ARTIFICIAL INTELLIGENCE

	L	T	P	C
BCSE306L -	3	0	0	3

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

Dr. S M SATAPATHY

Associate Professor Sr.,
School of Computer Science and Engineering,
VIT Vellore, TN, India – 632 014.

Module – 3

LOCAL SEARCH & ADVERSARIAL SEARCH

1. Local Search
2. Adversarial Search
3. Mini-max algorithm
4. Alpha-Beta Pruning

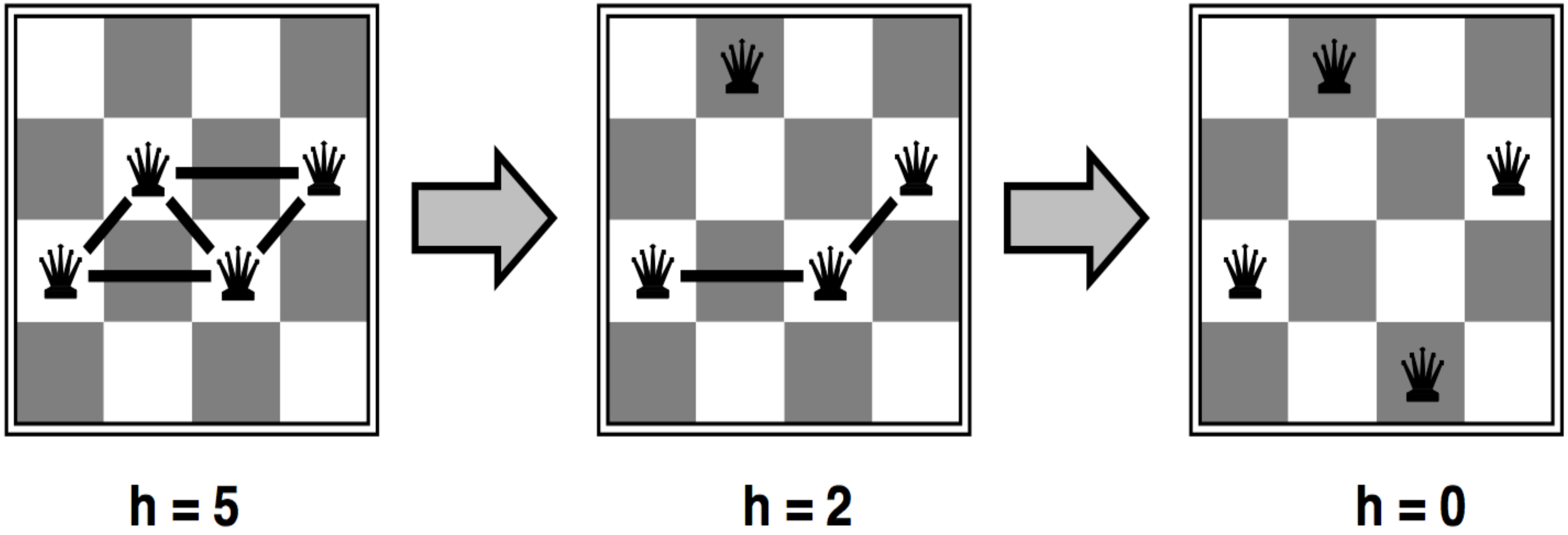
LOCAL SEARCH

Local Search :: Introduction

- Heuristic: It is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution
- Local search applied mostly to problems for which we don't need to know the path to the solution but only the solution itself.
- They operate using a single state or a small number of states and explore the neighbours of that state. They usually don't store the path.
- A particular case are optimization problems for which we search for the best solution according to an objective function. The distribution of values of the objective function in the state space is called a landscape.
- In many optimization problems, the path to the goal is irrelevant; the goal state itself is the solution
- State space = set of "complete" configurations
- Find configuration satisfying constraints, e.g., n-queens

Local Search :: Example - n-queens

❖ Put n queens on an n × n board with no two queens on the same row, column, or diagonal



Advantages:

❖ Memory requirement is less (as total search space is not searched) than normal blind search

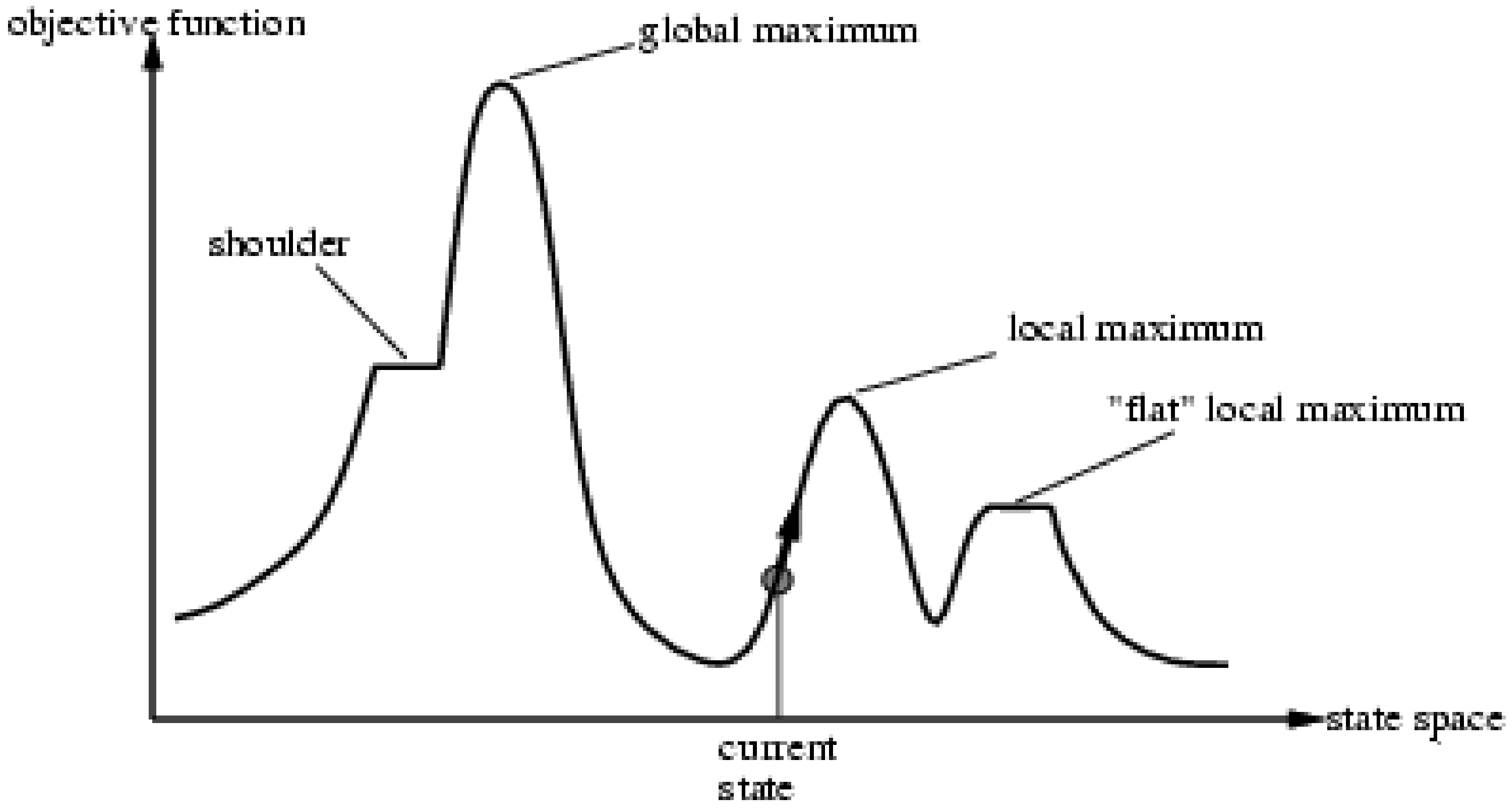
❖ They normally find reasonably fast solution in big search space

Local Search :: Terminologies

- A Landscape: both location (defined by the state) and elevation defined by the value of the heuristic cost function or objective function
- Global minimum: To find the lowest valley (if elevation corresponds to cost).
- Global maximum: Finding the highest peak (if elevation corresponds to an objective function). a state that maximizes the objective function over the entire landscape. A complete local search algorithm always finds a goal if one exists; an optimal algorithm always finds a global minimum/maximum.
- Local Maximum: A local maximum is a peak that is higher than each of its neighbouring states but lower than global maximum.
- Ridges: A sequence of local maxima that is very difficult for navigate
- Plateaux: A flat area of the state space landscape where the evaluation function is flat

Local Search :: Terminologies

State space landscape



Local Search :: Technique

1) Start with initial state. If it is not a goal state, go to step 2

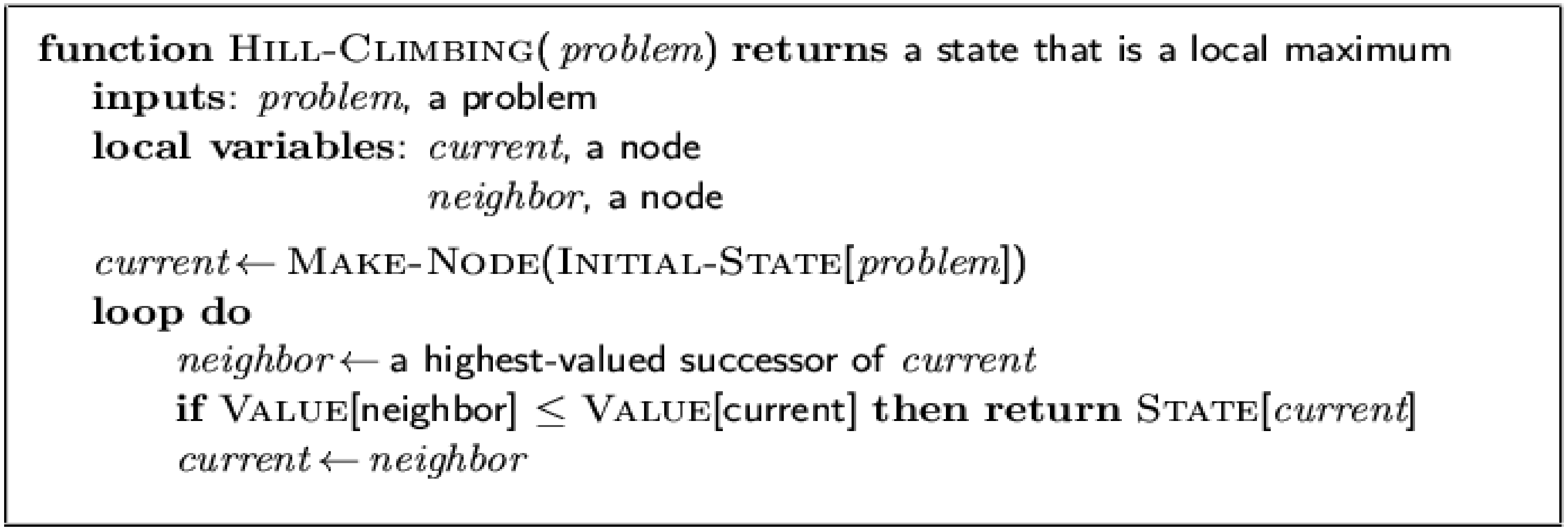
2) Repeat the following till either goal state is reached or applicable operations are over:
   1. Select yet unapplied operator, apply it to produce new state, which moves in the direction of increasing value of evaluation function.
   2. Evaluate new state. If it is goal state or no operator generates state having value of objective function more than previous one, then terminate.

Local Search :: Types

- Hill-climbing Search
- Simulation Annealing Search
- Local Beam search
- Genetic Algorithm

Local Search :: Hill-climbing Search

- It always moves towards the goal
- Using heuristics it finds which direction will take it closest to the goal
- It is actually combination of Generate-and-test + Direction to move
- Here the heuristic function is to estimate how close a given state is to a goal state
- "Like climbing Everest in thick fog with amnesia"
- Sometimes called greedy local search



```
function HILL-CLIMBING(problem) returns a state that is a local maximum
    inputs: problem, a problem
    local variables: current, a node
                     neighbor, a node

    current ← MAKE-NODE(INITIAL-STATE[problem])
    loop do
        neighbor ← a highest-valued successor of current
        if VALUE[neighbor] ≤ VALUE[current] then return STATE[current]
        current ← neighbor
```

Local Search :: Hill-climbing Algorithm

STEP 1: Evaluate the initial state, if it is goal state then quit otherwise make current state as initial state

STEP 2: Select a new operator that could be applied to this state and generate a new state

STEP 3: Evaluate the new state if this new state is closer to the goal state, then current state make the new state as the current if it is not better ignore this state and proceed with the current state

STEP 4: If the current state is goal state or no new operators are available, Quit. Otherwise go to STEP 2.

Local Search :: Hill-climbing Algorithm

```
function HILL-CLIMBING(problem) returns a state
    current ← make-node(problem.initial-state)
    loop do
        neighbor ← a highest-valued successor of current
        if neighbor.value ≤ current.value then
            return current.state
        current ← neighbor
```

Local Search :: Hill-climbing Search

- Stochastic hill climbing: chooses at random from among the uphill moves
- First-choice hill climbing: implements stochastic hill climbing by generating successors randomly until one is generated that is better than the current state. This is a good strategy when a state has many (e.g., thousands) of successors.
- Steepest Ascent Hill Climbing: This differs from the basic Hill climbing algorithm by choosing the best successor rather than the first successor that is better. This indicates that it has elements of the breadth first algorithm.
- Disadvantage: The hill-climbing algorithms described so far are incomplete—they often fail to find a goal when one exists because they can get stuck on local maxima.

Local Search :: Hill-climbing Search

- Random-restart hill climbing – complete algorithm - conducts a series of hill-climbing searches from randomly generated initial states, until a goal is found.
- Random sideways moves - Escape from shoulders, but loop forever on flat local maxima.

- A hill-climbing algorithm that never makes "downhill" moves toward states with lower value (or higher cost) is guaranteed to be incomplete, because it can get stuck on a local maximum.
- In contrast, a purely random walk—that is, moving to a successor chosen uniformly at random from the set of successors—is complete but extremely inefficient.
- Therefore, it seems reasonable to try to combine hill climbing with a random walk in some way that yields both efficiency and completeness.

Local Search :: Simulated annealing search

- In metallurgy, annealing is the process used to temper or harden metals and glass by heating them to a high temperature and then gradually cooling them, thus allowing the material to reach a low energy crystalline state
- Gradient descent – minimizing the cost
- Instead of picking the best move, it picks a random move
- It makes the procedure less sensitive to the starting point
- It avoid false foot hills based on the following changes done in this approach
  - Rather than creating maxima; minimisation is done
  - The term objective function is used rather than heuristic
- Idea: escape local maxima by allowing some "bad" moves but gradually decrease their frequency

Local Search :: Simulated annealing search Algorithm

STEP 1: Start with evaluating thee initial state

STEP 2: Apply each operator and loop until a goal state is found or till no new operators left to be applied:

```
    Set T according to an annealing schedule
    Select and applies a new operator
    Evaluate the new state: If it is a goal state quit; Otherwise
        $\Delta E$ = Val (current state) – Val (new state)
        If $\Delta E < 0$ then this is the new current state
        Else find a new current state with probability $e^{-E/T}$
```

Local Search :: Simulated annealing search

- One can prove: If T decreases slowly enough, then simulated annealing search will find a global optimum with probability approaching 1

- Widely used in VLSI layout, airline scheduling and other large scale optimization

Local Search :: Local Beam search

- In this search, instead of single state in memory K states are kept in memory
- Here states are generated in random fashion
- A successor function plays an important role by generating successor of all K states
- If any one successor state is goal state then no further processing is required
- In other case i.e. if goal state is not achieved it observes the K best successors from the list (current states and successor states) and process is repeated

Local Search :: Local Beam search

- Keep track of k states rather than just one
- Start with k randomly generated states
- At each iteration, all the successors of all k states are generated
- If any one is a goal state, stop; else select the k best successors from the complete list and repeat.

LIMITATIONS

- Lack of variation among the K states
- If the state concentrate on small area of state space then search becomes more expensive

Local Search :: Stochastic Beam search

- It’s a flavour of Local Beam search it resolve the limitations exist in Local Beam search
- It concentrate/focus on random selection of K successor instead of selecting K best successor from candidate successor
- The probability of random selection is increasing function of its success rate
- It is very similar to natural selection, whereby the “successors” (offspring) of a “state” (organism) populate the next generation according to its “value” (fitness).

Local Search :: Genetic Algorithm (GA)

- Variant of stochastic beam search
- A successor state is generated by combining two parent states
- Start with k randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation

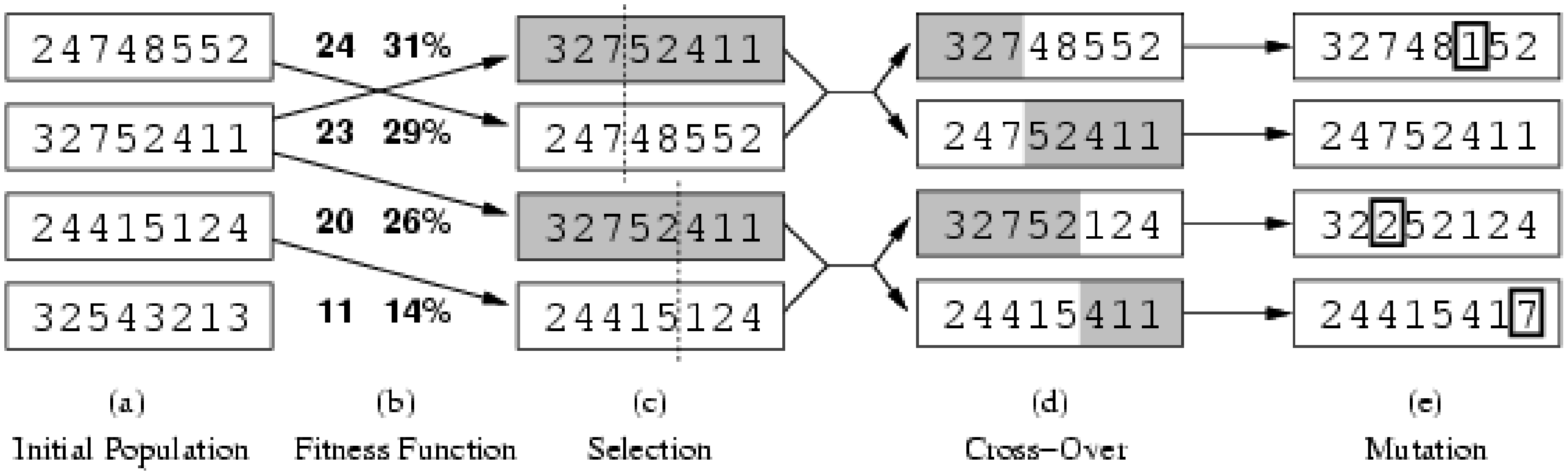
Local Search :: Genetic Algorithm (GA)

- Population: Population is set of states which are generated randomly
- Individual: Each state or individual is a string of finite alphabet (0’s & 1’s)
- Fitness function: The evaluation function which specifies the rating of each state is called fitness function
- Crossover: For each state pairs are divided that division point or meeting point is called crossover point
- Mutation: Mutation is one of the genetic operation. It works on random selections or changes.
- Schema: The schema is a substring in which position of some bit can be unspecified
- Instance: Strings that match the schema are called instances

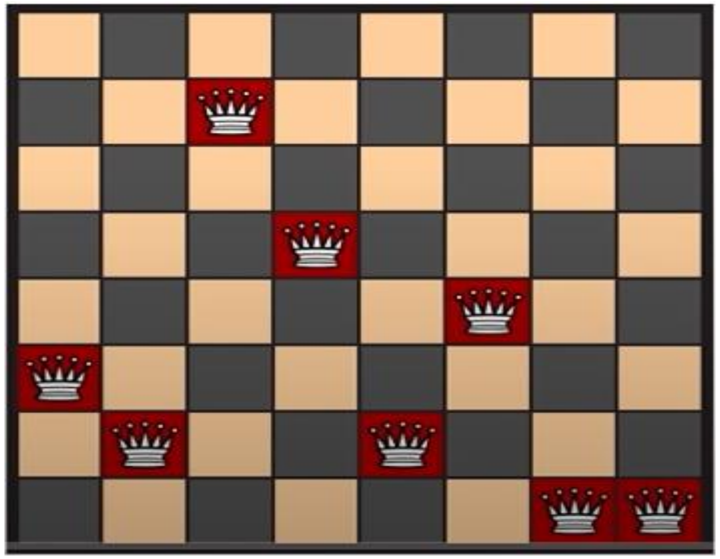
Local Search :: Genetic Algorithm (GA) Representation

1) Use number of state population or a set of individual
2) Use fitness function, function of rating
3) Create an individual ‘x’ (parent) by using random selection with fitness function ’A’ (rate A)
4) Create an individual ‘y’ (parent) by using random selection with fitness function ’B’ (rate B)
5) Child with good fitness is created for x+y
6) For small probability apply mutate operator on child
7) Add child to new population
8) The above process is repeated until child (an individual) in not fit as specified fitness function

Local Search :: Genetic Algorithm (GA)



- Fitness function: number of non-attacking pairs of queens (min = 0, max = 8 × 7/2 = 28)



- 24/(24+23+20+11) = 31%
- 23/(24+23+20+11) = 29% etc.,

ADVERSARIAL SEARCH

Adversarial Search :: Introduction

- In adversarial search we examine the problems that arise when we try to plan ahead in a world where other agents are planning against us.
- Adversarial search problems: agents have conflicting goals
  - Example: Game Playing

Game Playing :: Introduction

- Game playing is one of the oldest sub-filed of AI.
- Game playing involves abstract and pure form of competition that seems to require intelligence
- It is easy to represent the state and actions.
- The most common used AI technique in game is search
- Game playing research has contributed ideas on how to make the best use of time to reach good decisions
  - Game playing is a search problem defined by:
  - Initial state of the game
  - Operators defining legal moves
  - Successor function
  - Terminal test defining end of game states
  - Goal test
  - Path cost/ utility/ payoff function

Characteristics of game playing

- There is always an “Unpredictable” opponent:
  - Due to their uncertainty
  - He/she also wants to win
  - Solution for each problem is a strategy, which specifies a move for every possible opponent reply
- Time Limits:
  - Games are often played under strict time constraints and therefore must be very effectively handled
  - There are special games where the two players have exactly opposite goals.
  - It can be divided into two types
    - Perfect Information games (where both players have access to the same information)
    - Imperfect Information games (different information can be accessed)

Optimal Decisions in Games

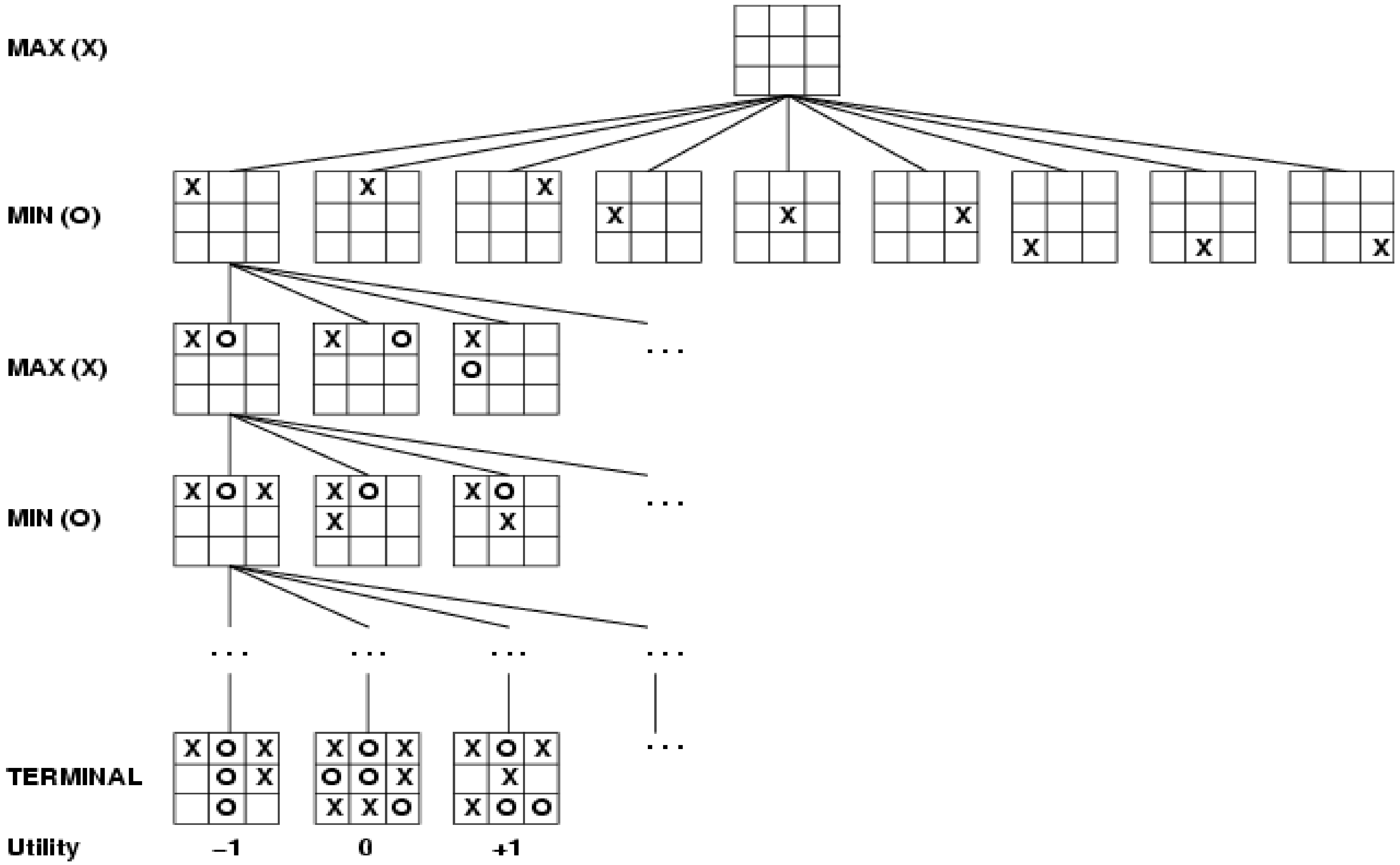
- MIN and MAX are two game players
- First move performed by MAX then turn is given to MIN and so on until the game is over
- A loser need to pay penalties and game points or credits are gifted to the winning player, at end of the game
- A game is defined as a search problem and broadly defined by the following components
  - The Initial State: It is start state position on the board, which leads to further move
  - A successor function: It is collection of (move, state) pairs, each specifies a legal move and output state. Here actions are considered
  - Terminal Test: It is final state which declares that the game is over or ended or terminal states

Optimal Decisions in Games

- Game Tree: Game tree is graphical representation of the initial state and legal moves for each side (two players – one by one) for a specific game
- MAX has nine possible move from initial state
- MAX with X and MIN with O symbol. The leaf nodes are entitled by utility value
- The MAX is assumed with high values and MIN is with low values. The best move is always determined by MAX by using search tree
- Optimal Strategies
  - A sequence of moves which achieves Win state/ Terminal state/ Goal state in search problem
- Optimal Strategy in Game
  - It is techniques which always leads to superior that any other strategy as opponent is playing in perfect manner

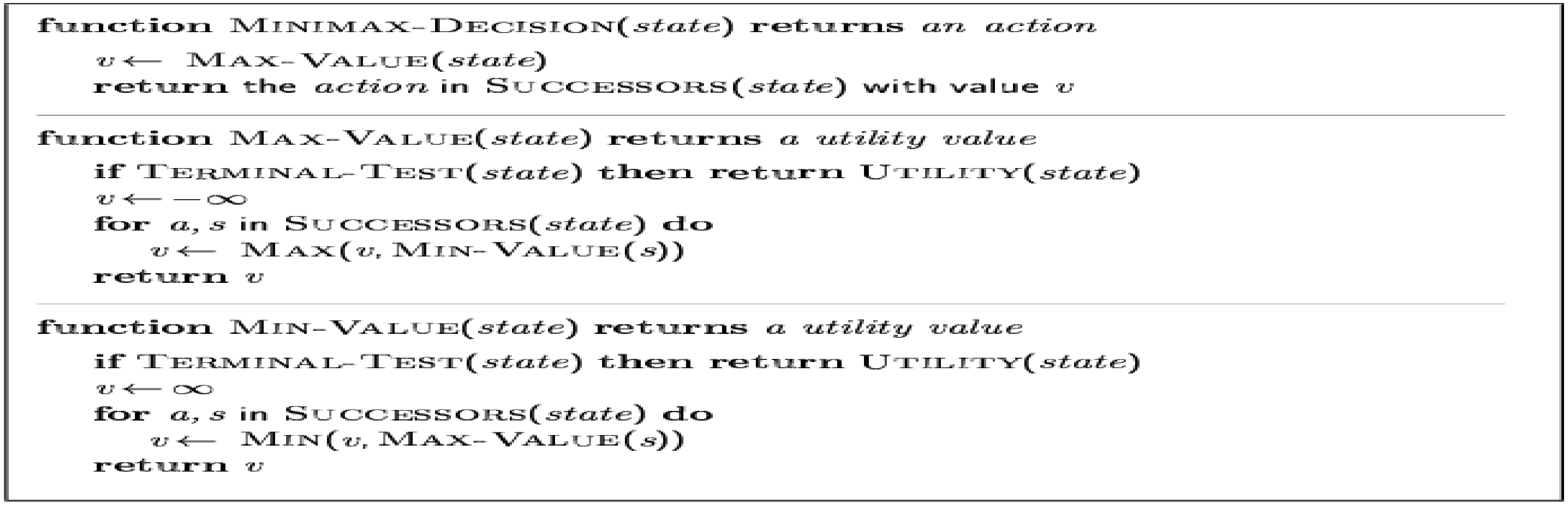
Optimal Decisions in Games

- Game tree (2-player, deterministic, turns)



MINIMAX Procedure

- Starting from the leaves of the tree (with final scores with respect to one player, MAX), and go backwards towards the root
- At each step, one player (MAX) takes the action that leads to the highest score, while the other player (MIN) takes the action that leads to the lowest score
- All nodes in the tree will be scored, and the path from root to the actual result is the one on which all nodes have the same score

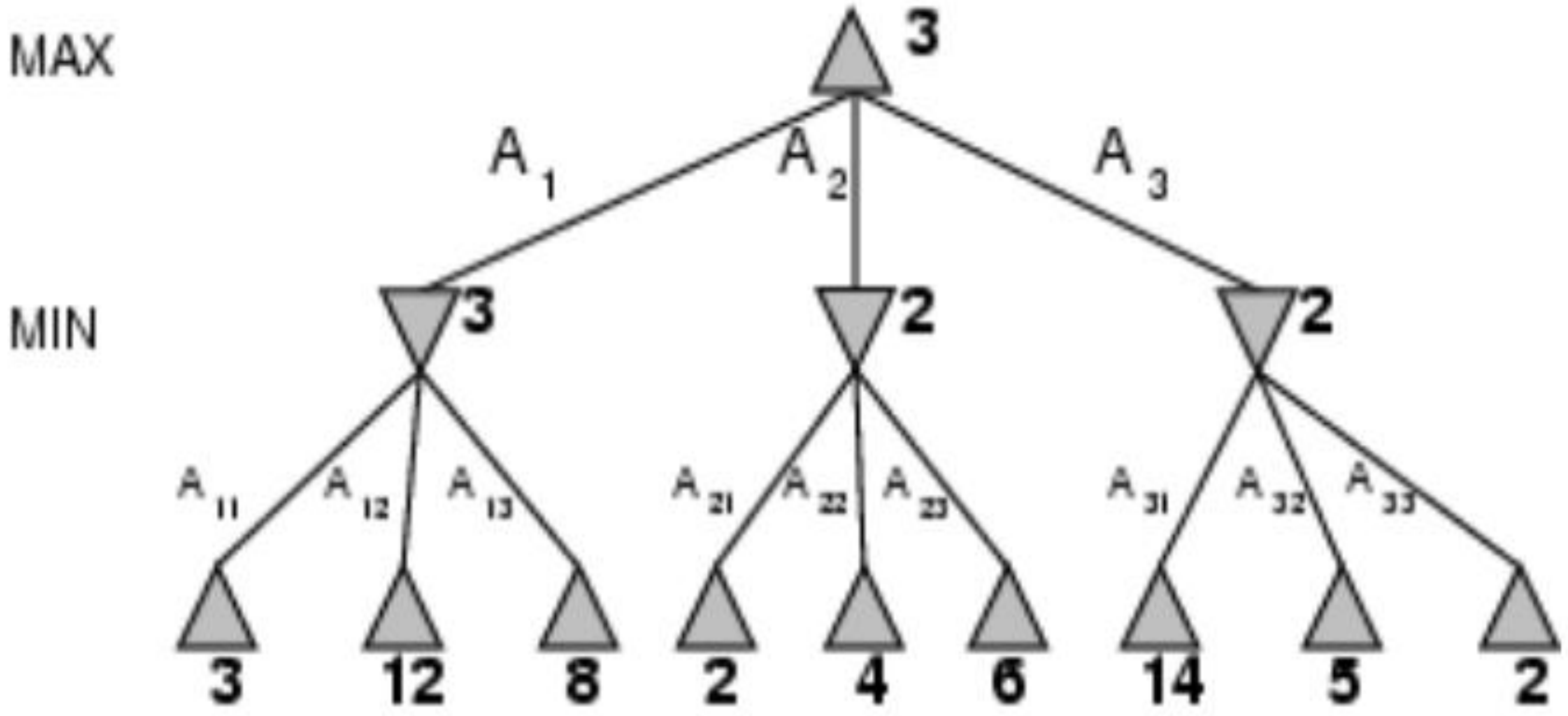


```
function MINIMAX-DECISION(state) returns an action
    $v \leftarrow$ MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value $v$

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow -\infty$
    for $a, s$ in SUCCESSORS(state) do
        $v \leftarrow$ MAX($v$, MIN-VALUE($s$))
    return $v$

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow \infty$
    for $a, s$ in SUCCESSORS(state) do
        $v \leftarrow$ MIN($v$, MAX-VALUE($s$))
    return $v$
```

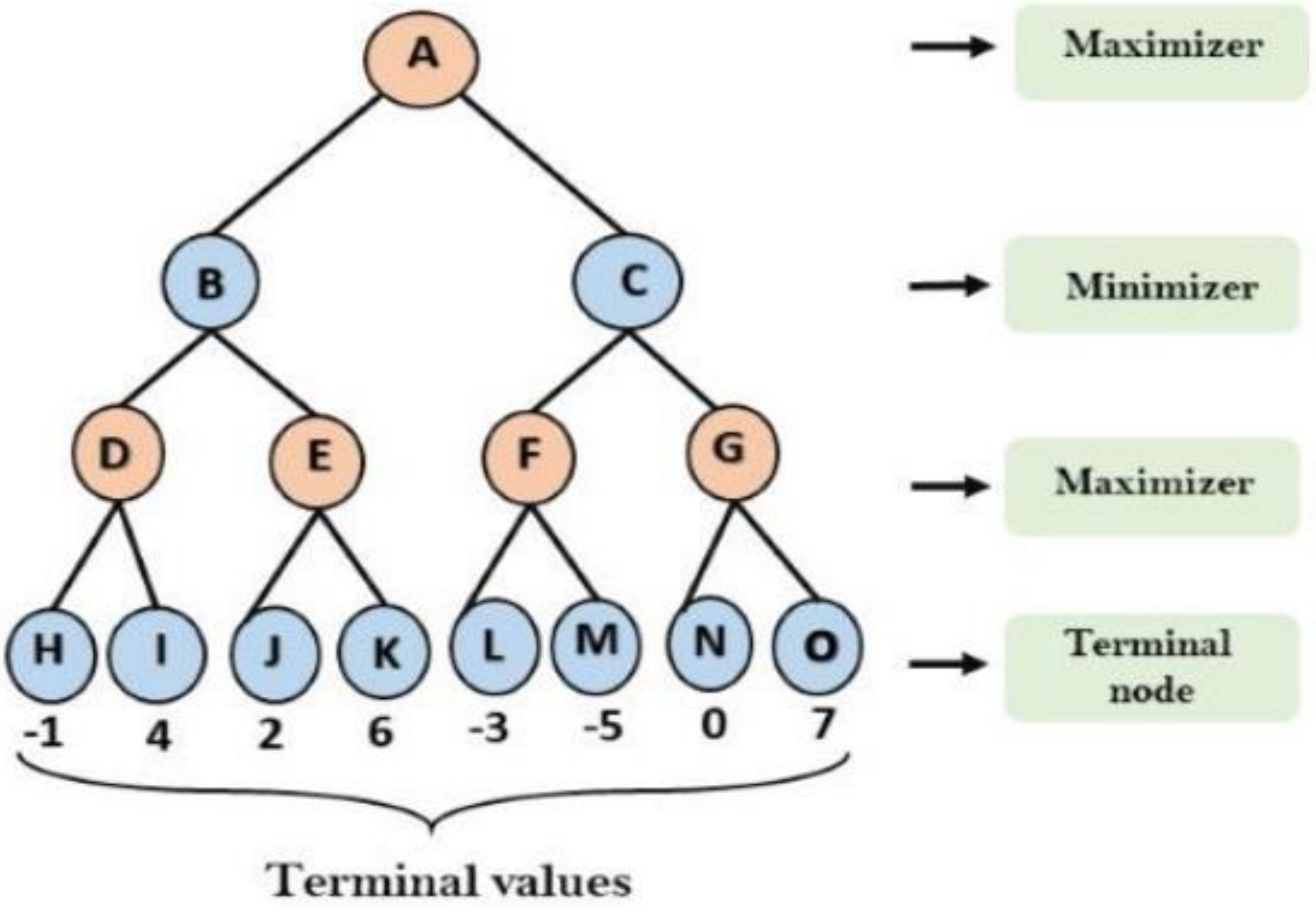
MINIMAX Procedure

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value

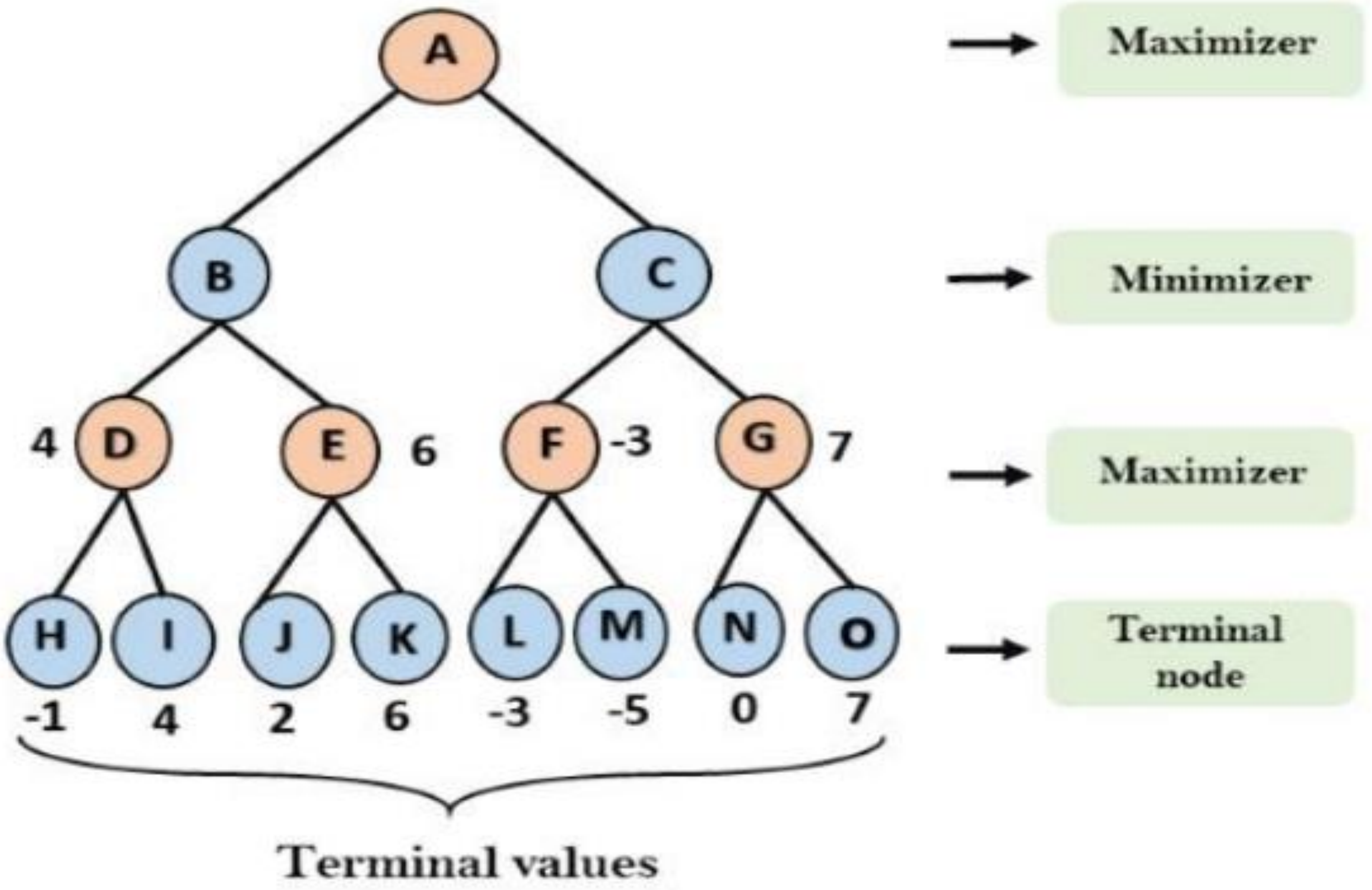
  = best achievable payoff against best play
- E.g., 2-ply game:



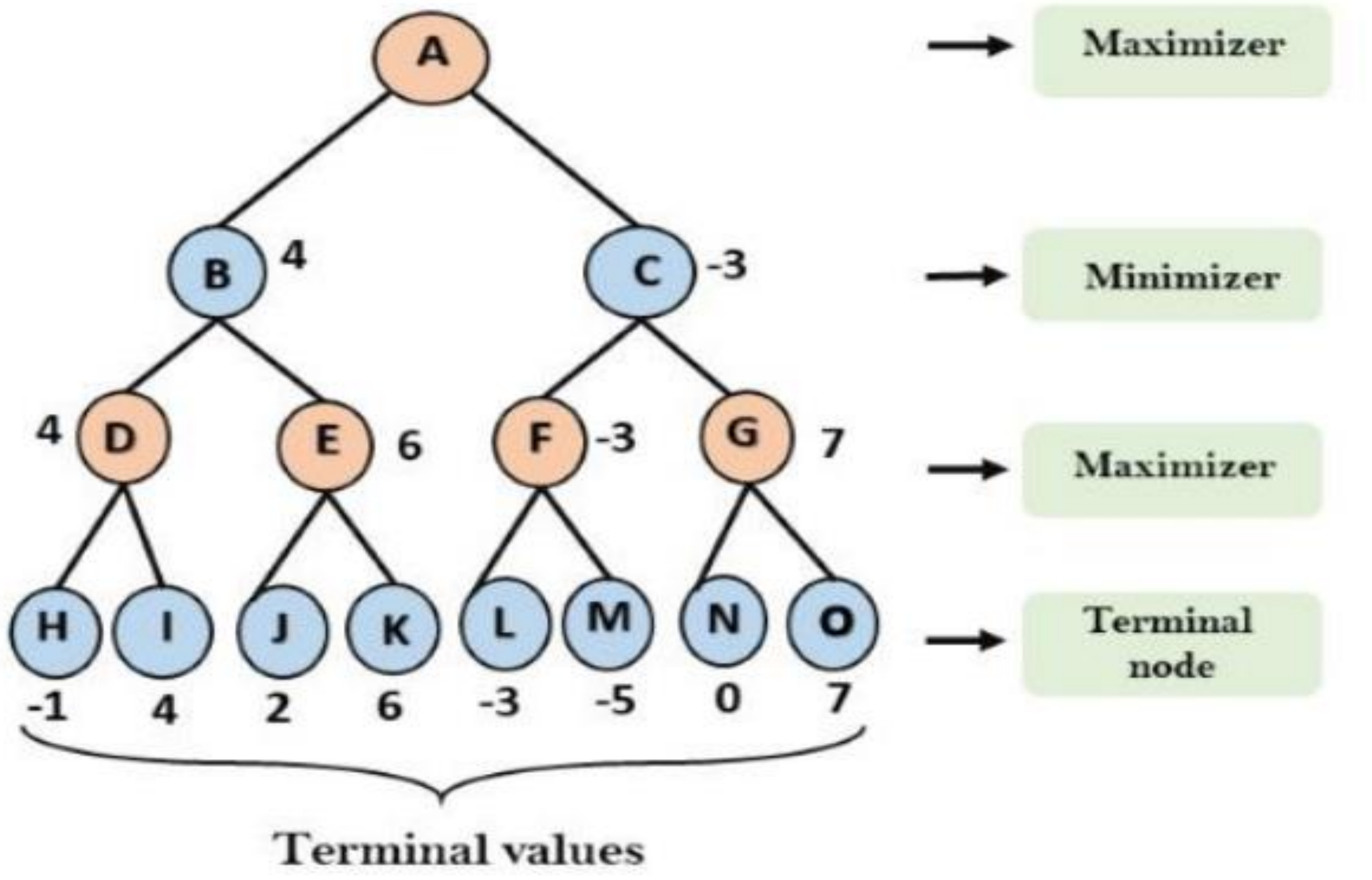
MINIMAX Procedure



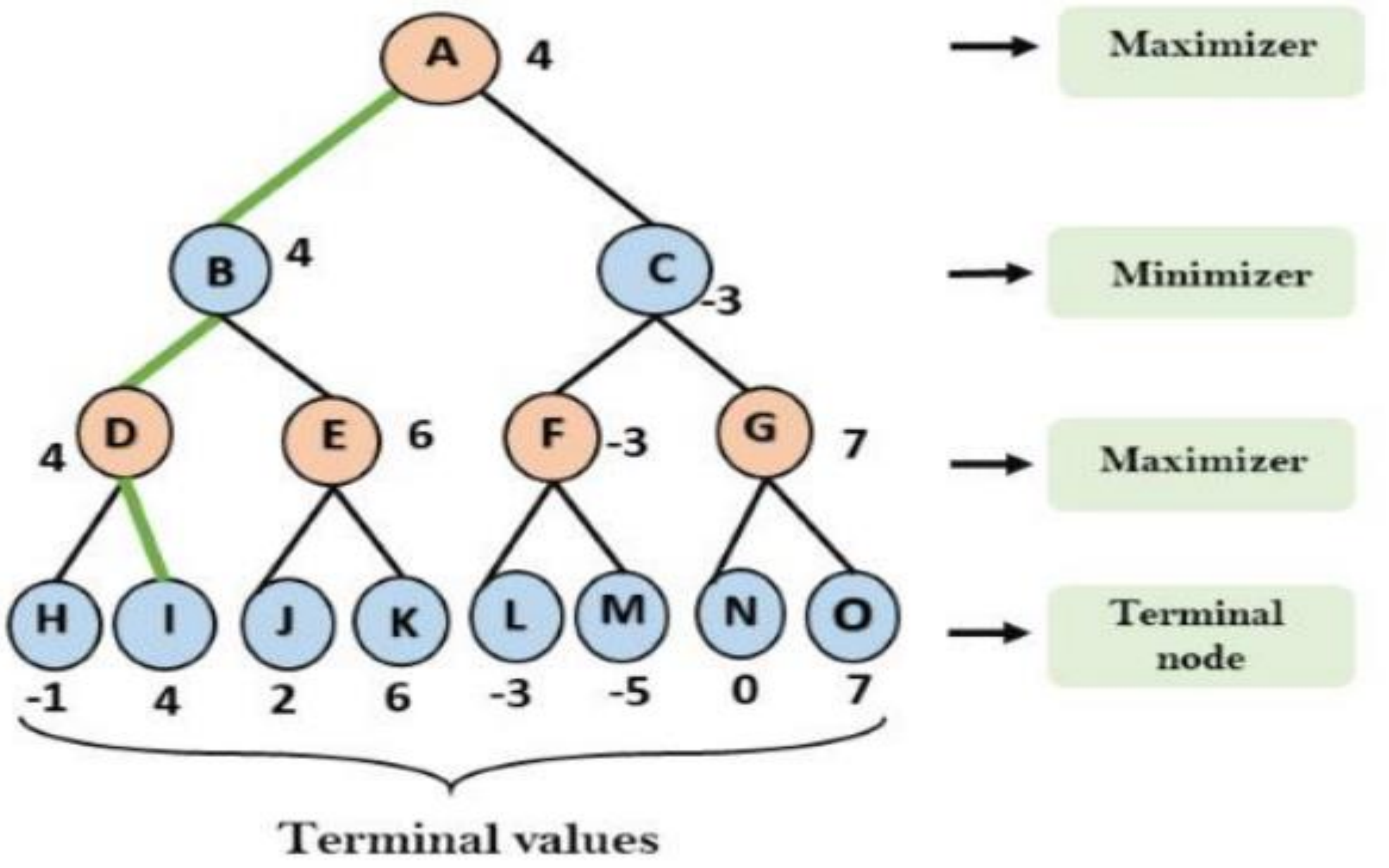
MINIMAX Procedure



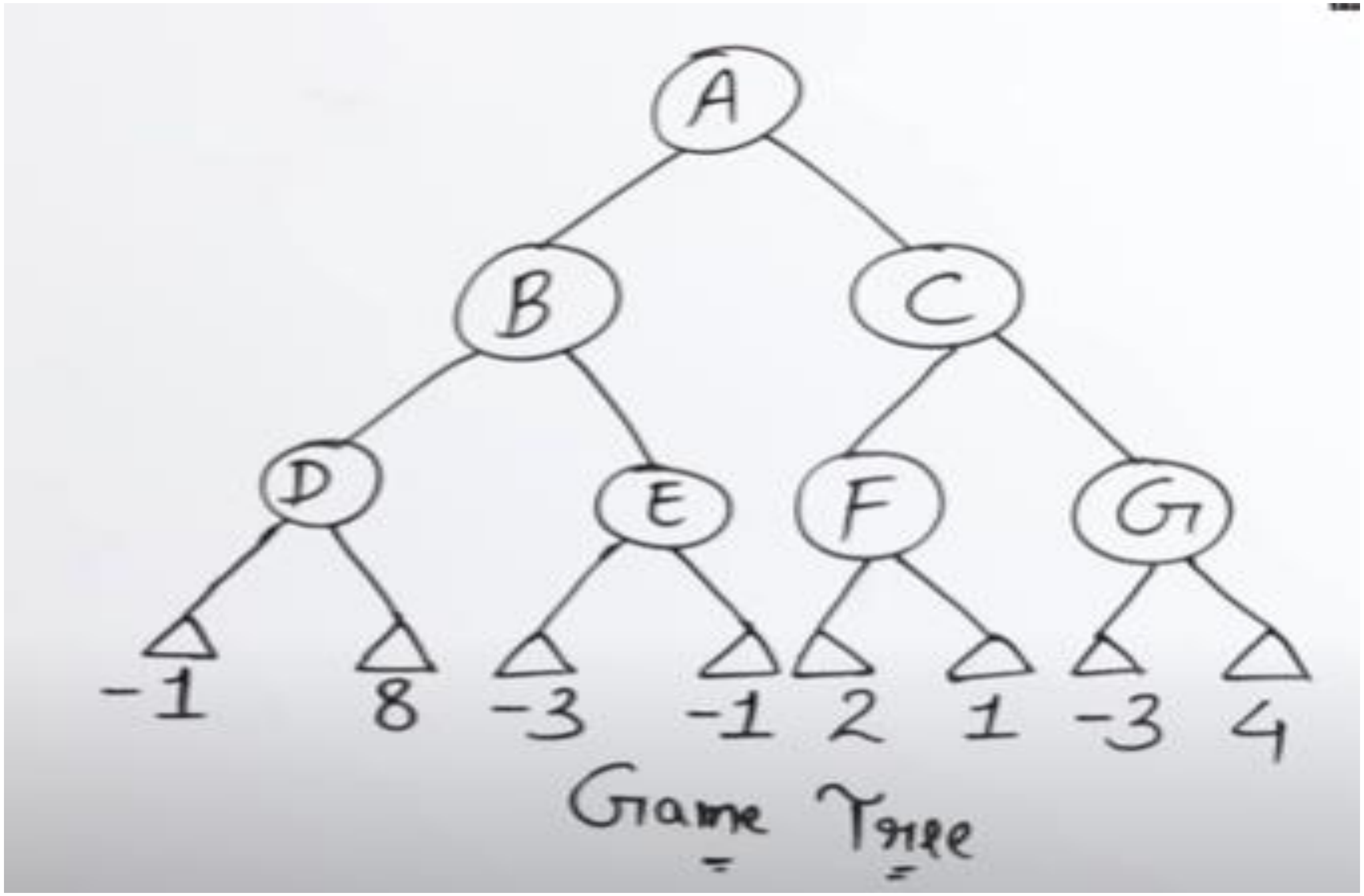
MINIMAX Procedure



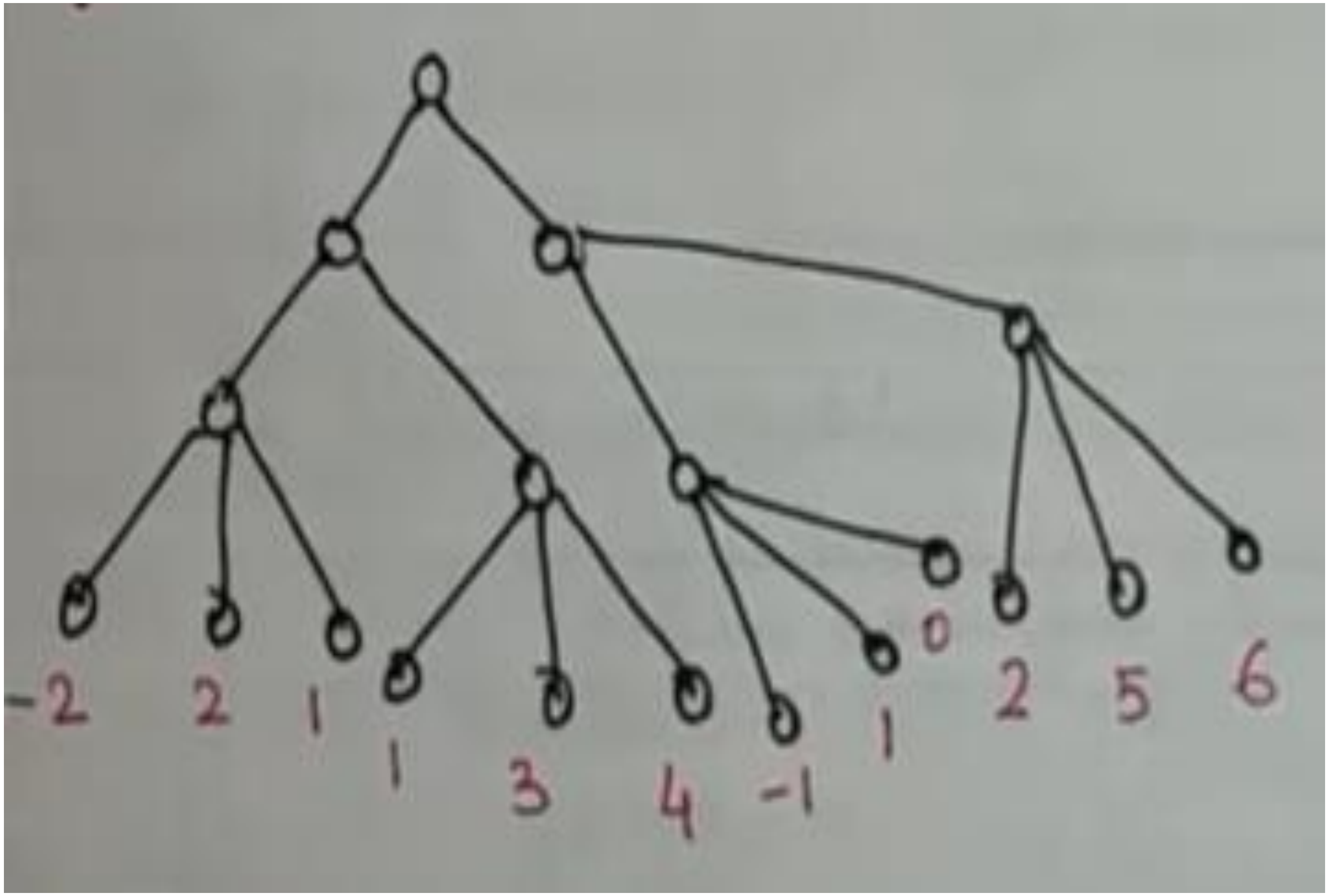
MINIMAX Procedure



MINIMAX Procedure



MINIMAX Procedure



MINIMAX Procedure

Time Complexity:

❖ O(b^m).

Space Complexity:

❖ O(bm) (depth-first exploration)

Completeness:

❖ Yes (if tree is finite)

Optimality:

❖ Yes (against an optimal opponent)

Alpha (α)- beta (β) pruning

- The basic idea of alpha-beta cutoffs is “It is possible to compute the correct minimax decision without looking at every node in the search tree”
- Allow the search process to ignore portion of the search tree that makes no difference to the final choice
- General principle of α-β pruning is
  - Consider a node n somewhere in the tree, such that a player has a chance to move to this node
  - If player has a better chance m either at the parent node of n then n will never be reached in actual play

Alpha (α)- beta (β) pruning :: Properties

- Pruning does not affect final result

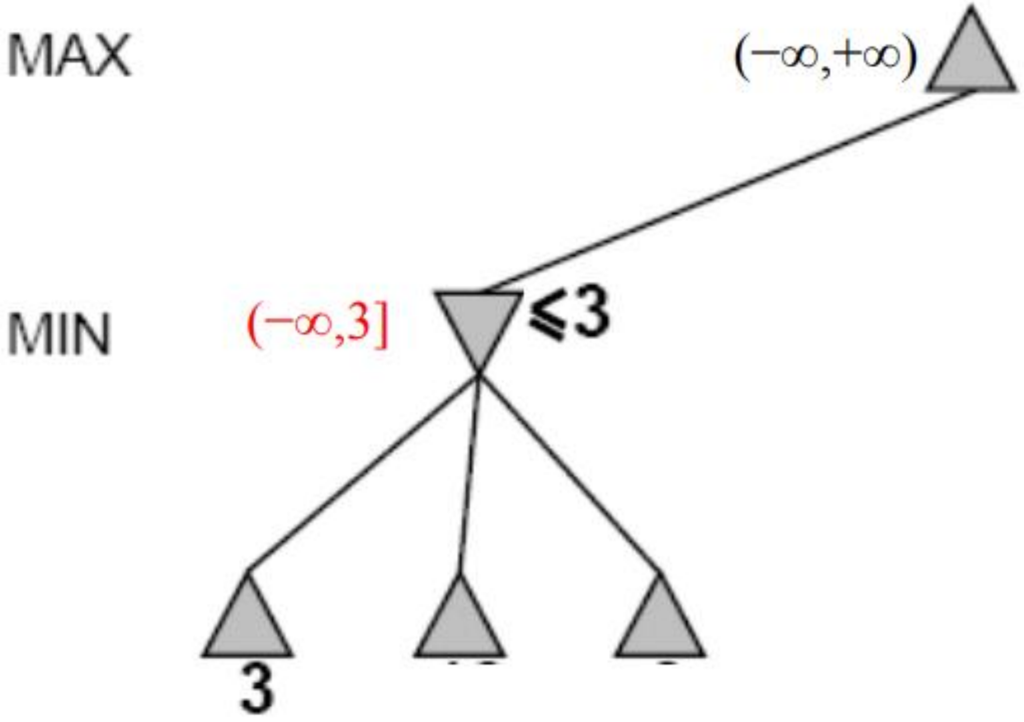
- Good move ordering improves effectiveness of pruning

- With "perfect ordering," time complexity = O($b^{m/2}$)
  - doubles depth of search

- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

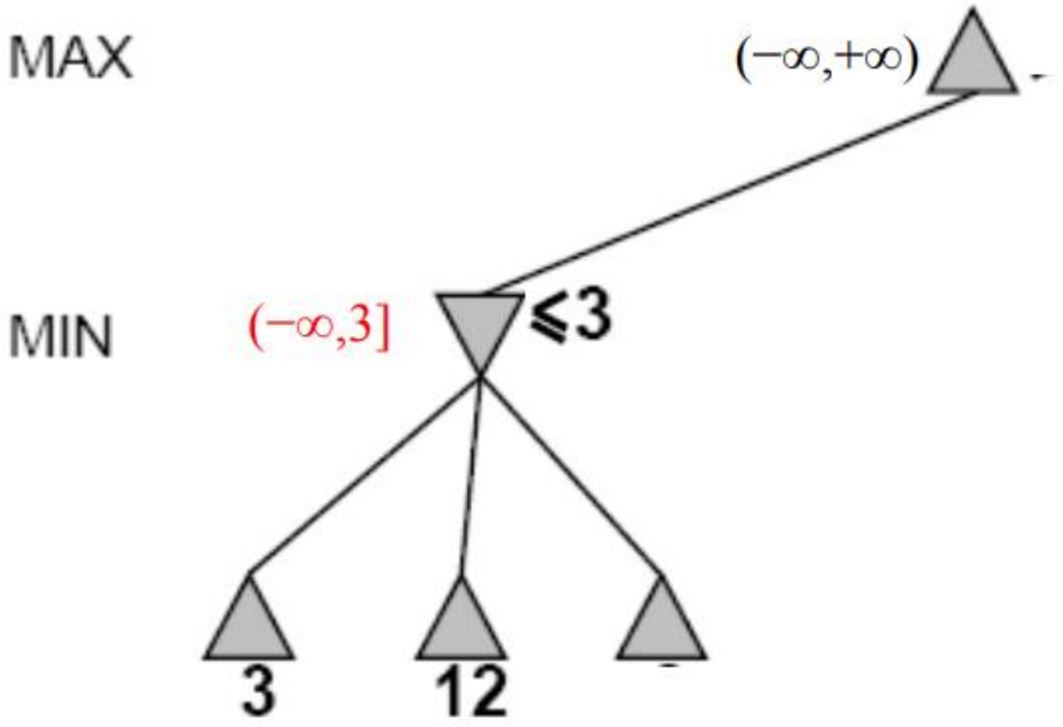
Alpha (α)- beta (β) pruning

- Alpha: The highest value that the maximizer can guarantee himself by making some move at the current node OR at some node earlier on the path
- Beta: The lowest value that the minimizer can guarantee by making some move at the current node OR at some node earlier on the path to this node



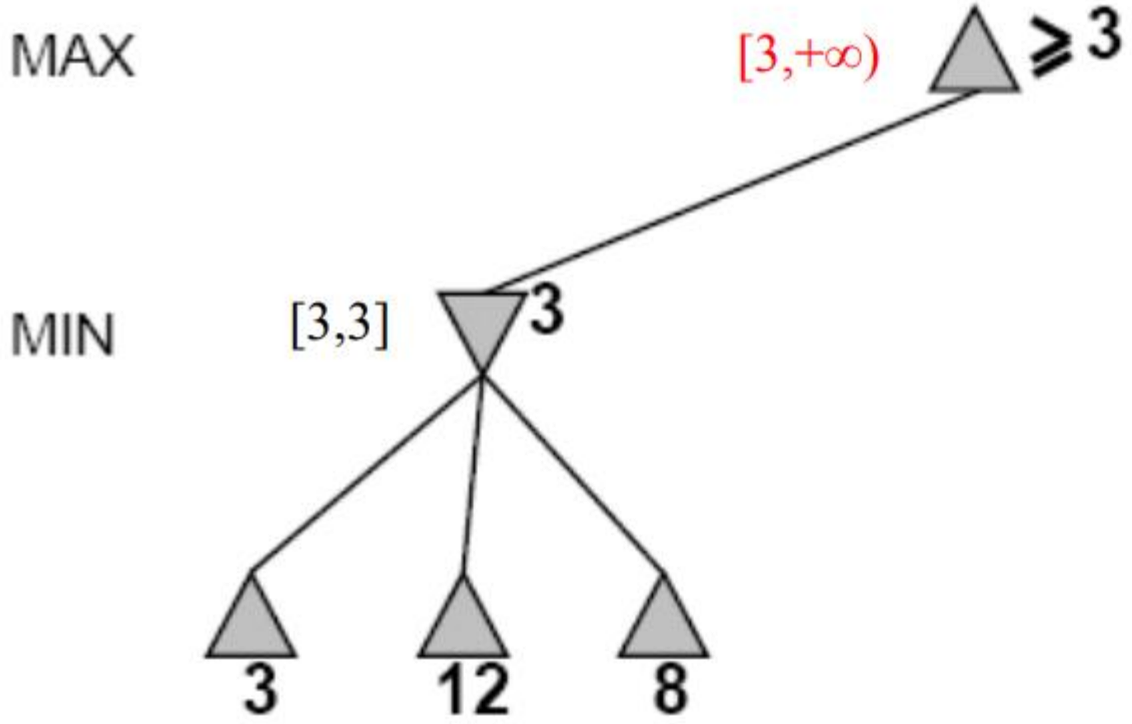
Alpha (α)- beta (β) pruning

- Alpha: The highest value that the maximizer can guarantee himself by making some move at the current node OR at some node earlier on the path
- Beta: The lowest value that the minimizer can guarantee by making some move at the current node OR at some node earlier on the path to this node



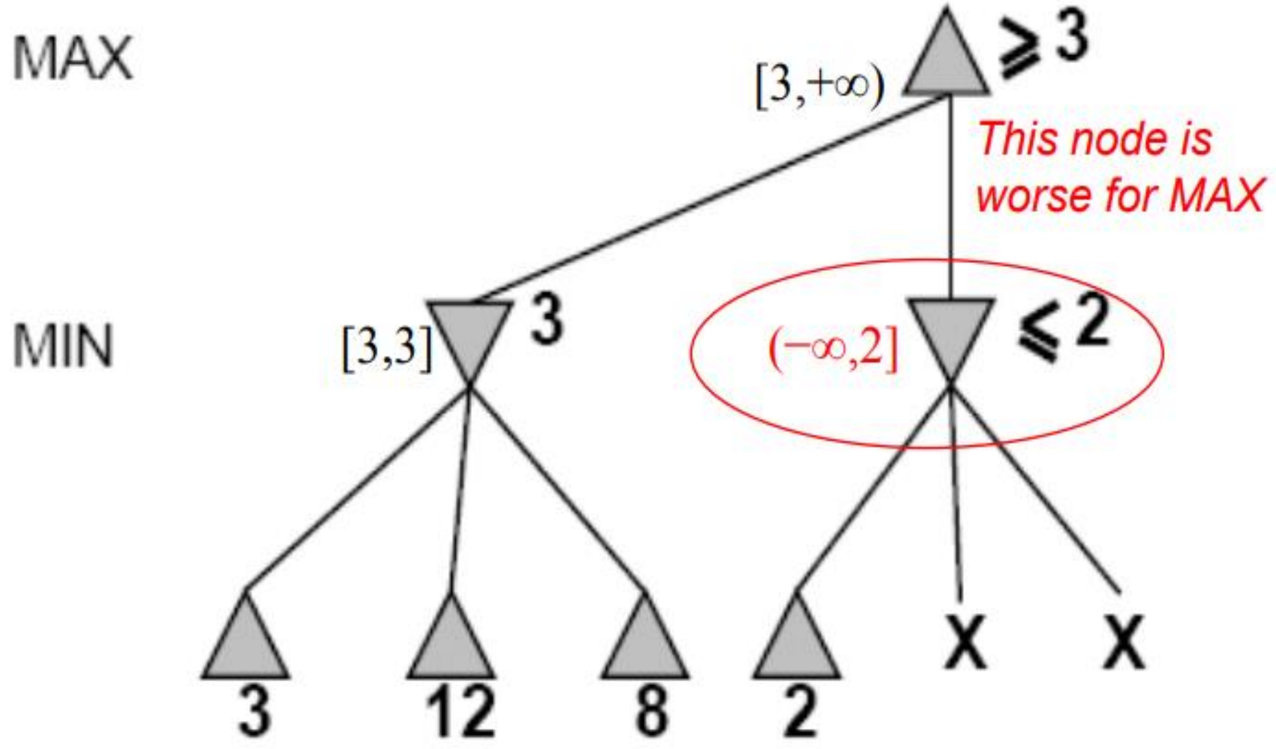
Alpha (α)- beta (β) pruning

- Alpha: The highest value that the maximizer can guarantee himself by making some move at the current node OR at some node earlier on the path
- Beta: The lowest value that the minimizer can guarantee by making some move at the current node OR at some node earlier on the path to this node



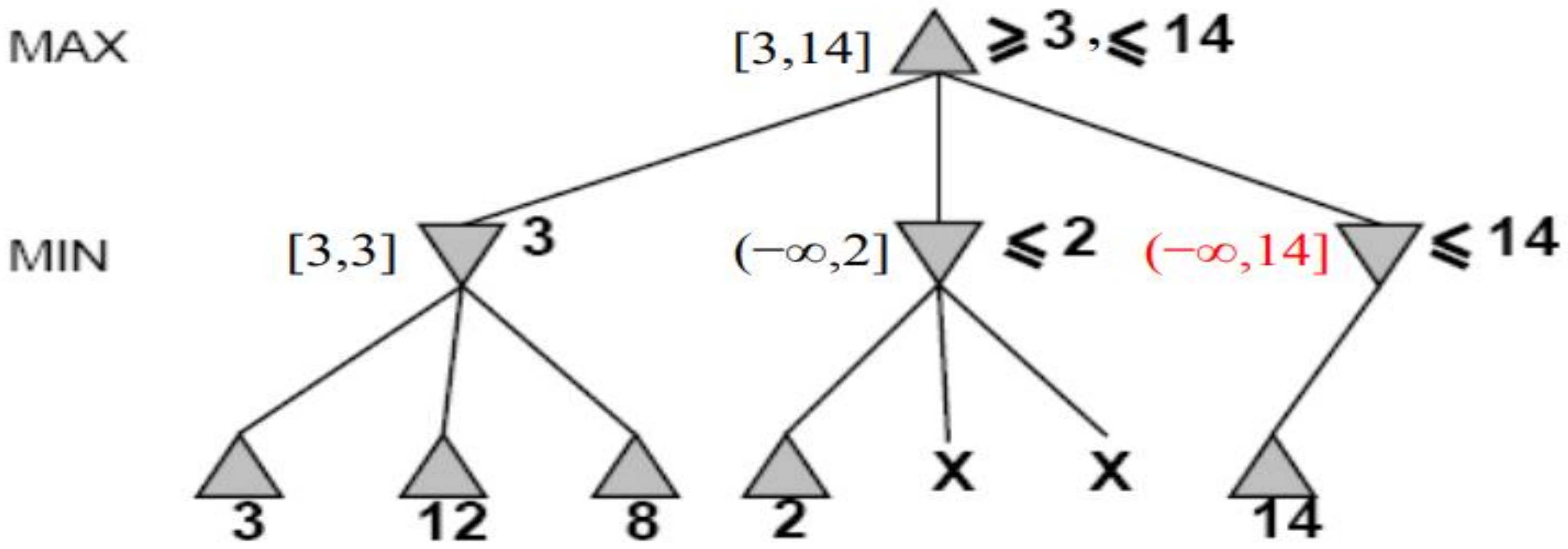
Alpha (α)- beta (β) pruning

- Alpha: The highest value that the maximizer can guarantee himself by making some move at the current node OR at some node earlier on the path
- Beta: The lowest value that the minimizer can guarantee by making some move at the current node OR at some node earlier on the path to this node



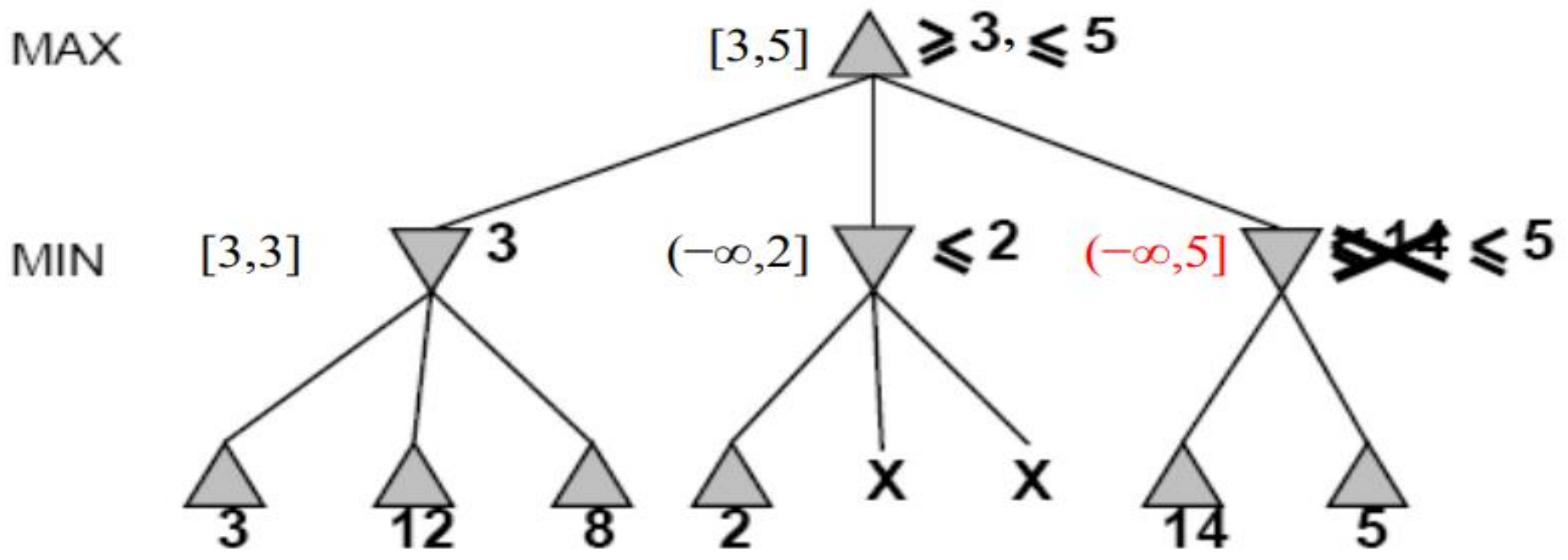
Alpha (α)- beta (β) pruning

- Alpha: The highest value that the maximizer can guarantee himself by making some move at the current node OR at some node earlier on the path
- Beta: The lowest value that the minimizer can guarantee by making some move at the current node OR at some node earlier on the path to this node



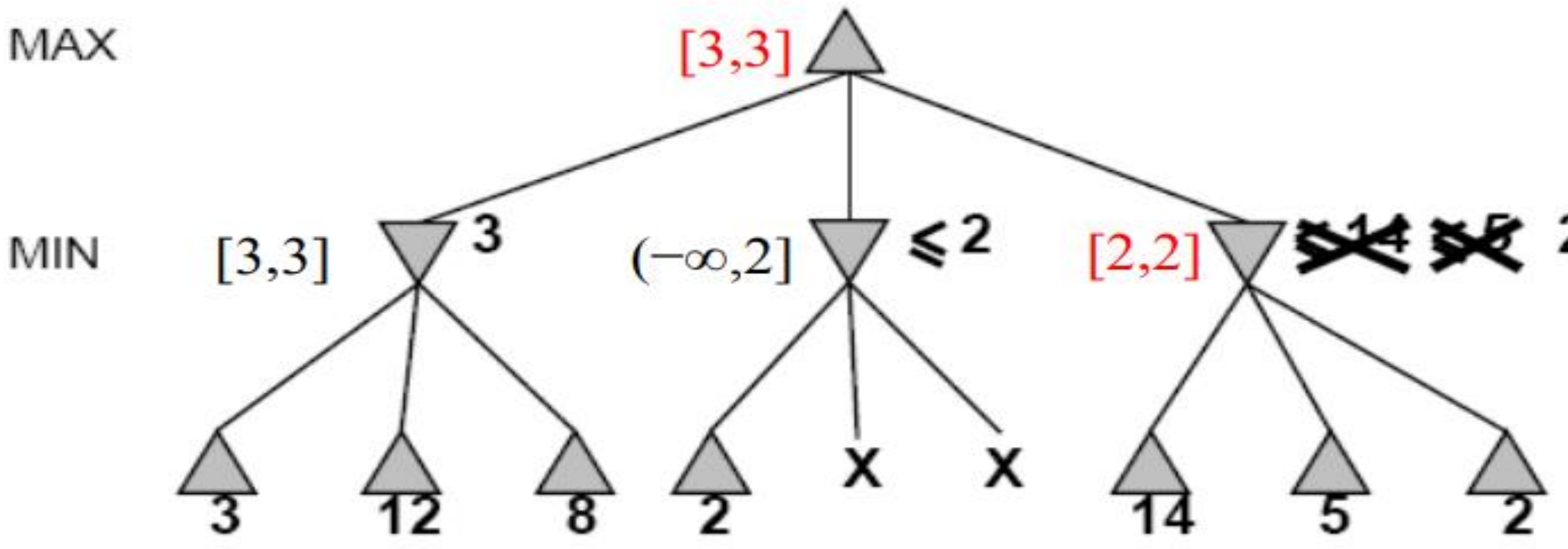
Alpha (α)- beta (β) pruning

- Alpha: The highest value that the maximizer can guarantee himself by making some move at the current node OR at some node earlier on the path
- Beta: The lowest value that the minimizer can guarantee by making some move at the current node OR at some node earlier on the path to this node



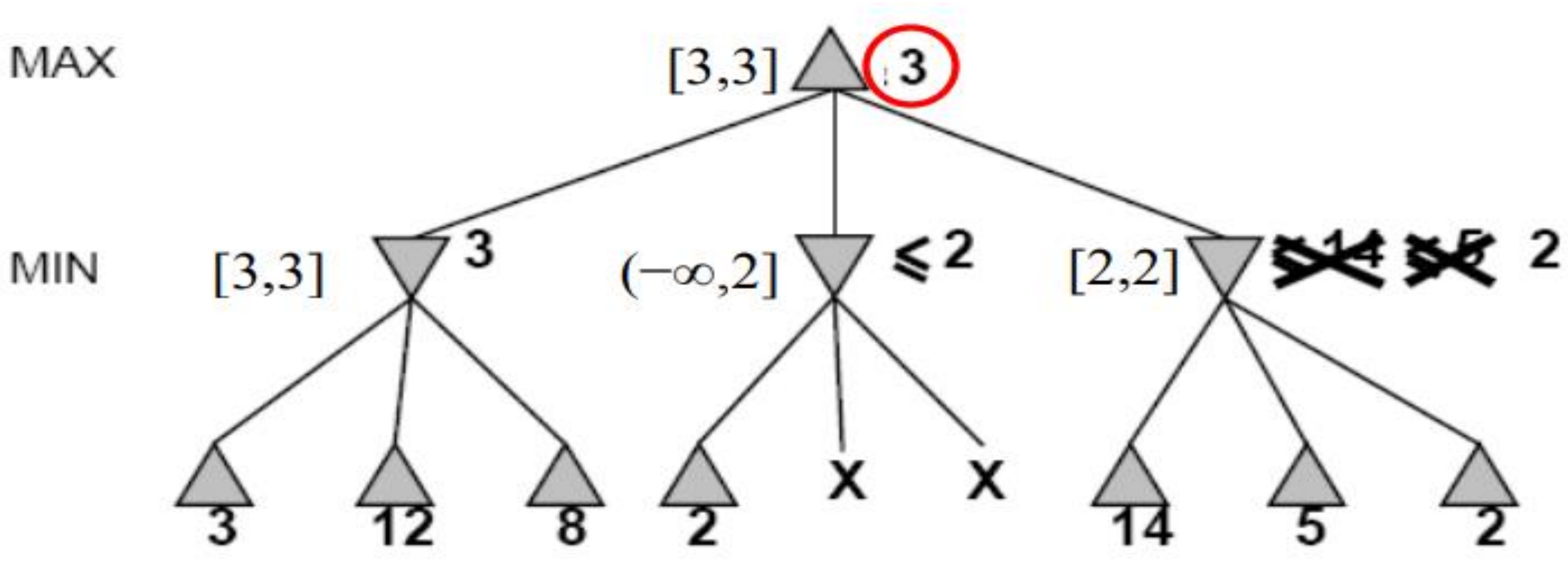
Alpha (α)- beta (β) pruning

- Alpha: The highest value that the maximizer can guarantee himself by making some move at the current node OR at some node earlier on the path
- Beta: The lowest value that the minimizer can guarantee by making some move at the current node OR at some node earlier on the path to this node

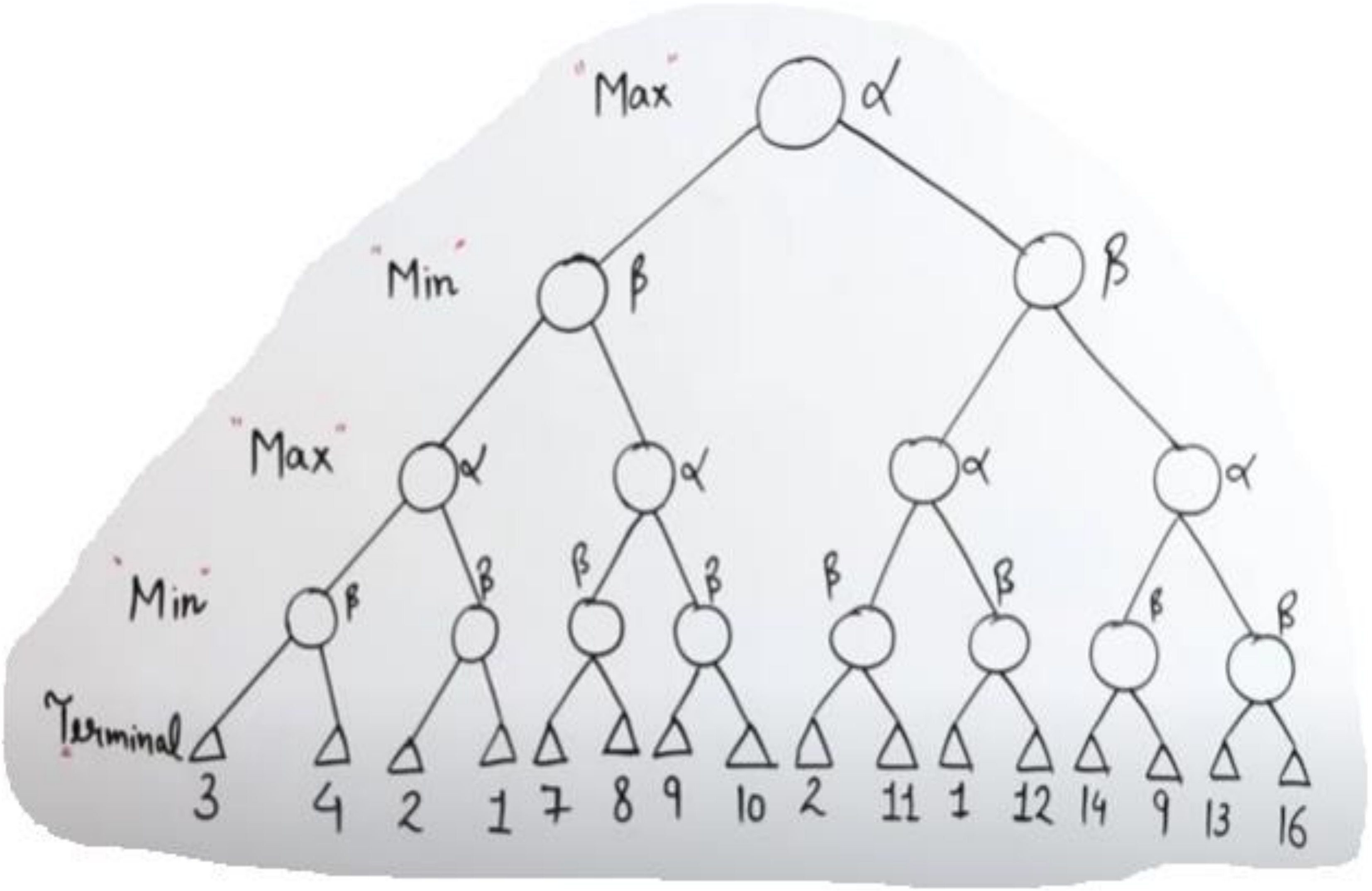


Alpha (α)- beta (β) pruning

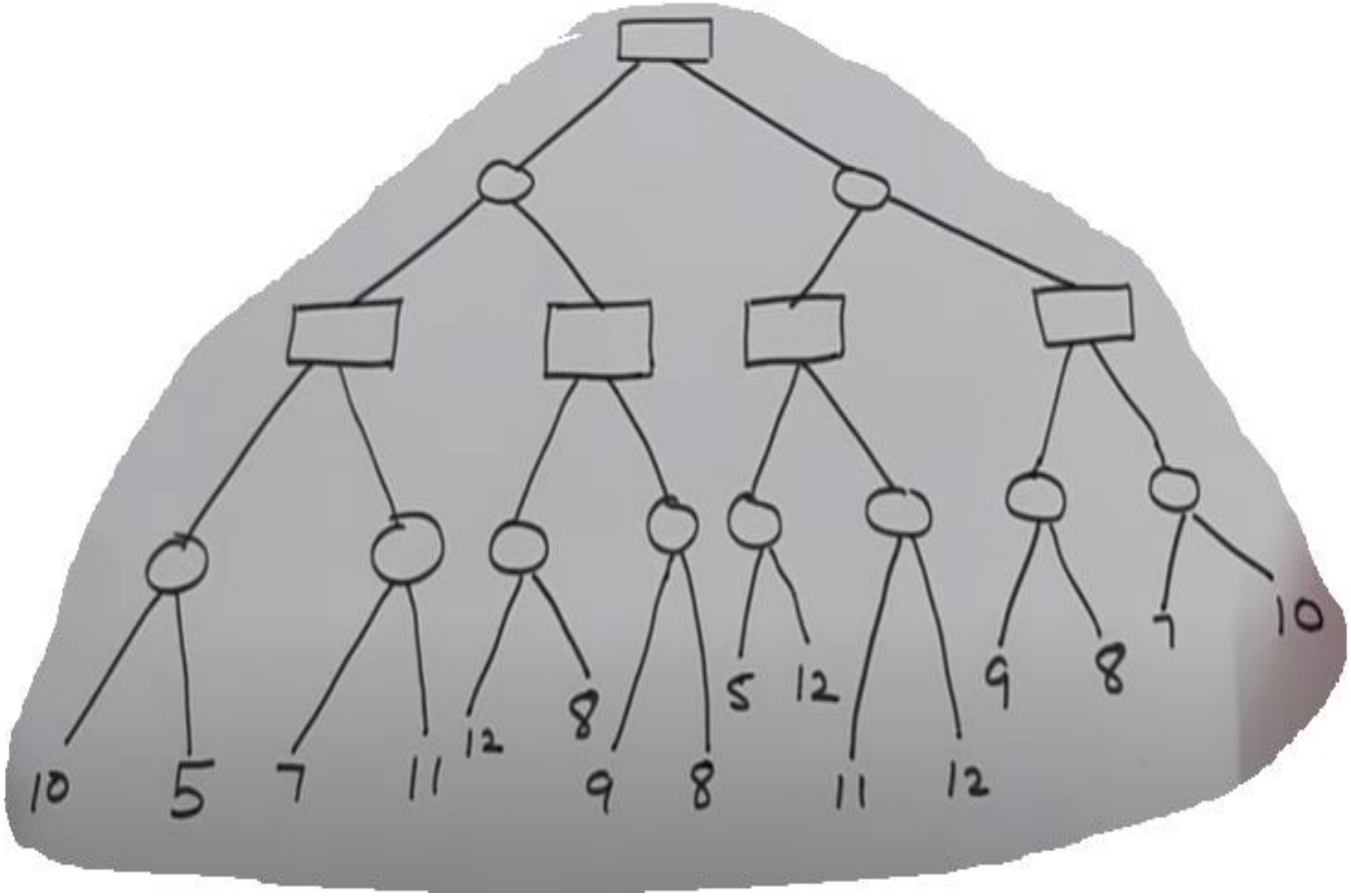
- Alpha: The highest value that the maximizer can guarantee himself by making some move at the current node OR at some node earlier on the path
- Beta: The lowest value that the minimizer can guarantee by making some move at the current node OR at some node earlier on the path to this node



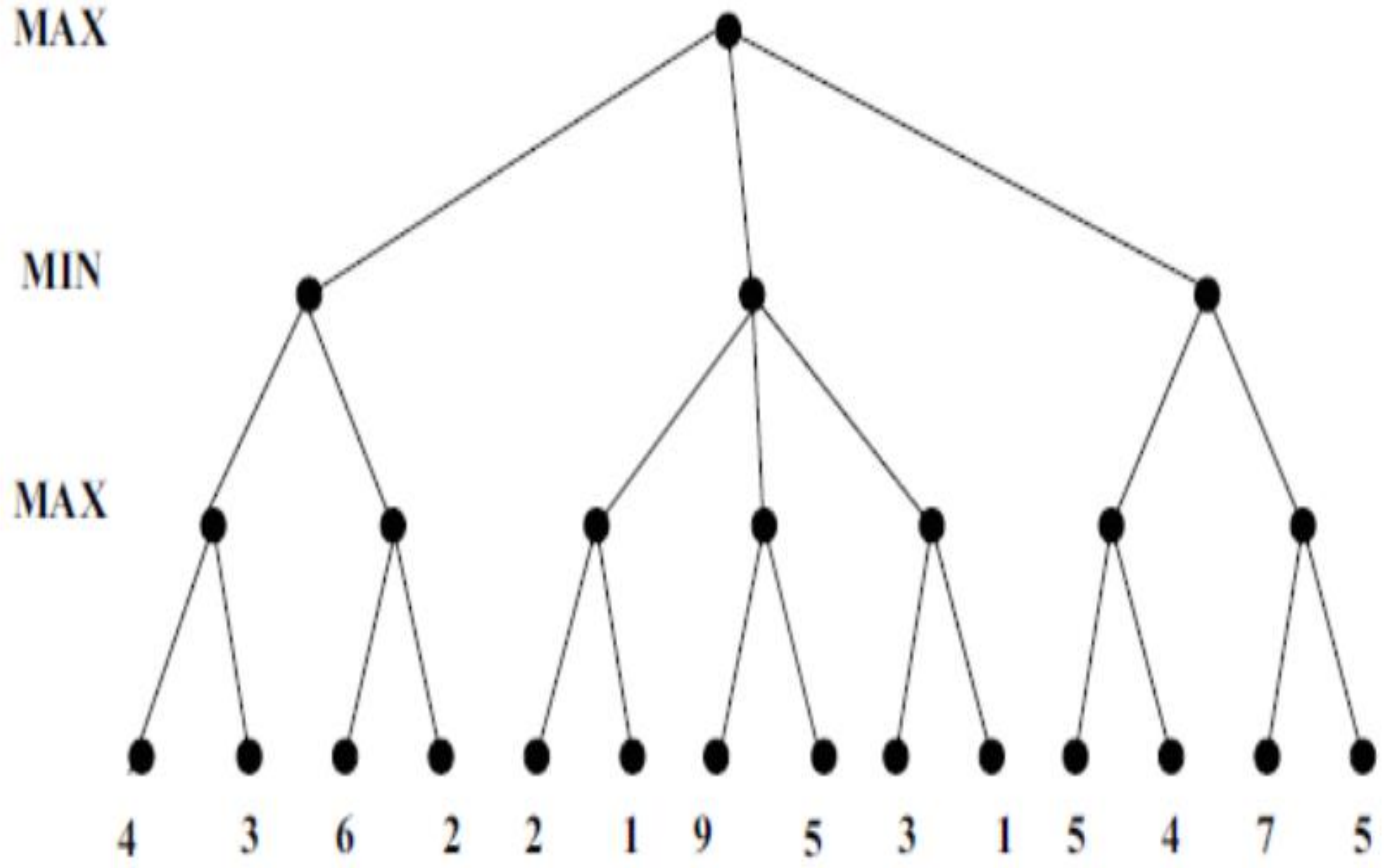
Alpha (α)- beta (β) pruning



Alpha (α)- beta (β) pruning

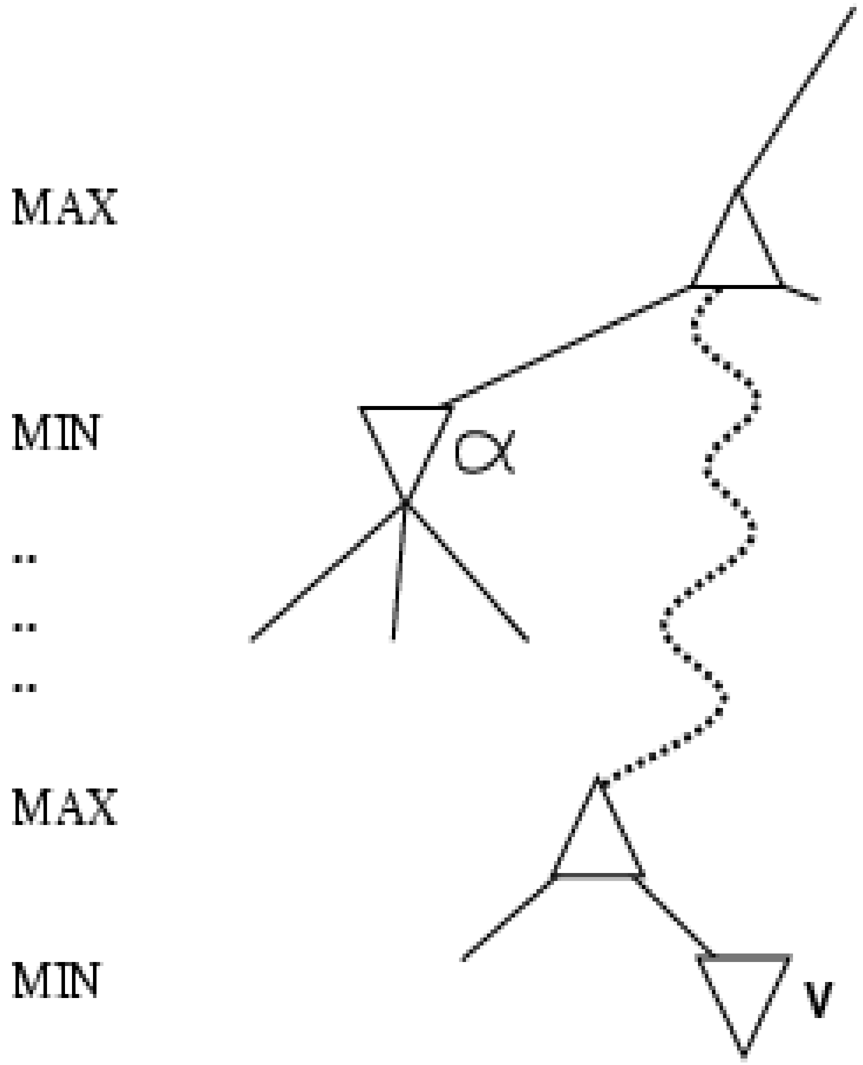


Alpha (α)- beta (β) pruning

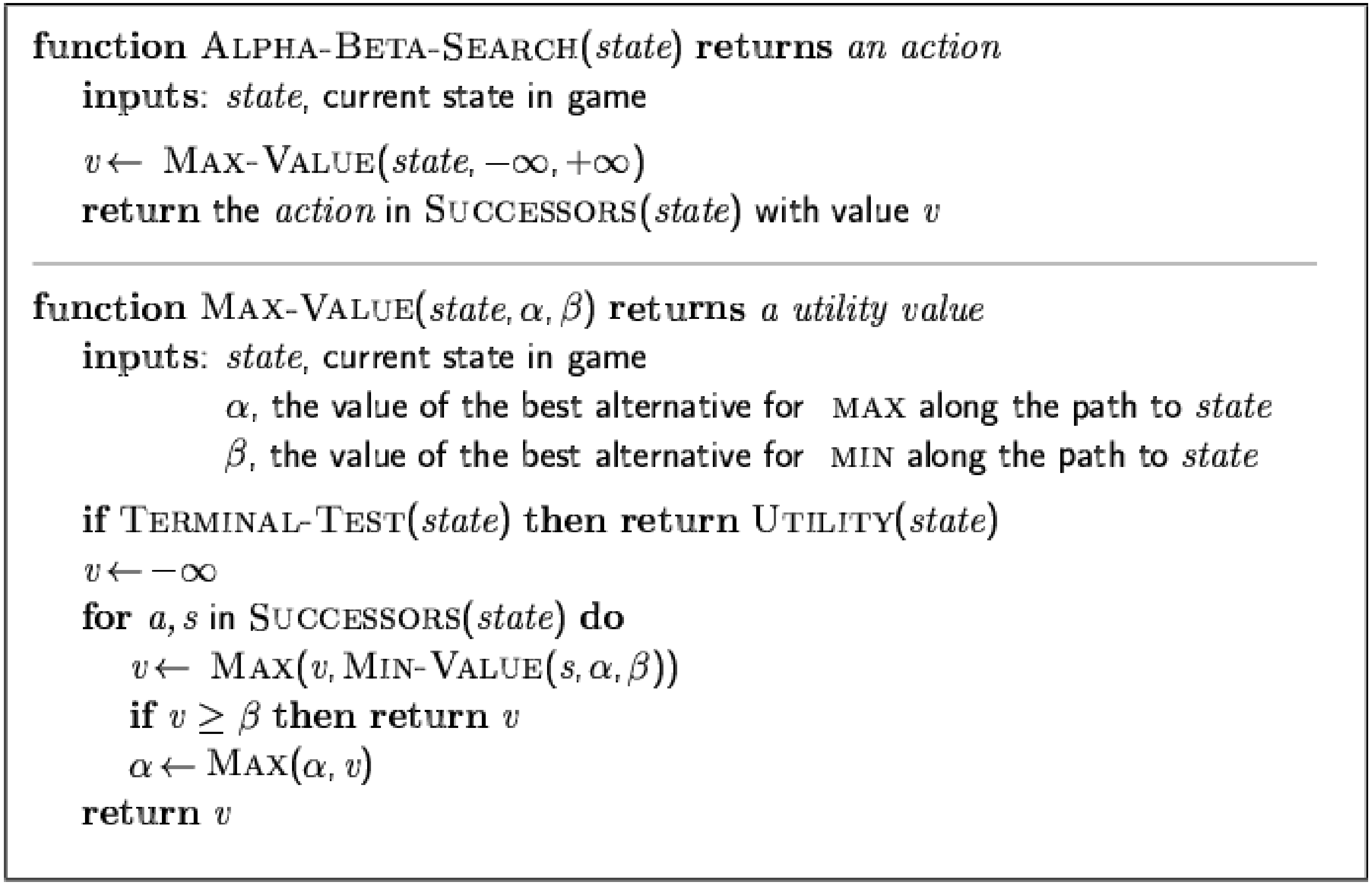


Alpha (α)- beta (β) pruning

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for max
- If v is worse than α, max will avoid it
  - prune that branch
- Define β similarly for min



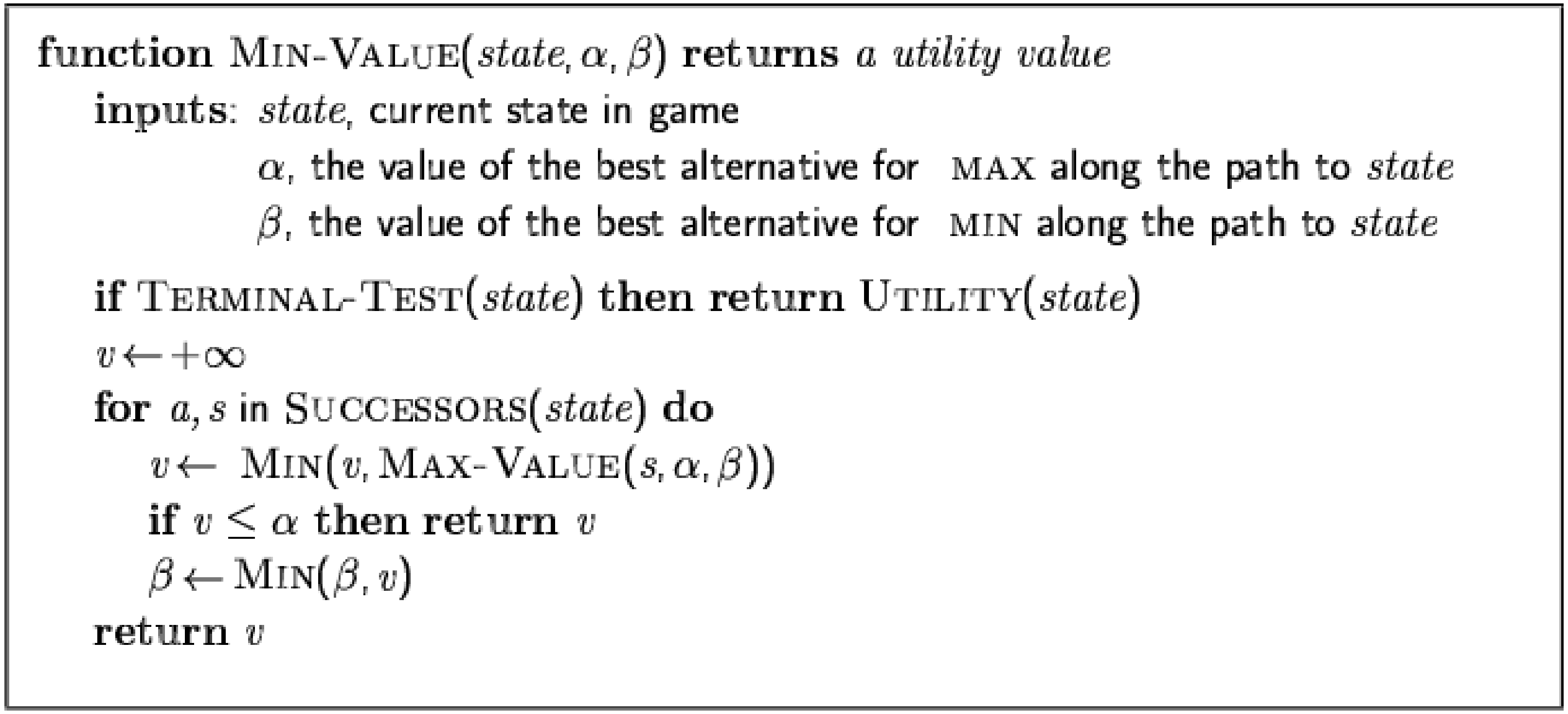
Alpha (α)- beta (β) pruning



```
function ALPHA-BETA-SEARCH($state$) returns $an$ $action$
    inputs: $state$, current state in game

    $v \leftarrow$ MAX-VALUE($state$, $-\infty$, $+\infty$)
    return the $action$ in SUCCESSORS($state$) with value $v$
```

```
function MAX-VALUE($state$, $\alpha$, $\beta$) returns $a$ $utility$ $value$
    inputs: $state$, current state in game
            $\alpha$, the value of the best alternative for MAX along the path to $state$
            $\beta$, the value of the best alternative for MIN along the path to $state$

    if TERMINAL-TEST($state$) then return UTILITY($state$)
    $v \leftarrow -\infty$
    for $a, s$ in SUCCESSORS($state$) do
        $v \leftarrow$ MAX($v$, MIN-VALUE($s$, $\alpha$, $\beta$))
        if $v \geq \beta$ then return $v$
        $\alpha \leftarrow$ MAX($\alpha$, $v$)
    return $v$
```

Alpha (α)- beta (β) pruning



```
function MIN-VALUE($state, \alpha, \beta$) returns a utility value
    inputs: state, current state in game
            $\alpha$, the value of the best alternative for MAX along the path to state
            $\beta$, the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    $v \leftarrow +\infty$
    for $a, s$ in SUCCESSORS(state) do
        $v \leftarrow$ MIN($v$, MAX-VALUE($s, \alpha, \beta$))
        if $v \leq \alpha$ then return $v$
        $\beta \leftarrow$ MIN($\beta, v$)
    return $v$
```

Online Tool

- https://raphsilva.github.io/utilities/minimax_simulator/

Disclaimer

The material for the presentation has been compiled from various sources such as prescribed text books by Russell and Norvig and other tutorials and lecture notes. The information contained in this lecture/ presentation is for educational purpose only.

Thank You *for* Your Attention !