Lecture No. 21-22

Artificial Intelligence

CS-3151 (V1)

Instructor: Dr. Muhammad Kabir

Assistant Professor - Department of Computer Science (SST)

University of Management and Technology, Lahore

Previous Lecture

- Constraint Satisfaction Problems (CSPs) & Examples
- Varieties of CSPs & Constraints
- Backtracking Search for CSPs
- Problem Structure and Decomposition
- Minimum Remaining Values (MRV)
- Degree Heuristic
- Least Constraining Value
- Forward Checking & Constraint Propagation
- Arc Consistency (AC-3 Algorithm)

Today's Lecture

- Games versus Search Problems
- Types of Games
- Game Tree
- Minimax Algorithm and its properties
- Multi-player Games
- Alpha-Beta Pruning - Example & Algorithm
- Why is it called α-β?
- Properties of α-β

Adversarial search

- In This search we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.

- In search strategies with a single agent aims to find the solution which often expressed in the form of a sequence of actions.

- But, It is possible that more than one agent is searching for the solution in the same search space, usually occurs in game playing.

Adversarial search

- The environment with more than one agent is termed as multi-agent environment, in which each agent is an opponent of other agent and playing against each other.
- Each agent needs to consider the action of other agent and effect of that action on their performance.
- 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.
- Games are modeled as a Search problem and heuristic evaluation function, and these factors which help to model and solve games in AI.

Search Versus Games

- **Search – No adversary**
  - Solution is (heuristic) method for finding goal
  - Heuristic techniques can find optimal solution
  - Evaluation function: estimate of cost from start to goal through given node
  - Examples: path planning, scheduling activities
- **Games – Adversary**
  - Solution is strategy (strategy specifies move for every possible opponent reply).
  - Optimality depends on opponent.
  - Time limits force an approximate solution
  - Evaluation function: evaluate “goodness” of game position
  - Examples: chess, checkers

Types of Games

	Deterministic	Chance
Perfect information	Chess, Checkers, Go, Othello	Backgammon, Monopoly
Imperfect information	Battleships, Blind Tic-Tac-Toe	Bridge, Poker, Scrabble, Nuclear War

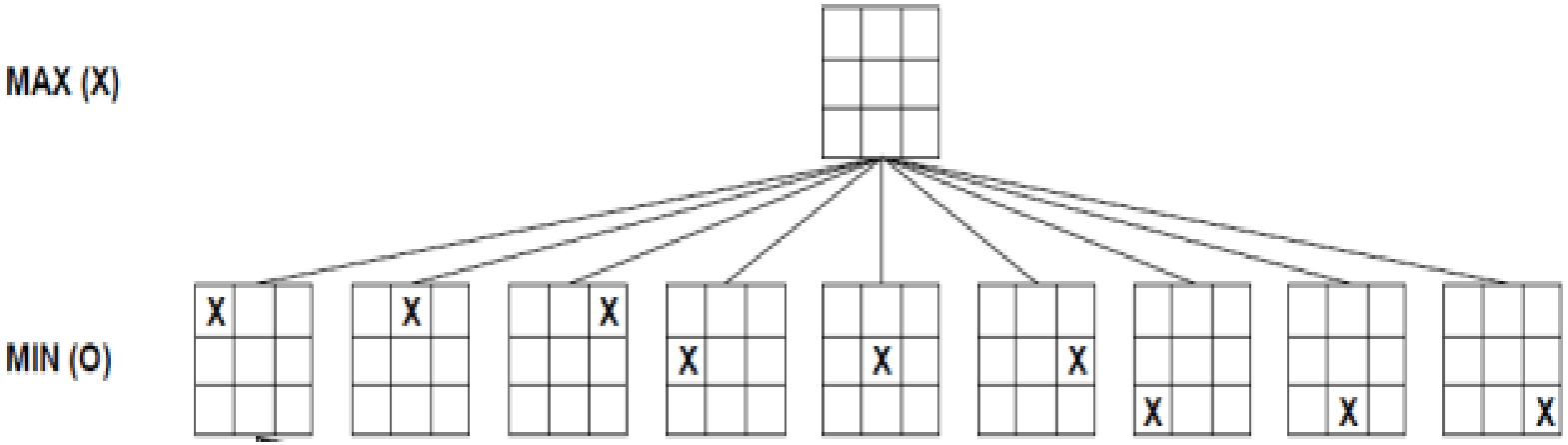
Types of Games

- **Perfect information**
  - A game with the perfect information is that in which agents can look into the complete board.
  - Agents have all the information about the game, and they can see each other moves also. Examples are Chess, Checkers, Go, etc.
- **Imperfect information**
  - If in a game agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information, such as tic-tac-toe, Battleship, blind, Bridge, etc.

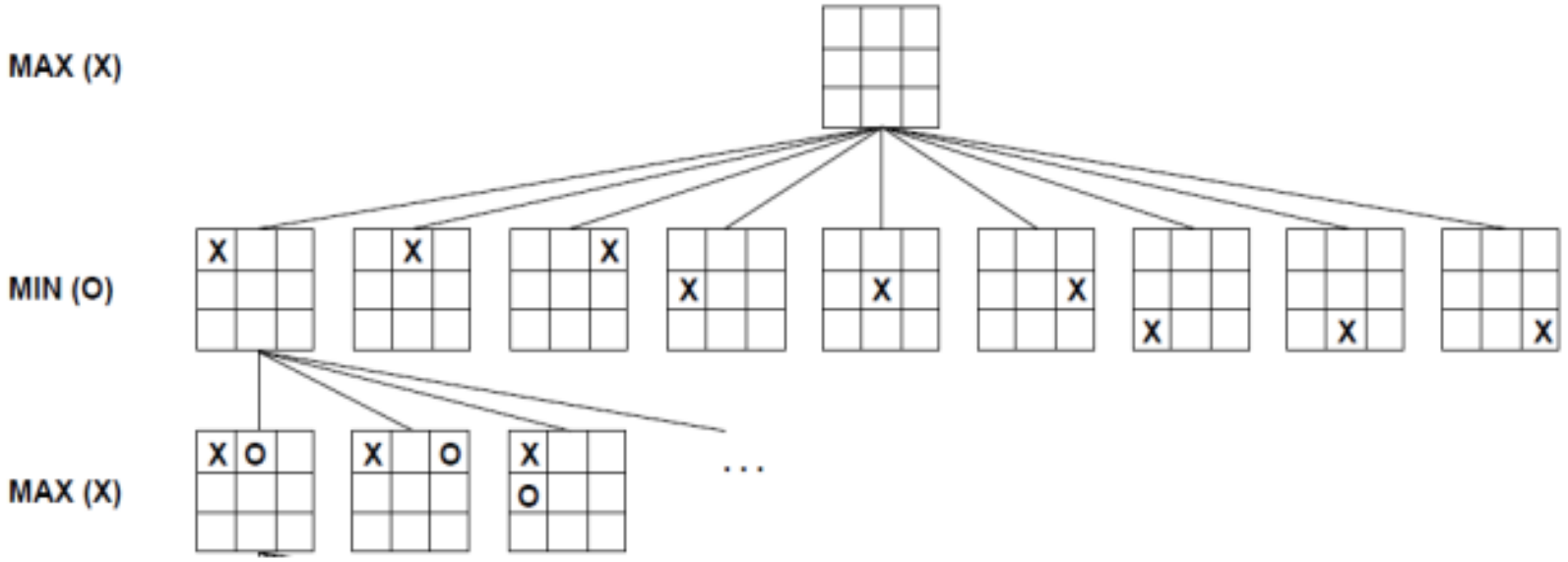
Types of Games

- **Deterministic games**
  - Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them.
  - Examples are chess, Checkers, Go, tic-tac-toe, etc.
- **Non-deterministic games (chance)**
  - Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck.
  - These are random, and each action response is not fixed. Such games are also called as stochastic games. Example: Backgammon, Monopoly, Poker, etc.

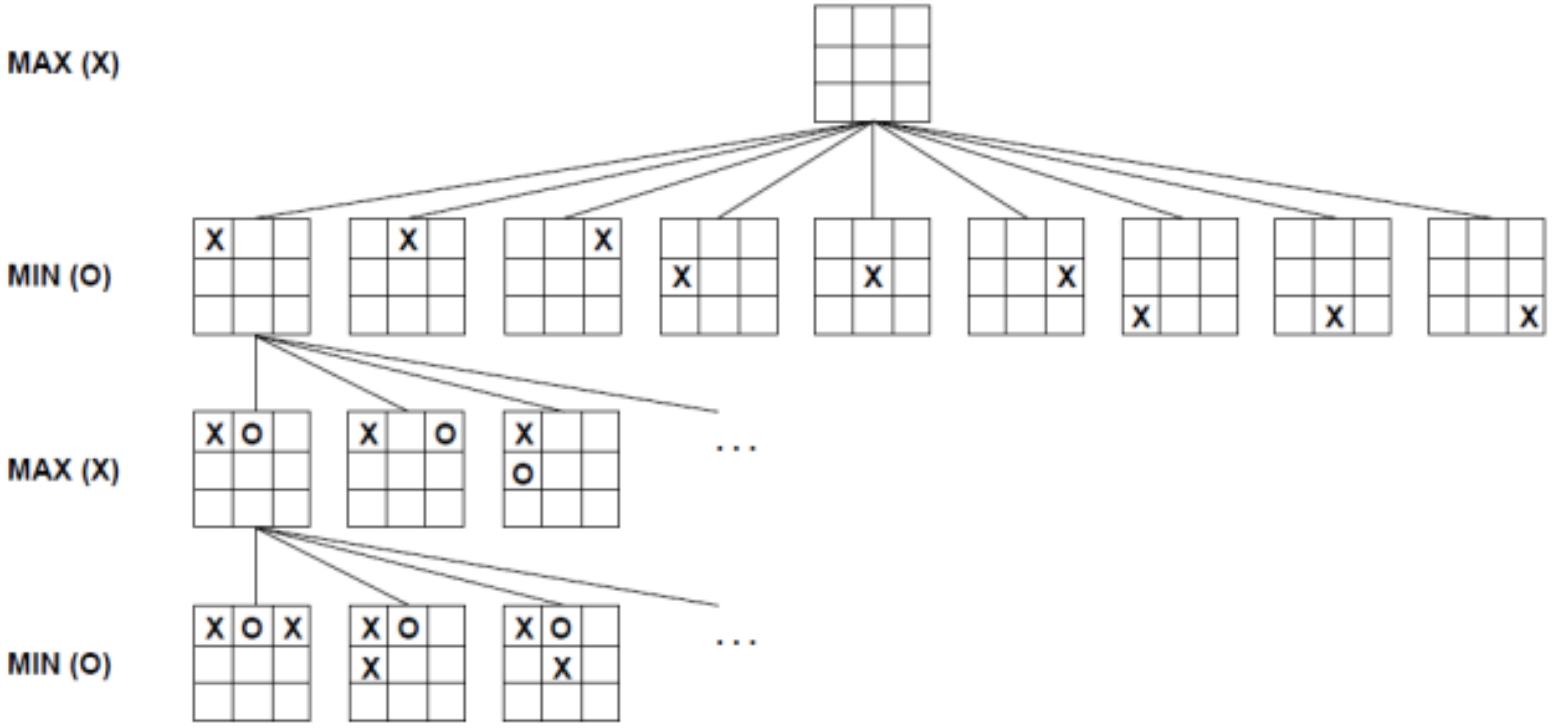
Game Tree
(2-player, deterministic, turns)



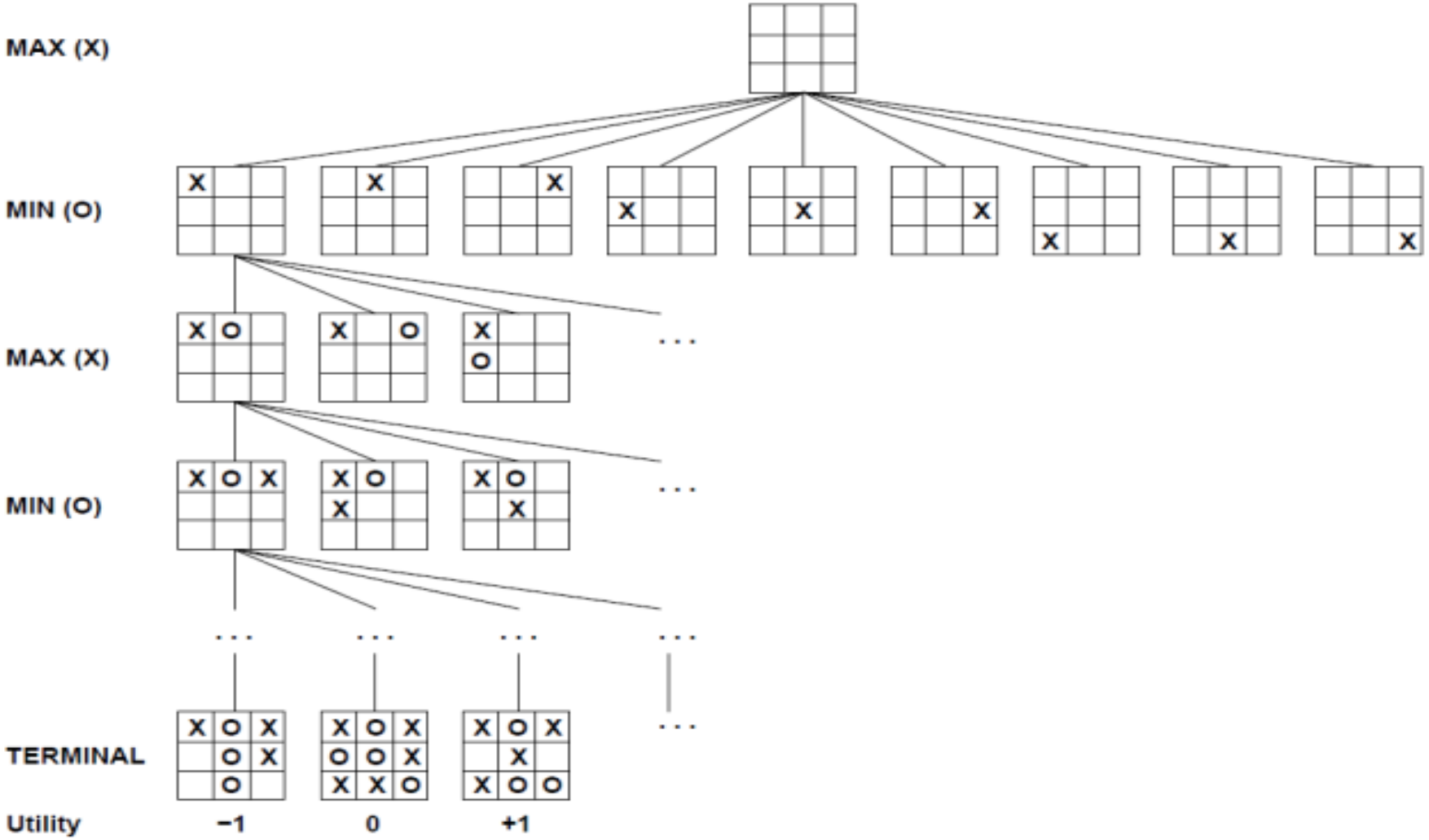
Game Tree
(2-player, deterministic, turns)



Game Tree
(2-player, deterministic, turns)



Game Tree
(2-player, deterministic, turns)



Minimax Algorithm

- Back Tracking Algorithm
- Depth First Search is Considered.
- Best Move Strategy will Use
- Max Will Try to Maximize the Utility (Best Move)
- Min Will Try to Minimize the Utility (Worst Case)
- Two players play the game, one is called MAX and other is called MIN.
- Both the players fight as the opponent player gets the minimum benefit while they get the maximum benefit.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go.

Minimax Algorithm

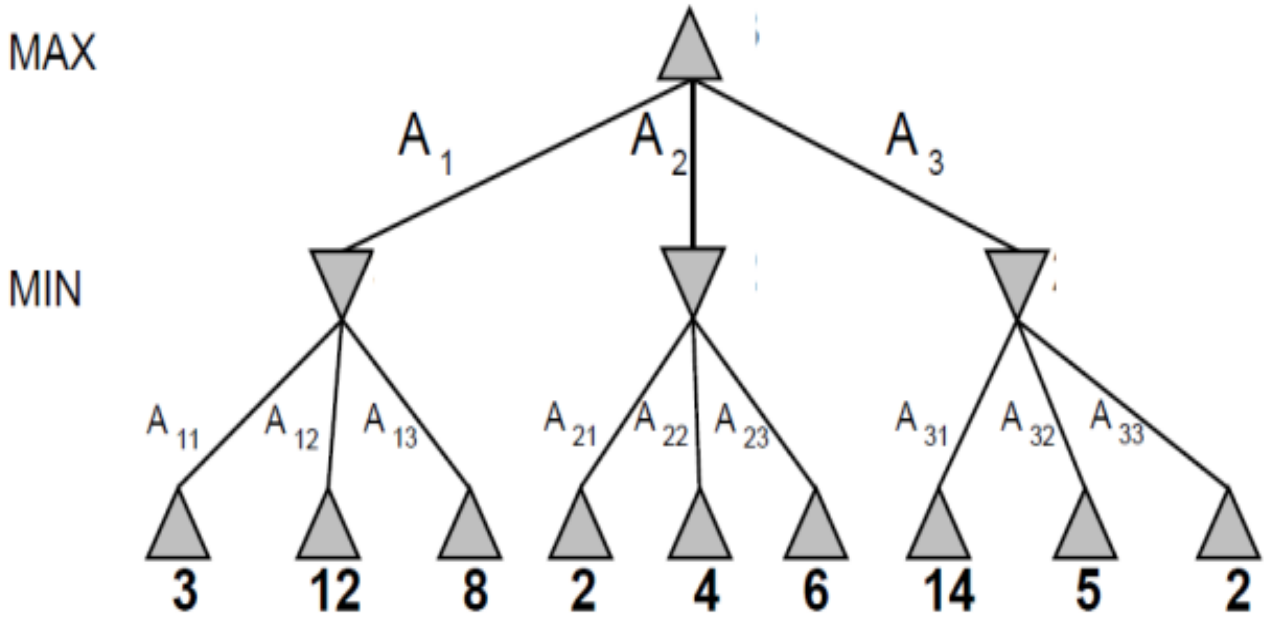
- Perfect play for deterministic, perfect information games
- Idea: choose move to position with highest ***minimax*** value

  = best achievable payoff against best play
- Example: 2-ply game:

Minimax Algorithm

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

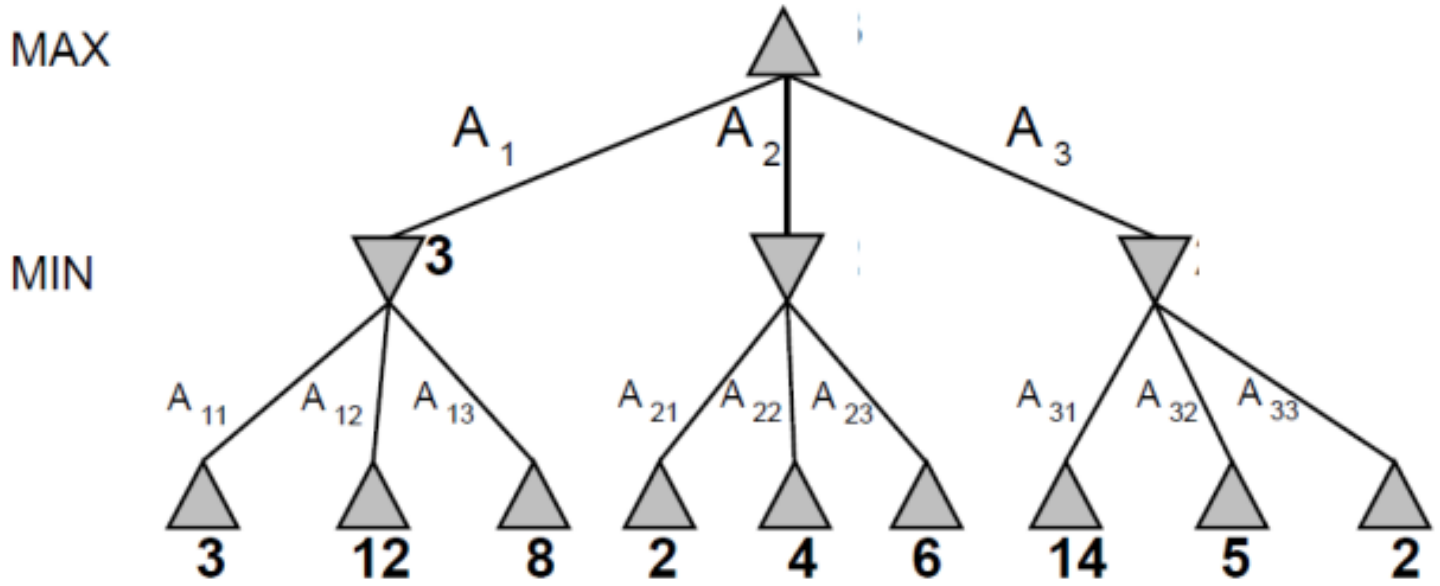
  = best achievable payoff against best play
- Example: 2-ply game:



Minimax Algorithm

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

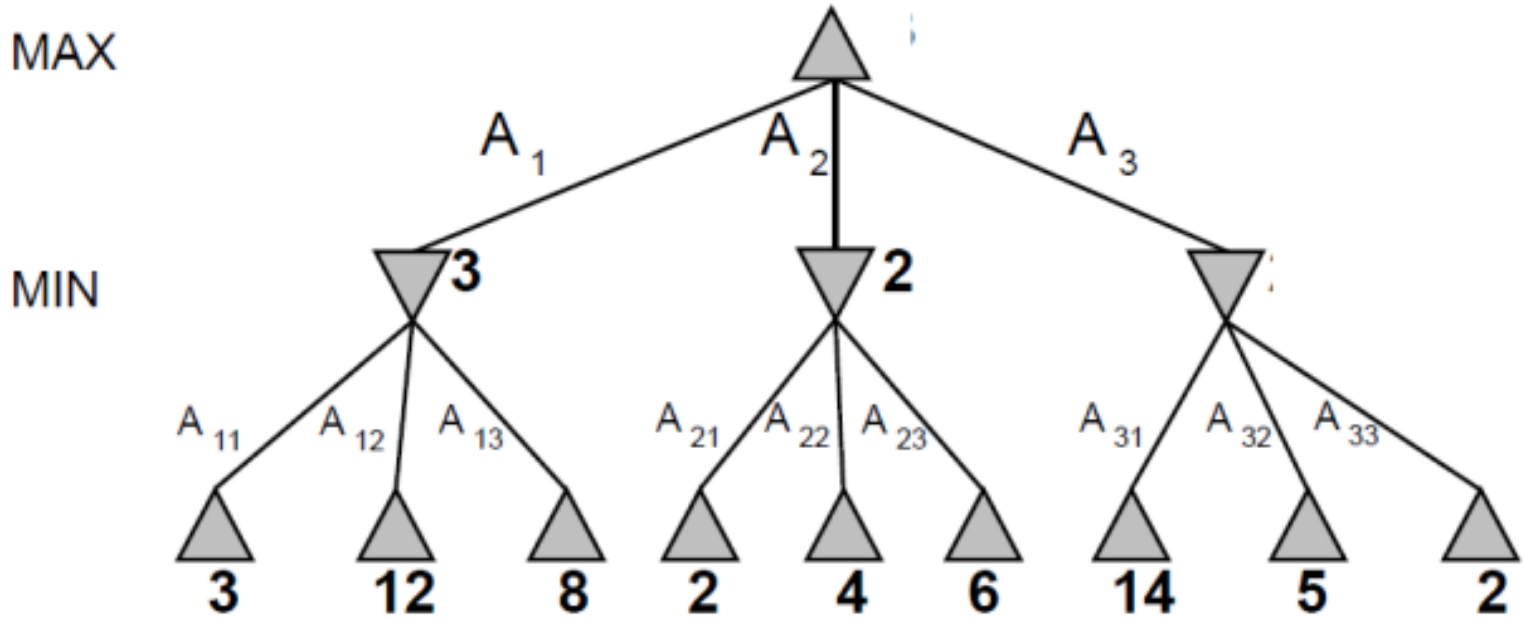
  = best achievable payoff against best play
- Example: 2-ply game:



Minimax Algorithm

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

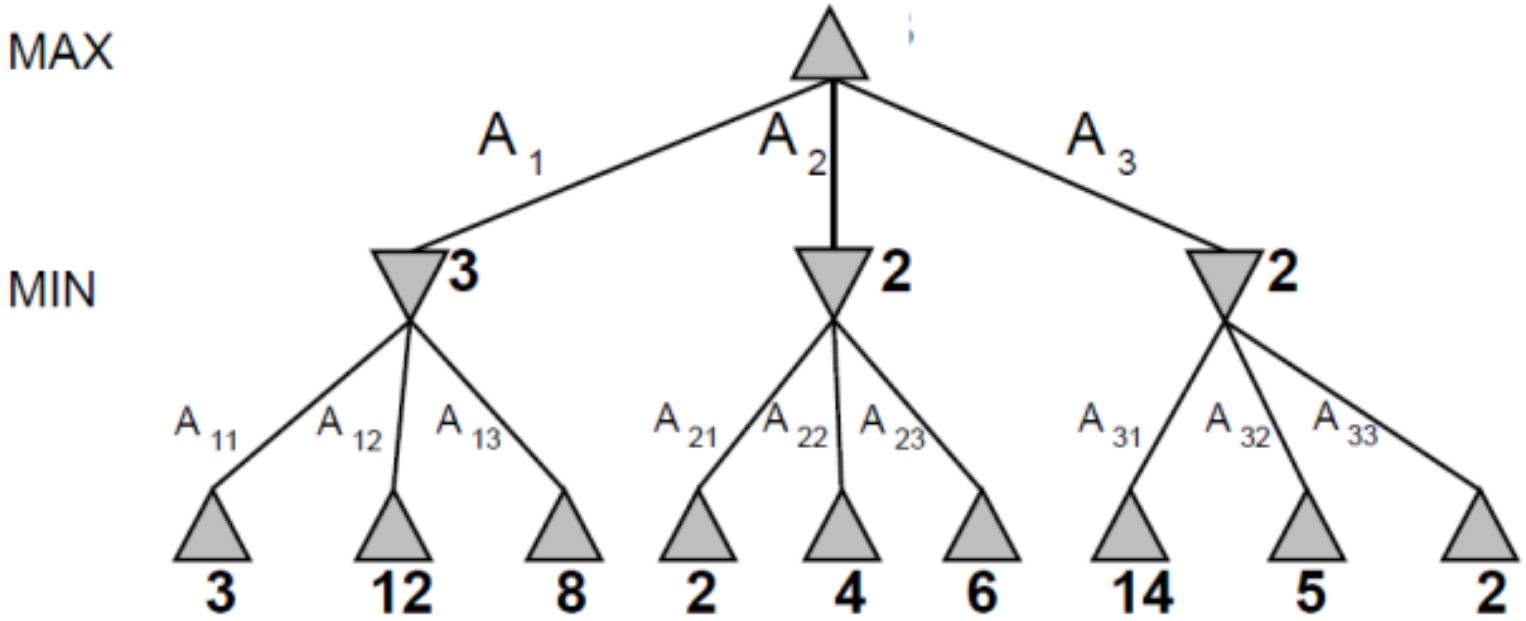
  = best achievable payoff against best play
- Example: 2-ply game:



Minimax Algorithm

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

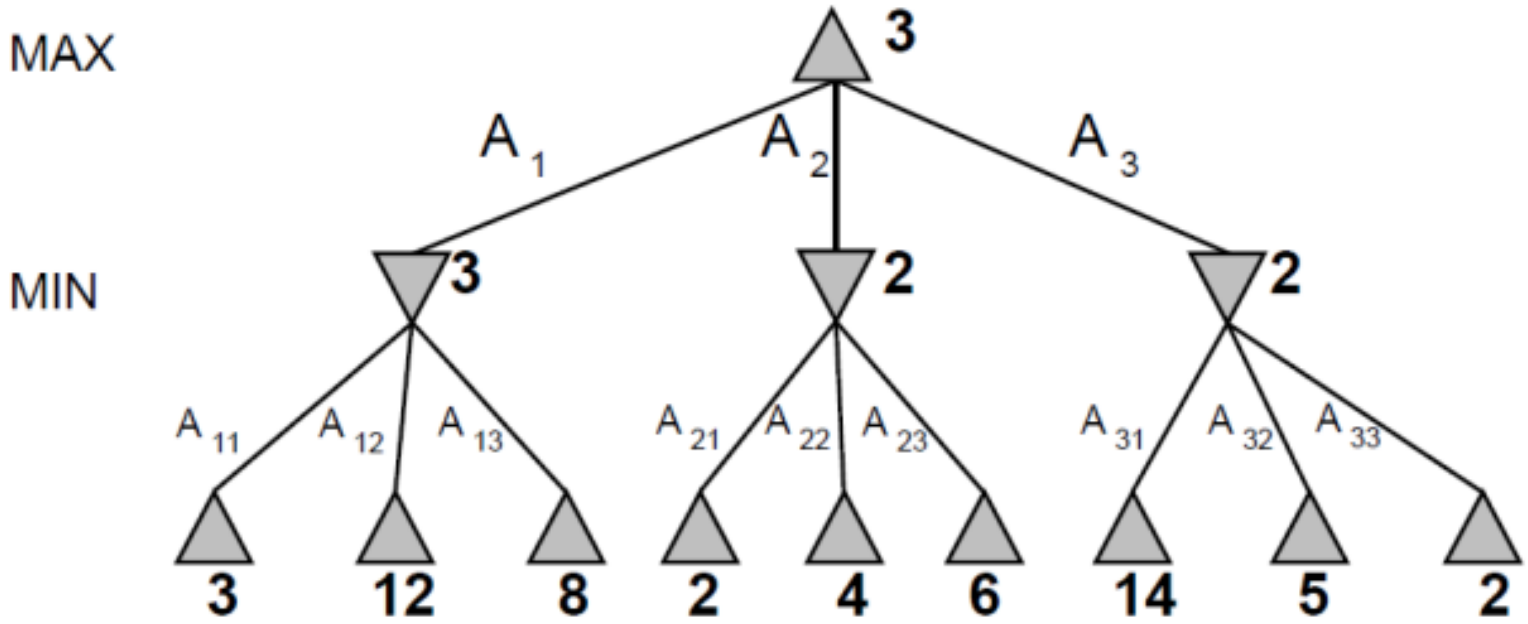
  = best achievable payoff against best play
- Example: 2-ply game:



Minimax Algorithm

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

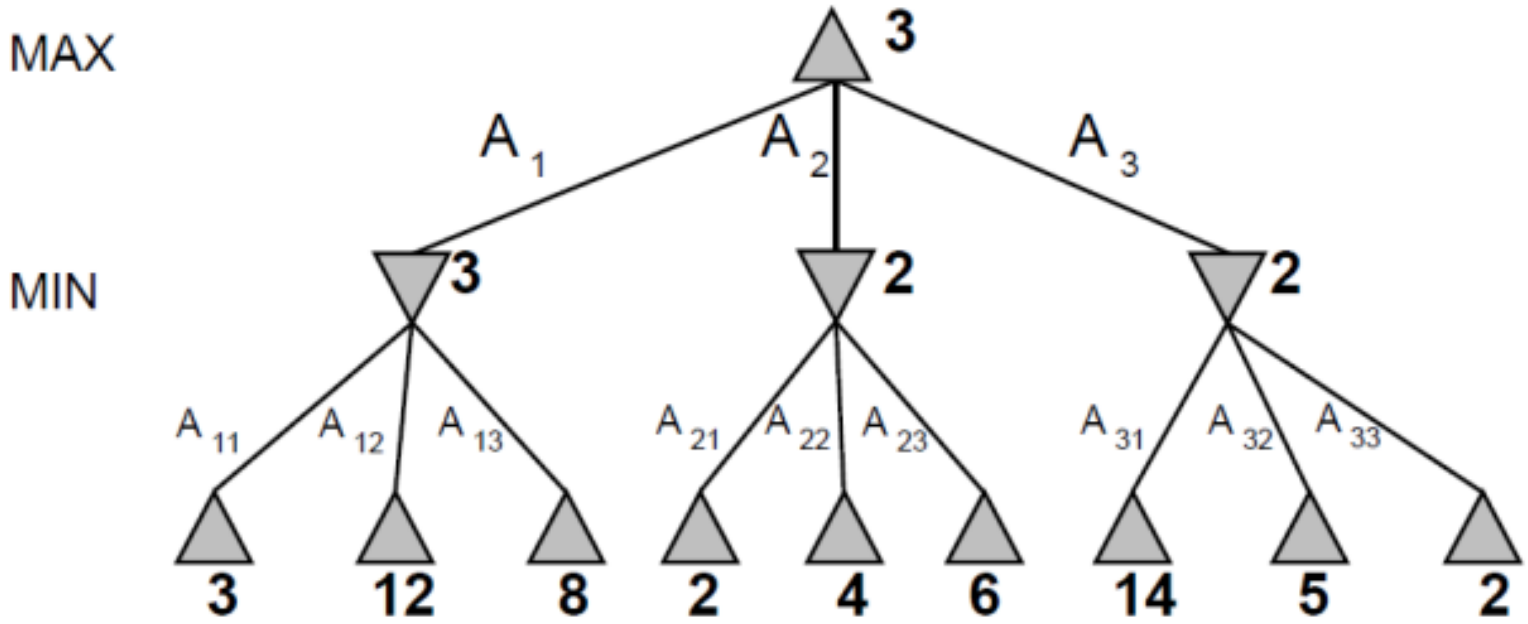
  = best achievable payoff against best play
- Example: 2-ply game:



Minimax Algorithm

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

  = best achievable payoff against best play
- Example: 2-ply game:



Minimax Algorithm

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game

    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

---

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v

---

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

Recap of Minimax

```
function minimax(position, depth, maximizingPlayer)
    if depth == 0 or game over in position
        return static evaluation of position

    if maximizingPlayer
        maxEval = -∞
        for each child of position
            eval = minimax(child, depth-1, false)
            maxEval = max(maxEval, eval)
        return maxEval
    else
        minEval = +∞
        for each child of position
            eval = minimax(child, depth-1, true)
            minEval = min(minEval, eval)
        return minEval
```

Properties of minimax

- Complete: Yes, if tree is finite (chess has specific rules for this)
- Optimal: Yes, against an optimal opponent
- Time complexity: $O\ (b^m)$
- Space complexity: $O(bm)$

Properties of minimax

- ***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:*** 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).

Multi-player Games

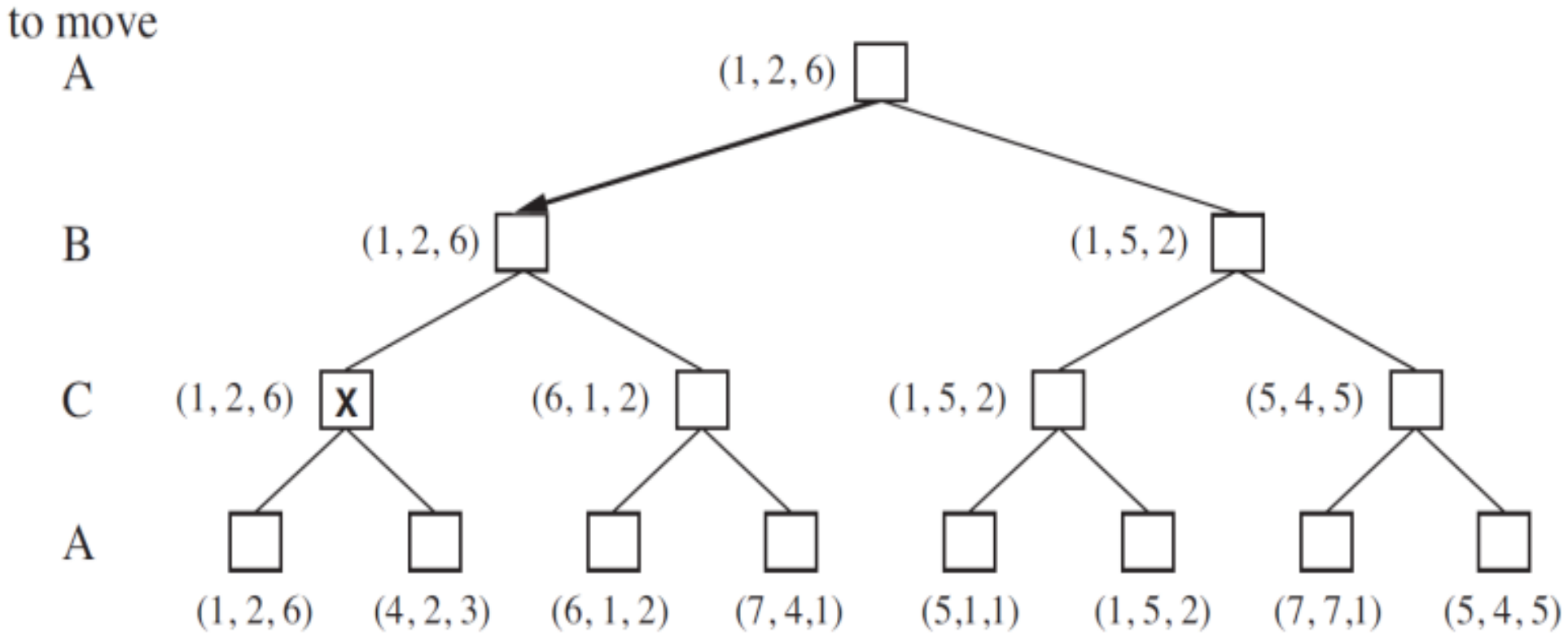


Figure 5.4 The first three plies of a game tree with three players (A, B, C). Each node is labeled with values from the viewpoint of each player. The best move is marked at the root.

Alpha-Beta Pruning

- Alpha-beta pruning is a ***modified version*** of the minimax algorithm.
- It is an ***optimization technique*** for the minimax algorithm.
- In the mini-max search algorithm the number of game states has to examine are exponential in depth of the tree.
- We cannot eliminate the exponent, but can cut it to half.
- This technique without checking each node of the game tree can compute the correct ***mini-max decision called pruning***.
- This ***involves two threshold parameter Alpha and beta*** for future expansion, so it is called ***alpha-beta pruning/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 Parameters

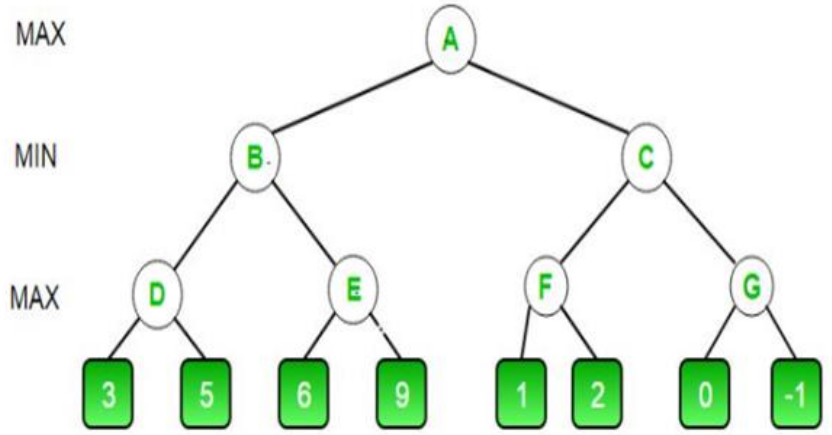
- The ***two-parameter*** can be defined as:
  - **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞.
  - **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞.
- The Alpha-beta pruning removes all the nodes which are not really affecting the final decision.
- By pruning those nodes, it makes the ***algorithm fast***.

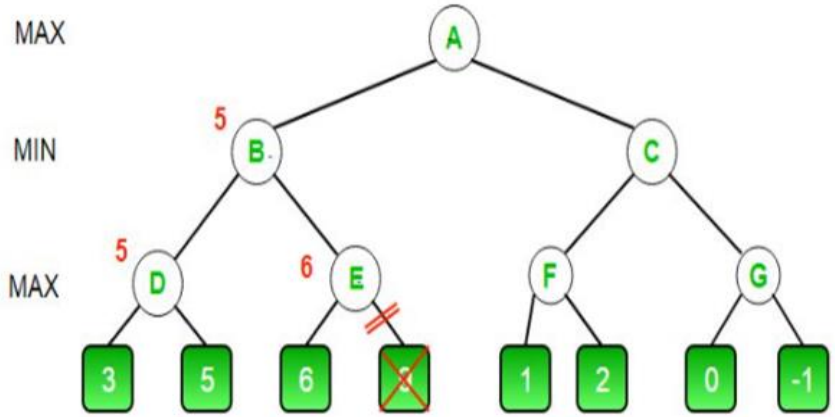
Condition for Alpha-beta pruning

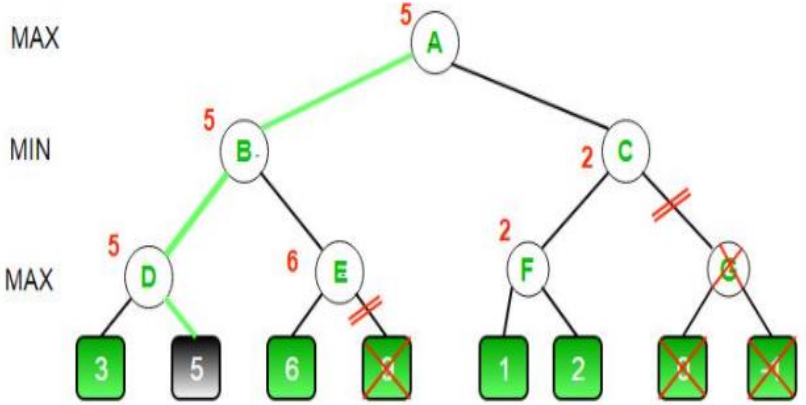
- The main condition which required for alpha-beta pruning is α>=β

Key points about alpha-beta pruning:

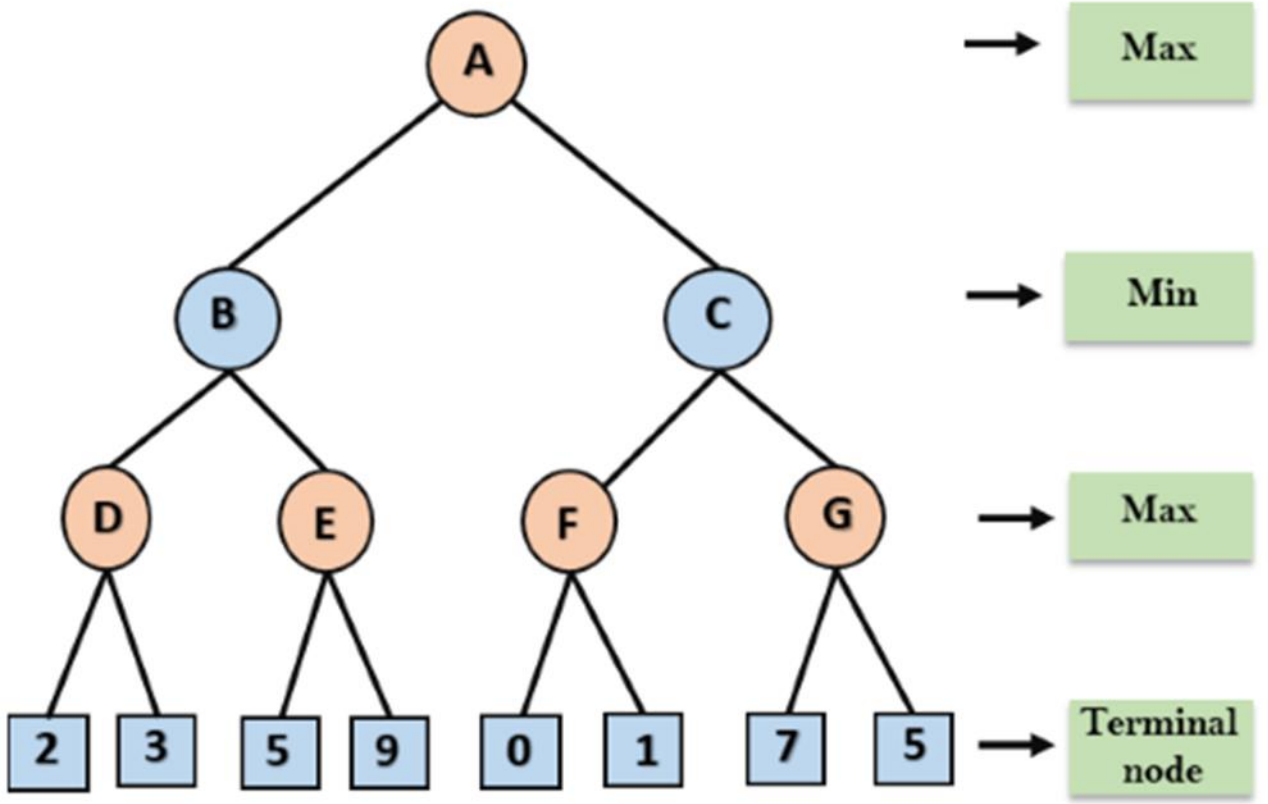
  - The Max player will only update the value of alpha.
  - The Min player will only update the value of beta.
  - While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.







Working of Alpha-Beta Pruning: Step-1

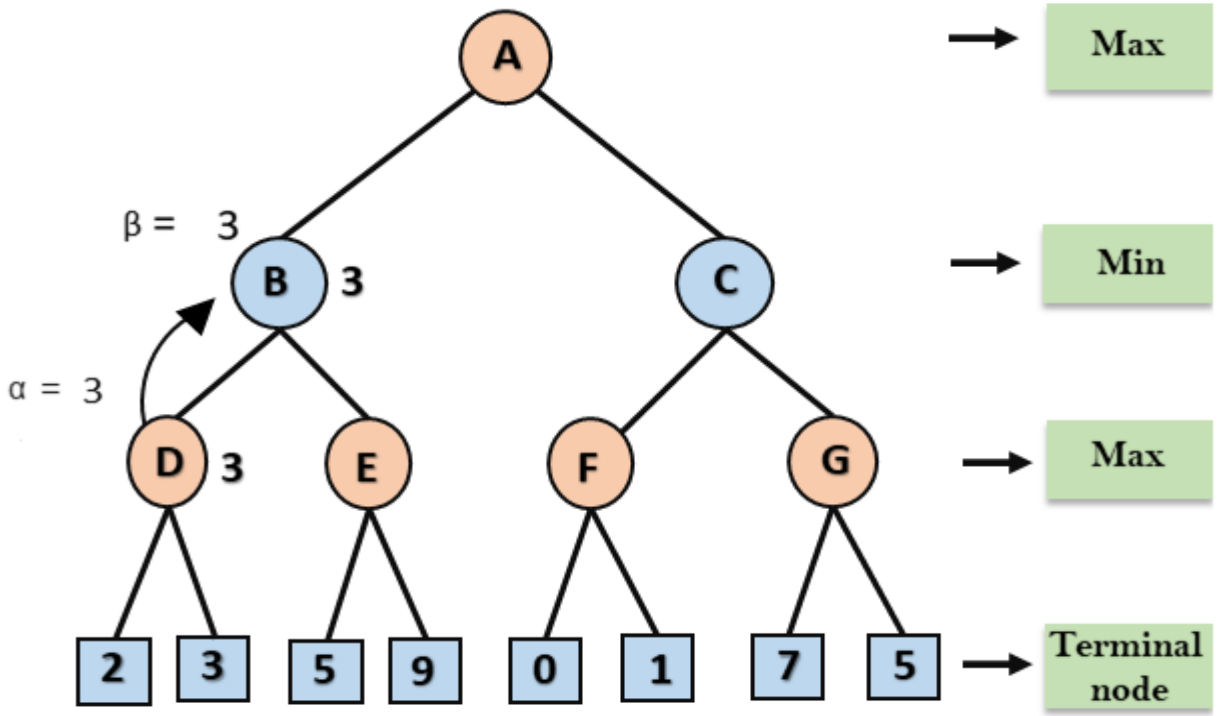


Step 1: At the first step the, Max player will start first move from node A and passed down to node B and Node B passes the same value to its child D.

Working of Alpha-Beta Pruning: Step-2,3

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.

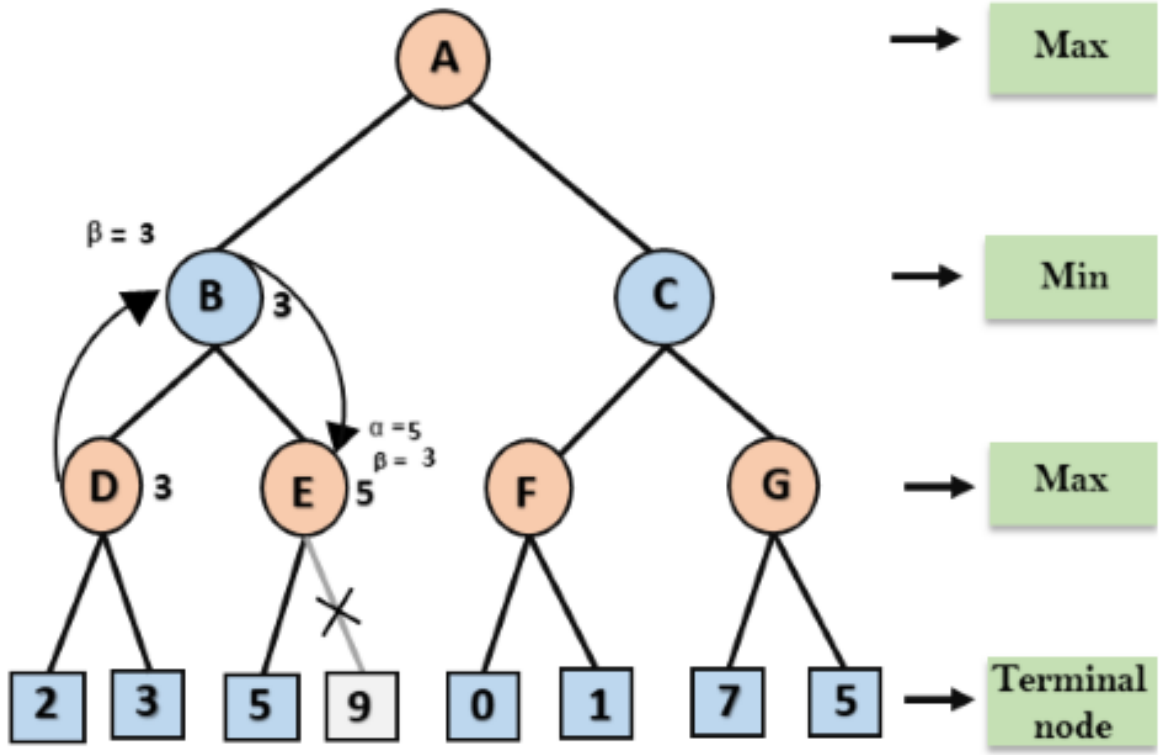
Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β will compare with the available subsequent nodes value, i.e. min = 3, hence at node B now and β= 3.



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

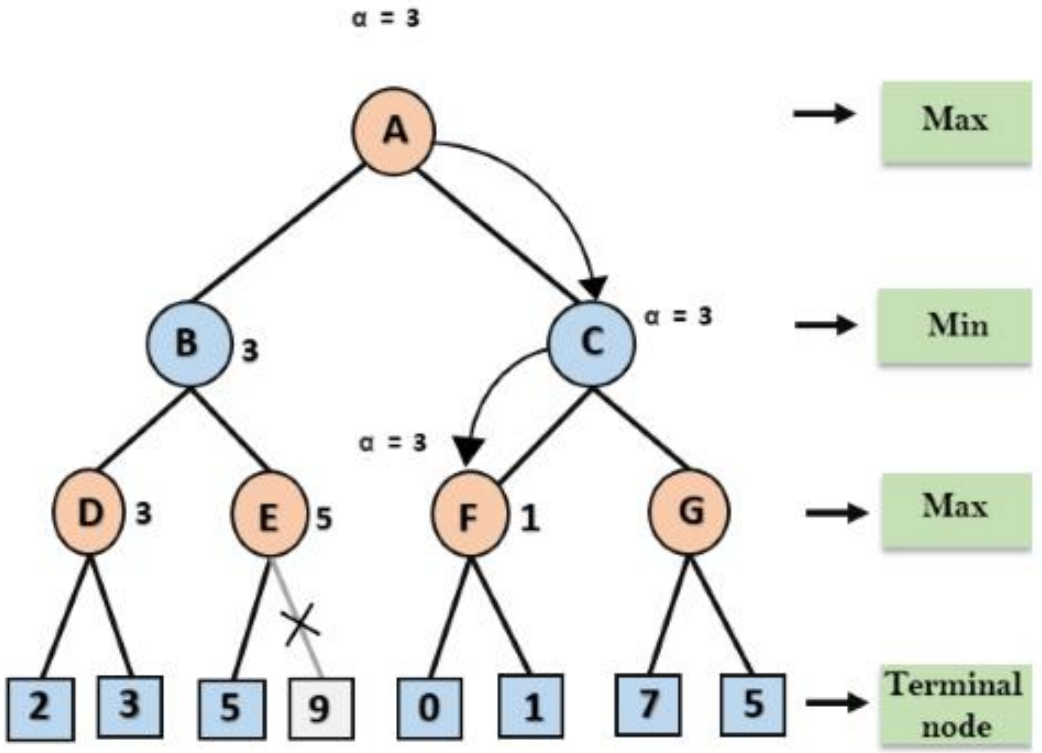
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



Working of Alpha-Beta Pruning: Step-5,6

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, and β= +∞, these two values now passes to right successor of A which is Node C.

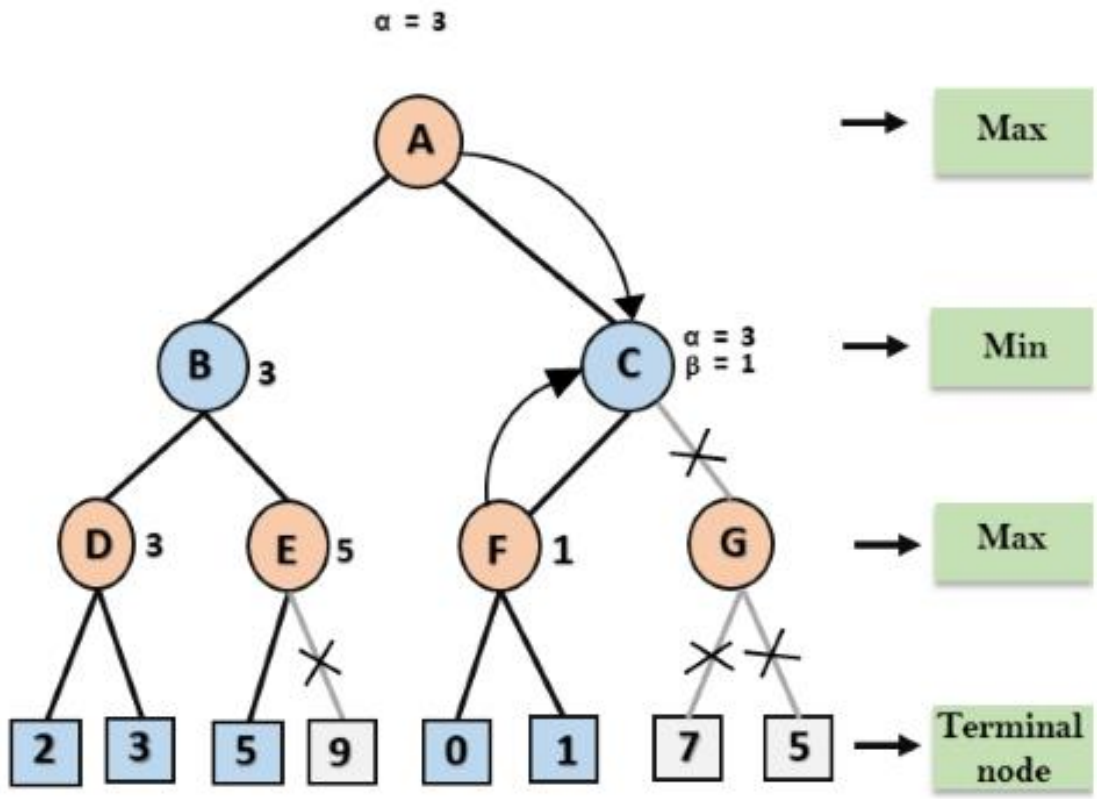
At node C, α=3 and β values will be passed on to node F.



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.

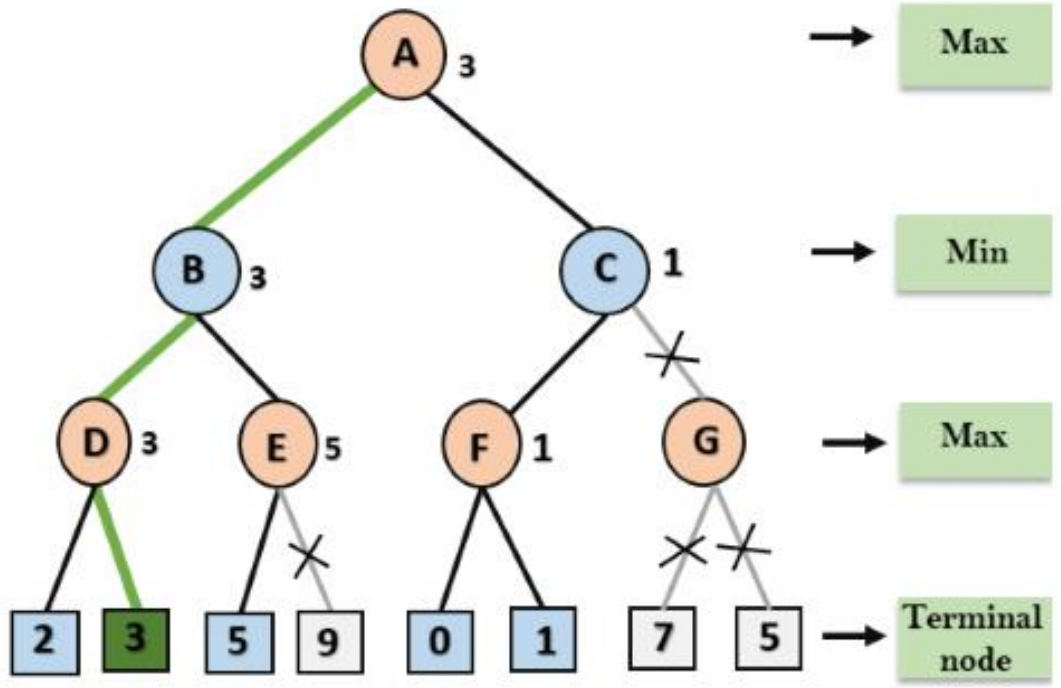
Working of Alpha-Beta Pruning: Step-7

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. 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.



The α-β Algorithm

```
function ALPHA-BETA-DECISION(state) returns an action
    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))

---

function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v

---

function MIN-VALUE(state, α, β) returns a utility value
    same as MAX-VALUE but with roles of α, β reversed
```

```
function minimax(position, depth, maximizingPlayer)
    if depth == 0 or game over in position
        return static evaluation of position

    if maximizingPlayer
        maxEval = -∞
        for each child of position
            eval = minimax(child, depth-1, false)
            maxEval = max(maxEval, eval)
        return maxEval
    else
        minEval = +∞
        for each child of position
            eval = minimax(child, depth-1, true)
            minEval = min(minEval, eval)
        return minEval
```

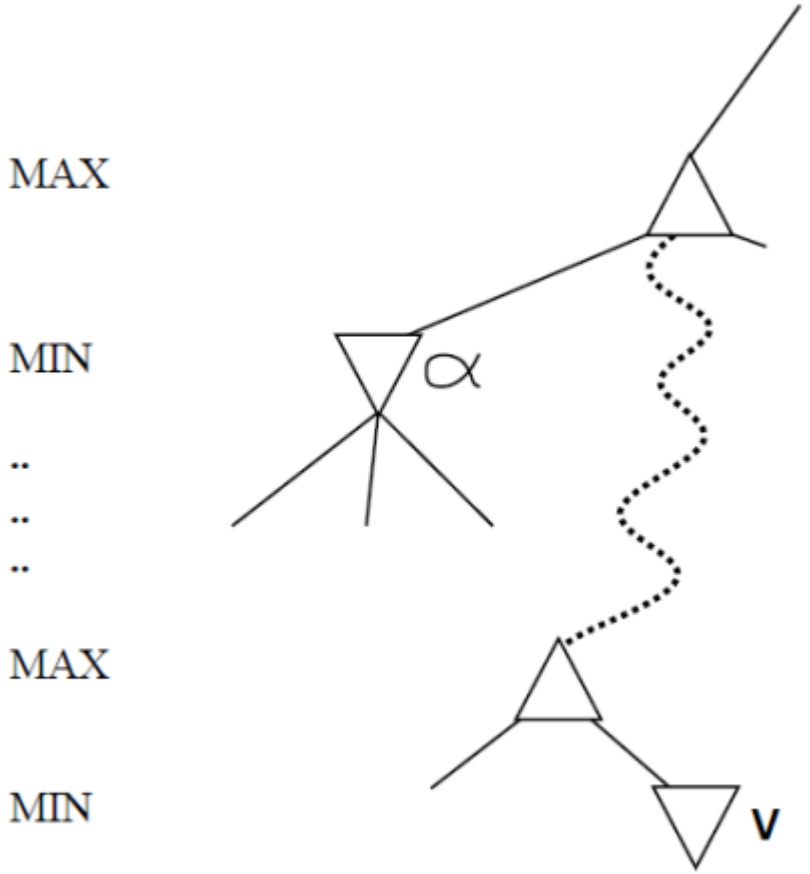
This was Minimax

α-β Pruning

```
function minimax(position, depth, alpha, beta,  maximizingPlayer)
        if depth == 0 or game over in position
                return static evaluation of position

        if maximizingPlayer
                maxEval = -∞
                for each child of position
                        eval = minimax(child, depth-1, alpha, beta, false)
                        maxEval = max(maxEval, eval)
                        alpha = max(alpha, eval)
                        if beta <= alpha
                                break
                return maxEval
        else
                minEval = +∞
                for each child of position
                        eval = minimax(child, depth-1, alpha, beta, true)
                        minEval = min(minEval, eval)
                        beta = min(beta, eval)
                        if beta <= alpha
                                break
                return minEval
```

Why is it called α-β?



- - ∝ is the best value (to MAX) found so far off the current path.
- If *V* is worse than ∝, MAX will avoid it → prune that branch.
- Define β similarly for MIN

Properties of α-β

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With perfect ordering, time complexity = $O(b^{\frac{m}{2}})$
  - Doubles solvable depth
- A simple example of the value of reasoning about which computations are relevant (a form of meta-reasoning)
- Unfortunately, 35^{50} is still not possible!
  - Remember: for chess, $b \approx 35, m \approx 100$, for "reasonable" games
    - → exact solution completely infeasible

The α-β Algorithm

I will try to include some more example slides for you people to understand this algorithm with different examples.

Chapter Reading

Chapter 5

Artificial Intelligence, A Modern Approach”

by

Stuart Russell and Peter Norivg, 4th edition.

Questions?