Problem Reduction Search

- Typically where we are planning how best to solve a problem that can be recursively decomposed into sub-problems in multiple ways
  - Matrix multiplication problem
  - Tower of Hanoi and Blocks World problems
  - Finding shortest proofs in theorem proving
- AND/OR Graphs
  - An OR node represents a choice between possible decompositions
  - An AND node represents a given decomposition
- Game Trees
  - Max nodes represent the choice of my opponent
  - Min nodes represent my choice

Problem Reduction

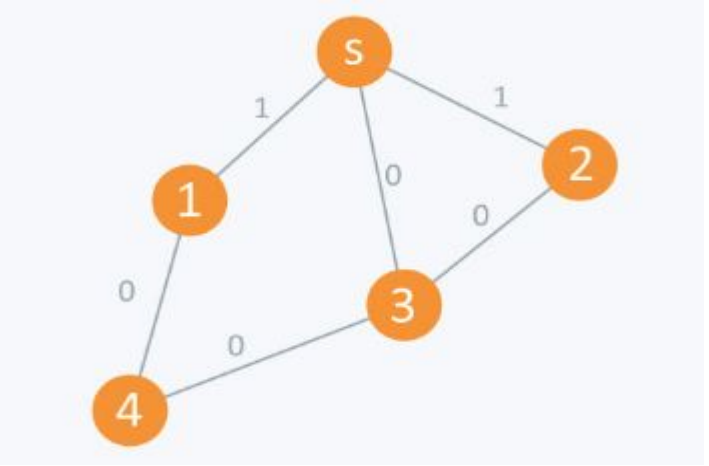
So far search strategies discussed were for OR graphs.

- Here several arcs indicate a different ways of solving problem.

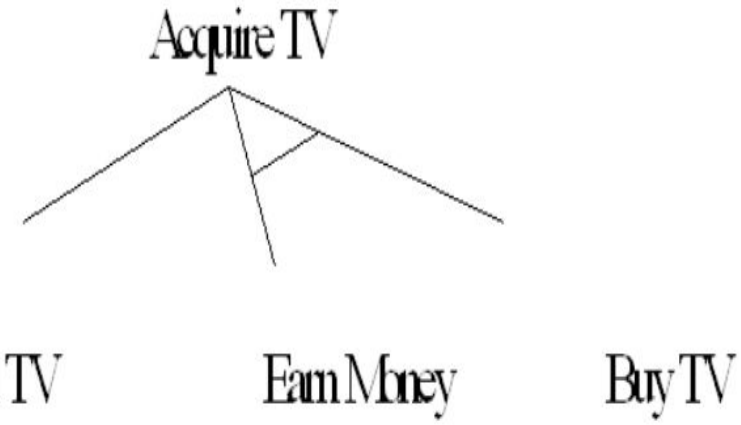
Another kind of structure is AND-OR graph (tree).

Useful for representing the solution of problem by decomposing it into smaller sub-problems.

- Each sub-problem is solved and final solution is obtained by combining solutions of each sub-problem.
- Decomposition generates arcs that we will call AND arc.
- One AND arc may point to any number of successors, all of which must be solved.
- Such structure is called AND–OR graph rather than simply AND graph.

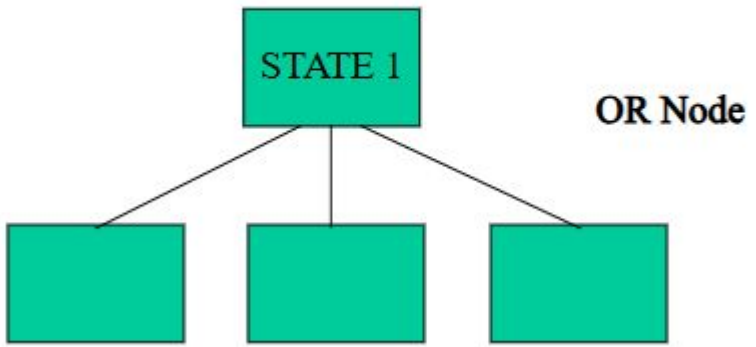


Example of AND-OR Tree



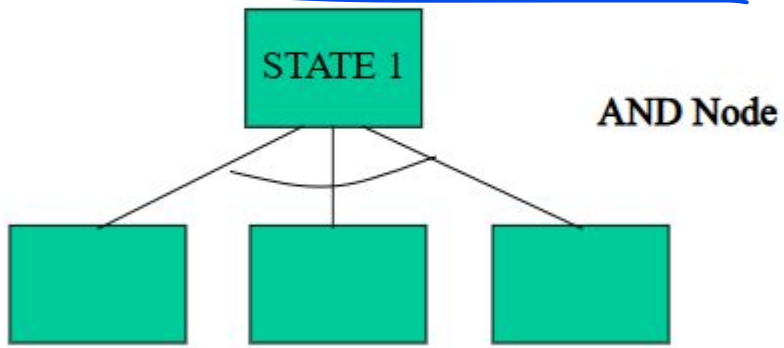
AND-OR Search Graphs

- OR Graphs: The state-space graphs dealt with previously were OR graphs; from any state one-of the applicable actions is chosen to reach the goal.



AND-OR Search Graphs

- AND-OR Graphs: In AND-OR graphs, from a state not one but all the paths may have to be pursued to reach the goal state.



AND-OR Search Graphs

- How do AND-OR graphs arise?
  - AND-OR graphs arise when it is advantageous to decompose a state into its components which are independent of each other.
- Example: Integration (See the handout)
- Example: Rewriting problem – found in syntax analysis.

AND–OR Graph

- To find a solution in AND–OR graph, we need an algorithm similar to A*
  - with the ability to handle AND arc appropriately.
- In search for AND-OR graph, we will also use the value of heuristic function **f** for each node.

AND–OR Graph Search

- Traverse AND-OR graph, starting from the initial node and follow the current best path.
- Accumulate the set of nodes that are on the best path which have not yet been expanded.
- Pick up one of these unexpanded nodes and expand it.
- Add its successors to the graph and compute f (using only h) for each of them.

AND–OR Graph Search – Contd..

- Change the f estimate of newly expanded node to reflect the new information provided by its successors.
  - Propagate this change backward through the graph to the start.
- Mark the best path which could be different from the current best path.
- Propagation of revised cost in AND-OR graph was not there in A*.

AO* Search Algorithm in Artificial Intelligence

- AO* algorithm is a heuristic search algorithm in AI.
- AO* algorithm uses the concept of AND-OR graphs to decompose any complex problem given into smaller set of problems which are further solved.
- **Working of AO* algorithm:**
- The AO* algorithm works on the formula given below :

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

- where,
- $g(n)$: The actual cost of traversal from initial state to the current state.
- $h(n)$: The estimated cost of traversal from the current state to the goal state.
- $f(n)$: The actual cost of traversal from the initial state to the goal state.

AO* Algorithm

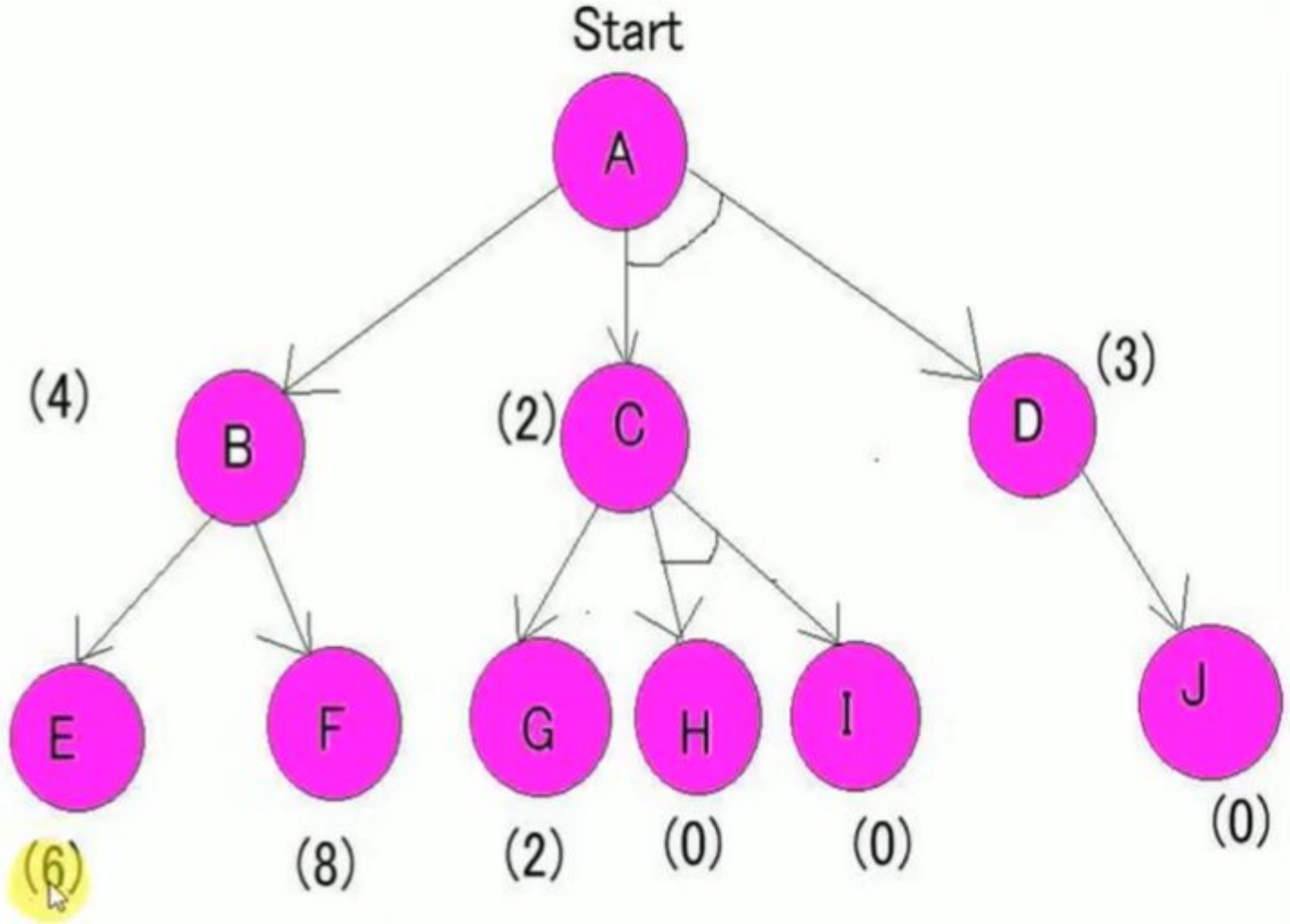
General search for AND-OR graphs:

1. Initialize: {D1,..,Dn} ← decompose initial state into independent components Di, if possible.
2. Loop until {Di} is *Empty*
3. Select: Select current path D* from {Di} according to some strategy and a *node* in D*.
4. Goal Check: Do all leaves in the current path D* satisfy *goal_test()*? If Yes, return solution.
5. Expand: Expand *node* selected from D*. Decompose each state generated and add the components to {Di}.
6. End.

AO* Search Algorithm in Artificial Intelligence

- Here, in the above example all numbers in brackets are the heuristic value i.e h(n).
- Each edge is considered to have a value of 1 by default.

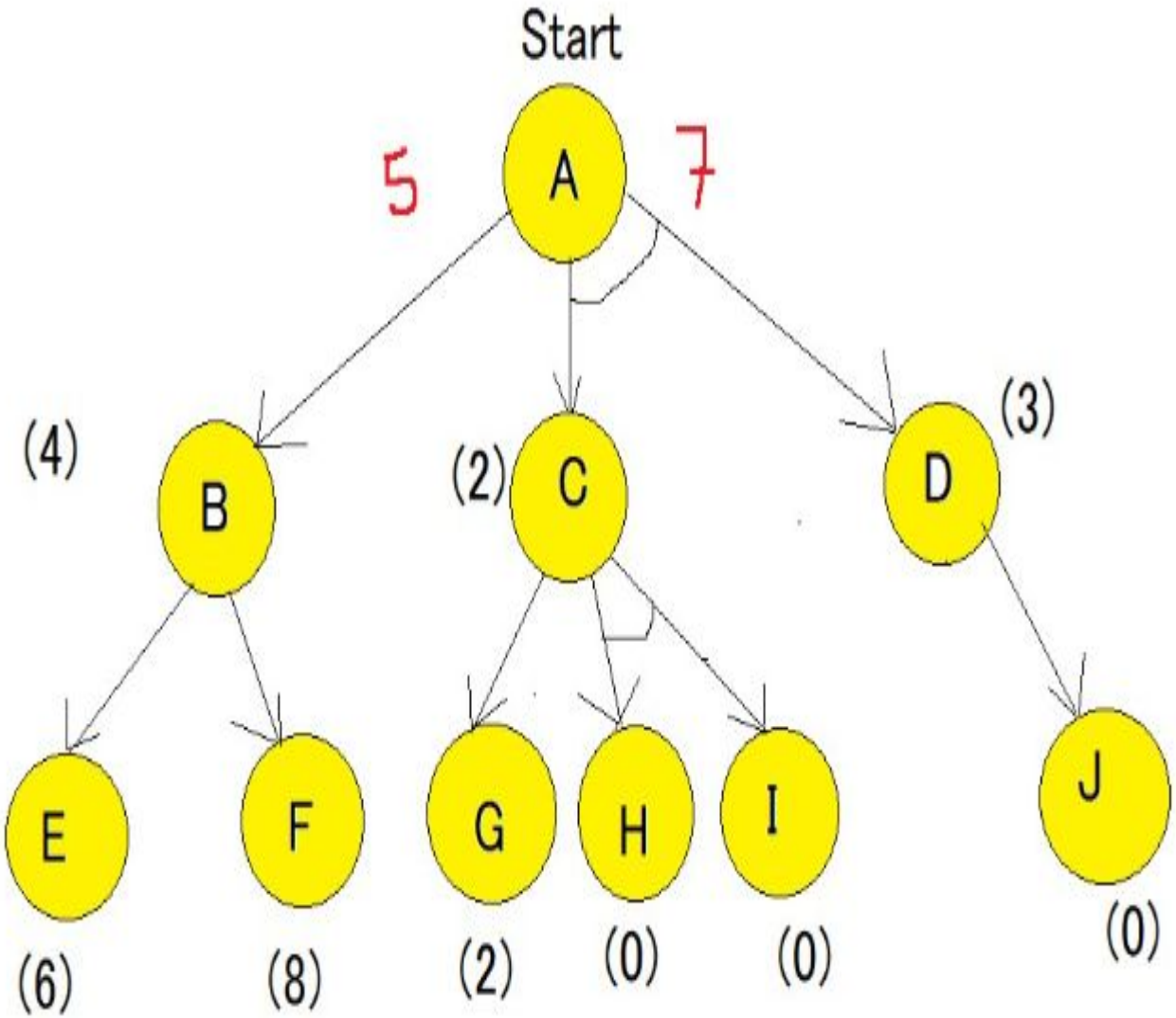
Here the numbers listed in the circular brackets ( ) are estimated cost and the revised costs are enclosed in square brackets [ ].



AO* Search Algorithm in Artificial Intelligence

Step-1

- Starting from node A, we first calculate the best path.
- **f(A-B) = g(B) + h(B) = 1+4= 5** , where 1 is the default cost value of travelling from A to B and 4 is the estimated cost from B to Goal state.
- **f(A-C-D) = g(C) + h(C) + g(D) + h(D) = 1+2+1+3 = 7**, here we are calculating the path cost as both C and D because they have the AND-Arc.
- The default cost value of travelling from A-C is 1, and from A-D is 1, but the heuristic value given for C and D are 2 and 3 respectively hence making the cost as 7.



AO* Search Algorithm in Artificial Intelligence

Step-2

- From the B node,

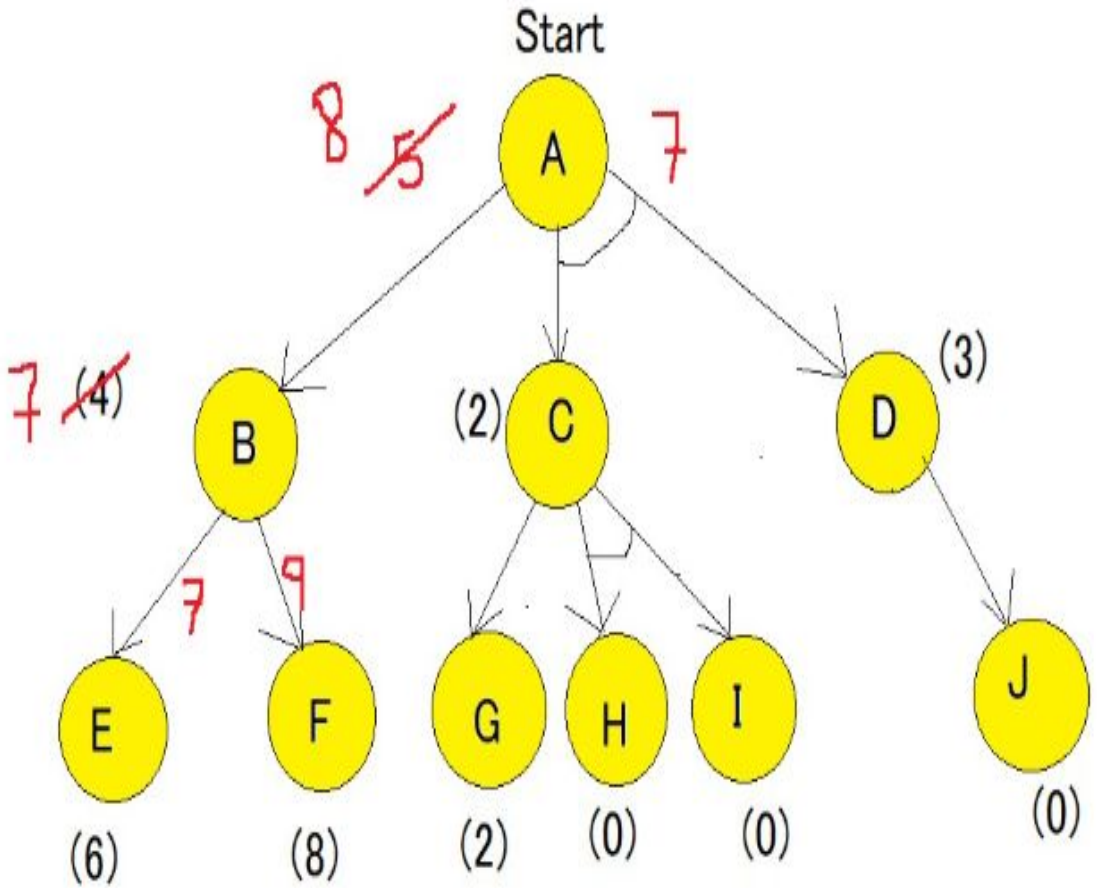
  f(B-E) = 1 + 6 = 7

  f(B-F) = 1 + 8 = 9

- Hence, the B-E path has lesser cost. Now the heuristics have to be updated since there is a difference between actual and heuristic value of B.
- The minimum cost path is chosen and is updated as the heuristic , in our case the value is 7.
- And because of change in heuristic of B there is also change in heuristic of A which is to be calculated again.

  f(A-B) = g(B) + updated((h(B)) = 1+7=8

f(A-B) = g(B) + updated((h(B)) = 1+7=8



AO* Search Algorithm in Artificial Intelligence

Step-3

- Now the current node becomes C node and the cost of the path is calculated,

  f(C-G) = 1+2 = 3

  f(C-H-I) = 1+0+1+0 = 2

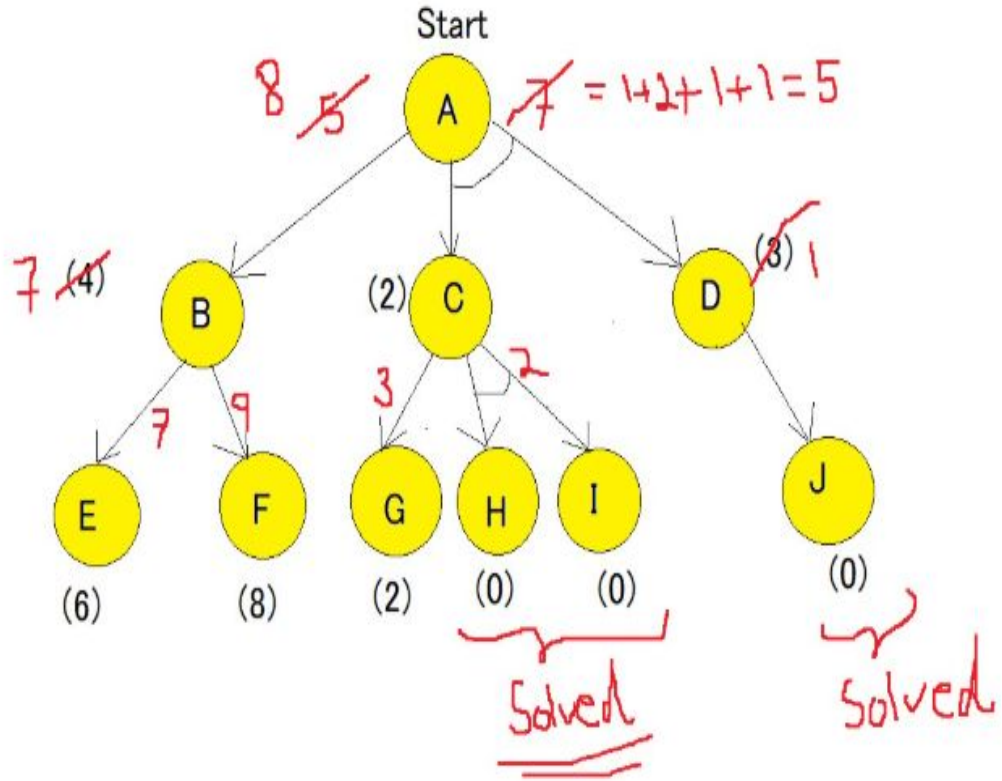
  f(C-H-I) is chosen as minimum cost path.
- Heuristic of path of H and I are 0 and hence they are solved,
- But Path A-D also needs to be calculated , since it has an AND-arc.
- **f(D-J) = 1+0 = 1**, hence heuristic of D needs to be updated to 1.
- And finally the f(A-C-D) needs to be updated.

  f(A-C-D) = g(C) + h(C) + g(D) + updated((h(D))

  = 1+2+1+1 =5.

f(D-J) = 1+0 = 1, hence heuristic of D needs to be updated to 1. And finally the f(A-C-D) needs to be updated.

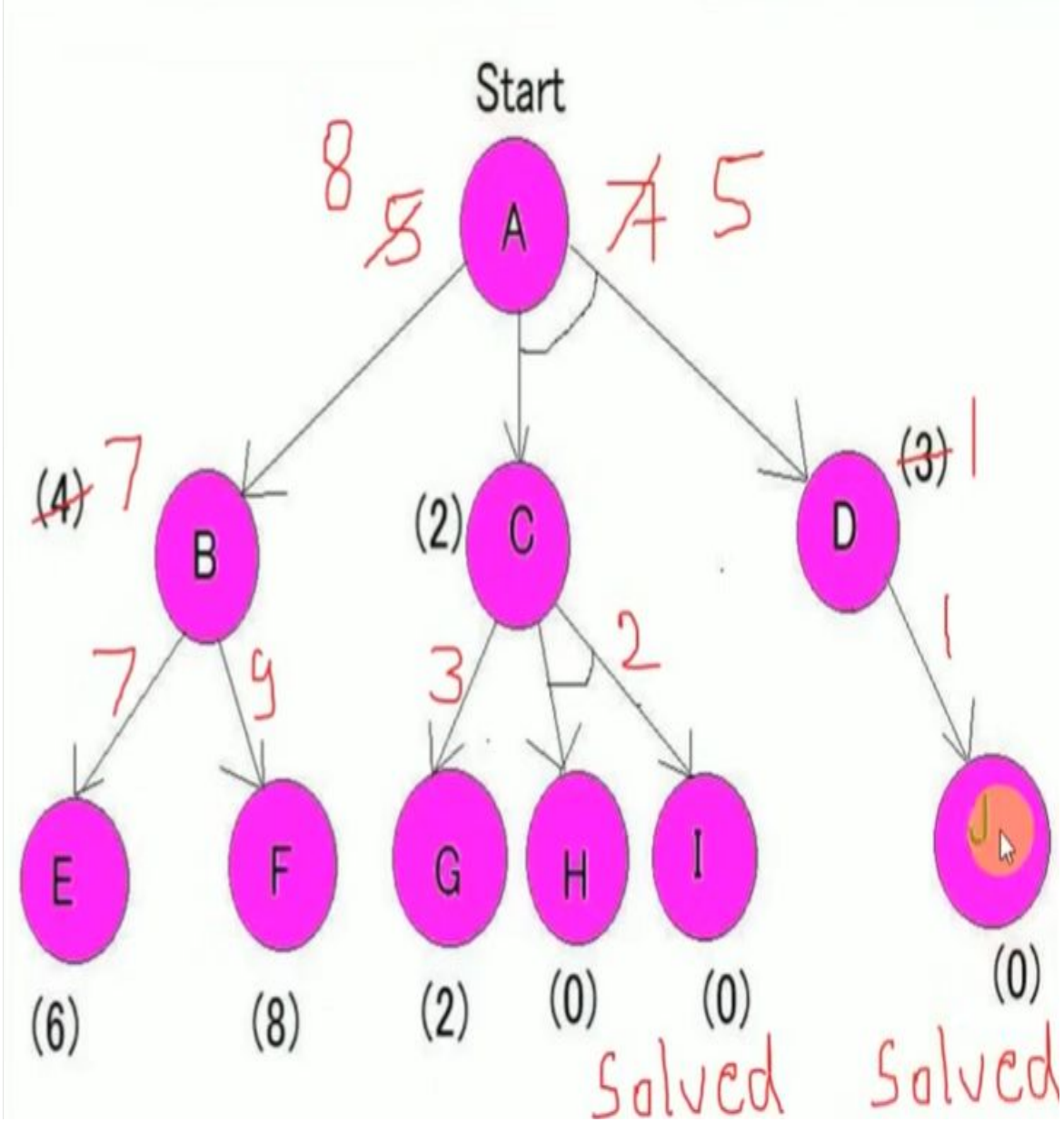
f(A-C-D) = g(C) + h(C) + g(D) + updated((h(D)) = 1+2+1+1 =5.



As we can see that the solved path is f(A-C-D).

The "Solve" labeling Procedure

- A terminal node is labeled as
  - "solved" if it is a goal node (representing a solution of sub-problem)
  - "unsolved" otherwise (as we can not further reduce it)
- A non-terminal AND node labeled as
  - "solved" if all of its successors are "solved".
  - "unsolved" as soon as one of its successors is labeled "unsolved".
- A non-terminal OR node is labeled as
  - "solved" as soon as one of its successors is labeled "solved".
  - "unsolved" if all its successors are "unsolved".



GAME PLAYING

GAME PLAYING

Definition:

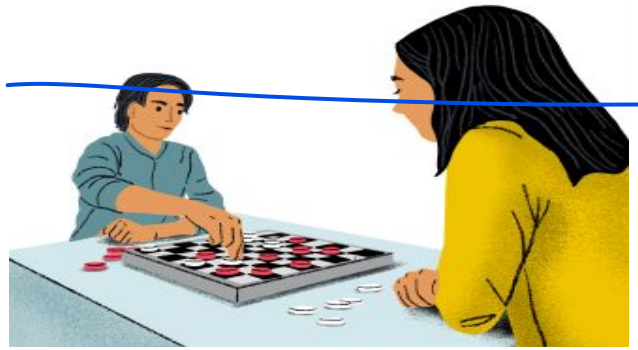
Game is defined as a sequences of choices where each choices is made from a number of discrete alternatives.

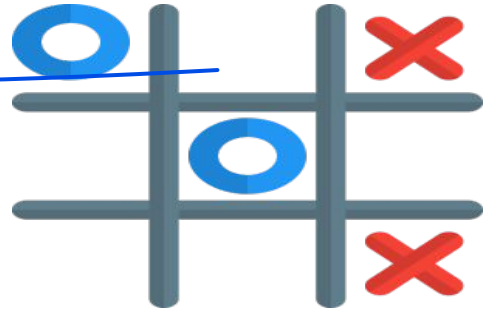
Each sequence ends in a certain outcome and every outcome has a definite value for opening player

Types :

Perfect information games- Both the players have access to the same information about the game in progress **(Chess, checker, tic-tac-toe)**

EX :





Imperfect information games- Player do not have the access to complete information about the game in progress

Ex:

Game Problem Vs State Space(Search) Problem

State Space Problem	Game Problem
States	Legal Board positions
Rule	Legal moves
Goal	Wining positions

Games don't require much knowledge;
we need to provide the **rules, legal moves, & conditions** of **winning** or **losing** the game.

Both players try to win the game. So, both of them try to make the best move possible at each turn.

- **Generate procedure so that only good moves are generated**.
- **Test procedure so that the best move can be explored first**

"The goal of game playing in AI is to develop algorithms that can learn how to play games and make decisions that will lead to winning outcomes".

Game Tree

A Game Tress represent **TWO Player** Game

It is a graphical representation **moves available** in the game.

It provides information about the **players, strategies**, and the **order of moves**.

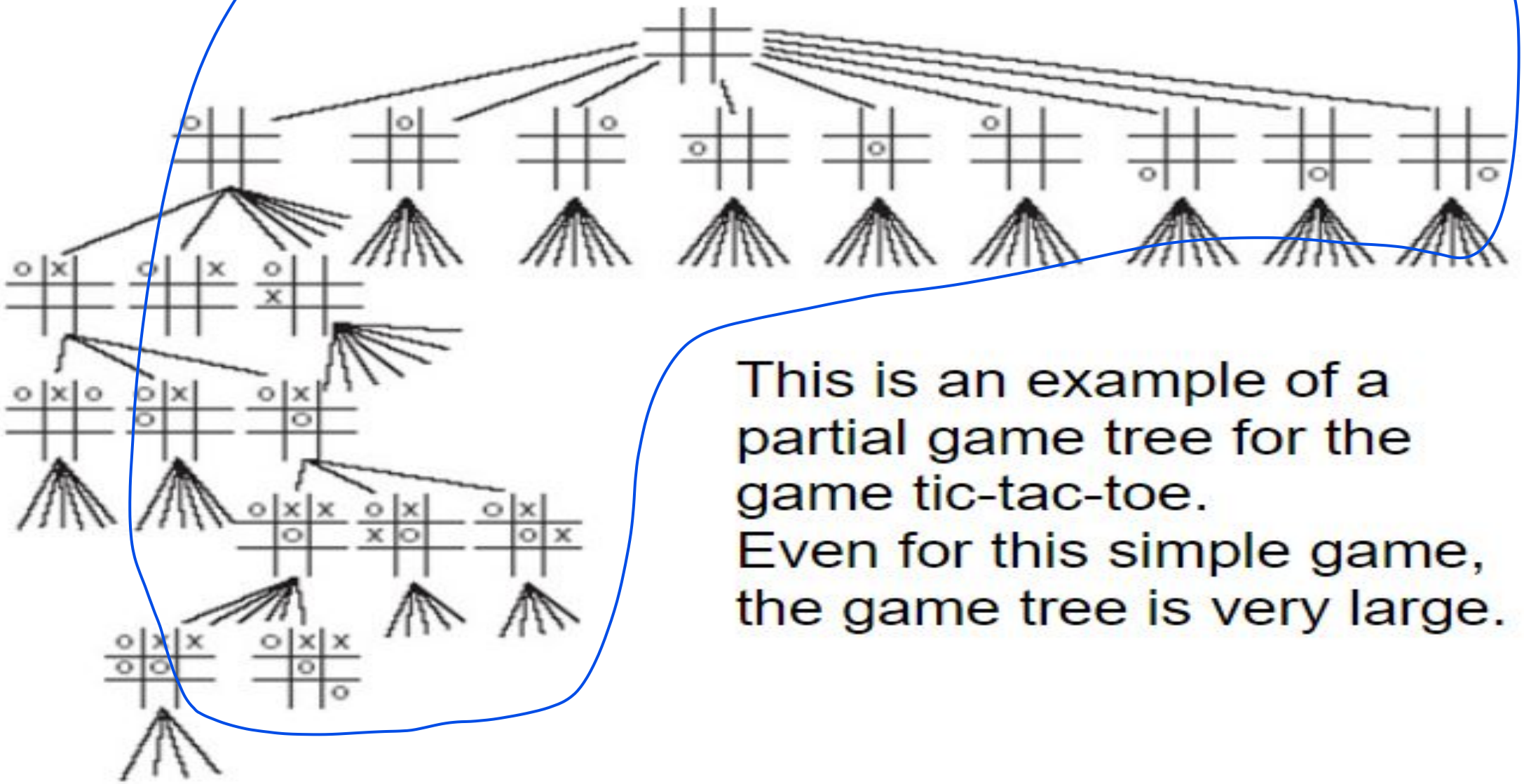
The game tree consists of,

Nodes = players position,
connected by edges = move.
Initial (or root) = first decision to be made
Terminal node = an end to the game.

Each terminal node is labeled with the **status labels** earned by each player

WIN, LOSS, DRAW

Game Trees



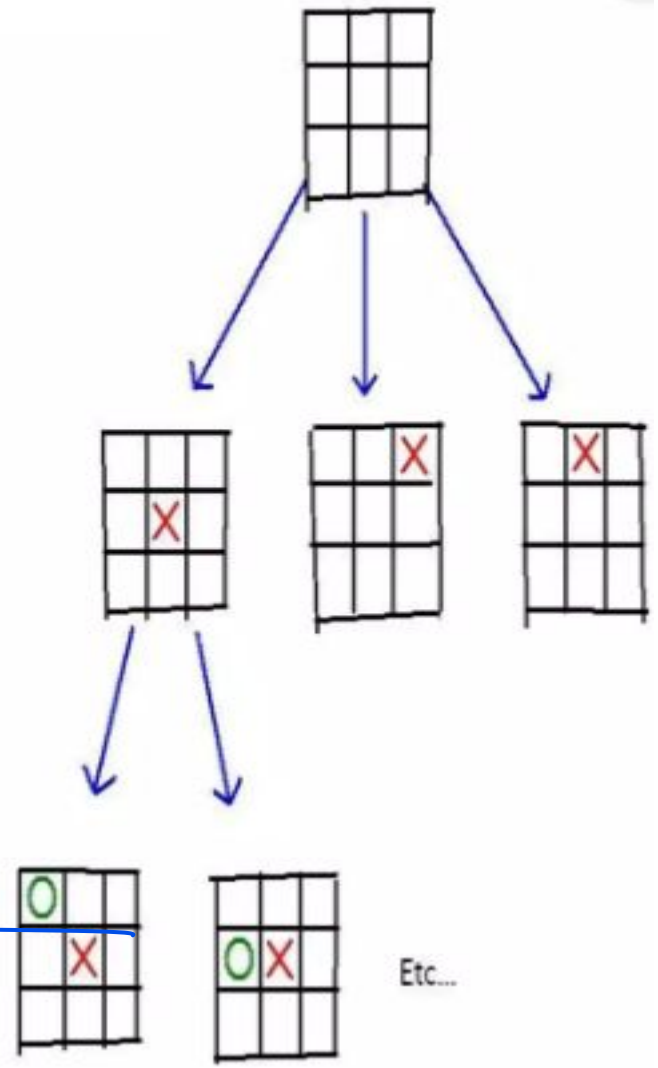
This is an example of a partial game tree for the game tic-tac-toe.
Even for this simple game, the game tree is very large.

Assumptions

- In talking about game playing systems, we make a number of assumptions:
  - The opponent is rational – will play to win.
  - The game is zero-sum – if one player wins, the other loses.
  - Usually, the two players have complete knowledge of the game. For games such as poker, this is clearly not true.

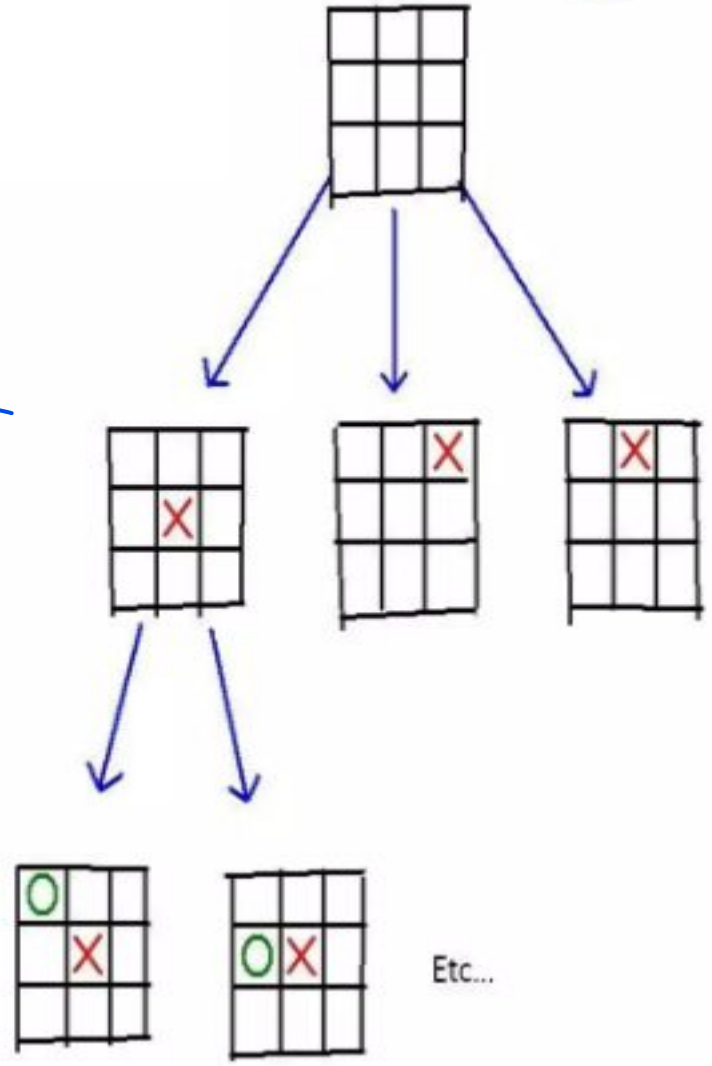
MINI-MAX ALGORITHM IN AI

- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player
- Both Players of the game are opponent 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 all the way down to the terminal node of the tree, then backtrack the tree as the recursion.



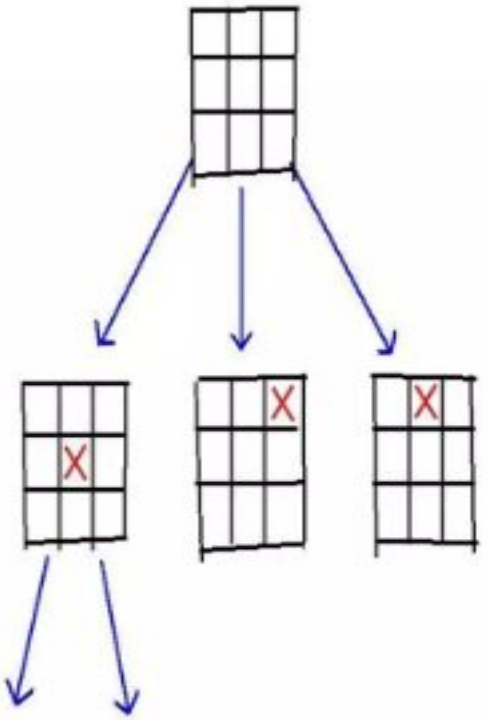
MINI-MAX ALGORITHM IN AI

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory.
- It provides an optimal move for the player assuming that opponent is also playing optimally.
- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.



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.

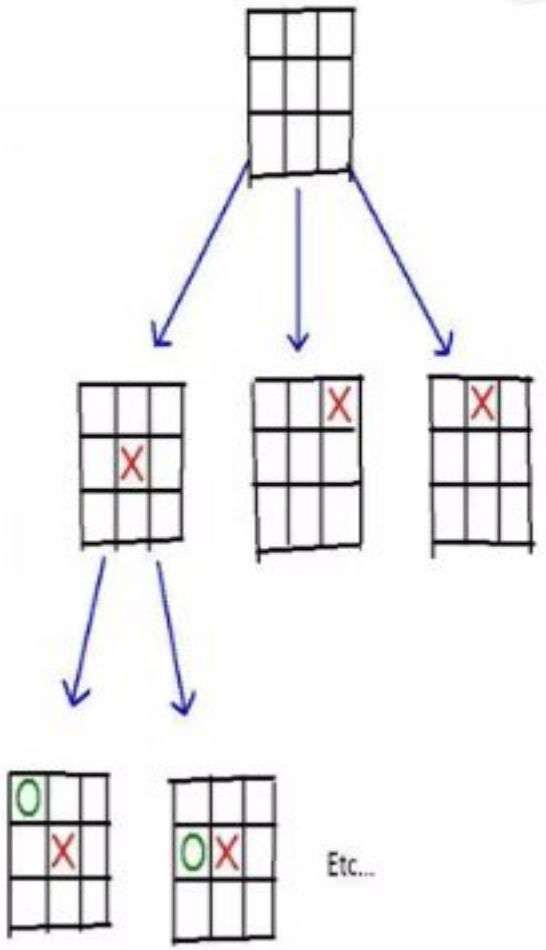
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.

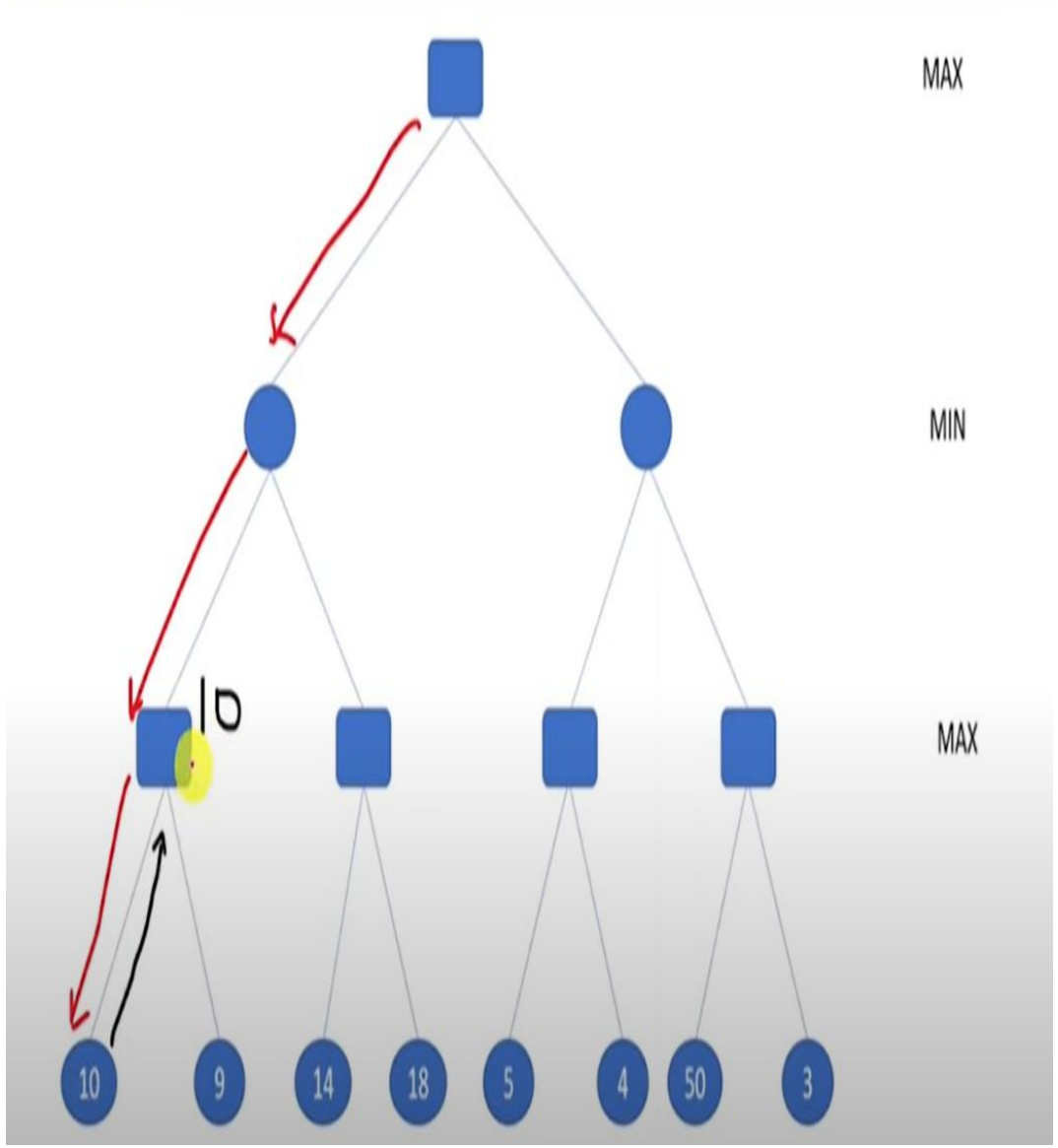
WORKING OF MIN-MAX ALGORITHM:

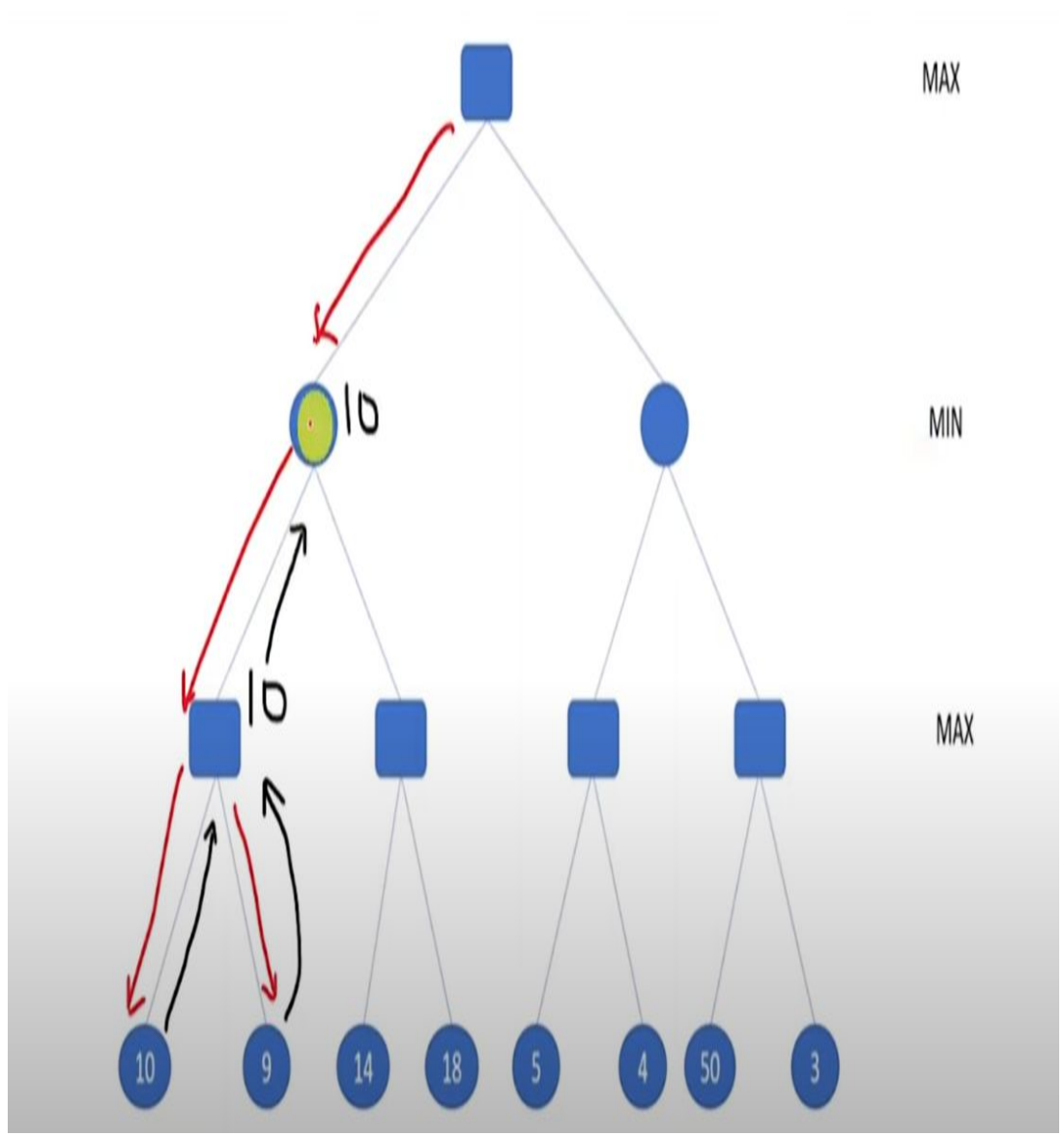
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.

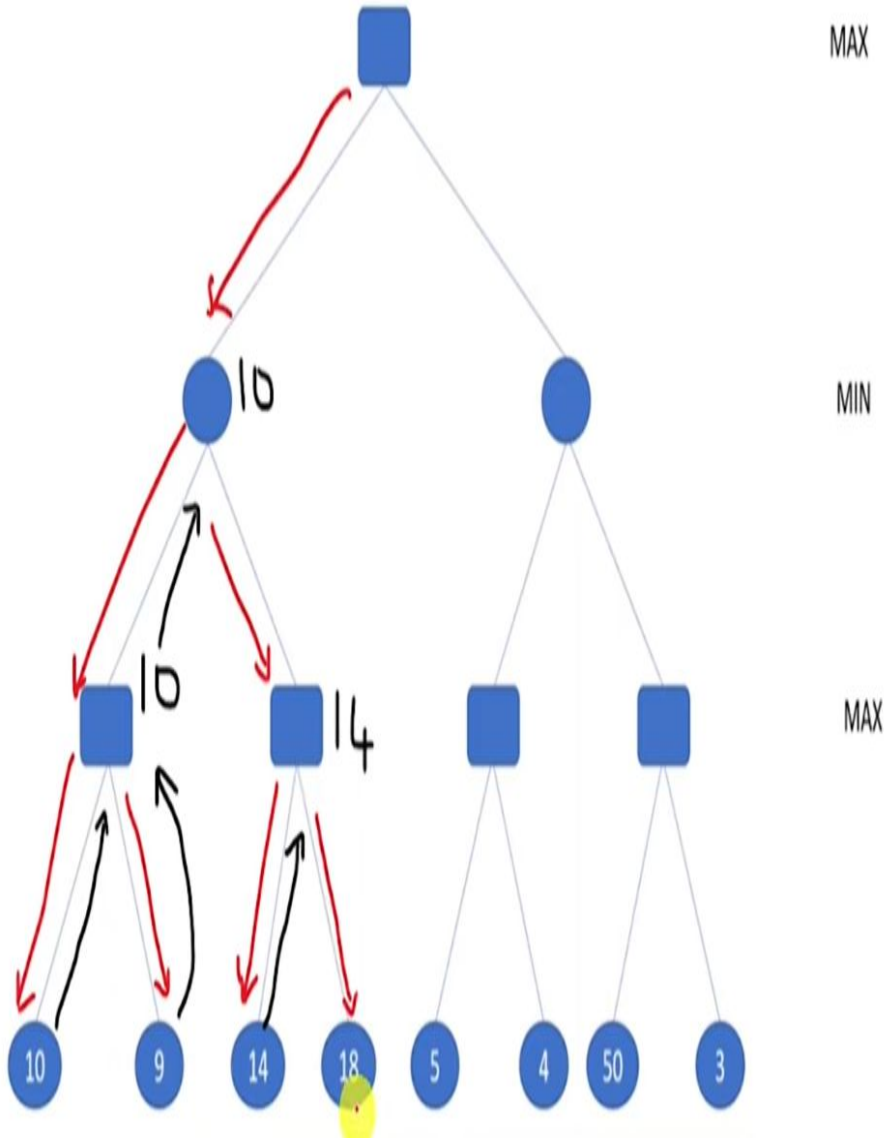
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs.

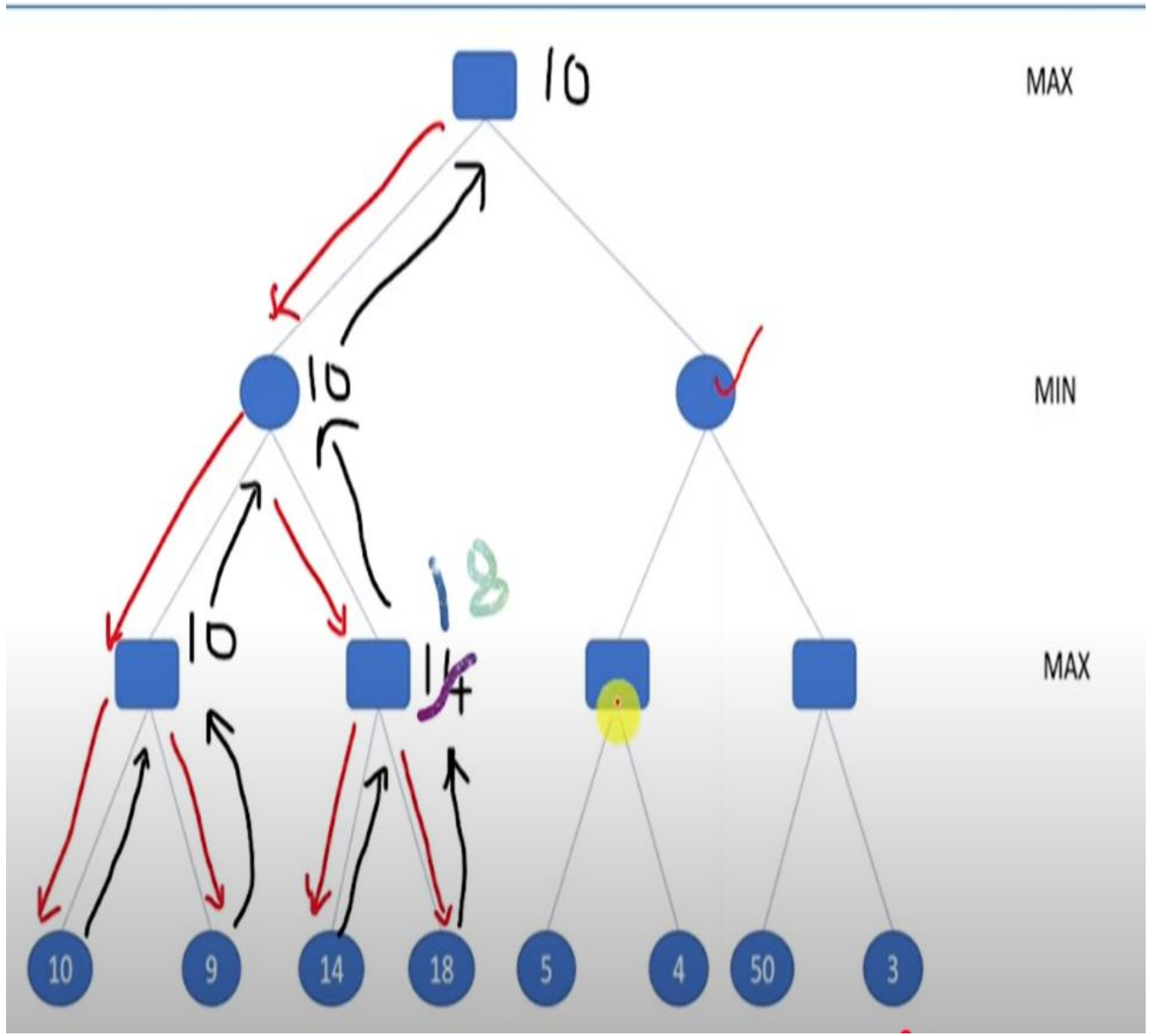
- Following are the main steps involved in solving the two-player game tree:

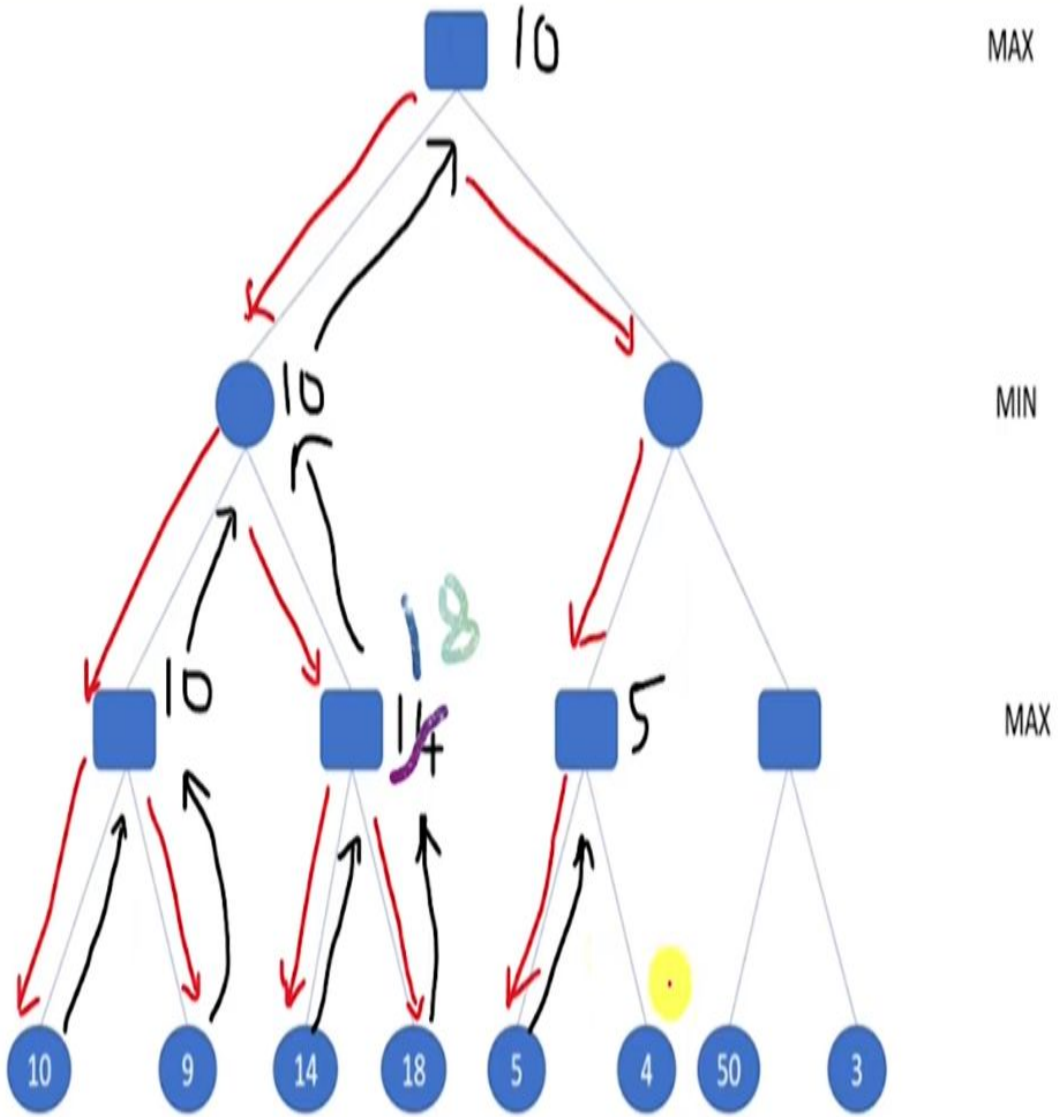


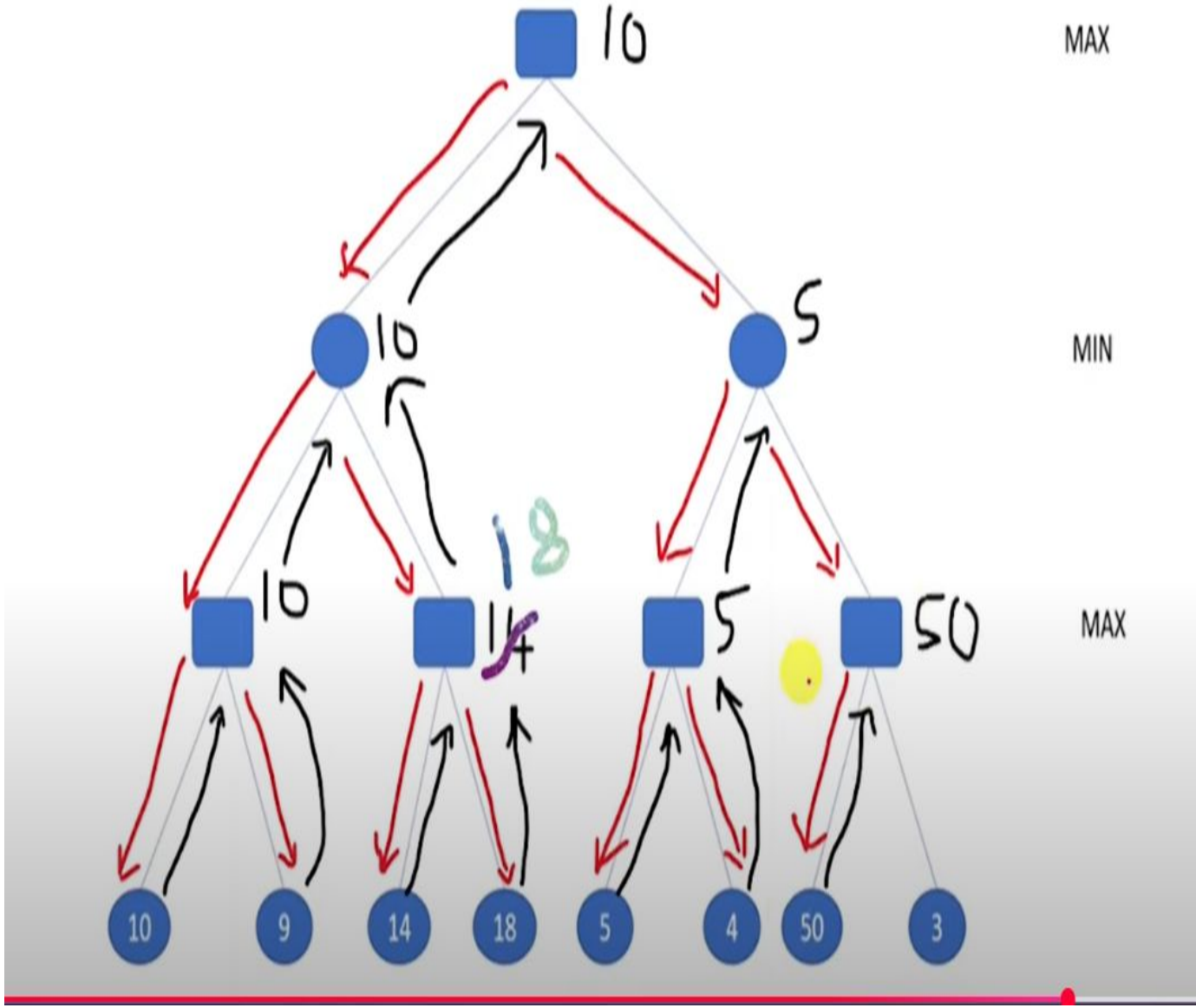

MAX
MIN
10
MAX
10
9
14
18
5
4
50
3

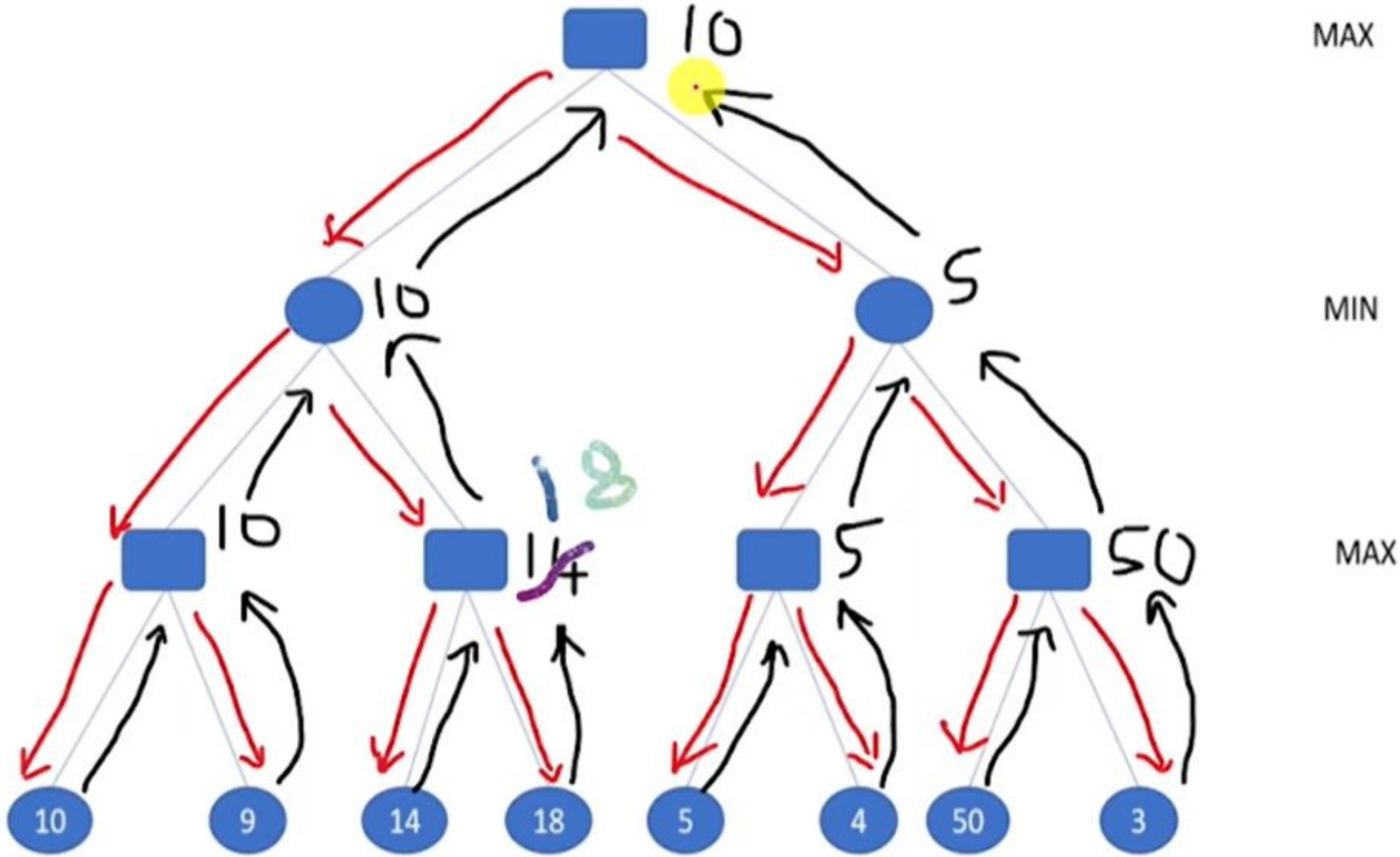

MAX
10
MIN
10
MAX
10
9
14
18
5
4
50
3

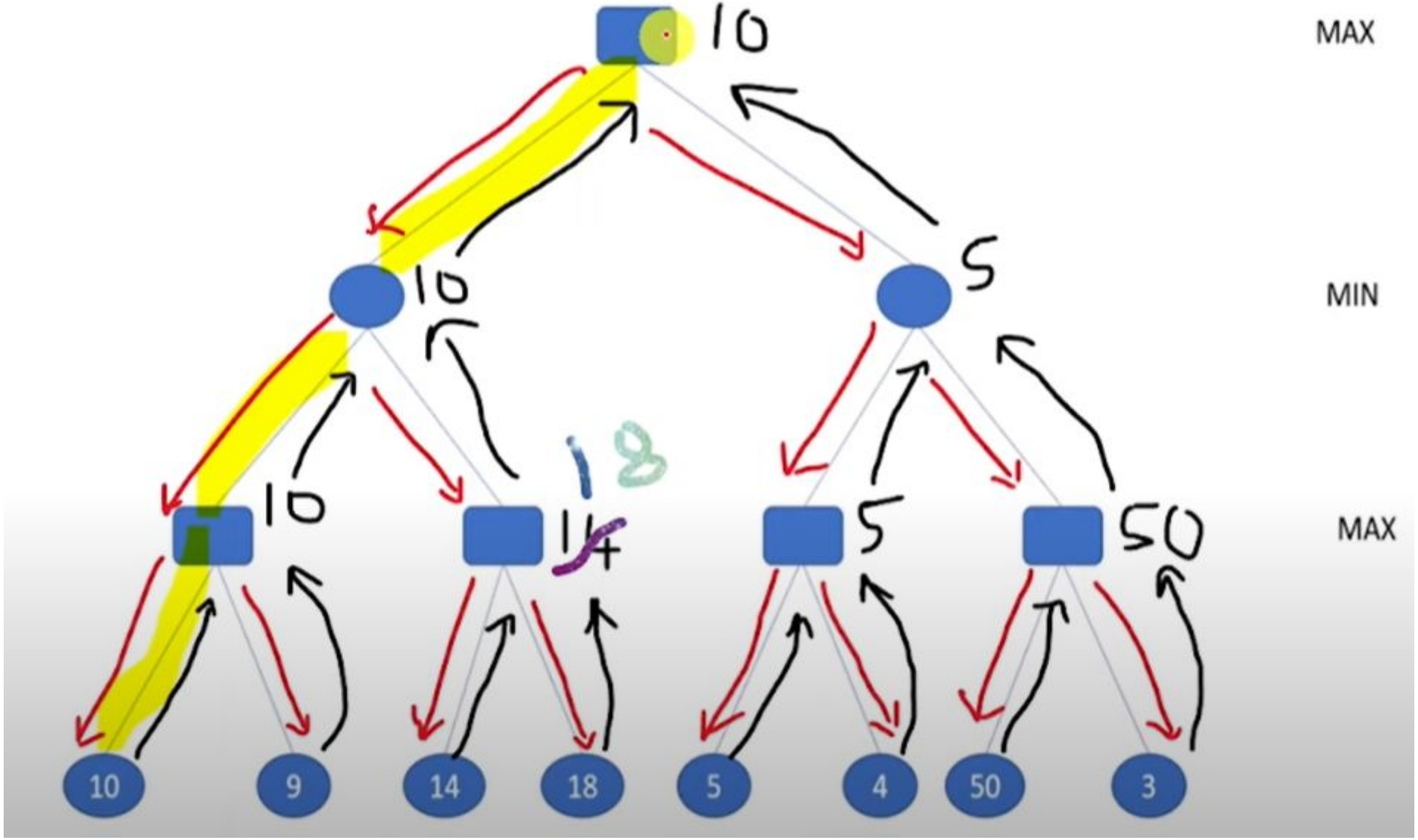

MAX
MIN
MAX
10
10
14
10
9
14
18
5
4
50
3


10
MAX
10
MIN
18
10
14
MAX
10
9
14
18
5
4
50
3


10
MAX
10
MIN
10
18
14
MAX
10
9
14
18
5
4
50
3


10
MAX
10
5
MIN
10
18
14
5
50
MAX
10
9
14
18
5
4
50
3

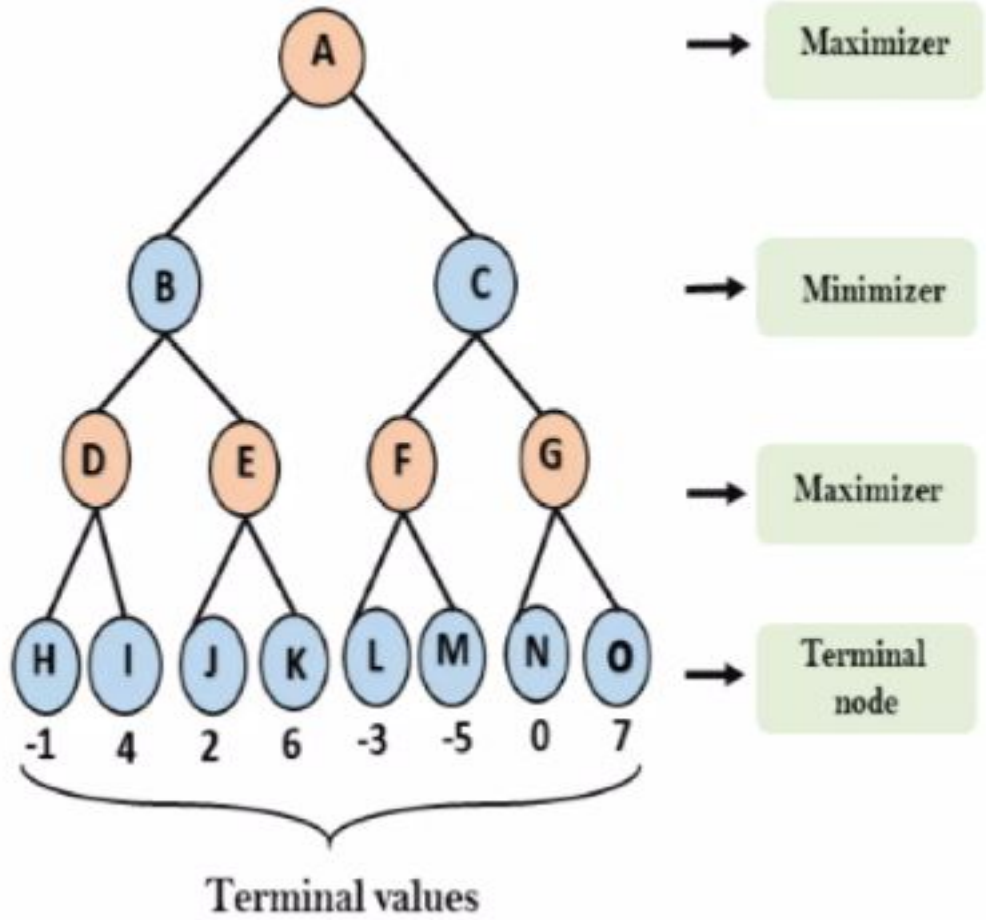

10
MAX
10
5
MIN
10
18
14
5
5
50
MAX
10
9
14
18
5
4
50
3


MAX
10
MIN
10
5
MAX
10
18
14
5
50
10
9
14
18
5
4
50
3

Working of Min-Max Algorithm:

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

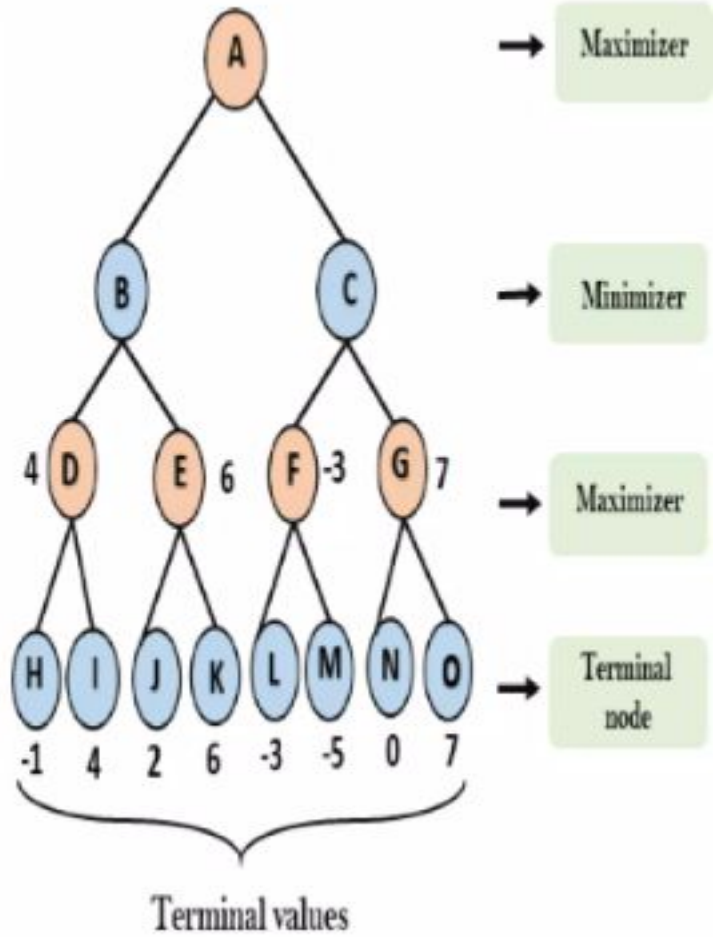
- **Step-1:** In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states.
- In the below tree diagram, let's take A is the initial state of the tree.
- Suppose maximizer takes first turn which has worst-case initial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



WORKING OF MIN-MAX ALGORITHM:

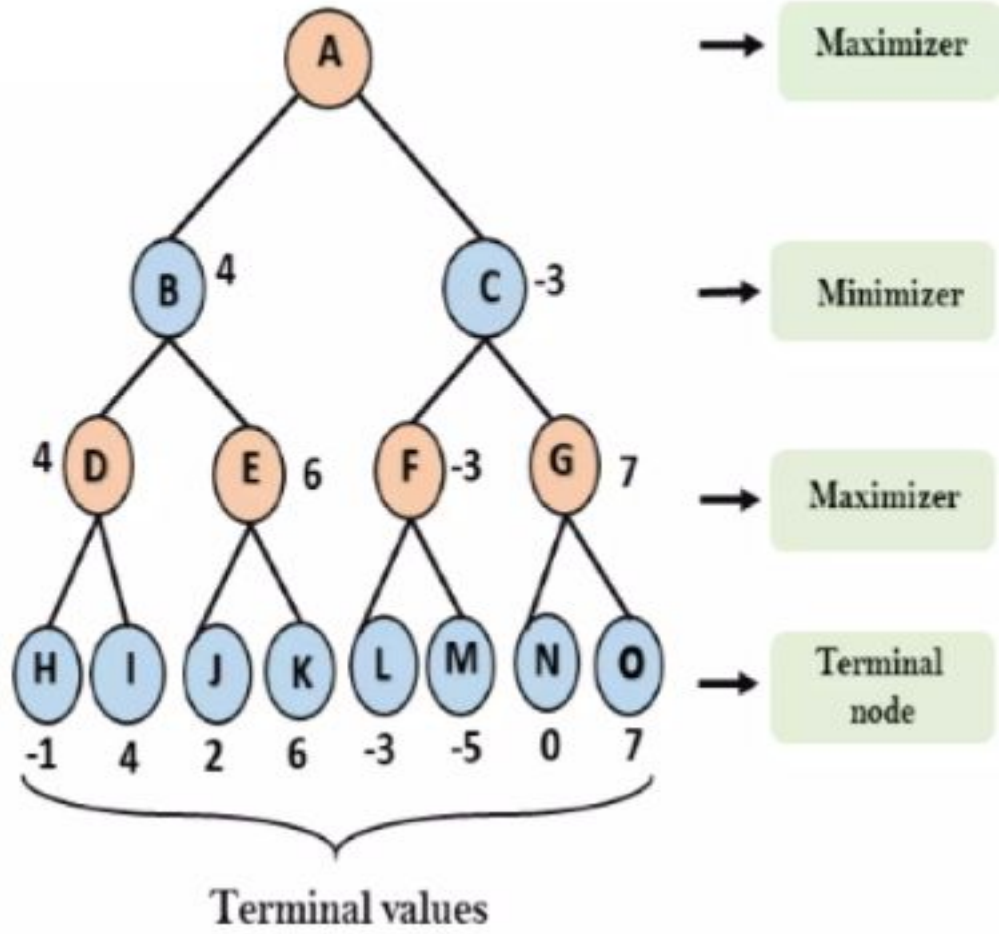
- **Step 2:** Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values.
- It will find the maximum among the all.

- 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



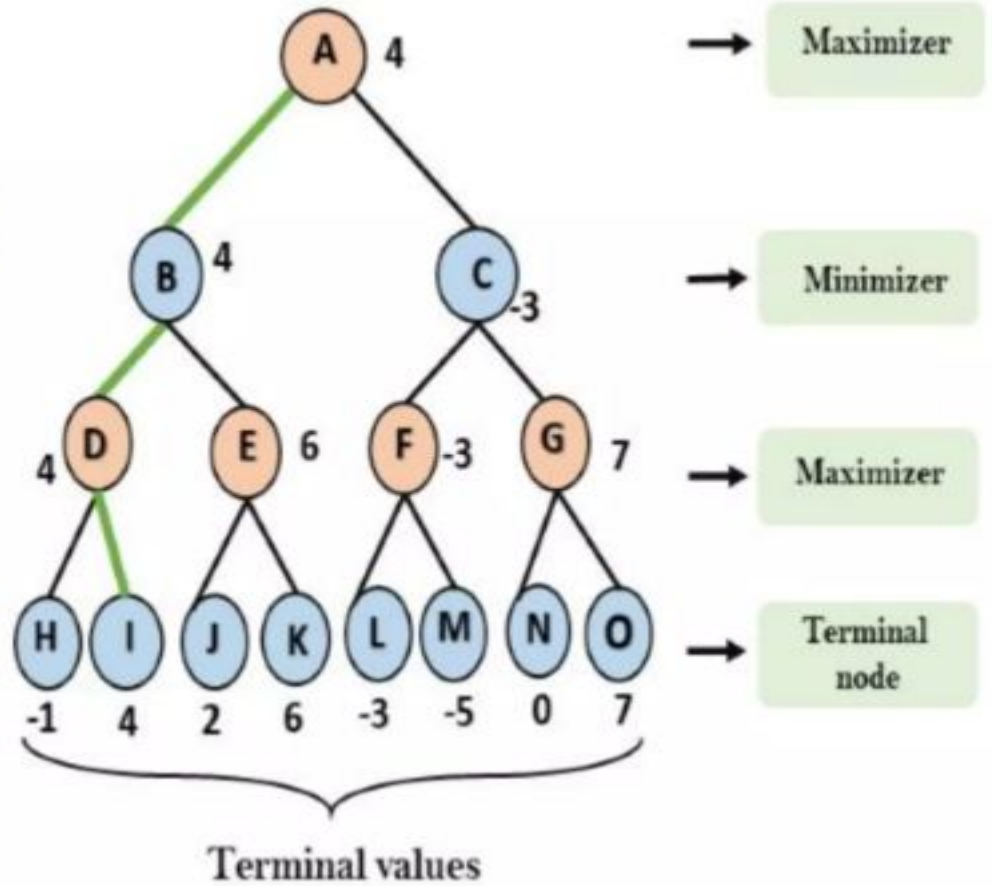
WORKING OF MIN-MAX ALGORITHM:

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



Working of Min-Max Algorithm:

- **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, -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(bm), 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 which we have discussed in the next topic.

ALPHA-BETA PRUNING IN AI

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half.
- 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.

ALPHA-BETA PRUNING IN AI

The two-parameter can be defined as:

- **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞.
- **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. 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 by pruning these nodes, it makes the algorithm fast.

ALPHA-BETA PRUNING IN AI

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.

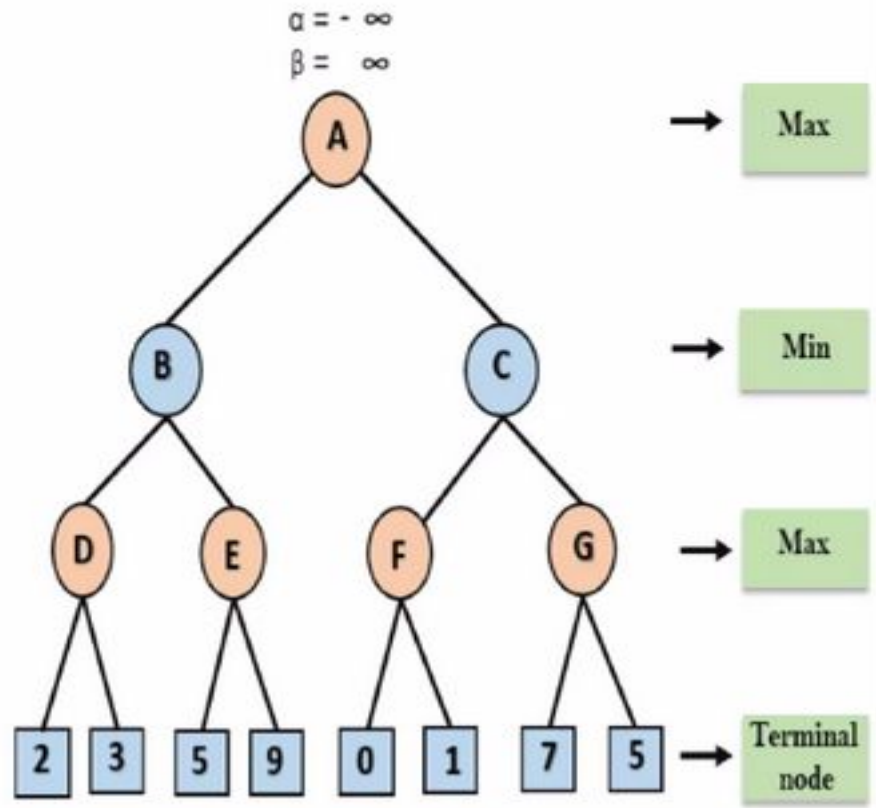
WORKING OF ALPHA-BETA PRUNING:

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 $\alpha = -\infty$ and $\beta = +\infty$, these value of alpha and beta passed down to node B where again $\alpha = -\infty$ and $\beta = +\infty$, and Node B passes the same value to its child D.



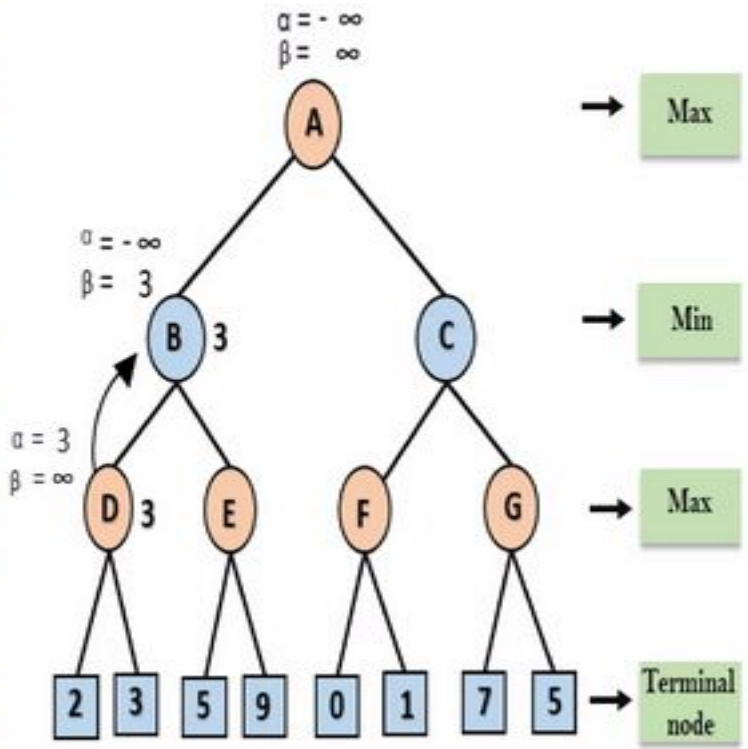
WORKING OF ALPHA-BETA PRUNING:

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:

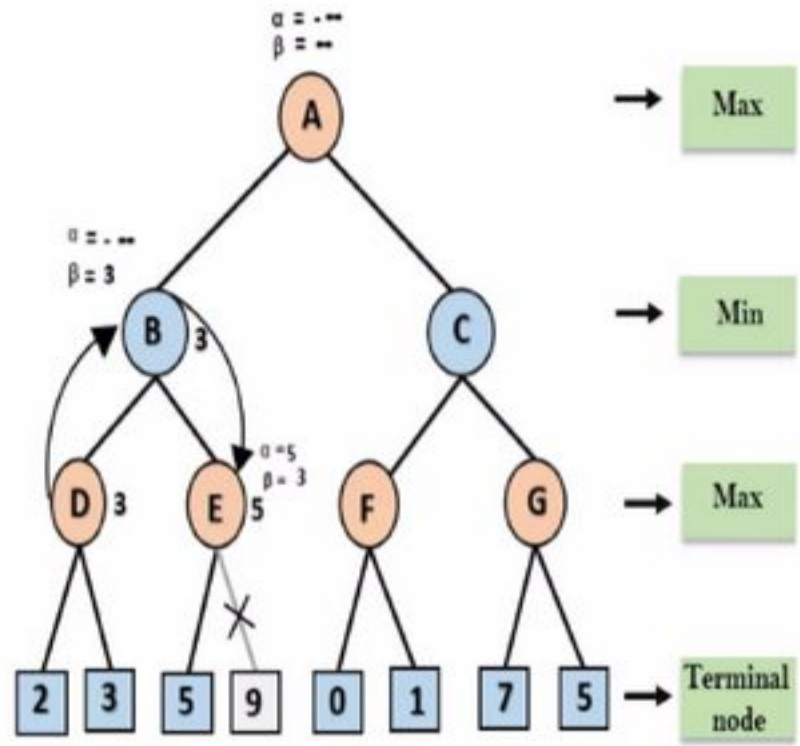
- Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.
- In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed.



WORKING OF ALPHA-BETA PRUNING:

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.

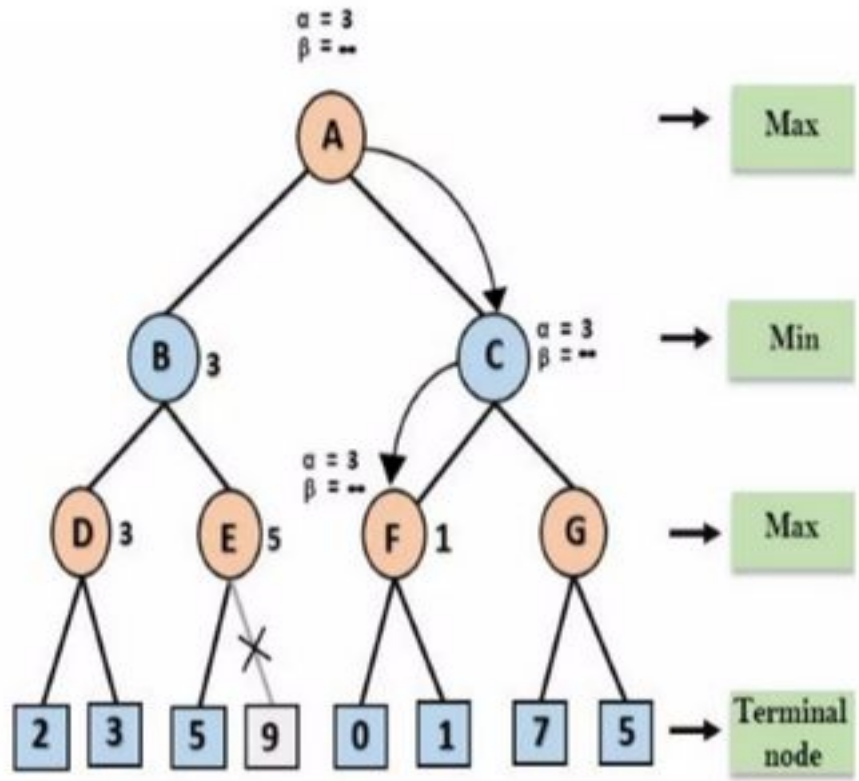


WORKING OF ALPHA-BETA PRUNING:

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.

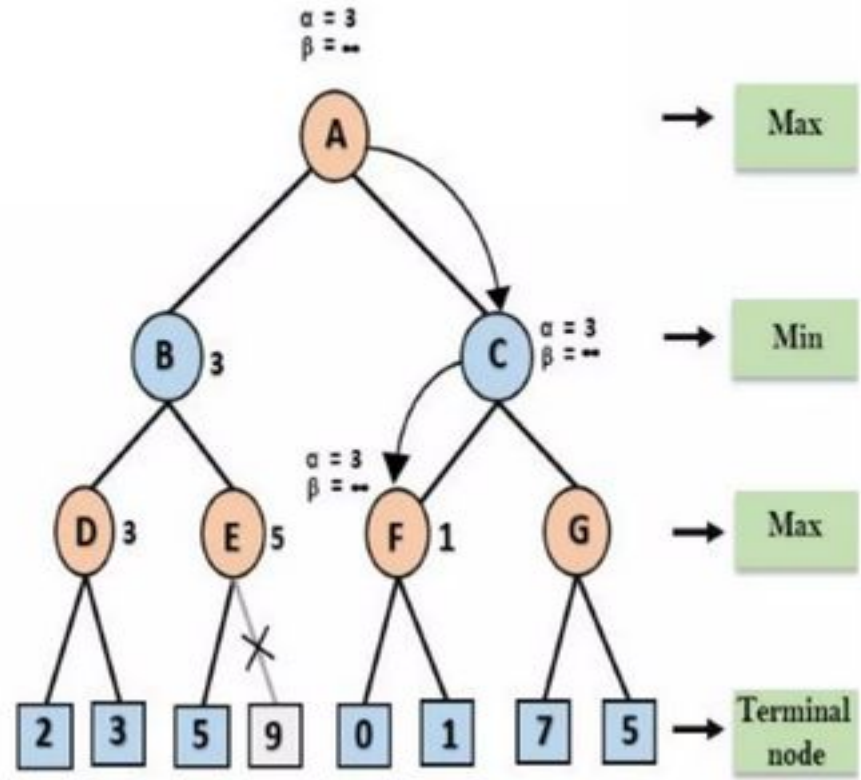
At node C, α=3 and β= +∞, and the same values will be passed on to node F.



WORKING OF ALPHA-BETA PRUNING:

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.

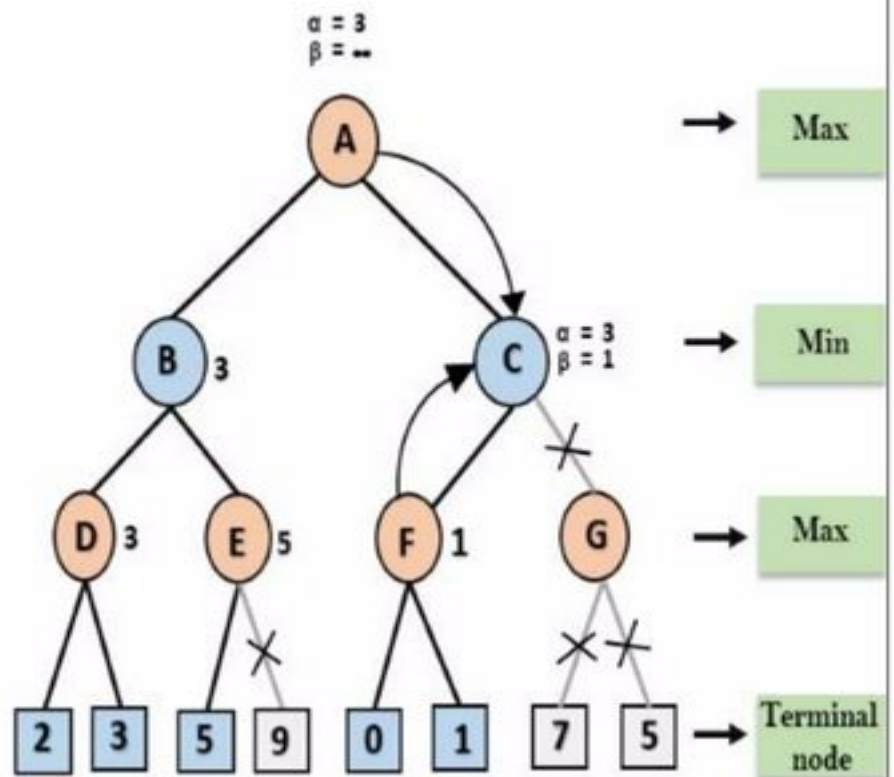


WORKING OF ALPHA-BETA PRUNING:

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.



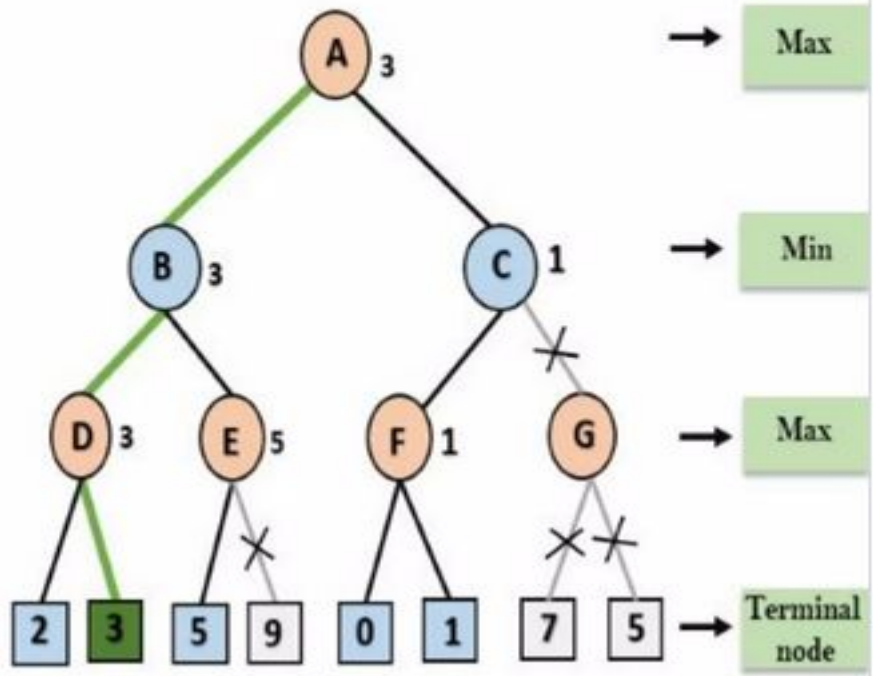
WORKING OF ALPHA-BETA PRUNING:

Step 8:

- Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3.

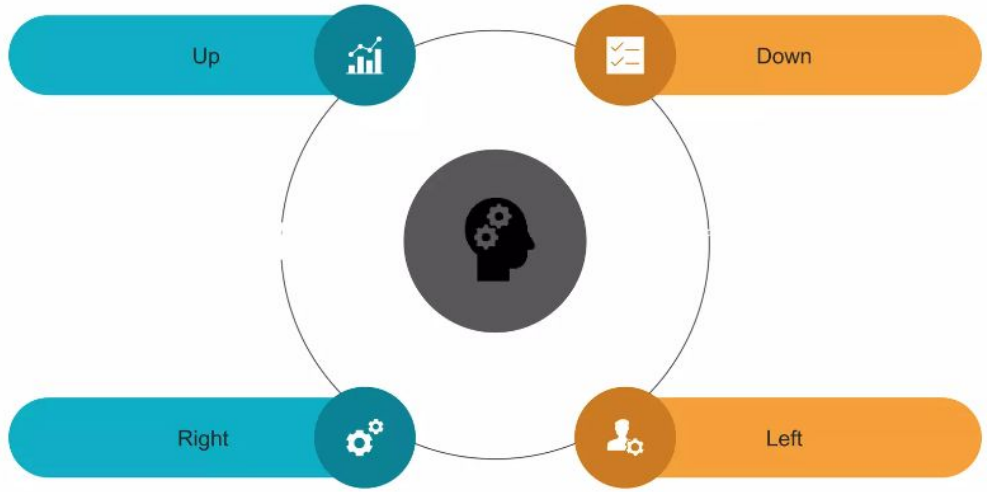
- Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed.

- Hence the optimal value for the maximizer is 3 for this example.



❑ What is 8 puzzle Problem?

➢ The 8-puzzle problem is a puzzle invented and popularized by Noyes Palmer Chapman in the 1870s. It is played on a 3-by-3 grid with 8 square blocks labeled 1 through 8 and a blank square.



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

2	8	3
1	6	4
7		5

Initial State

1	2	3
8		4
7	6	5

Final State

Initial State

Final State

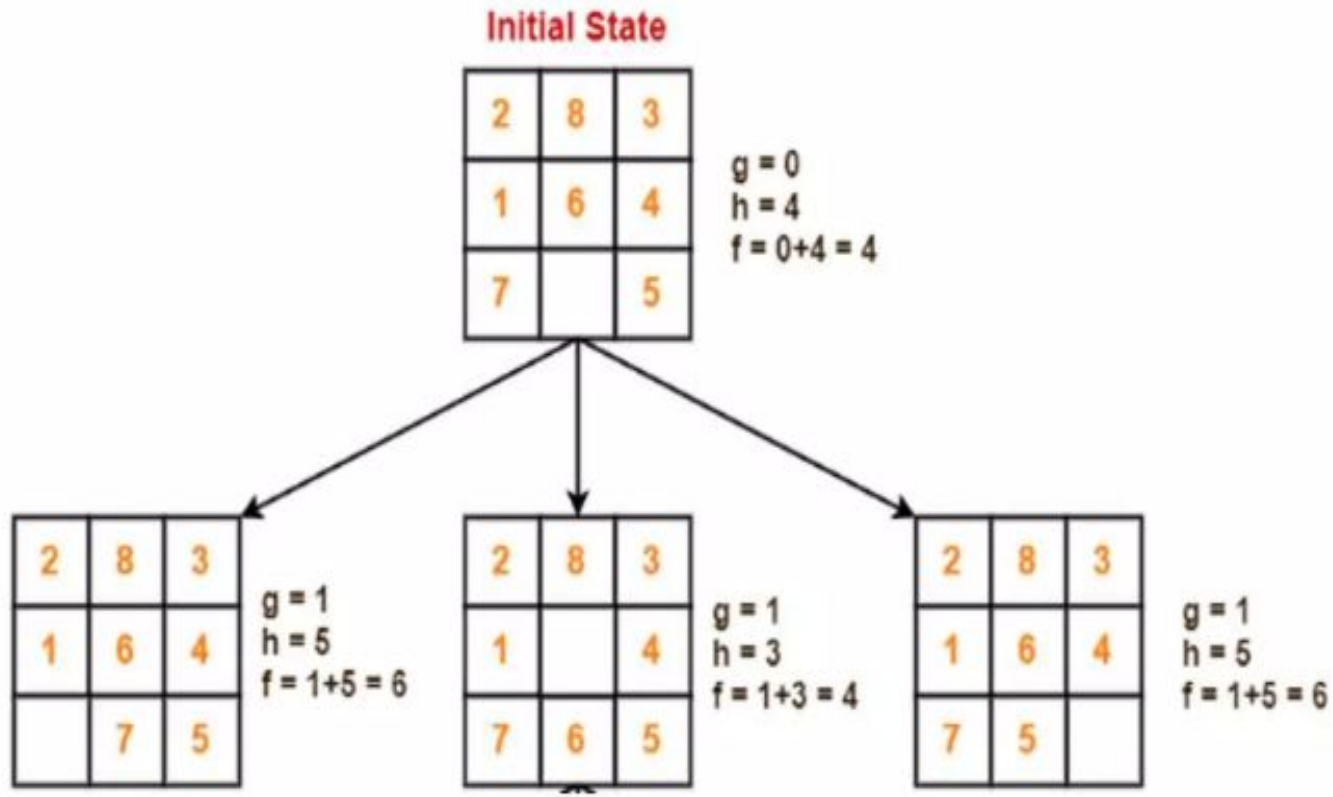
Initial State

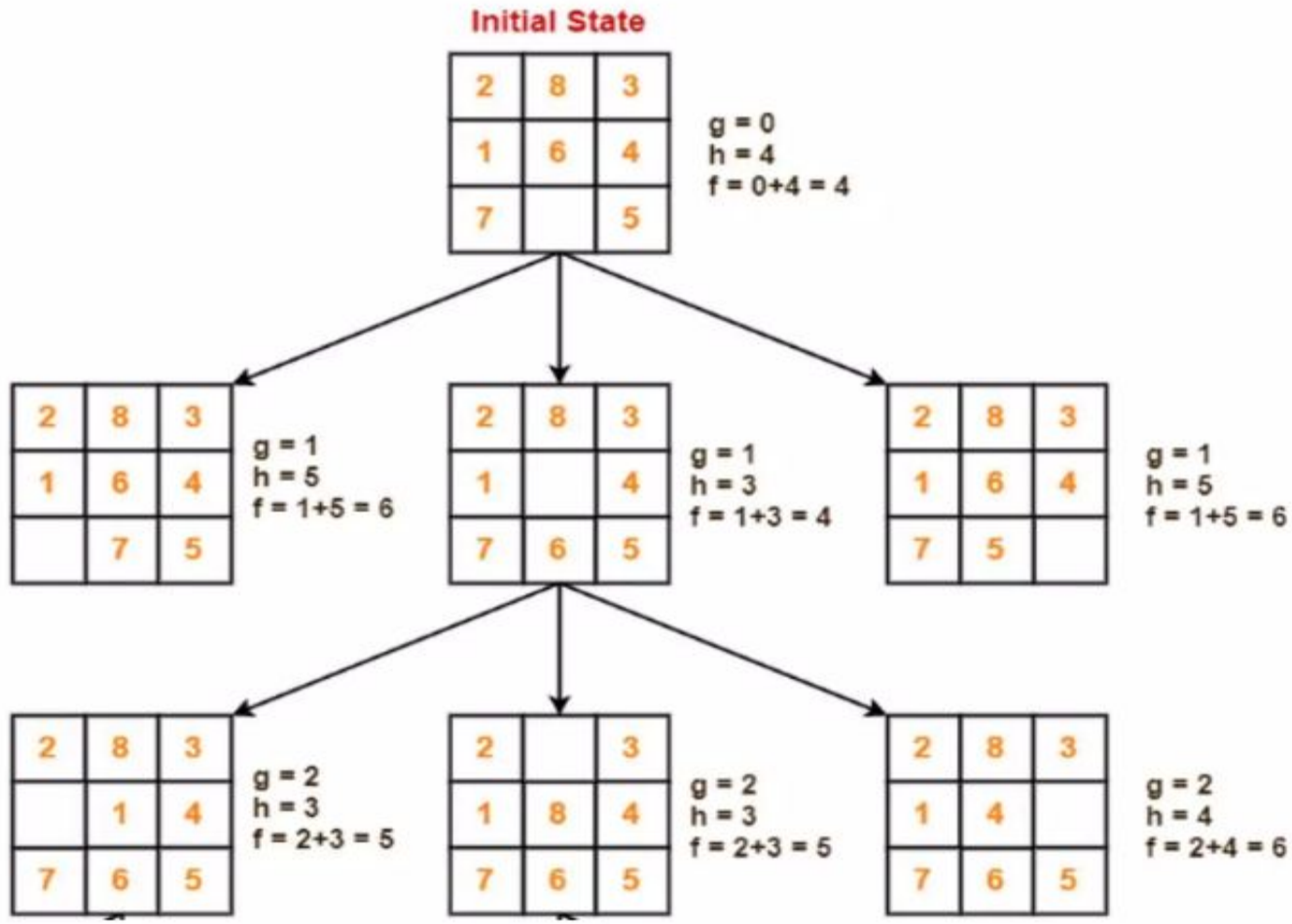
2	8	3
1	6	4
7		5

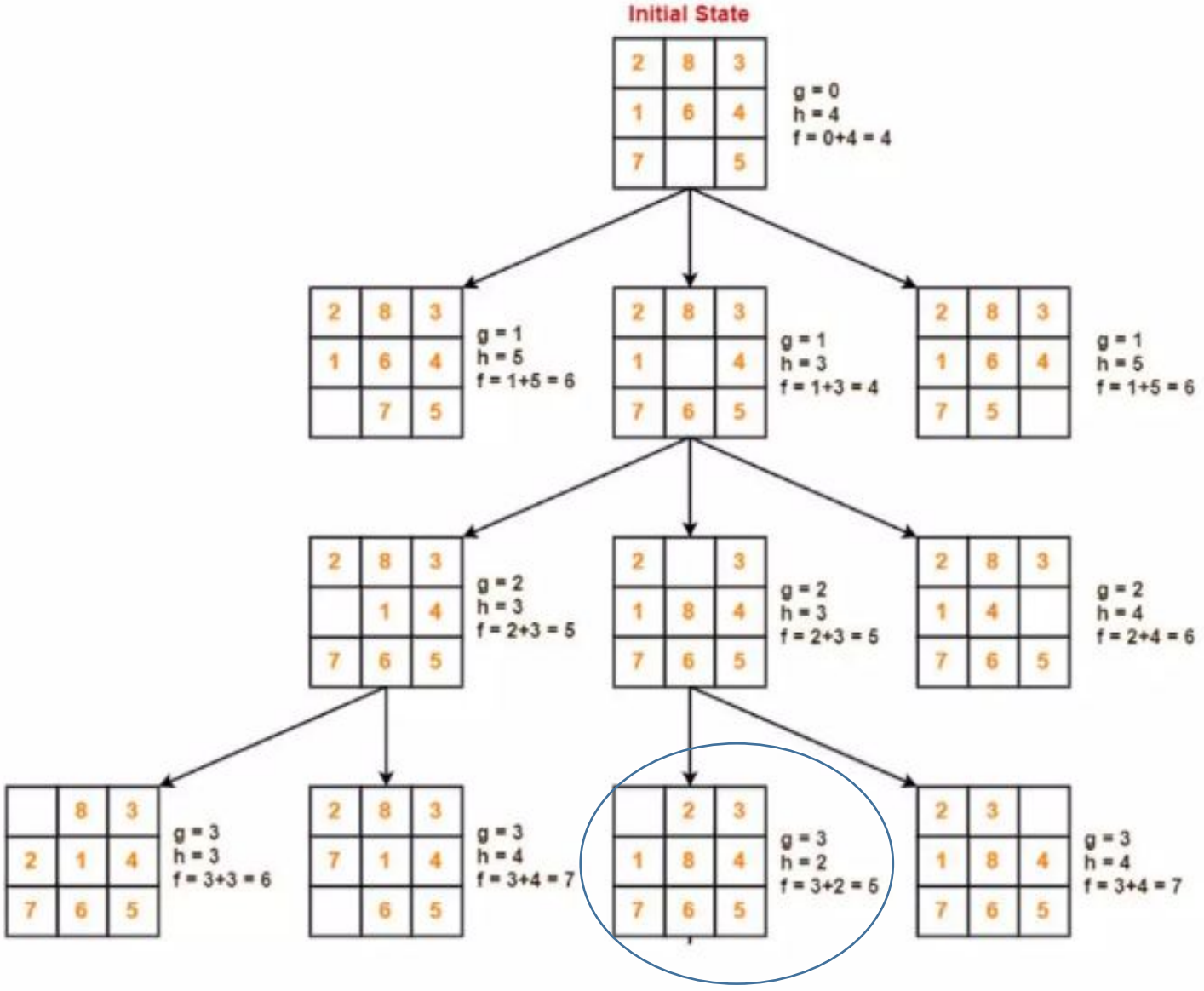
g = 0
h = 4
f = 0+4 = 4

Consider ,
g(n) = Depth of node
and
h(n) = Number of misplaced tiles.

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


Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6


Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 3
1 8 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 8 3
1 4
7 6 5
g = 2
h = 4
f = 2+4 = 6


Initial State
2 8 3
1 6 4
7 5
g = 0
h = 4
f = 0+4 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 1
h = 3
f = 1+3 = 4
2 8 3
1 6 4
7 5
g = 1
h = 5
f = 1+5 = 6
2 8 3
1 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 3
1 8 4
7 6 5
g = 2
h = 3
f = 2+3 = 5
2 8 3
1 4
7 6 5
g = 2
h = 4
f = 2+4 = 6
8 3
2 1 4
7 6 5
g = 3
h = 3
f = 3+3 = 6
2 8 3
7 1 4
6 5
g = 3
h = 4
f = 3+4 = 7
2 3
1 8 4
7 6 5
g = 3
h = 2
f = 3+2 = 5
2 3
1 8 4
7 6 5
g = 3
h = 4
f = 3+4 = 7

https://slideplayer.com/slide/7049935/
https://www.slideshare.net/vikasdhakane/i-minimax-algorithm-in-ai

Minimax: How Computers Play Games (youtube.com)

https://www.youtube.com/watch?v=NiY32wS2UVw
https://www.youtube.com/watch?v=mXqp6-TK7m8