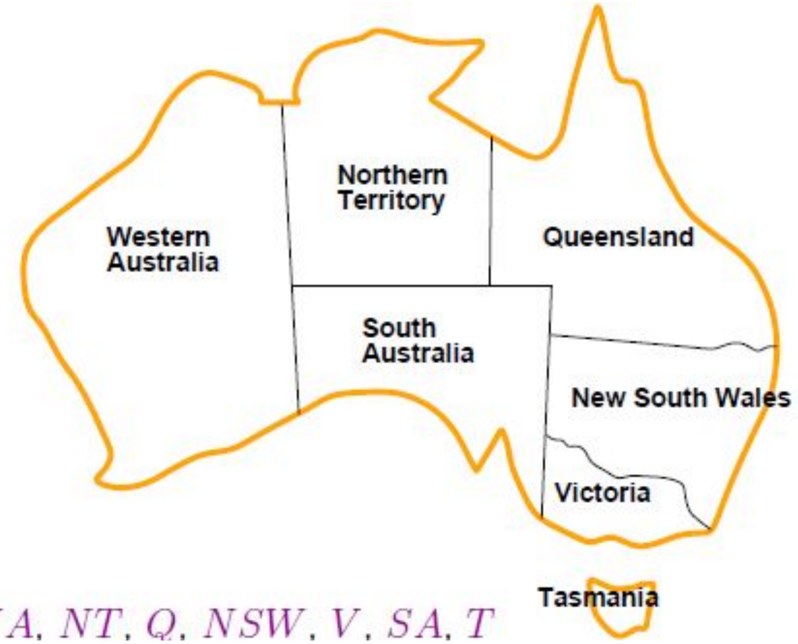


Variables WA, NT, Q, NSW, V, SA, T
Domains $D_i = \{red, green, blue\}$
Constraints: adjacent regions must have different colors
e.g., $WA \neq NT$ (if the language allows this), or
$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

Example: Map-Coloring

- There are nine constraints:

$$C = \{SA \neq WA, SA \neq NT, SA \neq Q, SA \neq NSW, SA \neq V, WA \neq NT, NT \neq Q, Q \neq NSW, NSW \neq V\}.$$

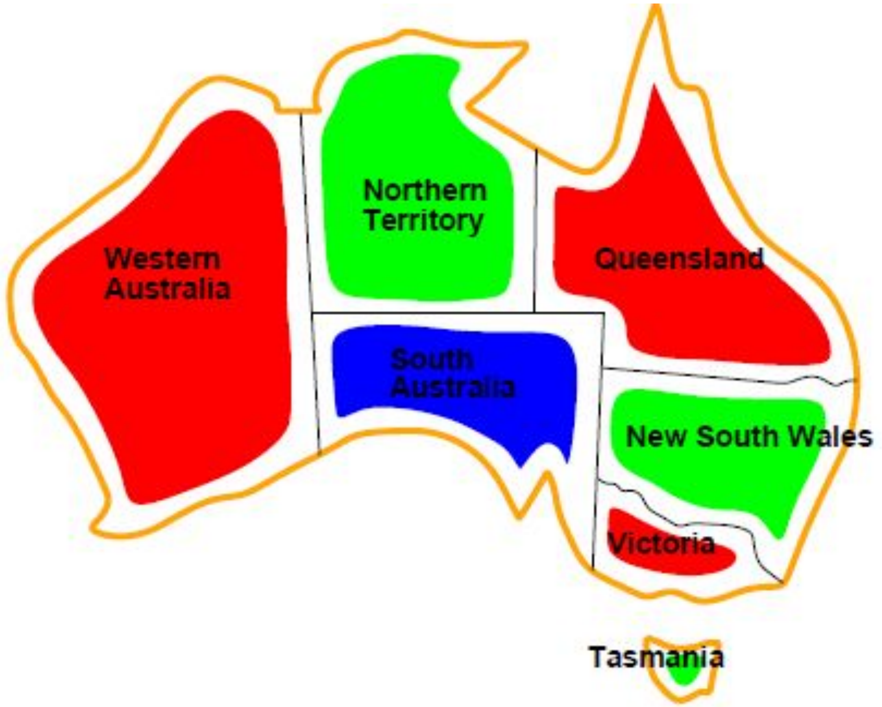


Variables WA, NT, Q, NSW, V, SA, T

- Here we are using abbreviations;
- is a shortcut Where it can be fully enumerated in turn as

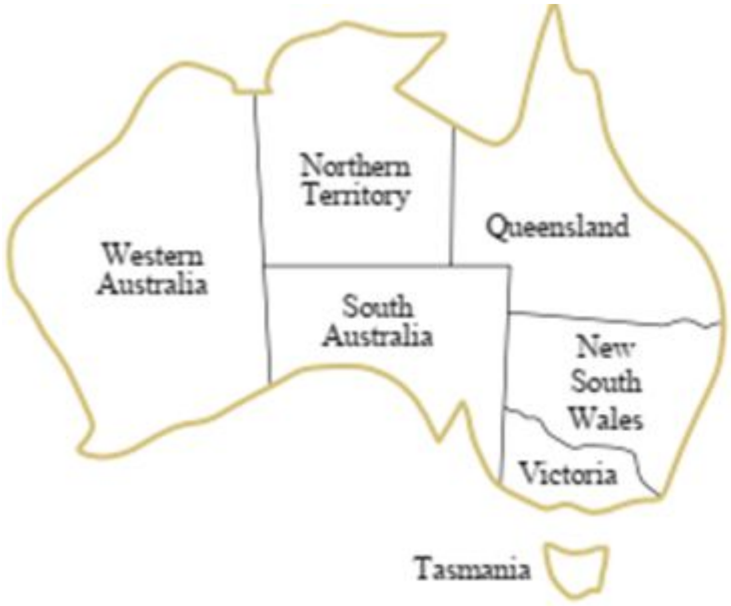
$$SA \neq WA$$

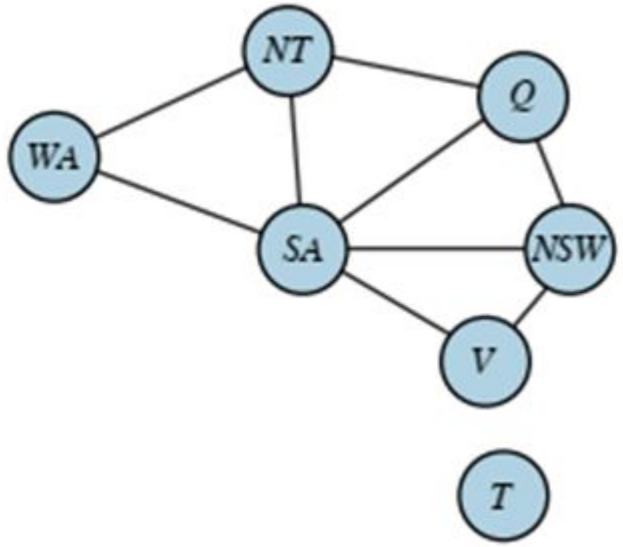
Example: Map-Coloring



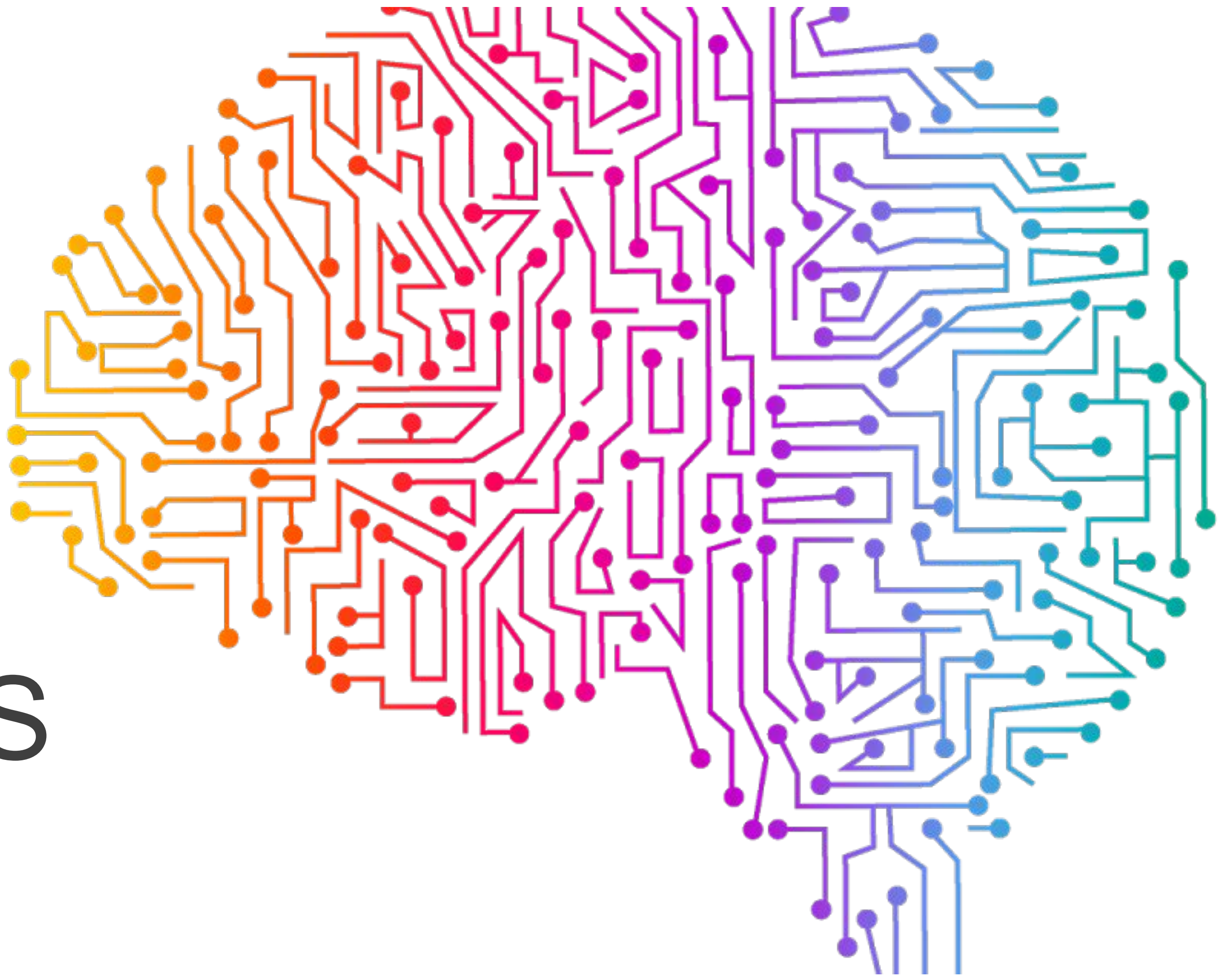
Solutions are assignments satisfying all constraints, e.g.,
$\{WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green\}$

Constraint graphs





APPLICATIONS OF CSP



Real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Hardware configuration
- Spreadsheets
- Transportation scheduling
- Factory scheduling
- Floorplanning

Notice that many real-world problems involve real-valued variables

References

- Russell, S. and Norvig, P. “Artificial Intelligence. A Modern Approach”, 4th ed,
  Prentice Hall, Inc., 2020. **Chapter#05**
- Adapted from http://aima.cs.berkeley.edu/

Questions
?
Answers
?
?