Unit-2
Game Playing

Dr. Sonam Mittal

Associate Professor

Adversarial Search in Artificial Intelligence

- Adversarial search is a **game-playing** technique where the agents are surrounded by a competitive environment.
- A conflicting goal is given to the agents (multi-agent). These agents compete with one another and try to defeat one another in order to win the game. Such conflicting goals give rise to the adversarial search.
- Here, game-playing means discussing those games where **human intelligence** and **logic factor** is used, excluding other factors such as **luck factor**.
- **Tic-tac-toe, chess, checkers**, etc., are such type of games where no luck factor works, only mind works.

Adversarial Search in Artificial Intelligence

- Mathematically, this search is based on the concept of **'Game Theory.'** ***According to game theory, a game is played between two players. To complete the game, one has to win the game and the other looses automatically.'***

Techniques required to get the best optimal solution

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

Pruning: A technique which allows ignoring the unwanted portions of a search tree which make no difference in its final result.

Heuristic Evaluation Function: It allows to approximate the cost value at each level of the search tree, before reaching the goal node.

Elements of Game Playing search

- To play a game, we use a game tree to know all the possible choices and to pick the best one out. **There are following elements of a game-playing:**

S_0: It is the initial state from where a game begins.

PLAYER (s): It defines which player is having the current turn to make a move in the state.

ACTIONS (s): It defines the set of legal moves to be used in a state.

RESULT (s, a): It is a transition model which defines the result of a move.

TERMINAL-TEST (s): It defines that the game has ended and returns true.

UTILITY (s, p): It defines the final value with which the game has ended. This function is also known as **Objective function** or **Payoff function**.

Elements of Game Playing search

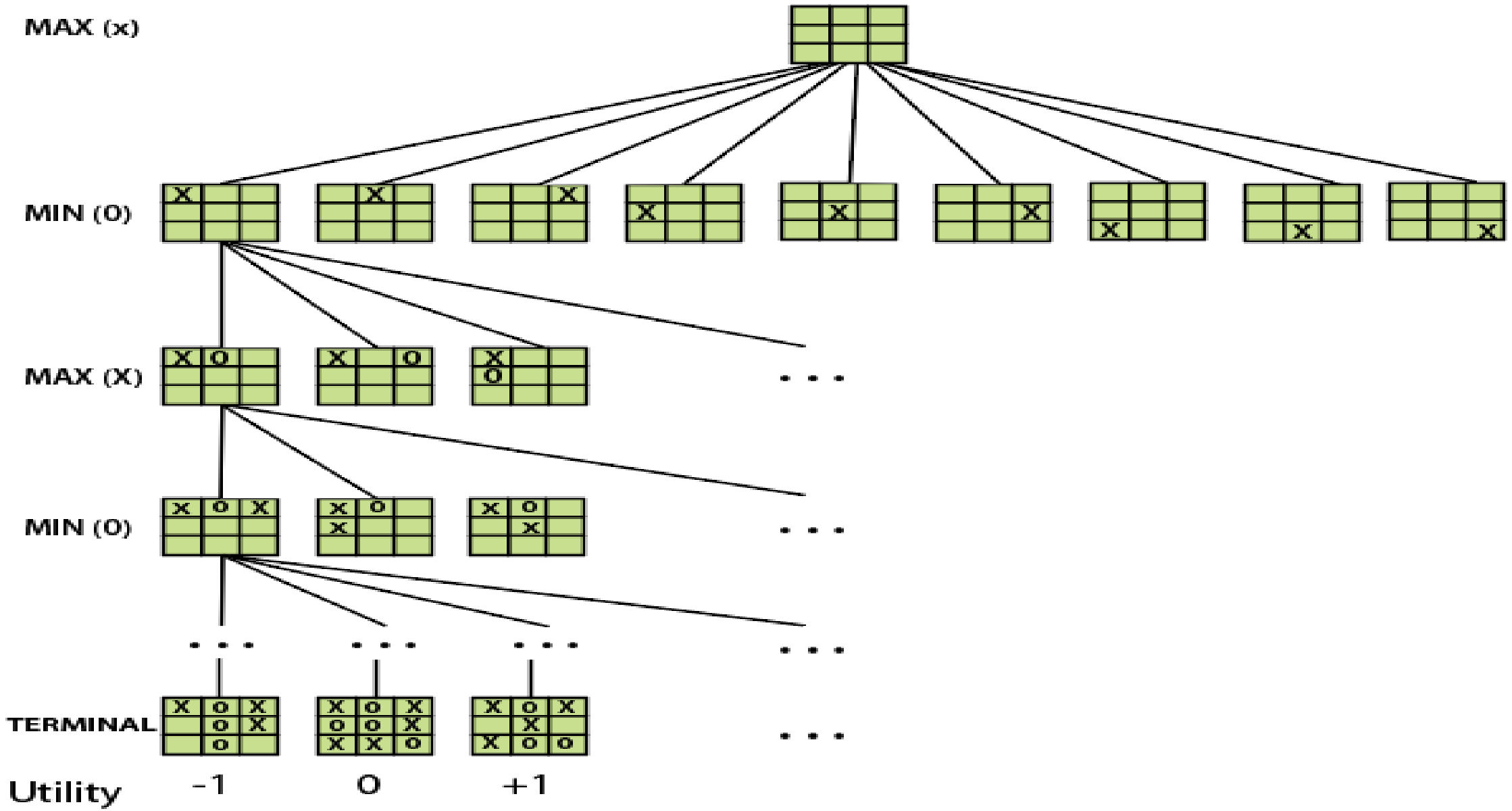
- The price which the winner will get i.e.

(-1): If the PLAYER loses.

(+1): If the PLAYER wins.

(0): If there is a draw between the PLAYERS.

- **For example**, in **chess, tic-tac-toe,** we have two or three possible outcomes. Either to win, to lose, or to draw the match with values **+1,-1 or 0.**
- Eg. a game tree designed for **tic-tac-toe**. Here, the node represents the game state and edges represent the moves taken by the players.


MAX (x)
MIN (O)
MAX (X)
MIN (O)
TERMINAL
Utility
-1
0
+1

- **INITIAL STATE (S_0):** The top node in the game-tree represents the initial state in the tree and shows all the possible choice to pick out one.
- **PLAYER (s):** There are two players, **MAX and MIN**. **MAX** begins the game by picking one best move and place **X** in the empty square box.
- **ACTIONS (s):** Both the players can make moves in the empty boxes chance by chance.
- **RESULT (s, a):** The moves made by **MIN** and **MAX** will decide the outcome of the game.
- **TERMINAL-TEST(s):** When all the empty boxes will be filled, it will be the terminating state of the game.
- **UTILITY:** At the end, we will get to know who wins: **MAX** or **MIN,** and accordingly, the price will be given to them**.**

Example Explanation:

•From the initial state, MAX has 9 possible moves as he starts first. MAX place x and MIN place o, and both player plays alternatively until we reach a leaf node where one player has three in a row or all squares are filled.

•Both players will compute each node, minimax, the minimax value which is the best achievable utility against an optimal adversary.

•Suppose both the players are well aware of the tic-tac-toe and playing the best play. Each player is doing his best to prevent another one from winning. MIN is acting against Max in the game.

•So in the game tree, we have a layer of Max, a layer of MIN, and each layer is called as **Ply**. Max place x, then MIN puts o to prevent Max from winning, and this game continues until the terminal node.

•In this either MIN wins, MAX wins, or it's a draw. This game-tree is the whole search space of possibilities that MIN and MAX are playing tic-tac-toe and taking turns alternately.

Types of algorithms in Adversarial search

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

There are following types of adversarial search:

- **Minmax Algorithm**
- **Alpha-beta Pruning.**

Minimax Strategy

- In artificial intelligence, minimax is a **decision-making** strategy under **game theory,** which is used to minimize the losing chances in a game and to maximize the winning chances.
- This strategy is also known as '**Minmax,' 'MM,' or 'Saddle point.'**
- Basically, it is a two-player game strategy where *if one wins, the other loose the game*.
- This strategy simulates those games that we play in our day-to-day life. Like, if two persons are playing chess, the result will be in favor of one player and will unfavor the other one.
- The person who will make his best *try,efforts as well as cleverness, will surely win.*
- We can easily understand this strategy via **game tree**– where the *nodes represent the states of the game and edges represent the moves made by the players in the game*. **Players will be two namely:**

MIN: Decrease the chances of **MAX** to win the game.

MAX: Increases his chances of winning the game.

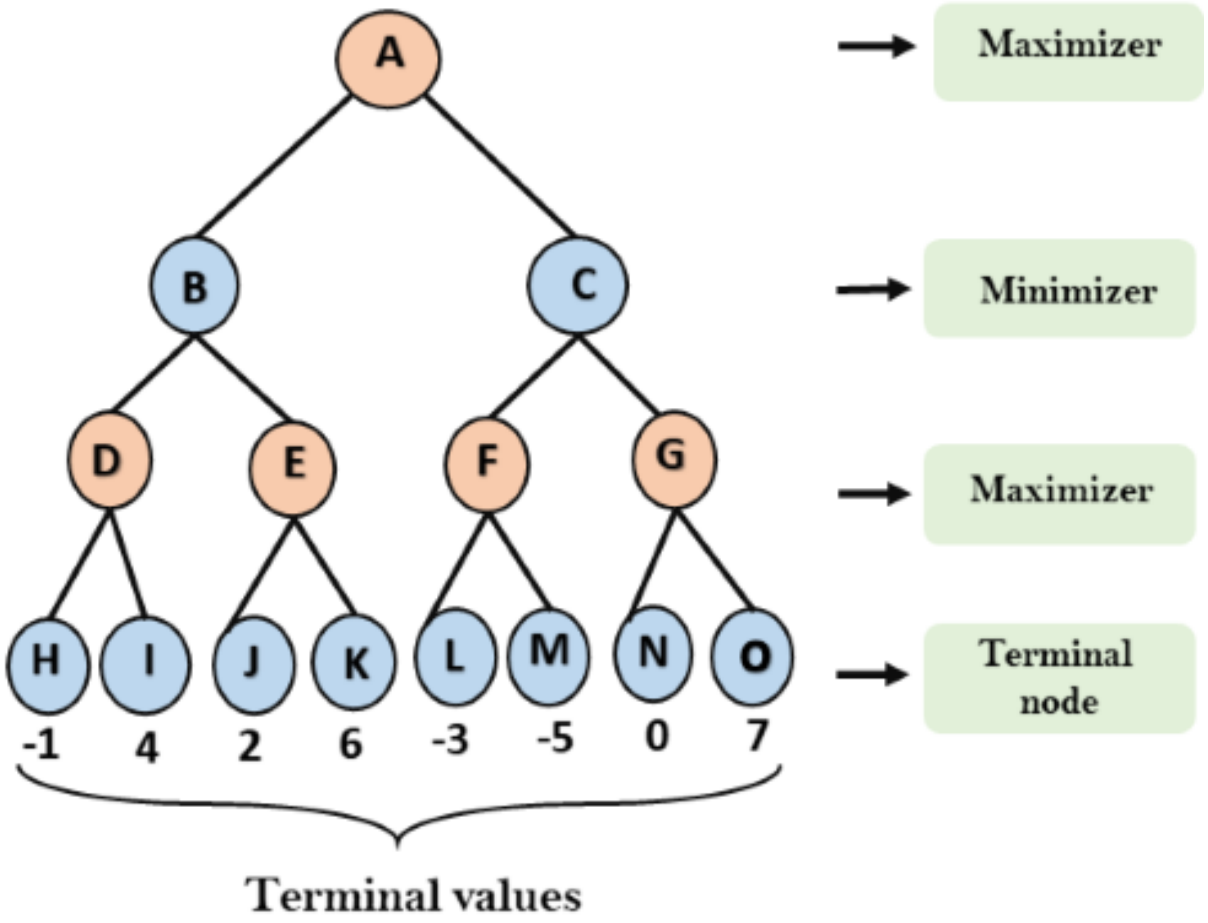
- They both play the game alternatively, i.e., turn by turn and following the above strategy, i.e., if one wins, the other will definitely lose it.
- Both players look at one another as competitors and will try to defeat one-another, giving their best.
- In minimax strategy, the result of the game or the utility value is generated by a **heuristic function** by propagating from the initial node to the root node.
- It follows the **backtracking technique** and backtracks to find the best choice.
- MAX will choose that path which will increase its utility value and MIN will choose the opposite path which could help it to minimize MAX's utility value.

Working of Min-Max Algorithm

- The working of the minimax algorithm can be easily described using an example.
- Below we have taken an example of game-tree which is representing the two-player game.
- **In this example**, there are two players one is called Maximizer and 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.

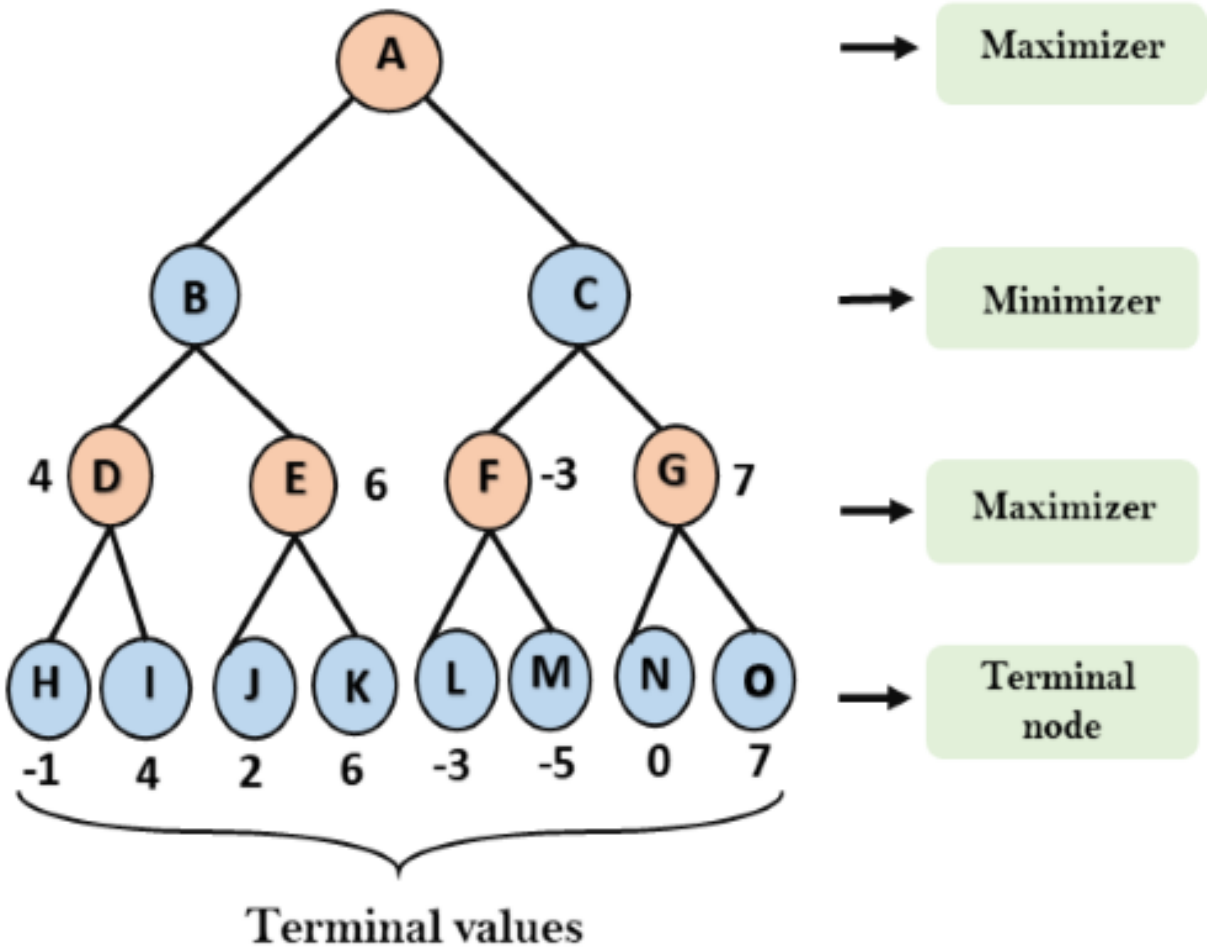
•Following are the main steps involved in solving the two-player game tree:

Step-1: In the first step, the algorithm generates the entire game-tree and apply 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.



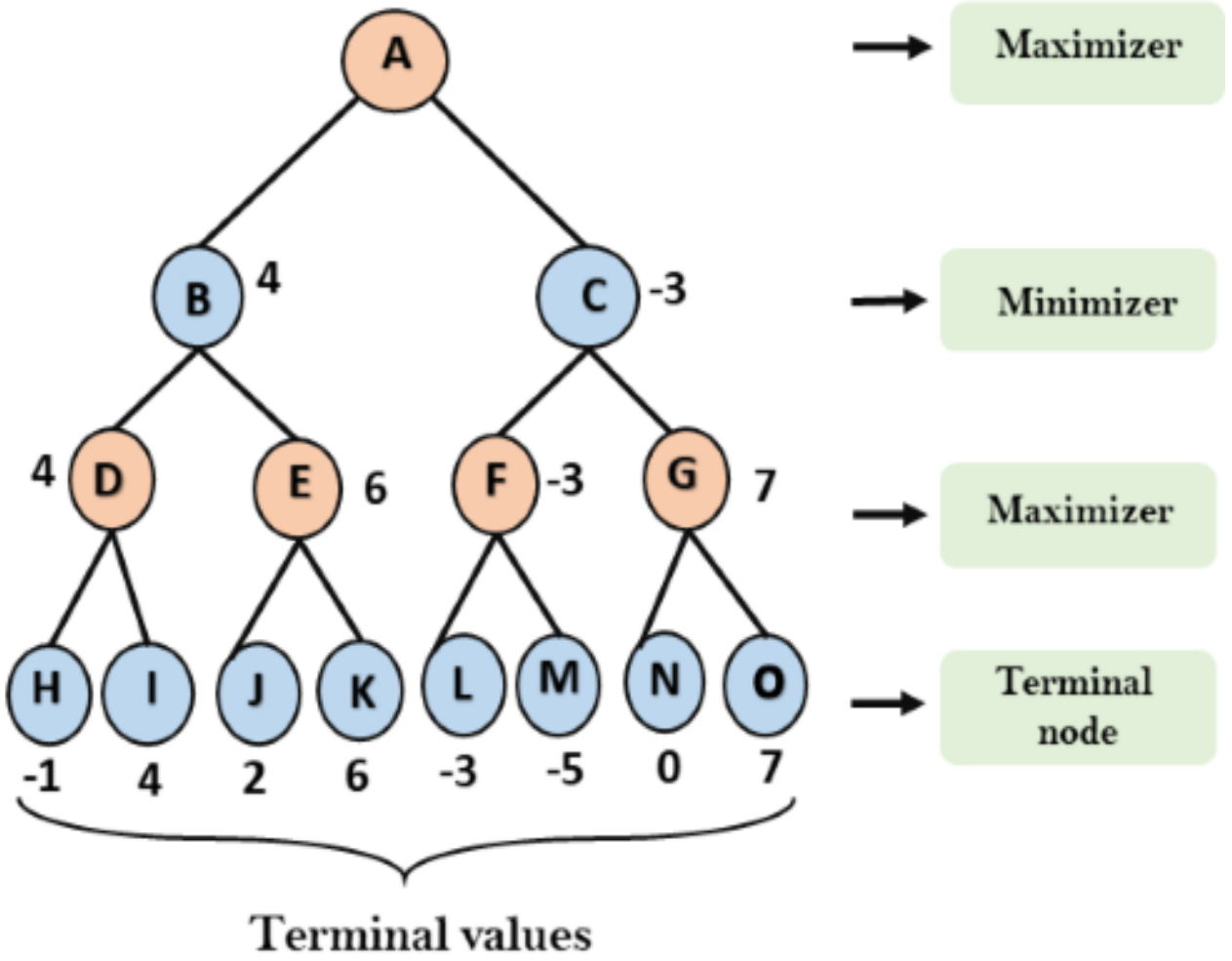
Step 2: Now, first we find the utilities value for the Maximizer, so we will compare each value in terminal state and determines the higher nodes values. It will find the maximum among the all.

- For node D max(-1,4)= 4
- For Node E max(2, 6)= 6
- For Node F max(-3,-5) = -3
- For node G max(0, 7) = 7



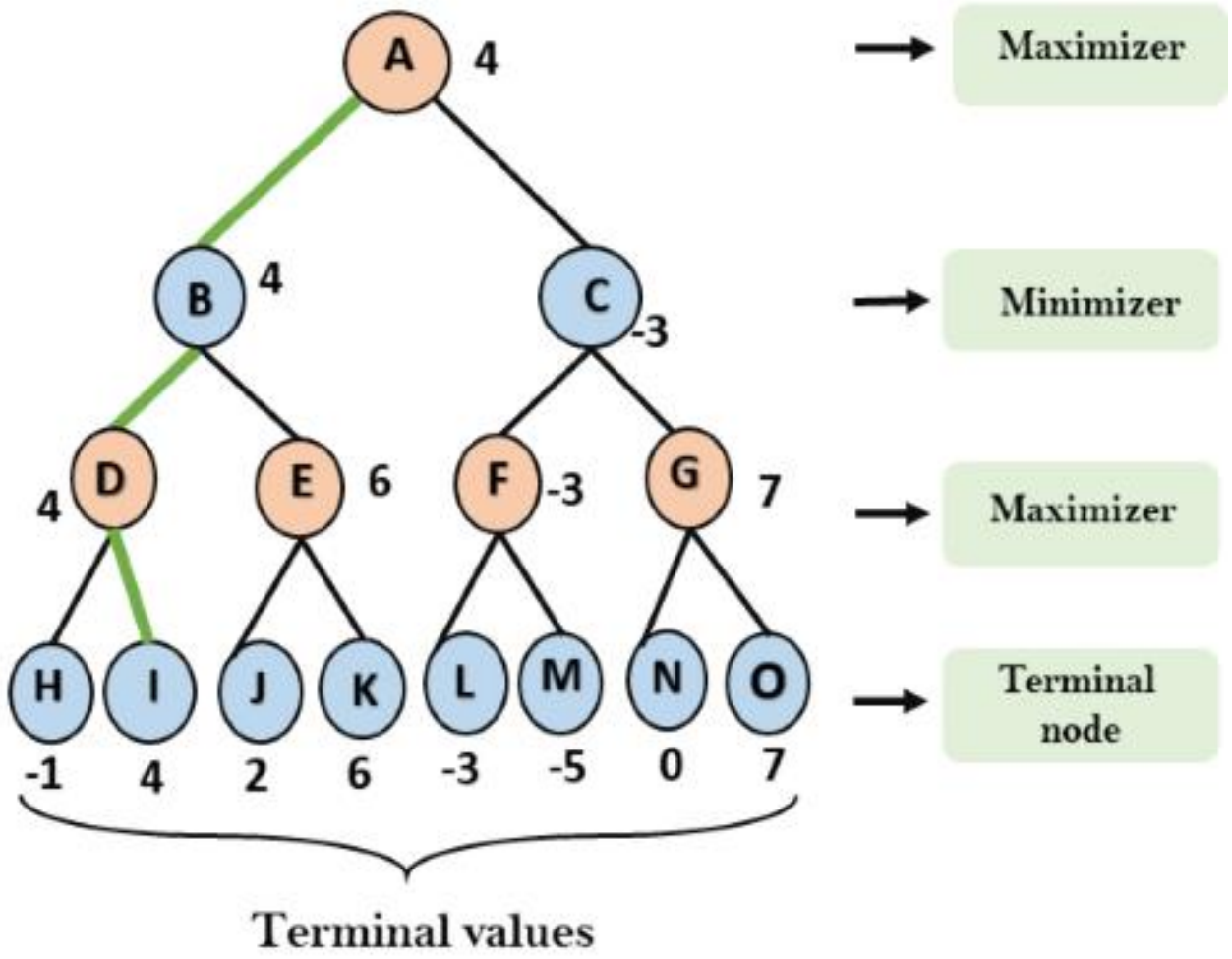
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value and choose the minimum value.

•For node B= min(4,6) = 4
•For node C= min (-3, 7) = -3



Step 3: 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, -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(bm)**.

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 is an advance version of MINIMAX algorithm.
- The drawback of minimax strategy is that it explores each node in the tree deeply to provide the best path among all the paths. This increases its time complexity.
- But as we know, the performance measure is the first consideration for any optimal algorithm. Therefore, alpha-beta pruning reduces this drawback of minimax strategy by less exploring the nodes of the search tree.
- The method used in alpha-beta pruning is that it **cutoff the search** by exploring less number of nodes. It makes the same moves as a minimax algorithm does, but it prunes the unwanted branches using the pruning technique.

- **Alpha-beta pruning works on two threshold values**, i.e., **α (alpha)** and **β (beta).**

α: It is the best highest value, a **MAX** player can have. It is the lower bound, which represents negative infinity value(**-∞**).

β: It is the best lowest value, a **MIN** player can have. It is the upper bound which represents positive infinity(**+∞**).

- So, each MAX node has α-value, which never decreases, and each MIN node has β-value, which never increases.

Note: Alpha-beta pruning technique can be applied to trees of any depth, and it is possible to prune the entire subtrees easily.

Condition for Alpha-beta pruning:

The main condition which required for alpha-beta pruning is:

$$\alpha >= \beta$$

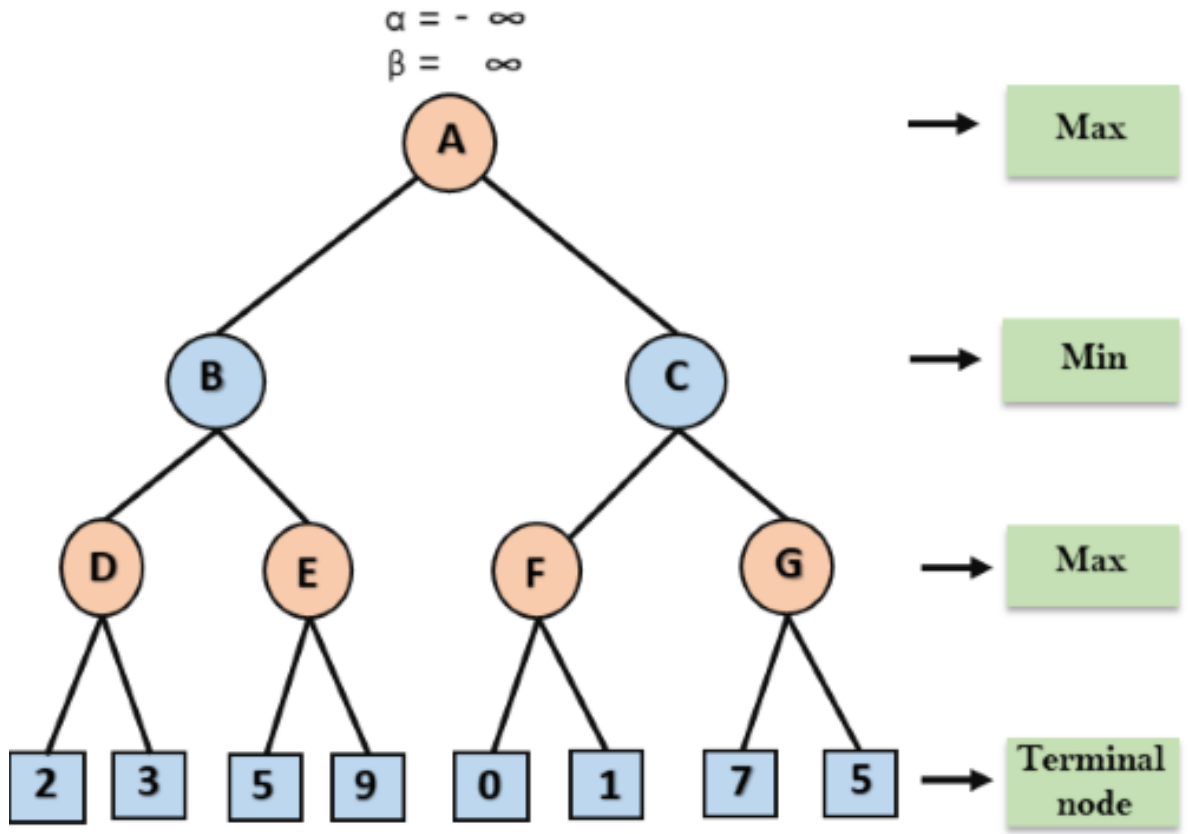
Key points about alpha-beta pruning:

- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

Working of Alpha-Beta Pruning:

Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

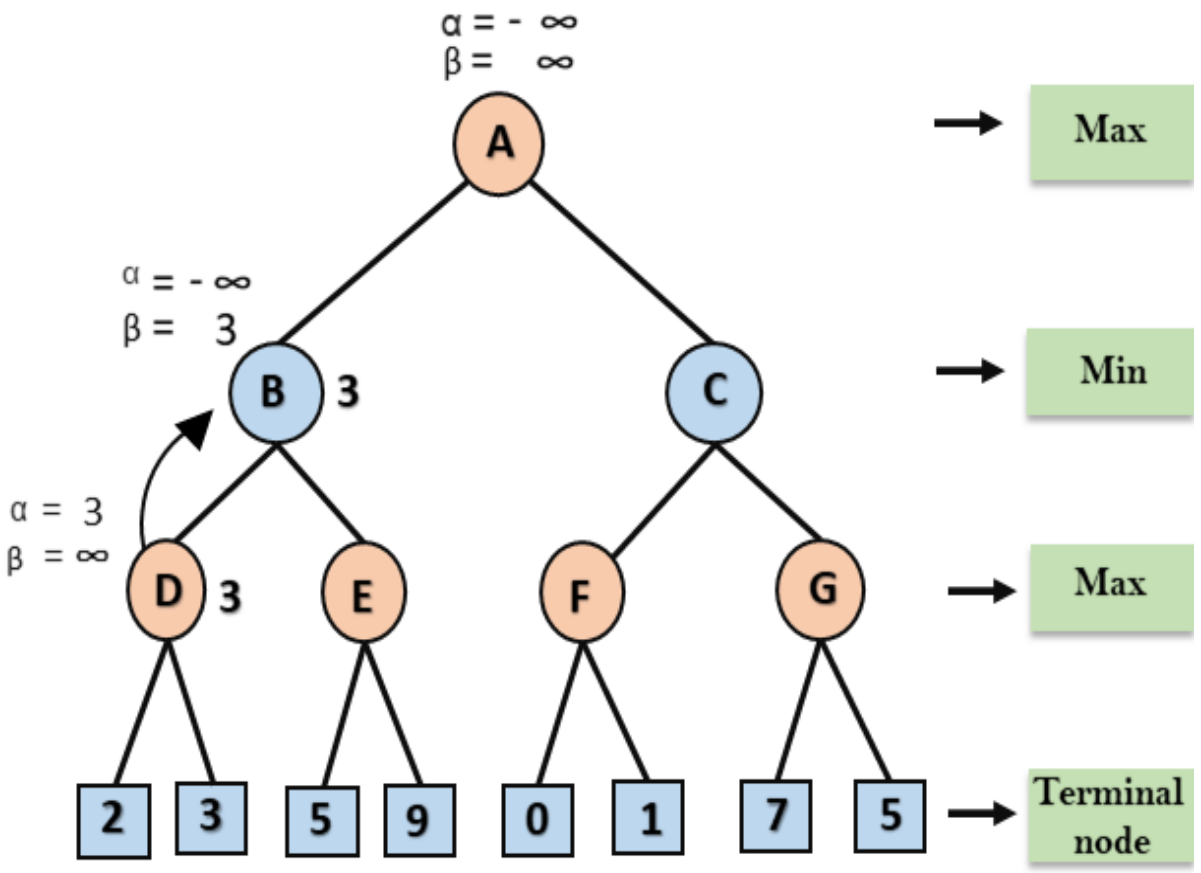
Step 1: At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



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.

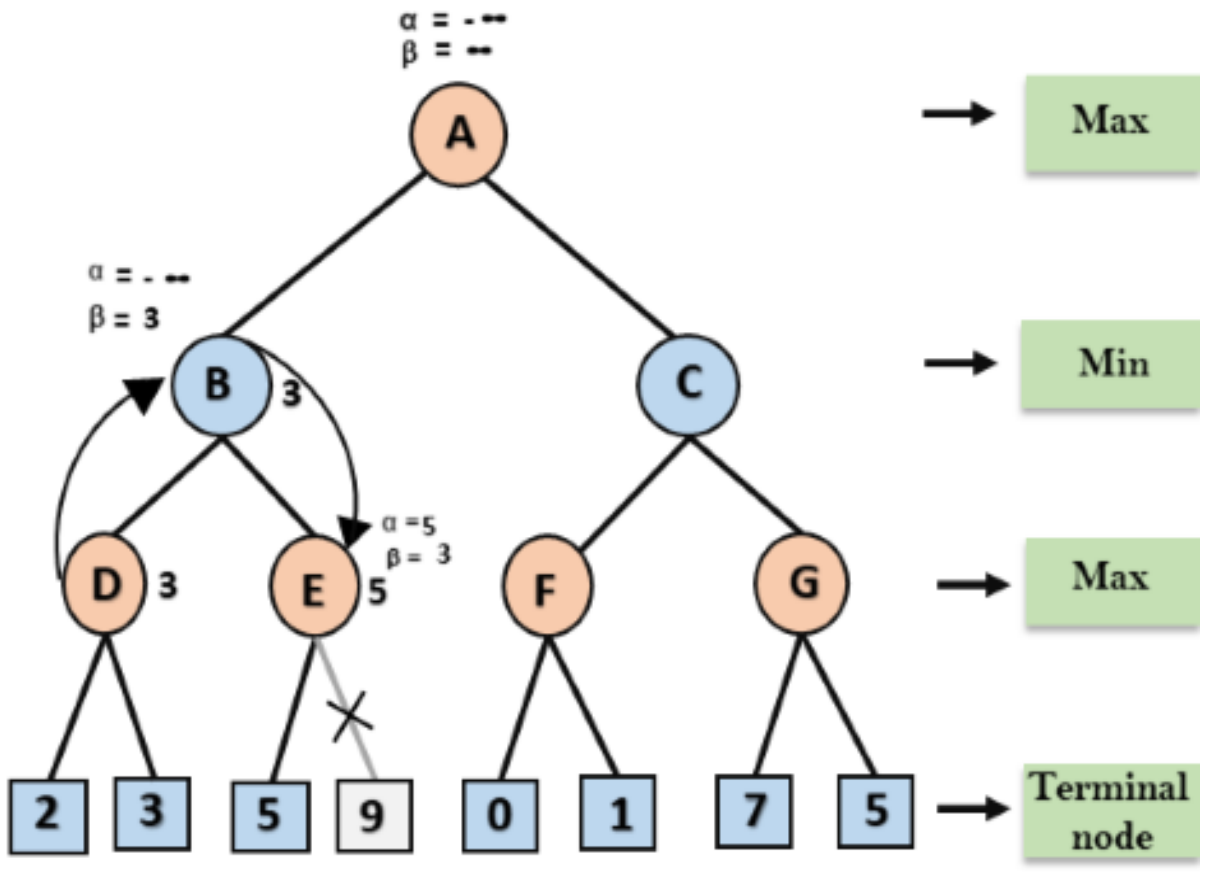
Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.

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.

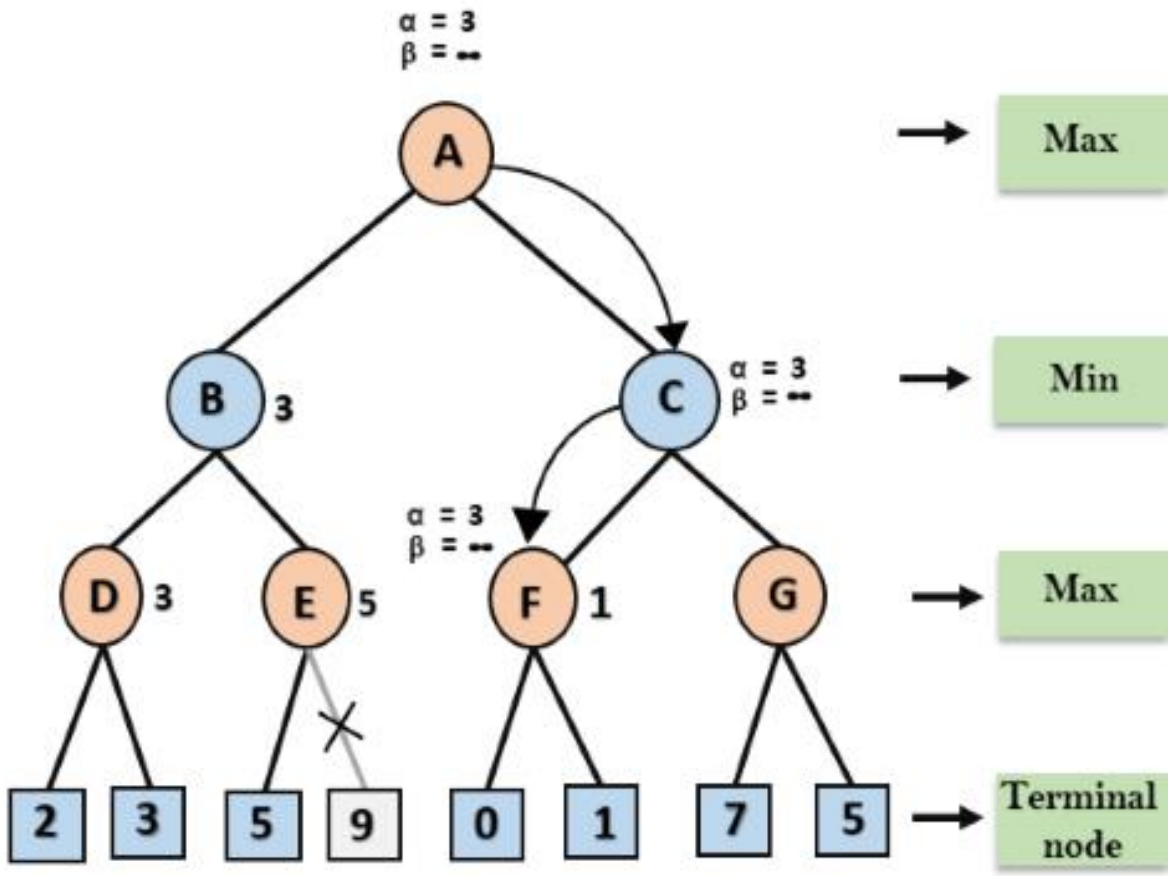


Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will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

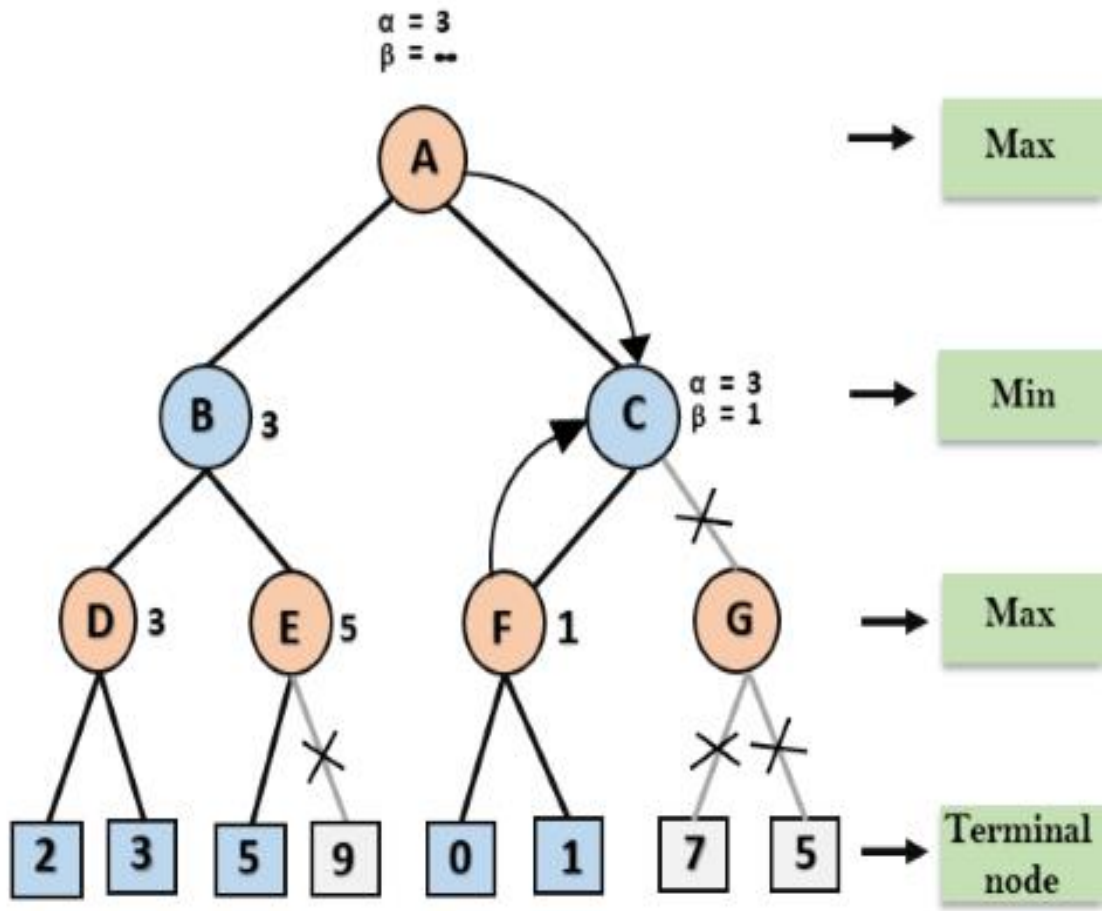
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.
At node C, α=3 and β= +∞, and the same values will be passed on to node F.



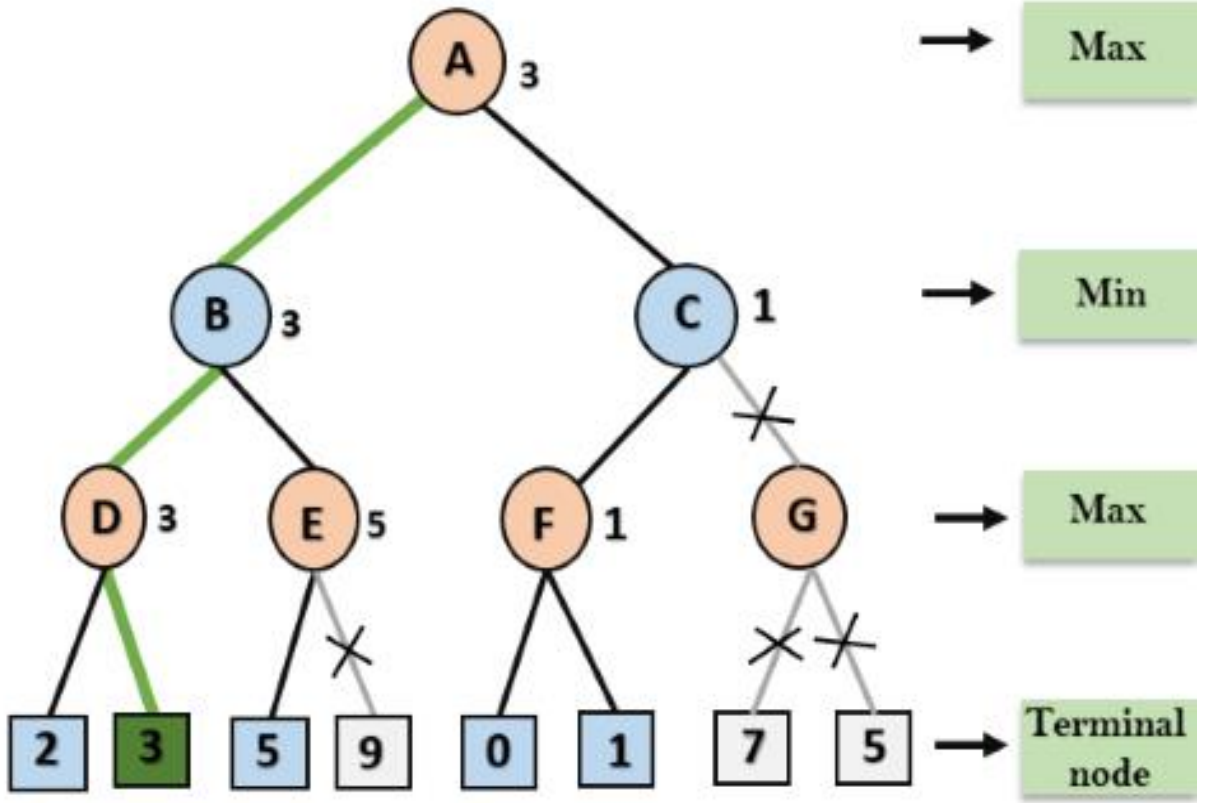
Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



Step 7: 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 8: 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.



Move Ordering in Alpha-Beta pruning

The effectiveness of alpha-beta pruning is highly dependent on the order in which each node is examined. Move order is an important aspect of alpha-beta pruning.
It can be of two types:

- **Worst ordering:** In some cases, alpha-beta pruning algorithm does not prune any of the leaves of the tree, and works exactly as minimax algorithm. In this case, it also consumes more time because of alpha-beta factors, such a move of pruning is called worst ordering. In this case, the best move occurs on the right side of the tree. The time complexity for such an order is $O(b^m)$.
- **Ideal ordering:** The ideal ordering for alpha-beta pruning occurs when lots of pruning happens in the tree, and best moves occur at the left side of the tree. We apply DFS hence it first search left of the tree and go deep twice as minimax algorithm in the same amount of time. Complexity in ideal ordering is $O(b^{m/2})$.

Water jug problem in Artificial Intelligence

In the **water jug problem in Artificial Intelligence**, we are provided with two jugs: one having the capacity to hold 3 gallons of water and the other has the capacity to hold 4 gallons of water. There is no other measuring equipment available and the jugs also do not have any kind of marking on them. So, the agent's task here is to fill the 4-gallon jug with 2 gallons of water by using only these two jugs and no other material. Initially, both our jugs are empty.

So, to solve this problem, following set of rules were proposed:

Production rules for solving the water jug problem

Here, let ***x*** denote the 4-gallon jug and ***y*** denote the 3-gallon jug.

S.No.	Initial State	Condition	Final state	Description of action taken
1.	(x,y)	If x<4	(4,y)	Fill the 4 gallon jug completely
2.	(x,y)	if y<3	(x,3)	Fill the 3 gallon jug completely
3.	(x,y)	If x>0	(x-d,y)	Pour some part from the 4 gallon jug
4.	(x,y)	If y>0	(x,y-d)	Pour some part from the 3 gallon jug
5.	(x,y)	If x>0	(0,y)	Empty the 4 gallon jug
6.	(x,y)	If y>0	(x,0)	Empty the 3 gallon jug
7.	(x,y)	If (x+y)<7	(4, y-[4-x])	Pour some water from the 3 gallon jug to fill the four gallon jug
8.	(x,y)	If (x+y)<7	(x-[3-y],y)	Pour some water from the 4 gallon jug to fill the 3 gallon jug.
9.	(x,y)	If (x+y)<4	(x+y,0)	Pour all water from 3 gallon jug to the 4 gallon jug
10.	(x,y)	if (x+y)<3	(0, x+y)	Pour all water from the 4 gallon jug to the 3 gallon jug

Solution of water jug problem according to the production rules:

S.No.	4 gallon jug contents	3 gallon jug contents	Rule followed
1.	0 gallon	0 gallon	Initial state
2.	0 gallon	3 gallons	Rule no.2
3.	3 gallons	0 gallon	Rule no. 9
4.	3 gallons	3 gallons	Rule no. 2
5.	4 gallons	2 gallons	Rule no. 7
6.	0 gallon	2 gallons	Rule no. 5
7.	2 gallons	0 gallon	Rule no. 9

8 Puzzle problem in AI

- We also know the **eight puzzle problem** by the name of **N puzzle problem** or **sliding puzzle problem.**
- **N-puzzle** that consists of N tiles (N+1 titles with an empty tile) where N can be 8, 15, 24 and so on.
- In our example **N = 8.** (that is **square root of (8+1) = 3 rows and 3 columns**).
- In the same way, if we have N = 15, 24 in this way, then they have Row and columns as follow **(square root of (N+1) rows and square root of (N+1) columns).**
- That is if **N=15** than number of rows and columns= 4, and if **N= 24** number of rows and columns= 5.
- So, basically in these types of problems we have given a **initial state or initial configuration (Start state) and a Goal state or Goal Configuration.**

Here We are solving a problem **of 8 puzzle** that is **a 3x3 matrix.**

Initial state

1	2	3
	4	6
7	5	8

Goal state

1	2	3
4	5	6
7	8	

Solution:

The puzzle can be solved by moving the tiles one by one in the single empty space and thus achieving the Goal state.

Rules of solving puzzle

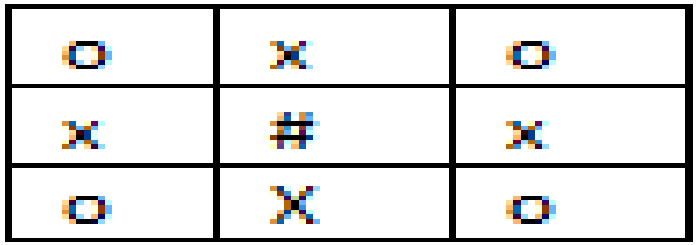
Instead of moving the tiles in the empty space we can visualize moving the empty space in place of the tile.

The empty space can only **move in four directions** (Movement of empty space)

1.Up
2.Down
3.Right or
4.Left

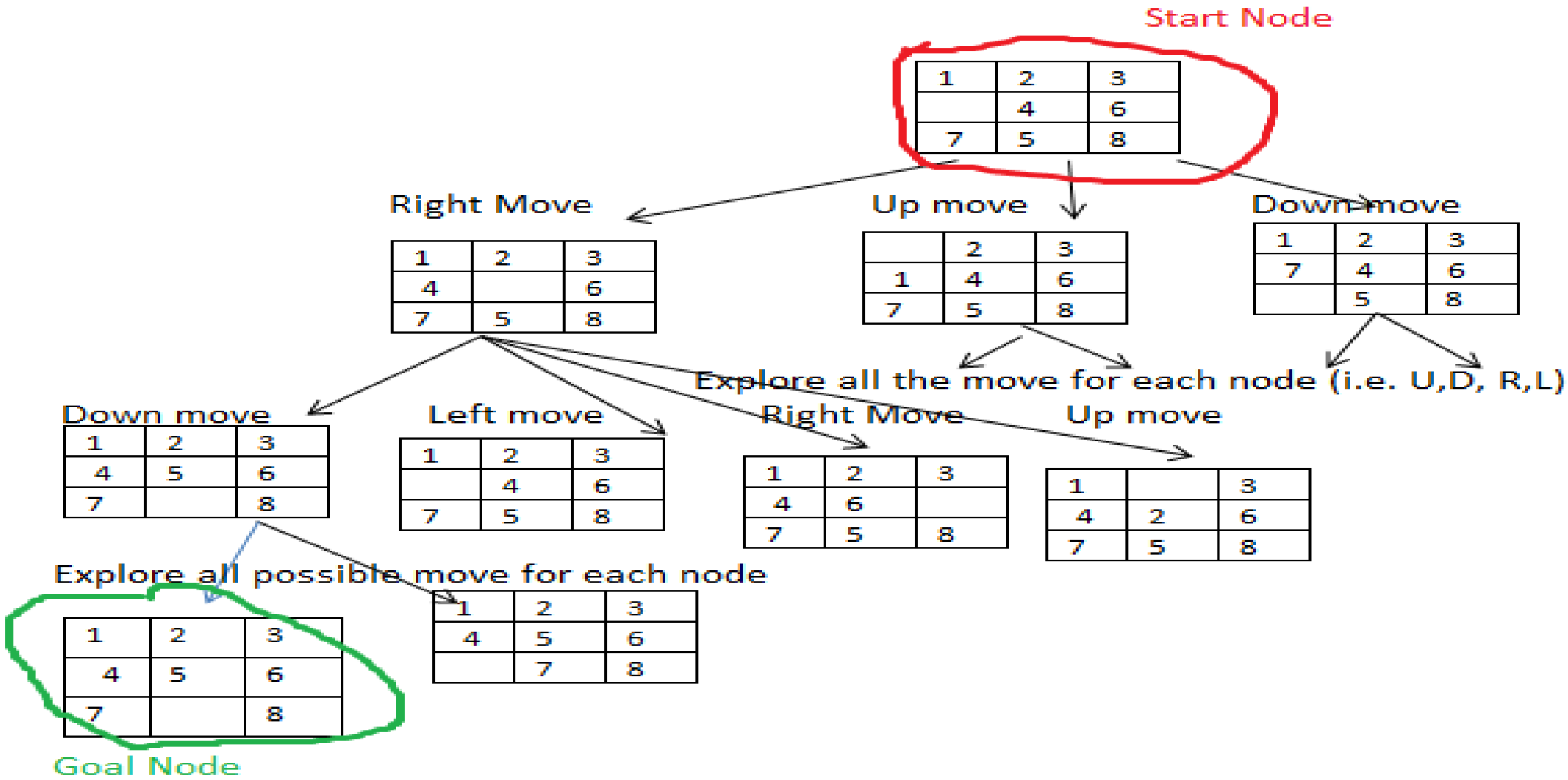
The empty space **cannot move diagonally** and can take **only one step at a time**.

All possible move of a Empty tile



o- Position total possible moves are **(2)**, **x - position** total possible moves are **(3)** and
#-position total possible moves **are (4)**
Let's solve the problem without **Heuristic Search** that is **Uninformed Search or Blind Search (Breadth First Search and Depth First Search)**

Breath First Search to solve Eight puzzle problem


Start Node
Right Move
Up move
Down move
Explore all the move for each node (i.e. U,D, R,L)
Down move
Left move
Right Move
Up move
Explore all possible move for each node
Goal Node

Note: If we solve this problem with depth first search, then it will go to depth instead of exploring layer wise nodes.
Time complexity: In worst case time complexity in **BFS is O(b^d) know as order of b raise to power d.**

b-branch factor
d-depth factor
Let's solve the problem with **Heuristic Search** that is **Informed Search (A* , Best First Search (Greedy Search))**
To solve the problem with Heuristic search or informed search we have to calculate Heuristic values of each node to calculate **cost function. (f=g+h)**

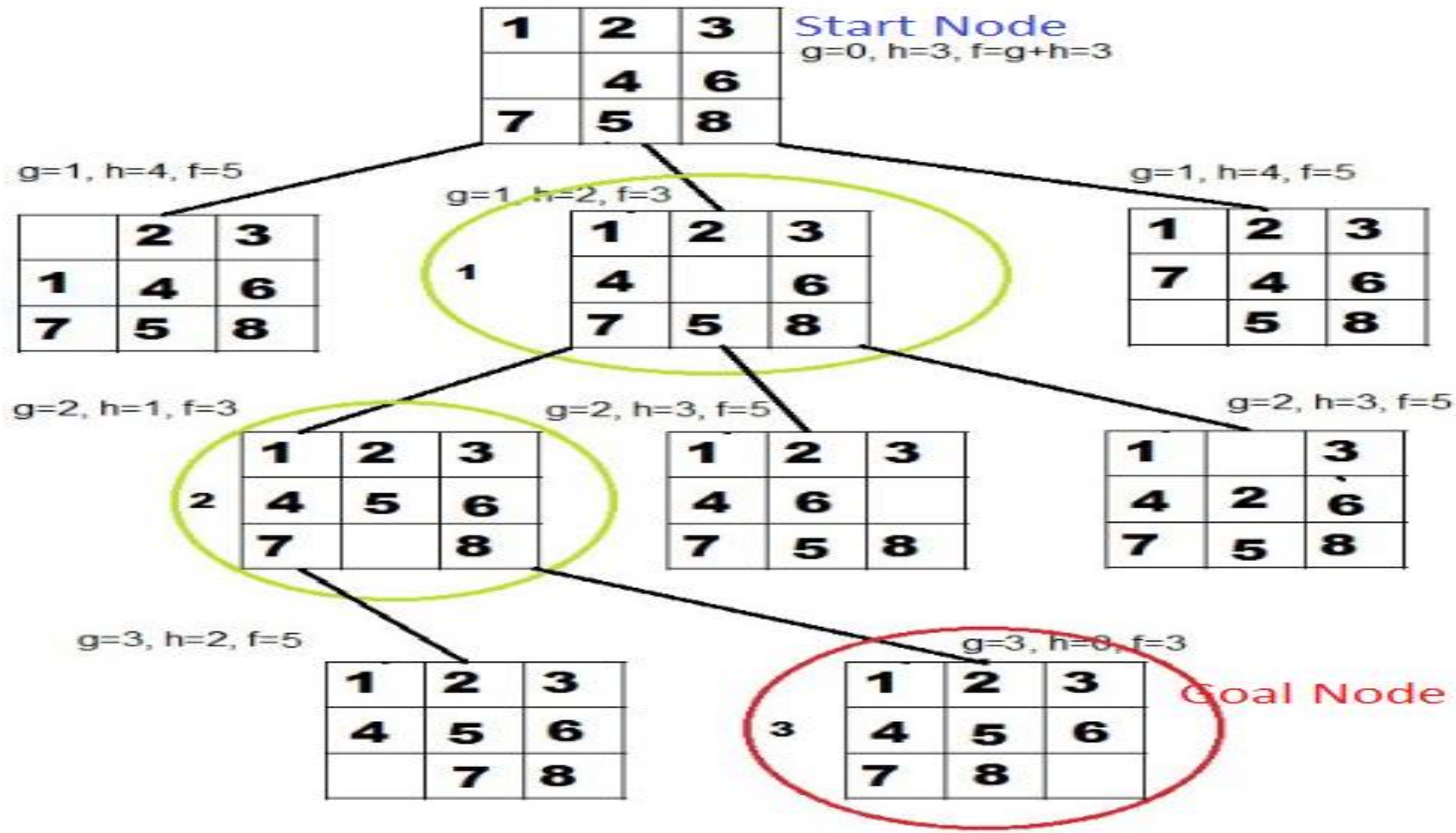
Initial state

1	2	3
	4	6
7	5	8

Goal state

1	2	3
4	5	6
7	8	

Note: See the initial state and goal state carefully all values except (4,5 and 8) are at their respective places. **so, the heuristic value for first node is 3.**(Three values are misplaced to reach the goal). And l**et's take actual cost (g) according to depth.**



N-Queens Problem/ Chess Problem

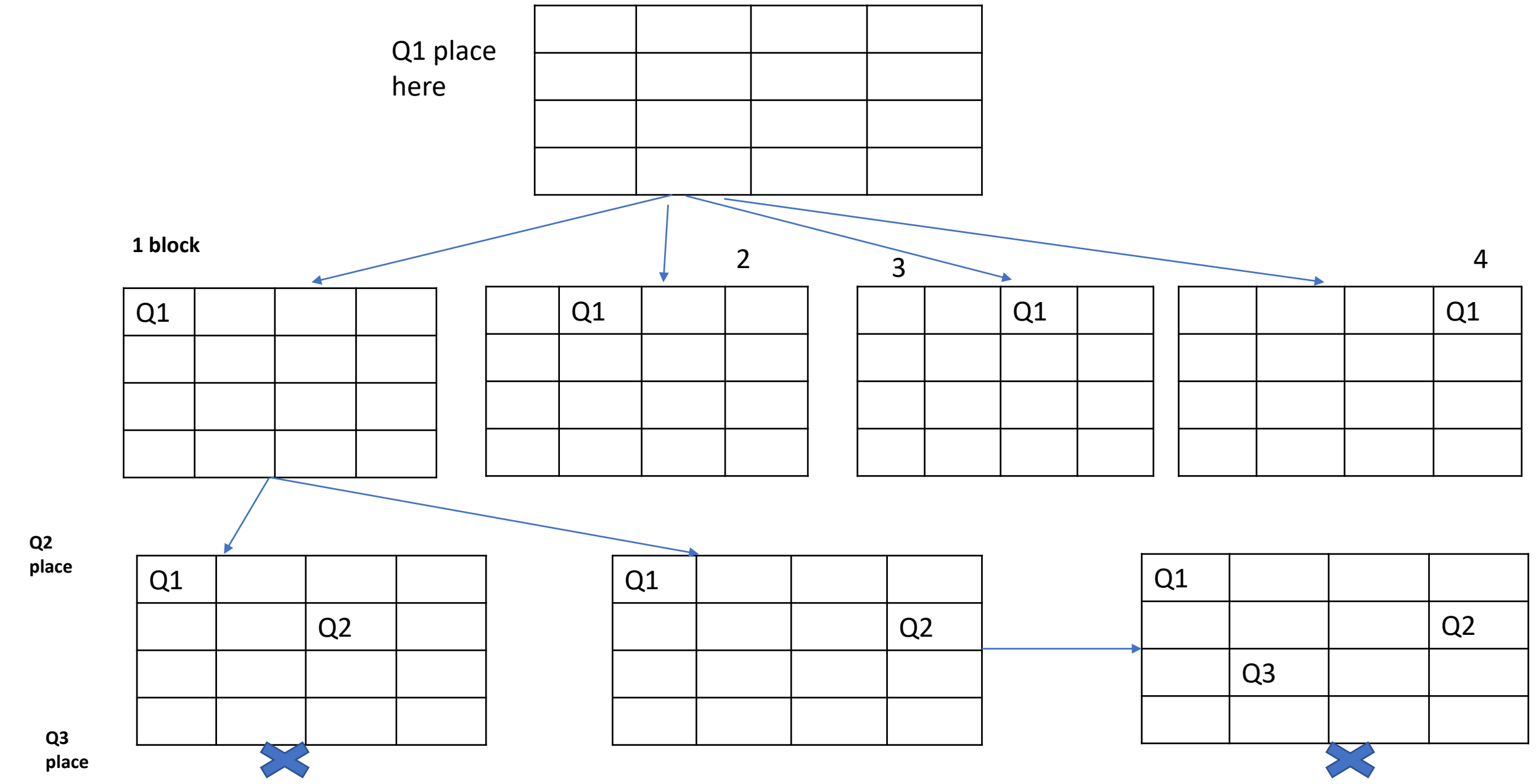
- **N - Queens problem** is to place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal.

- **Eg. 4 queen problem**

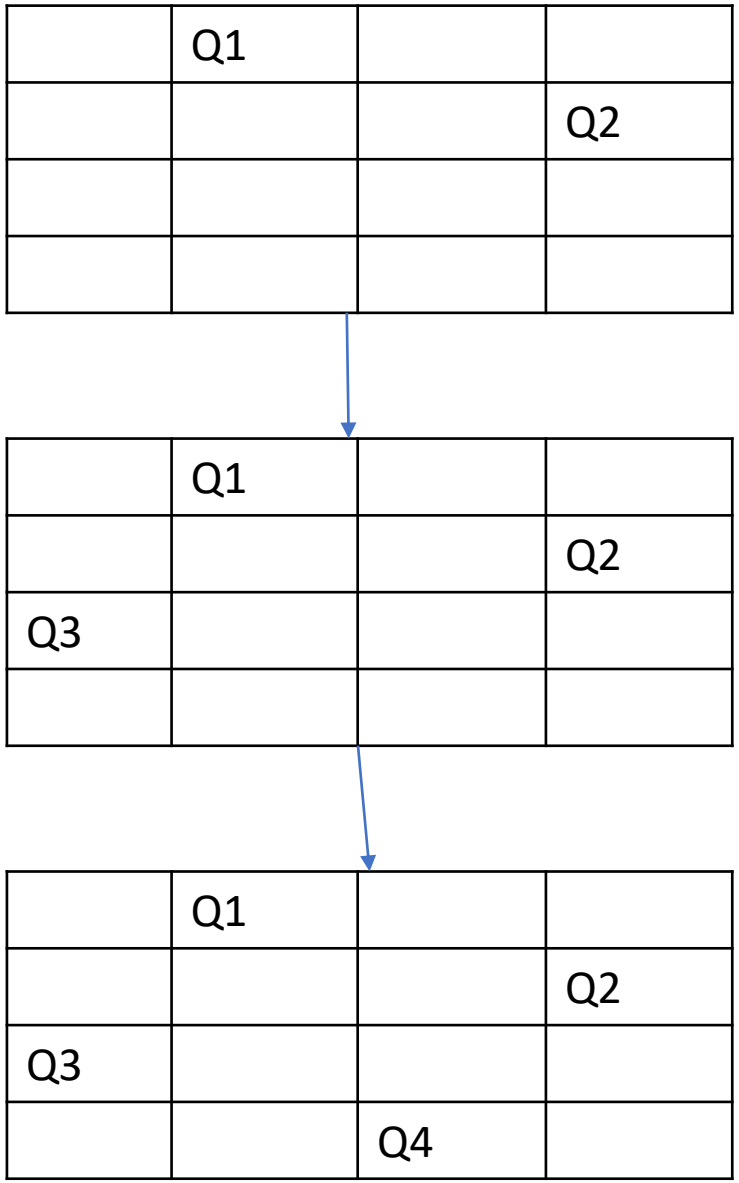
- **Rule 1:** 4 queen should be in separate row

Q1 in 1st row
Q2 in 2nd row
Q3 in 3rd row
Q4 in 4th row

- **Rule 2:** 4 queen cannot be in same column
- **Rule 3:** 4 queen can't be in diagonal


Q1 place here
1 block
2
3
4
Q1
Q2
Q3
Q2 place
Q3 place

Expand block 2
Q2 place here



Q3 place

Q4 place

Missionaries and Cannibals Problem

Missionaries and Cannibals is a problem in which 3 missionaries and 3 cannibals want to cross from the left bank of a river to the right bank of the river. There is a boat on the left bank, but it only carries at most two people at a time (and can never cross with zero people). If cannibals ever outnumber missionaries on either bank, the cannibals will eat the missionaries. How everything get across the river without the missionaries risking being eaten.

- A state can be represented by a triple, (m c b), where m is the number of missionaries on the left, c is the number of cannibals on the left, and b indicates whether the boat is on the left bank or right bank.
- For example, the initial state is (3 3 L) and the goal state is (0 0 R).
- **Operators are:**
- MM: 2 missionaries cross the river
- CC: 2 cannibals cross the river
- MC: 1 missionary and 1 cannibal cross the river
- M: 1 missionary crosses the river
- C: 1 cannibal crosses the river