CS 3011: Artificial Intelligence

Adversarial Search

Instructors: Dr. Durgesh Singh

CSE Discipline, PDPM IIITDM, Jabalpur -482005

Adversarial Search

- Search in competitive environments
    - Where agents' goals are in conflict
- Games are good examples of adversarial search
    - States are easy to represent
    - Agents are restricted to a small number of actions
        - Two agents whose actions must alternate
    - Outcome of agent actions defined by precise rules.
    - Utility values at the end of the game are always equal and opposite
        - +1 = Chess winner
        - -1 = Chess looser.

Games vs. Search problems

- In typical search problems, we optimize a measure to acquire the goal: there is no opponent
- In games, there is an "Unpredictable" opponent
  - Need to specify a move for every possible opponent reply

Game Formulation

- Imagine 2 players of tic-tac-toe: MAX and MIN
  - MAX moves first
  - The terminal states are at the leaves
- MAX should play in order to maximize its utility, which will minimize the utility for MIN
  - This is called a Zero-Sum Game.

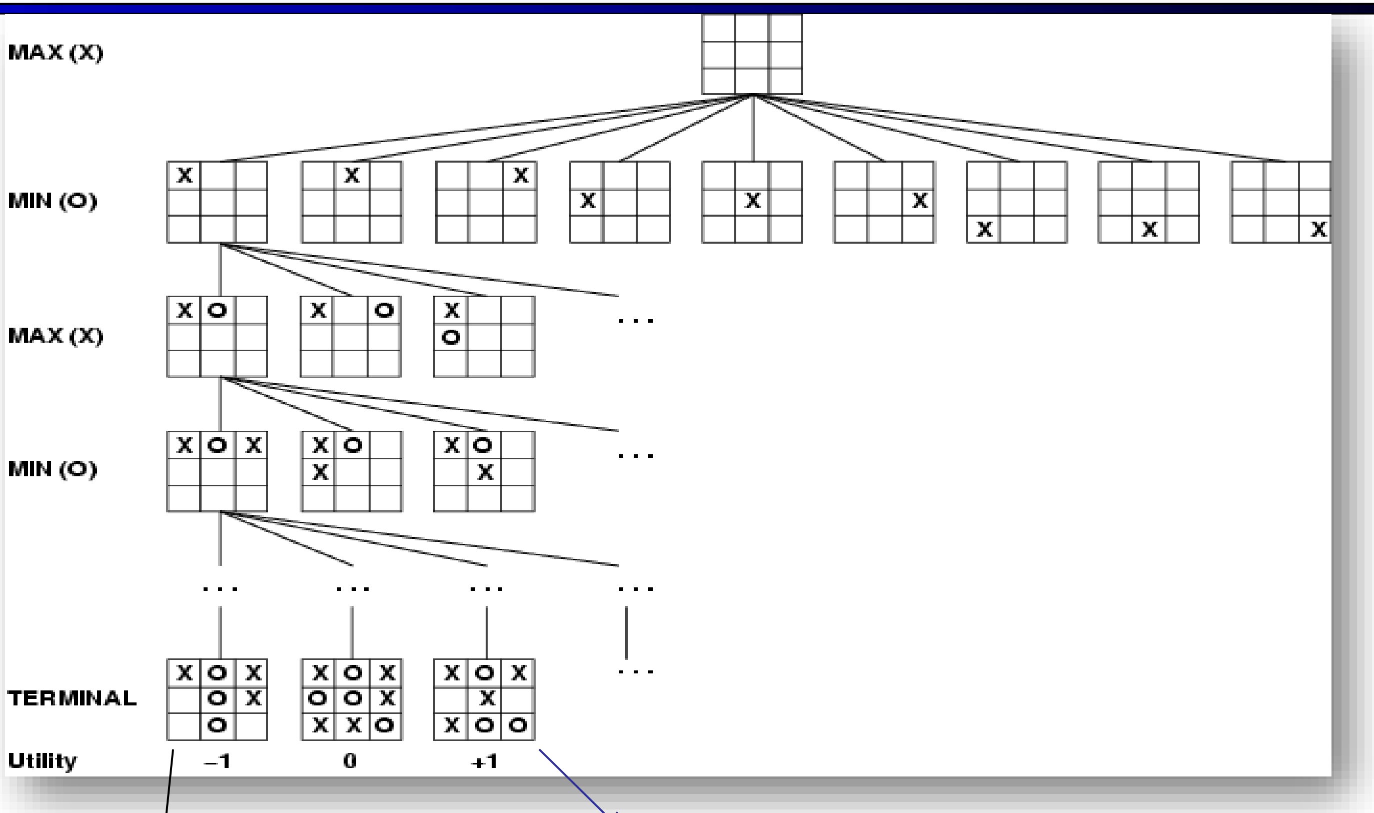
Game Formulation…

A game can be formally defined as search problem with the following elements:

- **Initial state :** S0
- **TO-MOVE(S) ::** Define which player has the move in a state S.
- **Action(S):** Returns the set of legal moves in a state S.
- **Results (S, a):** The transition model which defines the result of a move
- **Terminal-Test (S):** True when game is over, false otherwise

- **Utility(S, P):**
  - Also called an objective function or payoff function
  - Returns the final numeric value for a game that ends in a terminal state S for player P
  - For example, in Tic-Tac –Toe, the outcome is win, loss, or draw with values 1,-1, or 0
- Initial state, Actions function, and Result function define game tree for a game
  - Game tree: a tree where nodes are game states and edges are moves

Game tree for Tic Tac Toe



This terminal state is one of the worst for MAX and one of the best for MIN

This terminal state is one of the best for MAX and one of the worst for MIN

Optimal Strategy For MAX

- In discovering the sequence of optimal moves by MAX, we have to consider that MIN is taking moves as well
  - So, the optimal strategy will tell how should MAX play against MIN in order to win the game
- The optimal strategy can be determined from the Minimax value of each node

Minimax

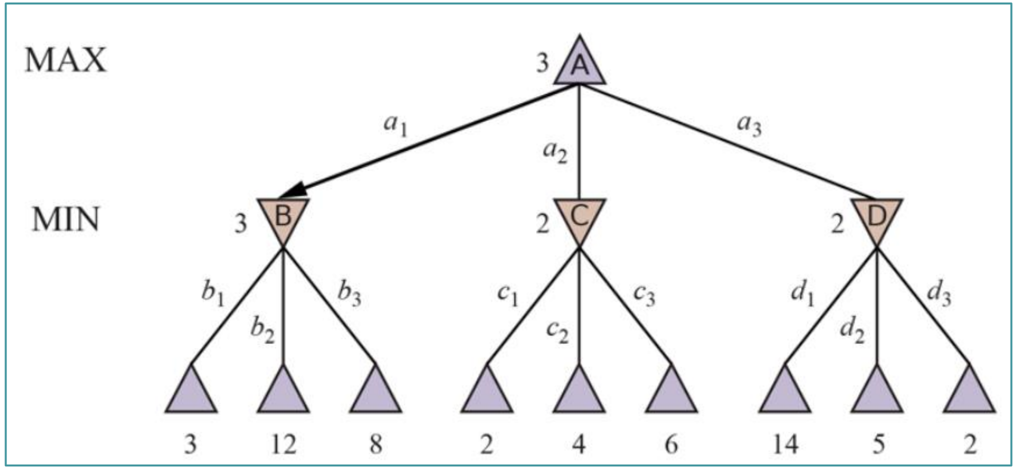
- When it is the turn of MAX, it will always take an action in order to maximize its utility, because it's winning configurations have high utilities
- When it is the turn of MIN, it will always take an action in order to minimize its utility, because it's winning configurations have low utilities
- In order to implement this, we need to define a measure in each state that takes the move of the opponent into account:
  - This measure is called Minimax.

Minimax

- Minimax represents the utility of a state, *given that both MAX and MIN will play optimally till the end of the game*
- In any state *s*, one or more actions are possible
- For every possible new state that can be transited into from *s*, we compute the minimax value
- The term "Minimax" is used because:
  - the opponent is always trying to minimize the utility of the player, and
  - the player is always trying to maximize this minimized selection of the opponent.

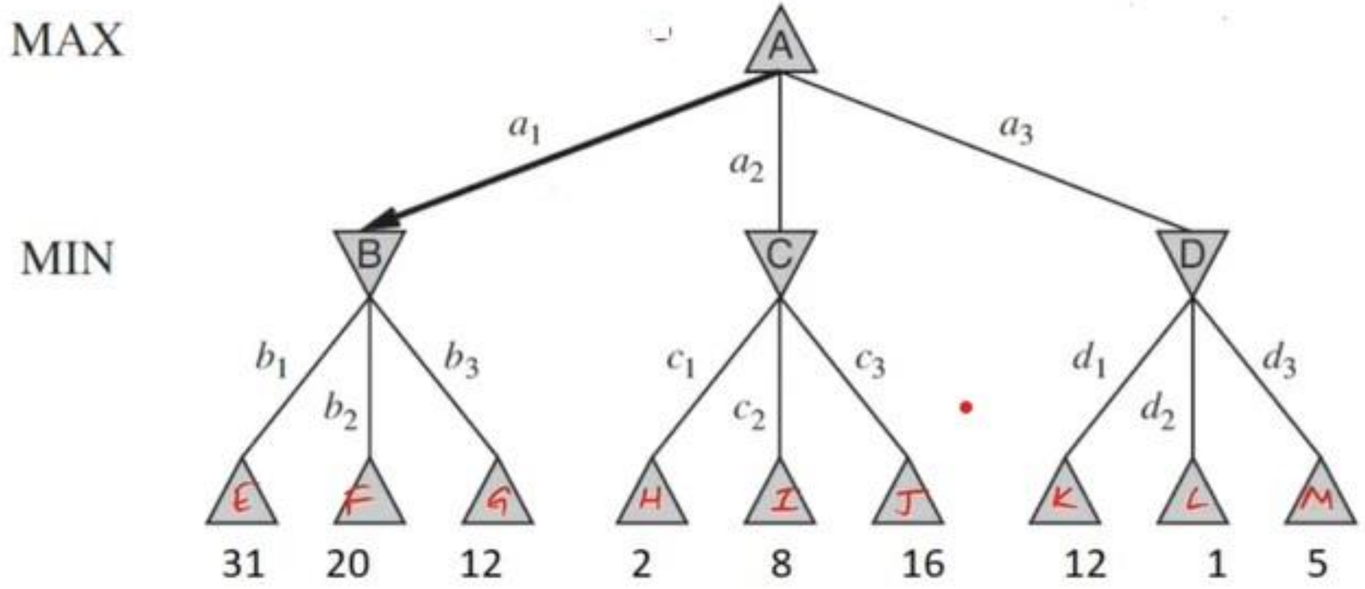
Minimax

- Minimax value of each state in the tree, which we write as MINIMAX(s).
- The minimax value of a terminal state is just its utility.

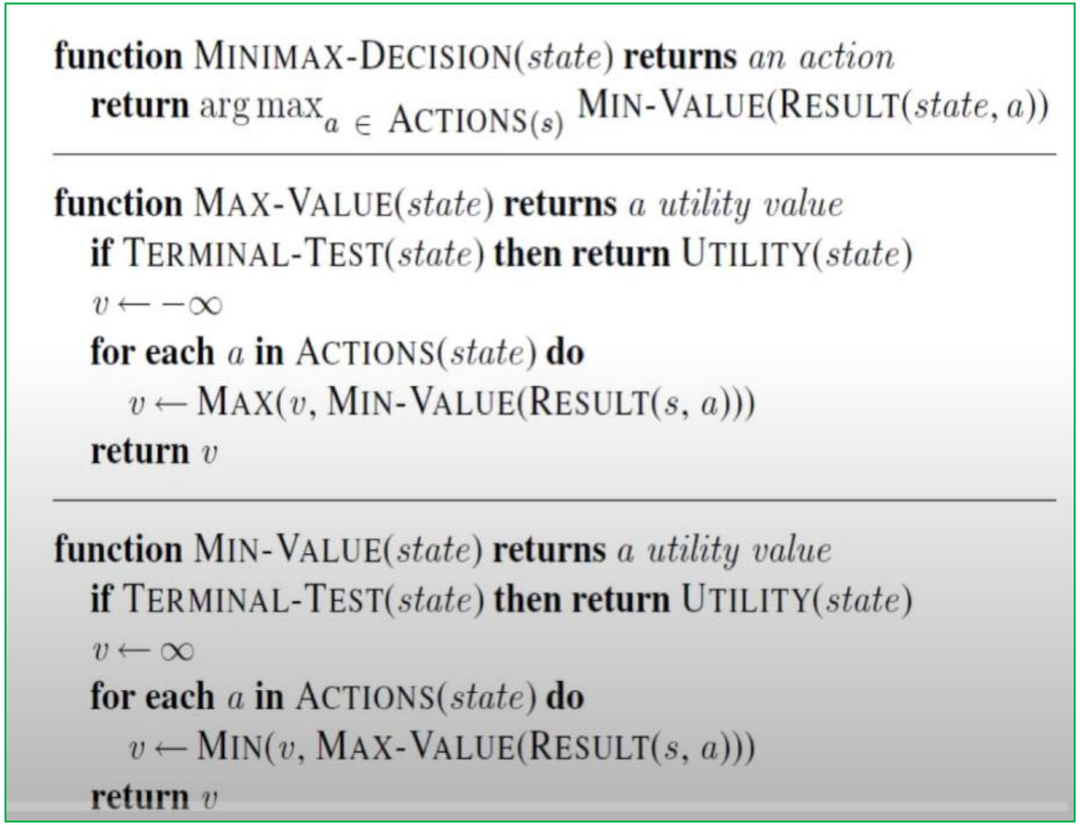


$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s, \text{MAX}) & \text{if IS-TERMINAL}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if TO-MOVE}(s) = \text{MIN} \end{cases}$$

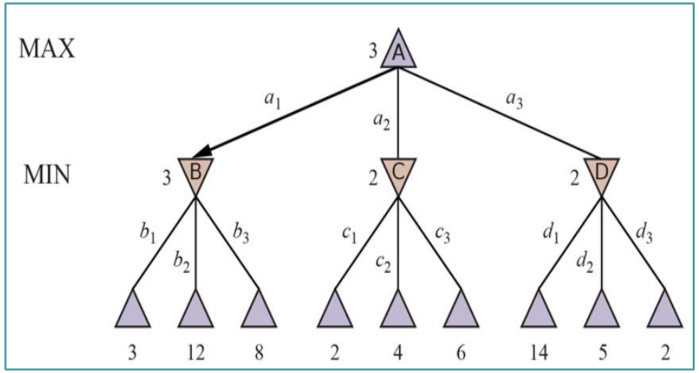
Find the Minimax (A)



Minimax algorithm



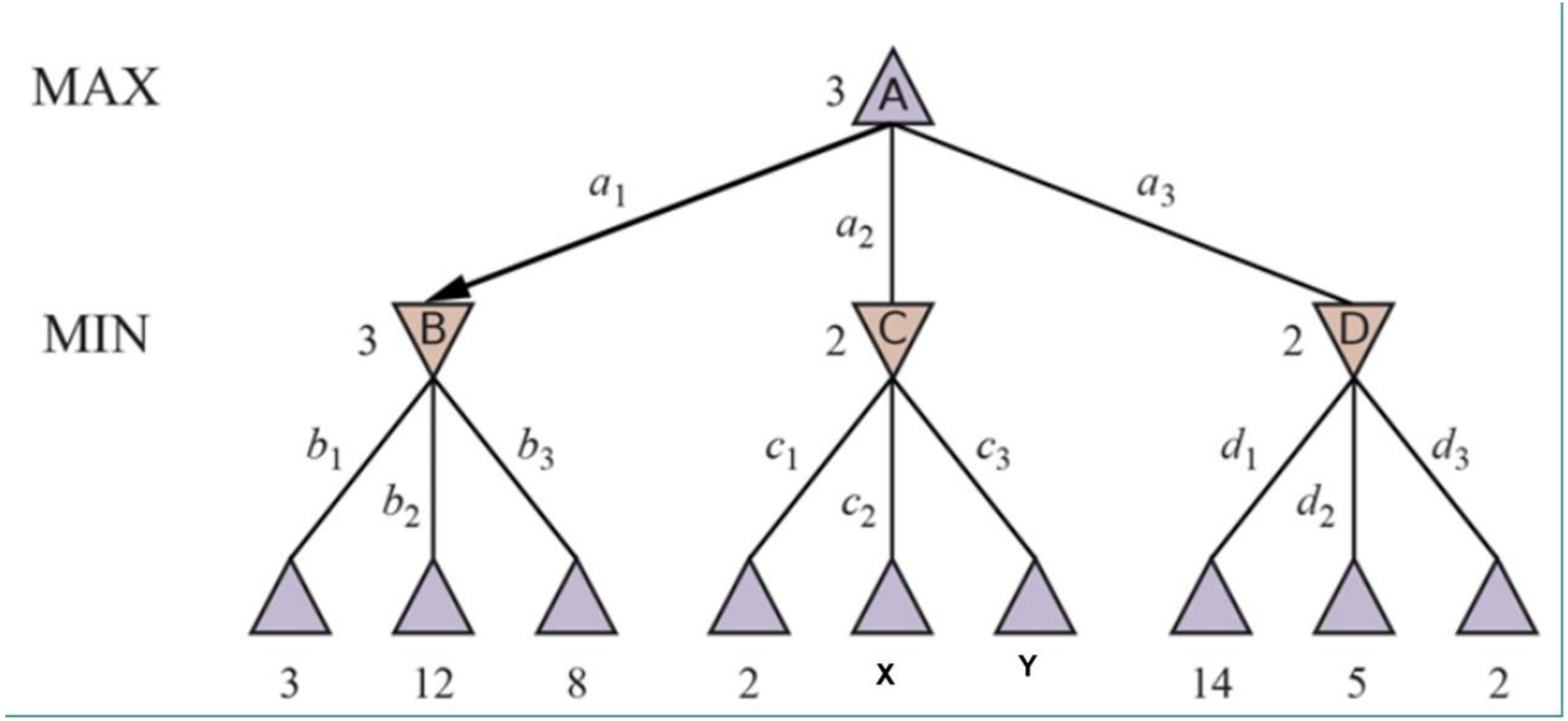
```
function MINIMAX-DECISION(state) returns an action
  return $\arg\max_{a \in \text{ACTIONS}(s)}$ MIN-VALUE(RESULT(state, a))

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

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



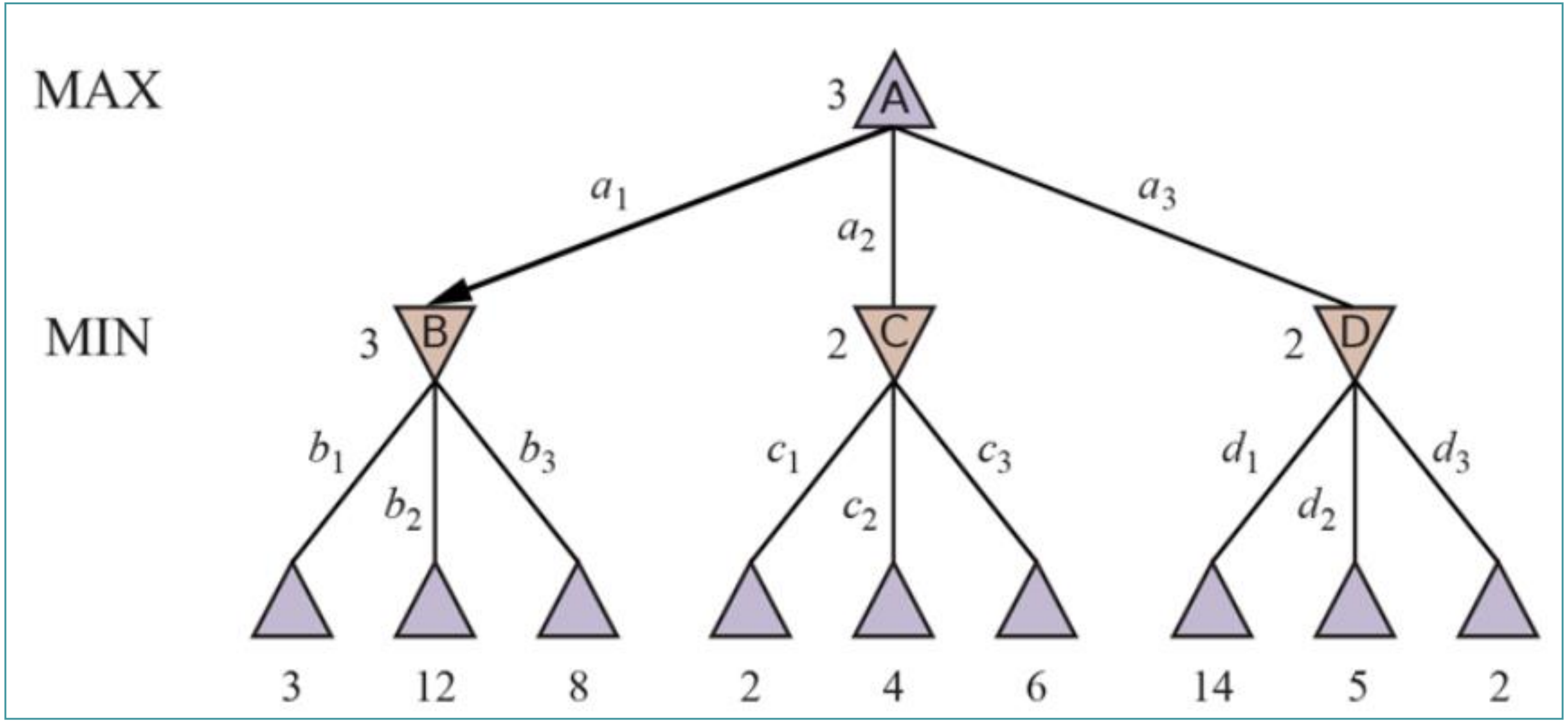
Properties of Minimax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? O(b^m)
- Space complexity? O(bm) (depth-first exploration)
- For chess, b ≈ 35, m ≈100 for "reasonable" games
  → exact solution completely infeasible
  - We need to think of a way to cut down the number of search paths.

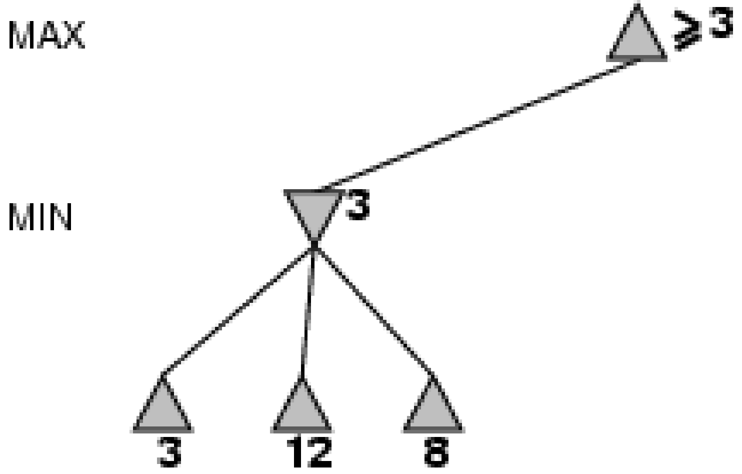
Find the Minimax value of root node



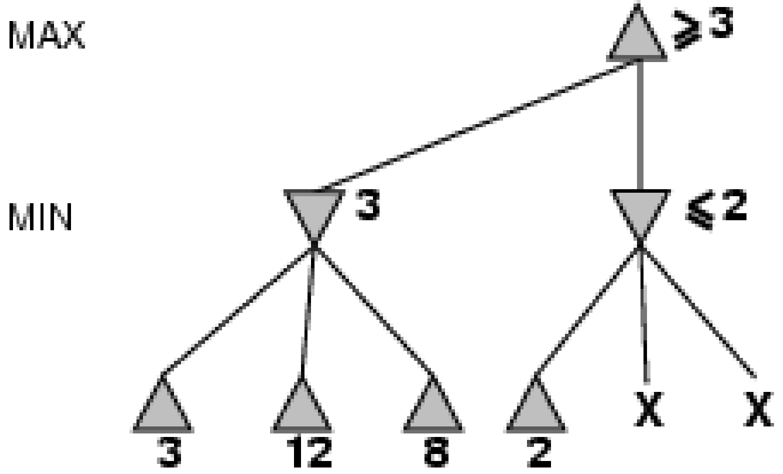
check the nodes that not need to explore



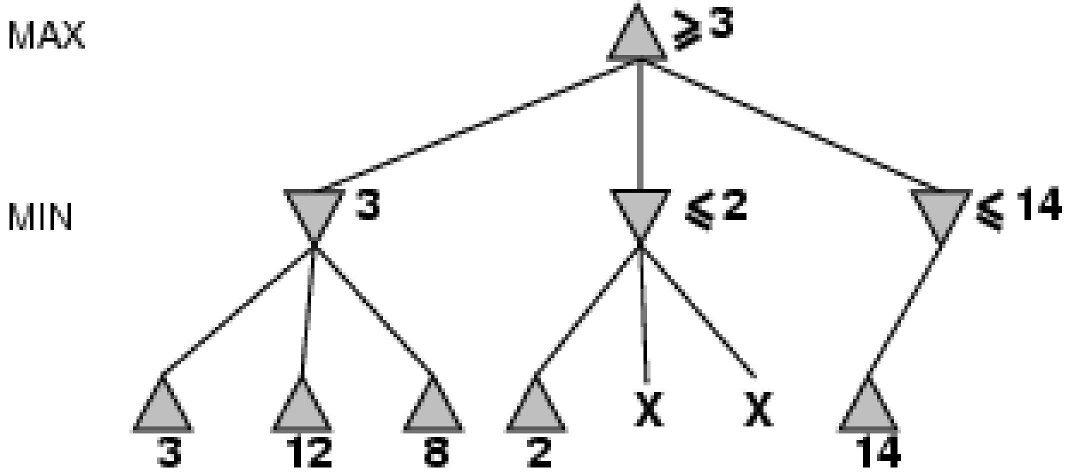
α-β pruning example



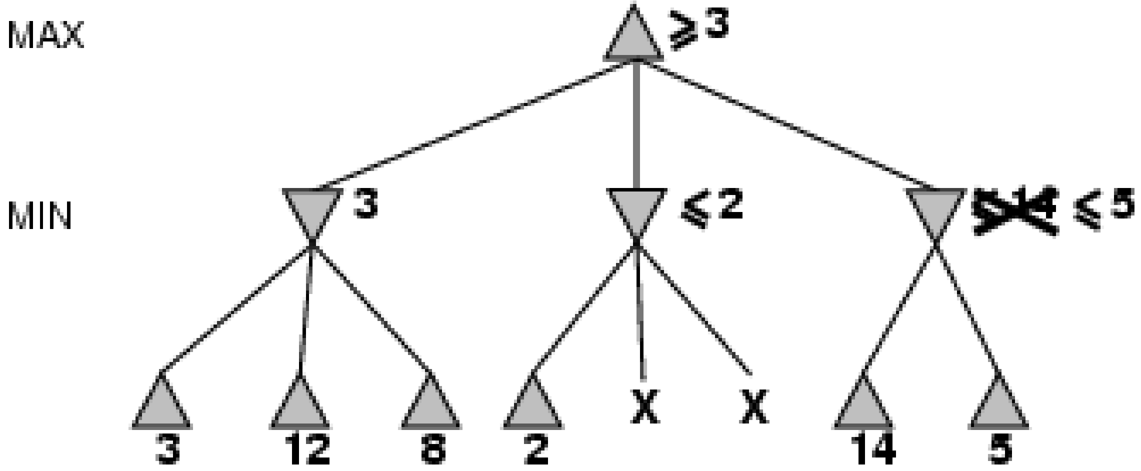
α-β pruning example



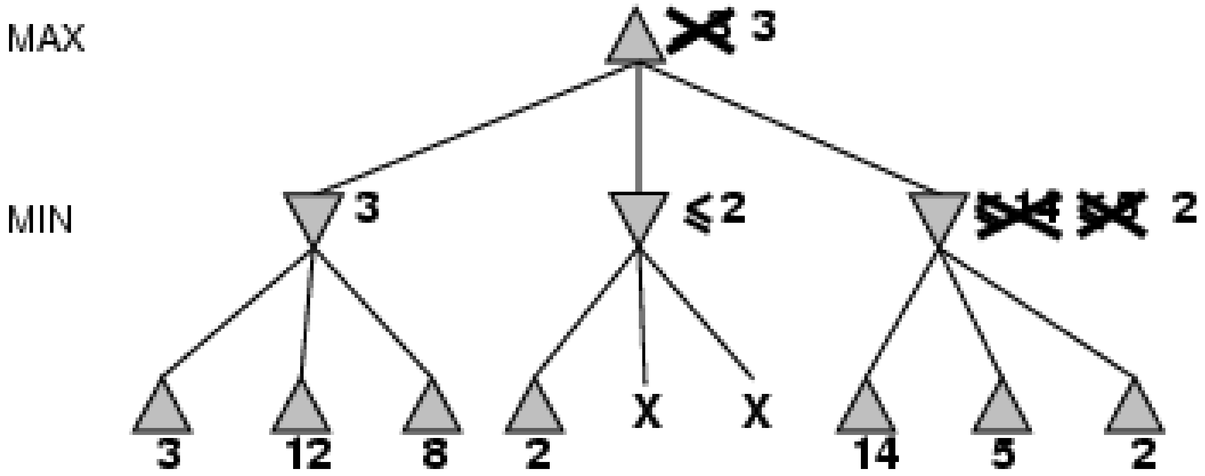
α-β pruning example



α-β pruning example



α-β pruning example

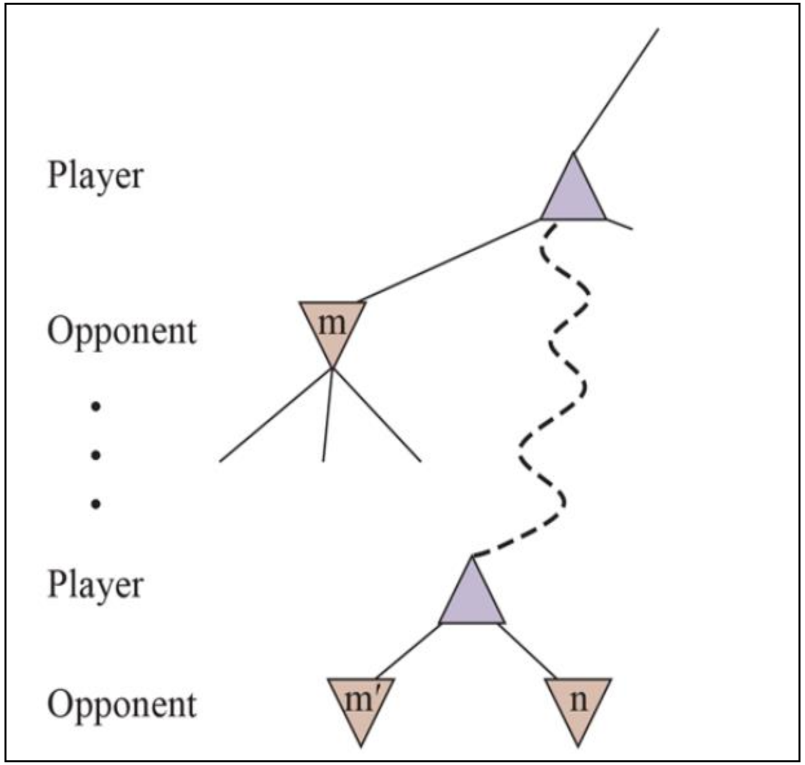


Alpha–Beta Pruning

- We can improve on the performance of the minimax algorithm through alpha-beta pruning
- Alpha–beta pruning can be applied to trees of any depth, and it is often possible to prune entire subtrees rather than just leaves.

Alpha–Beta Pruning: General Principle

- Consider a node n somewhere in the tree such that Player has a choice of moving to n .
  - If Player has a better choice either at the same level (i.e., m’) or at any point higher up in the tree (i.e., m), then Player will never move to n.
  - So, once we have found out enough about n to reach this conclusion, we can prune it.



How the Alpha-Beta Pruning Work?

- Minimax search is depth-first, so at any one time we just have to consider the nodes along a single path in the tree.
- Alpha–beta pruning gets its name from the two extra parameters used here **α and** β that describe bounds on the backed-up values that appear anywhere along the path.
- Alpha–beta search updates the values of **α and** β as it goes along and prunes the remaining branches at a node (i.e., terminates the recursive call) as soon as the value of the current node is known to be worse than the current **α or** β value for MAX or MIN, respectively.

What is Alpha

- Alpha (α) is the value of the best (i.e., highest-value) choice we have found so far at any choice point along the path for MAX. Think: α = “at least.”
- The MAX node only update the value of **α**
- **Initially, α = - ∞**

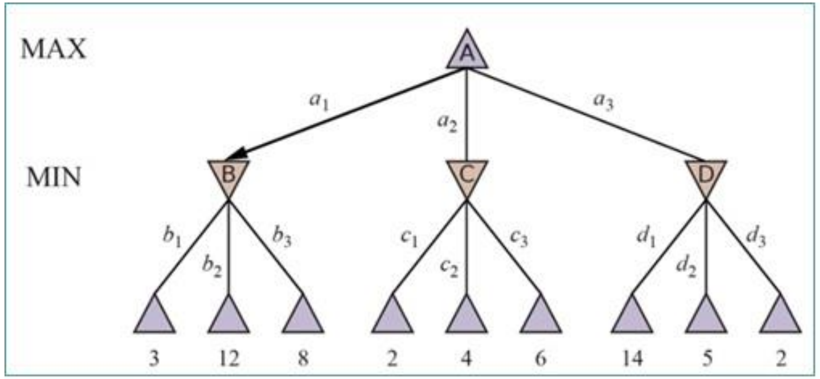
What is Beta

- **Beta (β)** is the value of the best (i.e., lowest-value) choice we have found so far at any choice point along the path for MIN.

  Think: β= “at most.”
- The MIN node only update the value of β
- **Initially,** β **= + ∞**

Alpha–Beta Pruning …

```
function ALPHA-BETA-SEARCH($state$) returns an action
    $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
    return the $action$ in ACTIONS($state$) with value $v$

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

function MIN-VALUE($state, \alpha, \beta$) returns $a$ $utility$ $value$
    if TERMINAL-TEST($state$) then return UTILITY($state$)
    $v \leftarrow +\infty$
    for each $a$ in ACTIONS($state$) do
        $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s$,$a$) , $\alpha$, $\beta$))
        if $v \leq \alpha$ then return $v$
        $\beta \leftarrow$ MIN($\beta$, $v$)
    return $v$
```



Effectiveness of alpha–beta pruning

- Alpha-beta is guaranteed to compute the same value for the root node as computed by minimax, with less or equal computation.
- The effectiveness of alpha–beta pruning is highly dependent on the order in which the states are examined.
- **Move order is an important aspect of alpha –beta pruning**
  - **Best case** is when each player's best move is the first alternative generated
  - **Worst case:** no pruning

Properties of α-β

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering" time complexity = $O(b^{m/2})$