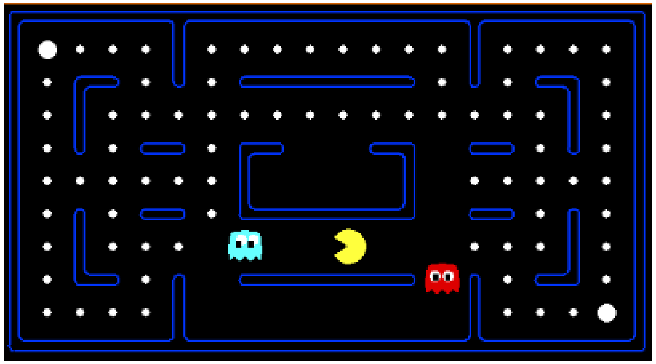
Lecture 10-11
Artificial Intelligence

Khola Naseem
khola.naseem@uet.edu.pk



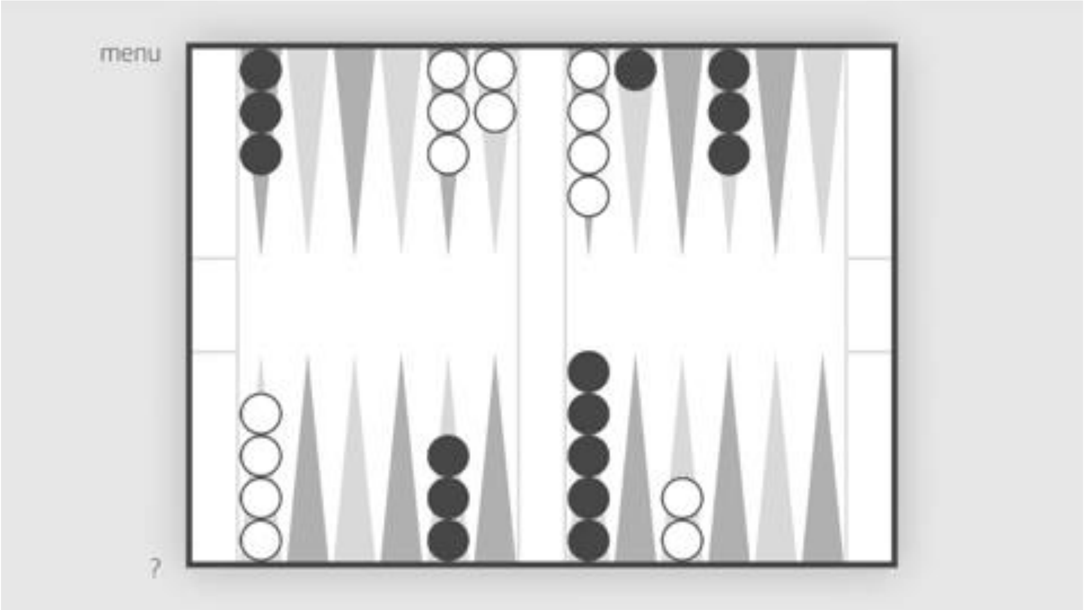
Credit: Khola Naseem

Adversarial Search

- Multi agent environments : any given agent will need to consider the actions of other agents and how they affect its own welfare.
- The unpredictability of these other agents can introduce many possible contingencies

Adversarial Search

- Examine the problems that arise when we try to plan ahead in a world where other agents are planning against us.
- Competitive environments, in which the agent's goals are in conflict require adversarial search
- A good example is in board games, Chess, Tic-tac-toe, backgammon and many more



Credit: Khola Naseem

Typical AI assumptions

- AI games are a specialized kind - deterministic, turn taking, two-player, zero sum games of perfect information
- a zero-sum game is a mathematical representation of a situation in which a participant's gain (or loss) of utility is exactly balanced by the losses (or gains) of the utility of other participant(s)
- In our terminology – deterministic, fully observable environments with two agents whose actions alternate and the utility values at the end of the game are always equal and opposite (+1 and –1)
- If a player wins a game of chess (+1), the other player necessarily loses (-1)

Search vs 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. Why?
  - Time limits force an approximate solution
  - Evaluation function: evaluate "goodness" of game position
  - Examples: chess, checkers

Size of search tree

- Games, unlike most of the toy problems are interesting because they are too hard to solve.
  - b = branching factor
  - d = number of moves by both players
  - Search tree is O(bd)
- For example, chess has an average branching factor of about 35, and games often go to 50 moves by each player, so the search tree has about 35^100 nodes

Games step up

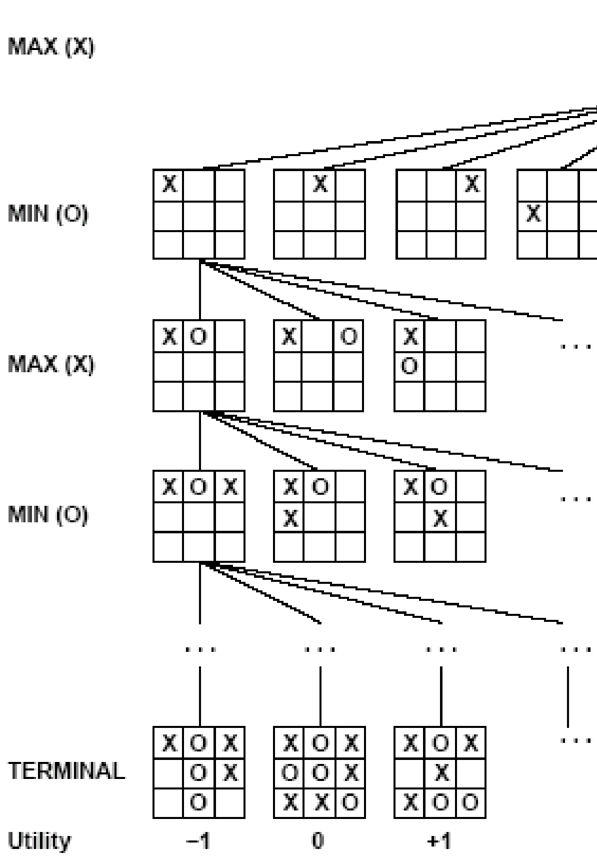
- Two players: MAX and MIN
- MAX moves first and they take turns until the game is over
  - Winner gets award, loser gets penalty.
- Games as search:
  - Initial state(S0): The initial state, which specifies how the game is set up at the start.
  - PLAYER(s): Defines which player has the move in a state.
  - ACTIONS(s): Returns the set of legal moves in a state.
  - RESULT(s, a): The transition model, which defines the result of a move.
  - TERMINAL-TEST(s): A terminal test, which is true when the game is over and false otherwise. States where the game has ended are called terminal states.
  - UTILITY(s, p): A utility function (also called an objective function or payoff function), defines the final numeric value for a game that ends in terminal state s for a player p. In chess, the outcome is a win, loss, or draw, with values +1, 0, or 1/2 .

Deterministic Games

- Many possible formalizations, one is:
  - States: S (start at s_0)
  - Players: P={1...N} (usually take turns)
  - Actions: A (may depend on player / state)
  - Transition Function: SxA →S
  - Terminal Test: S → {t,f}
  - Terminal Utilities: SxP → R

Partial Game Tree for Tic-Tac-Toe

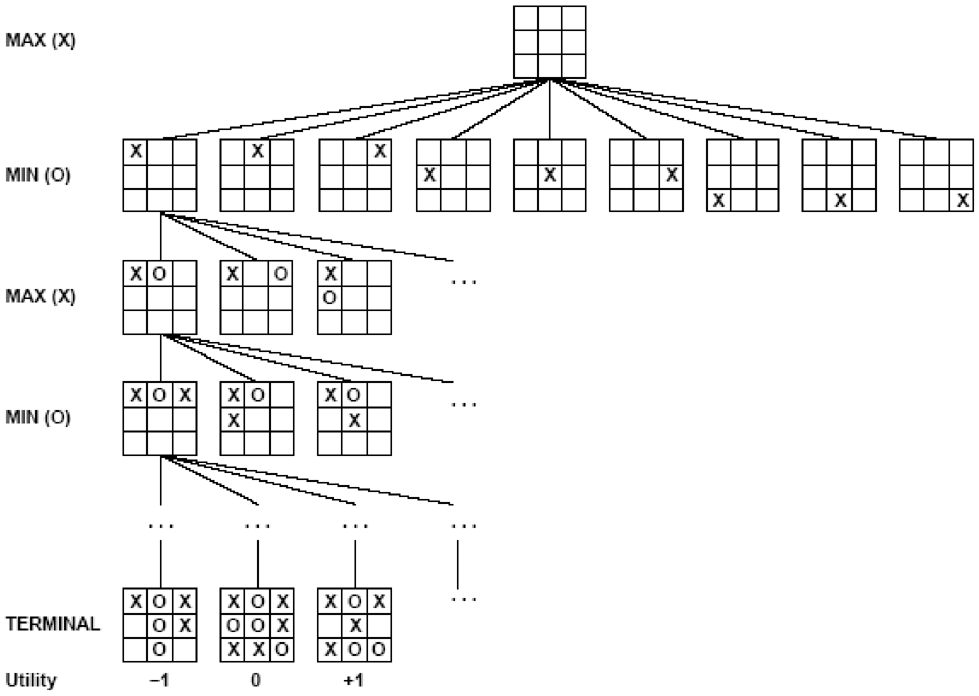
- The initial state, ACTIONS function, and RESULT function define the game tree for the game—a tree where the nodes are game states and the edges are moves



Credit: Khola Naseem

Partial Game Tree for Tic-Tac-Toe

- The initial state, ACTIONS function, and RESULT function define the game tree for the game—a tree where the nodes are game states and the edges are moves



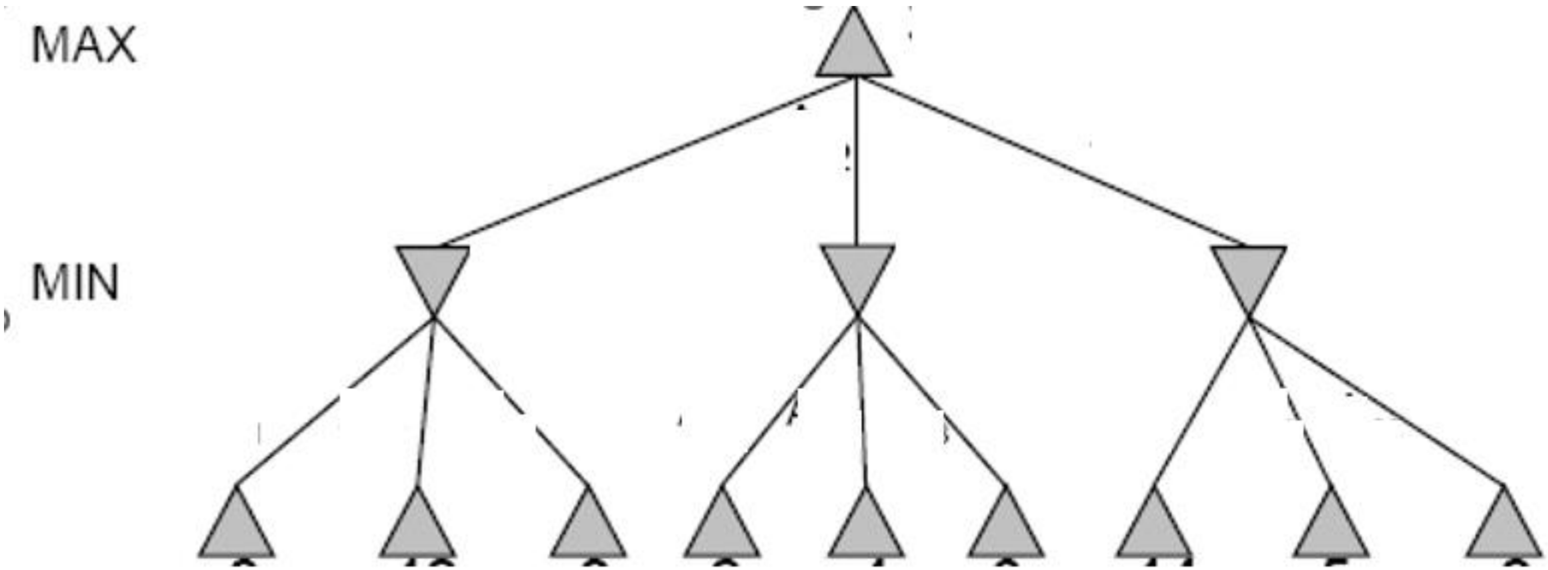
- For tic-tac-toe the game tree is relatively small—fewer than 9! = 362, 880 terminal nodes.

Minimax Algorithm:

- Find the optimal strategy for MAX assuming an infallible MIN opponent
  - Need to compute this all the down the tree
- Assumption: Both players play optimally!
- Given a game tree, the optimal strategy can be determined by using the minimax value of each node.

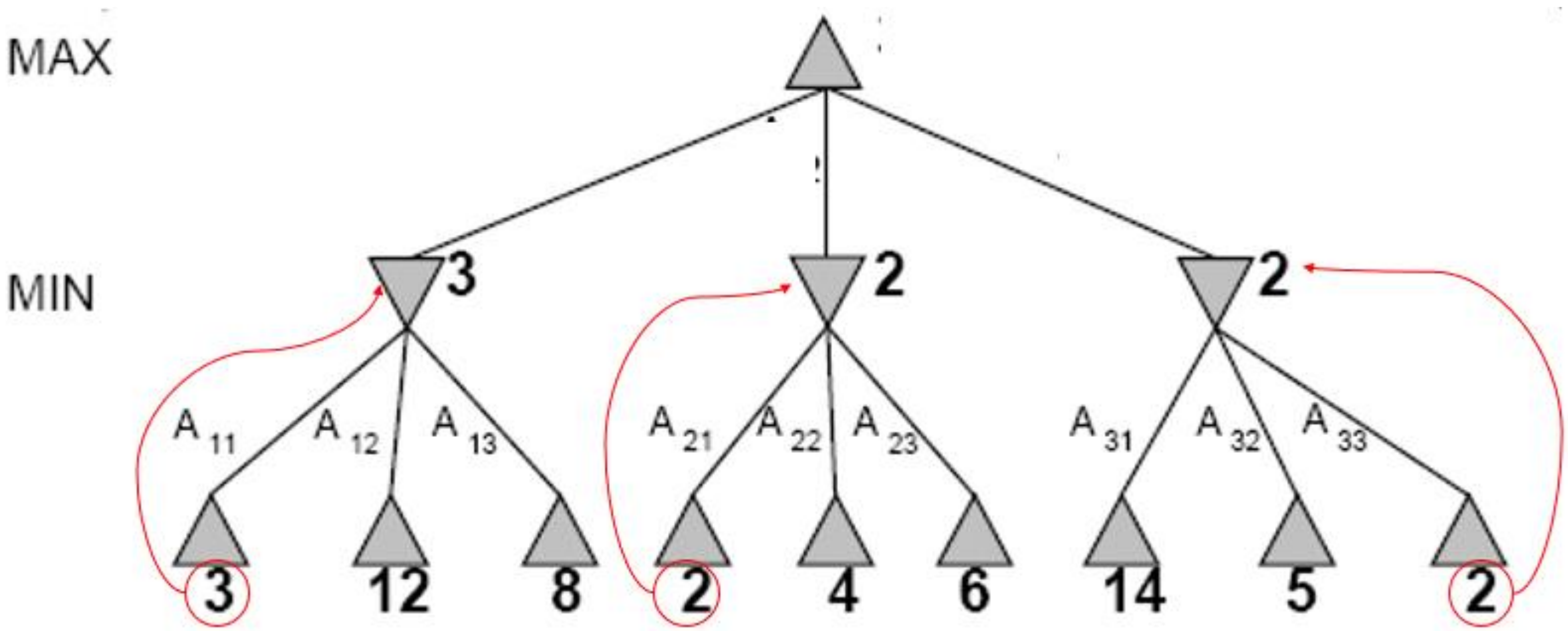
Minimax Algorithm:

- Two player game tree:



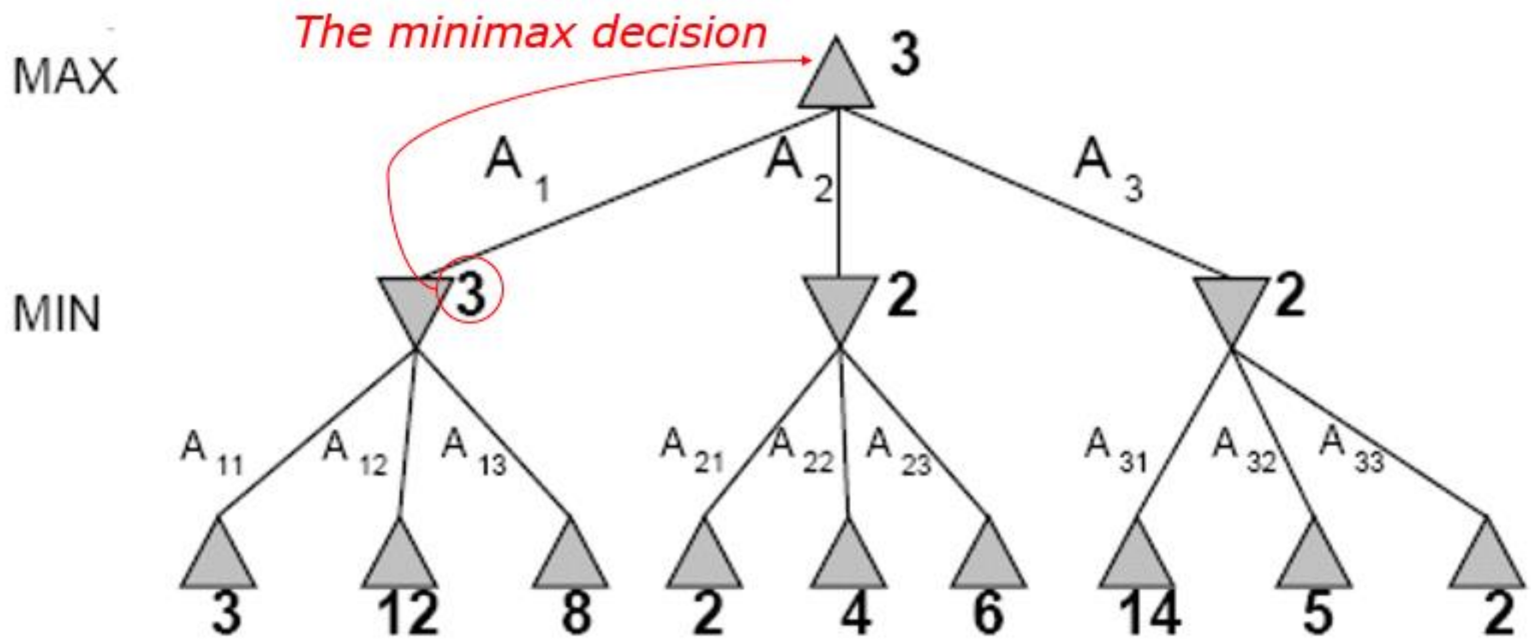
Minimax Algorithm:

- Two player game tree:



Minimax Algorithm:

- Two player game tree:



Minimax Algorithm:

- **What if MIN does not play optimally?**
- Definition of optimal play for MAX assumes MIN plays optimally:
  - maximizes worst-case outcome for MAX
- But if MIN does not play optimally, MAX will do even better
  - Can prove this

Minimax Algorithm:

- **Goal**

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

Minimax Algorithm:

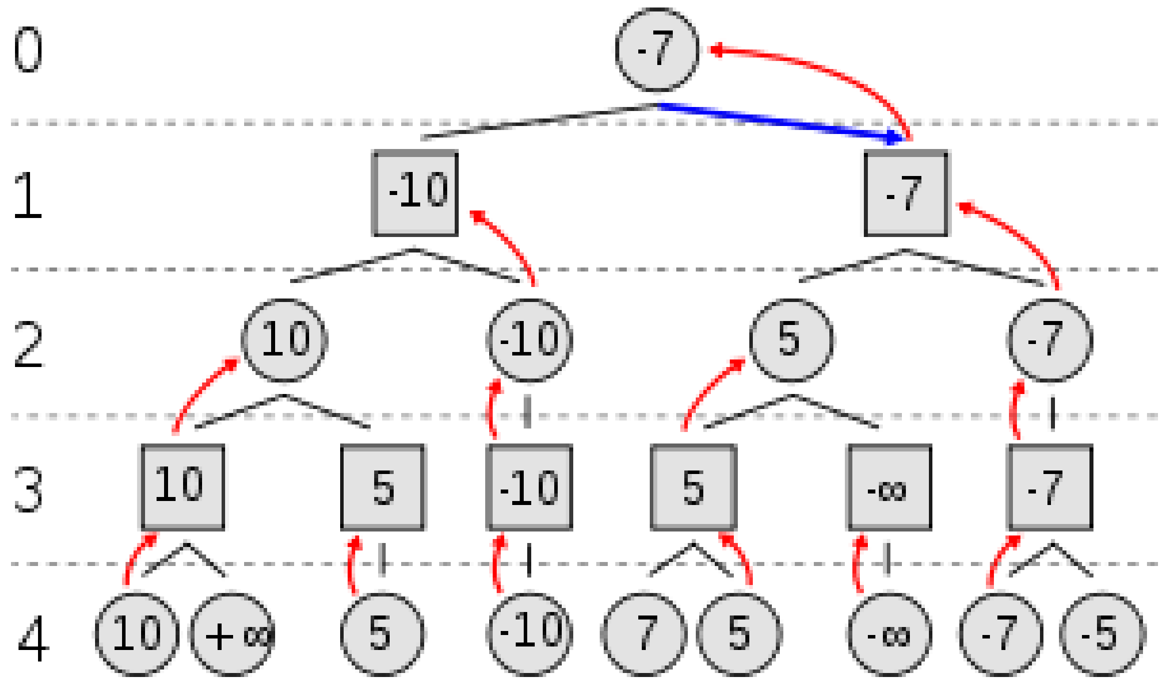
- **Pseudocode for Minimax Algorithm**


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

        if maximizingPlayer
                maxEval = -infinity
                for each child of position
                        eval = minimax(child, depth - 1, false)
                        maxEval = max(maxEval, eval)
                return maxEval

        else
                minEval = +infinity
                for each child of position
                        eval = minimax(child, depth - 1, true)
                        minEval = min(minEval, eval)
                return minEval


// initial call
minimax(currentPosition, 3, true)
```

Credit: Khola Naseem

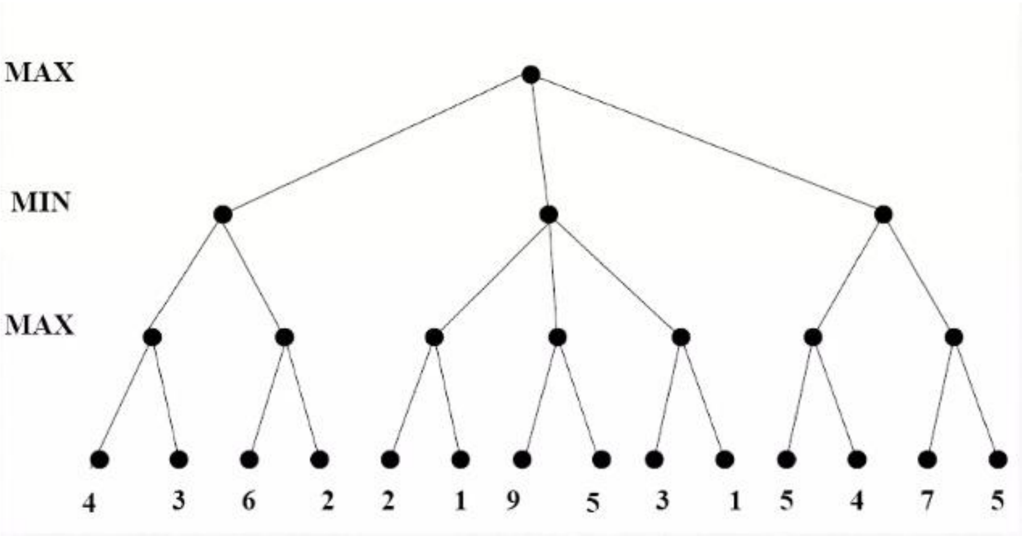
Minimax Algorithm:

- Two player game tree:



Minimax Algorithm:

- Two player game tree:




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

        if maximizingPlayer
                maxEval = -infinity
                for each child of position
                        eval = minimax(child, depth - 1, false)
                        maxEval = max(maxEval, eval)
                return maxEval

        else
                minEval = +infinity
                for each child of position
                        eval = minimax(child, depth - 1, true)
                        minEval = min(minEval, eval)
                return minEval


// initial call
minimax(currentPosition, 3, true)
```

Credit: Khola Naseem

Minimax Algorithm:

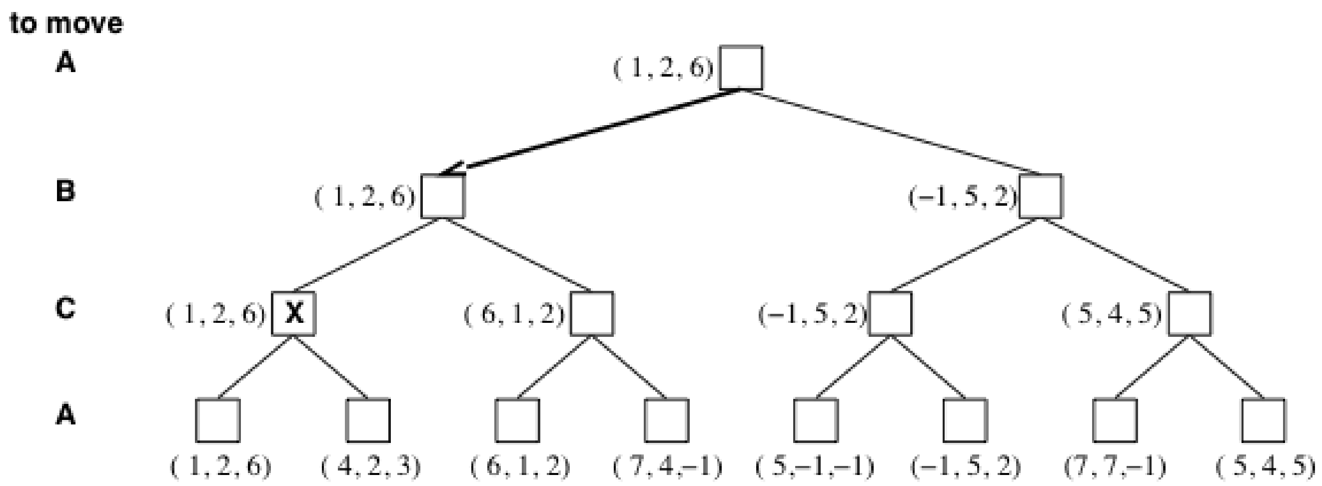
- Complete?
  - Yes (if tree is finite).
- Optimal?
  - Yes (against an optimal opponent).
  - Can it be beaten by an opponent playing sub-optimally?
- Time complexity?
  - O(b^m)
- Space complexity?
  - O(bm)   (depth-first search, generate all actions at once)

Minimax Algorithm:

- Complete depth-first exploration of the game tree
- Assumptions:
  - Max depth = m, b legal moves at each point
- then the time complexity of the minimax algorithm is O(b^m). The space complexity is O(bm) for an algorithm that generates all actions at once
- For real games, of course, the time cost is totally impractical, but this algorithm serves as the basis for the mathematical analysis of games and for more practical algorithms.

Multiplayer games:

- First, we need to replace the single value for each node with a vector of values. For example, in a three-player game with players A, B, and C, a vector (vA, vB, vC) is associated with each node.
- For terminal states, this vector gives the utility of the state from each player's viewpoint.



Credit: Khola Naseem

Multiplayer games:

- **Zero sum games:** zero-sum describes a situation in which a participant's gain or loss is exactly balanced by the losses or gains of the other participant(s).
- If the total gains of the participants are added up, and the total losses are subtracted, they will sum to zero

	B chooses B1	B chooses B2	B chooses B3
A chooses A1	+3	−2	+2
A chooses A2	−1	0	+4
A chooses A3	−4	−3	+1

Multiplayer games:

- **Alliance:**
- suppose A and B are in weak positions and C is in a stronger position.
- In this way, collaboration emerges from purely selfish behavior.
- as soon as C weakens the alliance loses its value, and either A or B could violate the agreement. In some cases, explicit alliances merely make concrete

Minimax Issue:

- Number of game states is exponential in the number of moves.
- Solution: Do not examine every node
  - => pruning
  - Remove branches that do not influence final decision

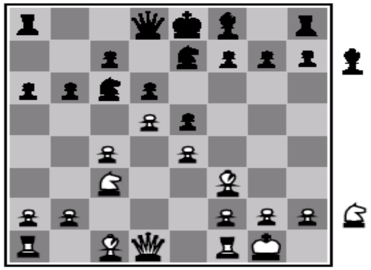
(Static) Heuristic Evaluation Functions:

- An Evaluation Function:
  - Estimates how good the current board configuration is for a player.
  - Typically, evaluate how good it is for the player, how good it is for the opponent, then subtract the opponent's score from the player's.
  - Often called "static" because it is called on a static board position.
  - Chess: Value of all white pieces - Value of all black pieces
- Typical values from -infinity (loss) to +infinity (win) or [-1, +1].
- If the board evaluation is X for a player, it's -X for the opponent
  - "Zero-sum game"

(Static) Heuristic Evaluation Functions:

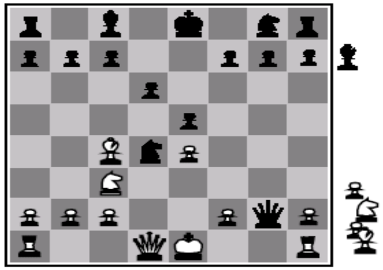
- material value for each piece: each pawn is worth 1, a knight or bishop is worth 3, a rook 5, and the queen 9 and so on
- Chess: Value of all white pieces - Value of all black pieces

Evaluation functions



Black to move

White slightly better



White to move

Black winning

For chess, typically *linear* weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

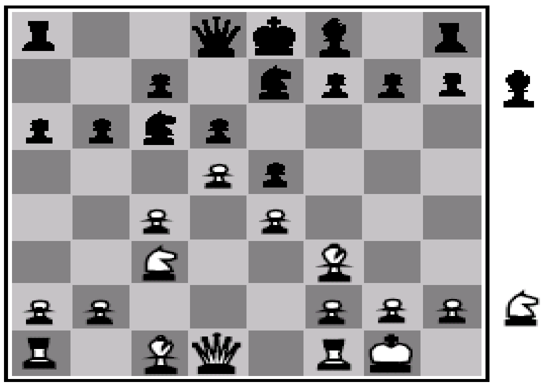
e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens), etc.

Chapter 5, Sections 1–5 14

Credit: Khola Naseem

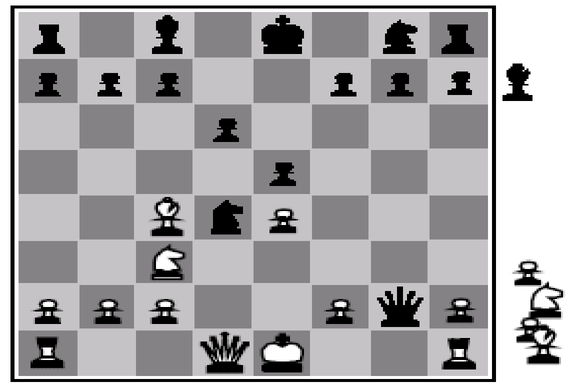
(Static) Heuristic Evaluation Functions:

Evaluation functions



Black to move

White slightly better



White to move

Black winning

For chess, typically *linear* weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

e.g., $w_1 = 9$ with
$f_1(s) =$ (number of white queens) – (number of black queens), etc.

Alpha-beta pruning:

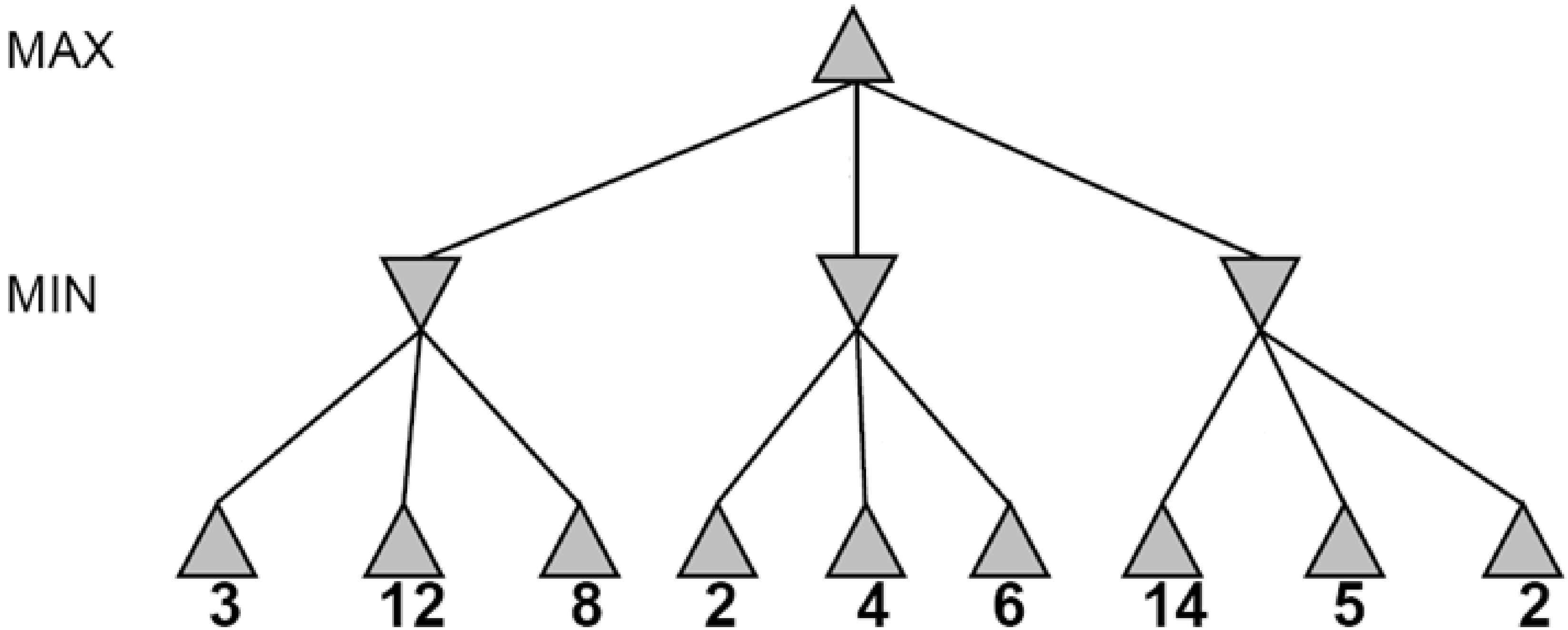
- The problem with minimax search is that the number of game states it has to examine is exponential in the depth of the tree.
- we can’t eliminate the exponent, but it turns out we can effectively cut it in half.
- It is possible to compute the exact minimax decision without expanding every node in the game tree
- That is, we can borrow the idea of pruning the particular technique we examine is called alpha–beta pruning
- Prune away branches that cannot possibly influence the final decision.

Alpha-beta pruning:

- Depth first search – only considers nodes along a single path at any time
- α = highest-value choice that we can guarantee for MAX so far in the current subtree.
- β = lowest-value choice that we can guarantee for MIN so far in the current subtree.
- update values of a and b during search and prunes remaining branches as soon as the value is known to be worse than the current a or b value for MAX or MIN.

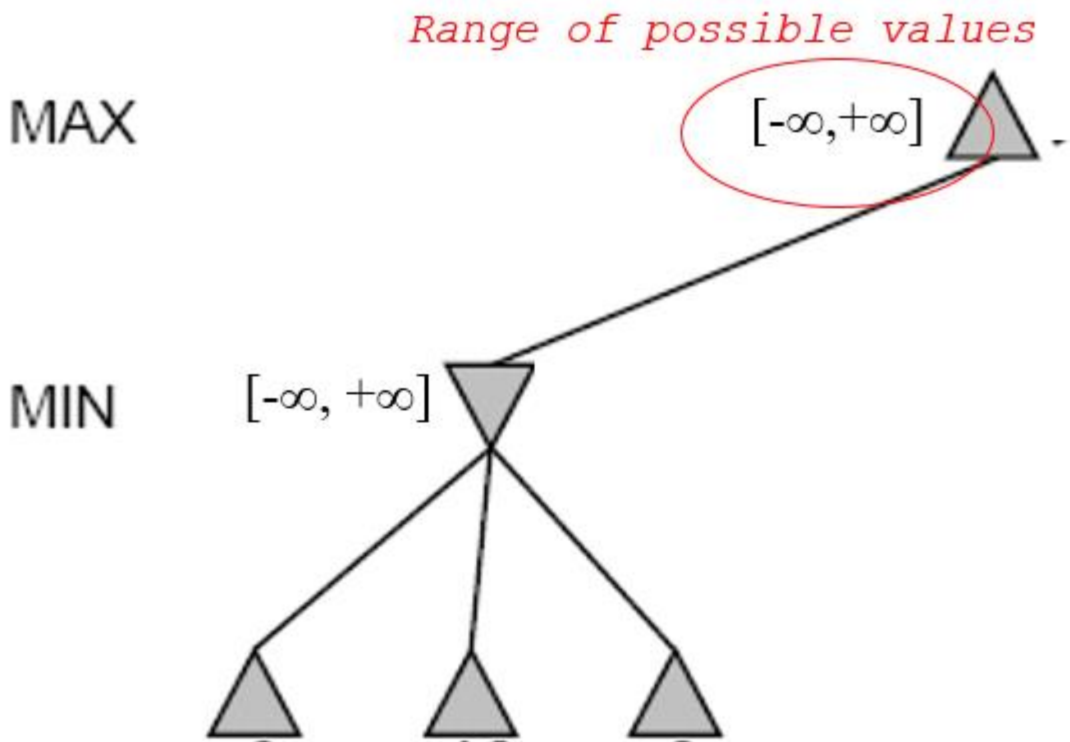
Alpha-beta pruning:

- It is possible to compute the exact minimax decision without expanding every node in the game tree



Alpha-beta pruning:

- It is possible to compute the exact minimax decision without expanding every node in the game tree

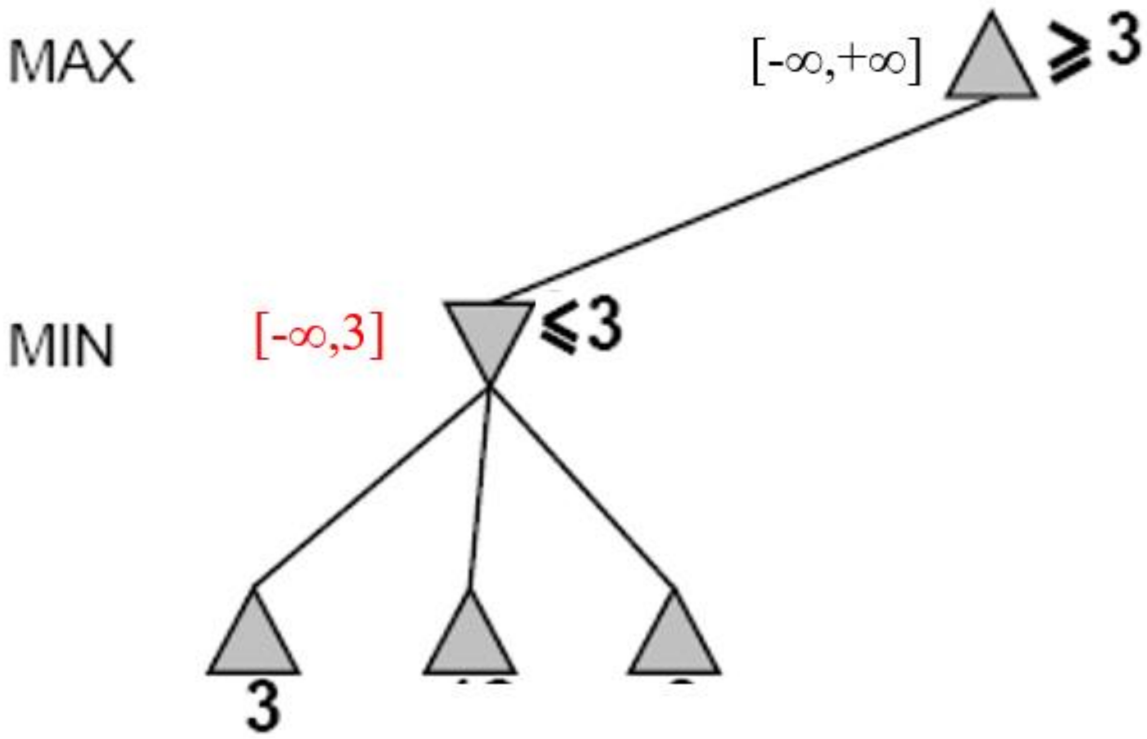


Alpha-beta pruning:

- When to Prune
- Prune whenever alpha ≥ beta
- Prune below a Max node whose alpha value becomes greater than or equal to the beta value of its ancestors. alpha ≥ beta
  - Max nodes update alpha based on children's returned values.
- Prune below a Min node whose beta value becomes less than or equal to the alpha value of its ancestors. beta <=alpha
  - Min nodes update beta based on children's returned values.

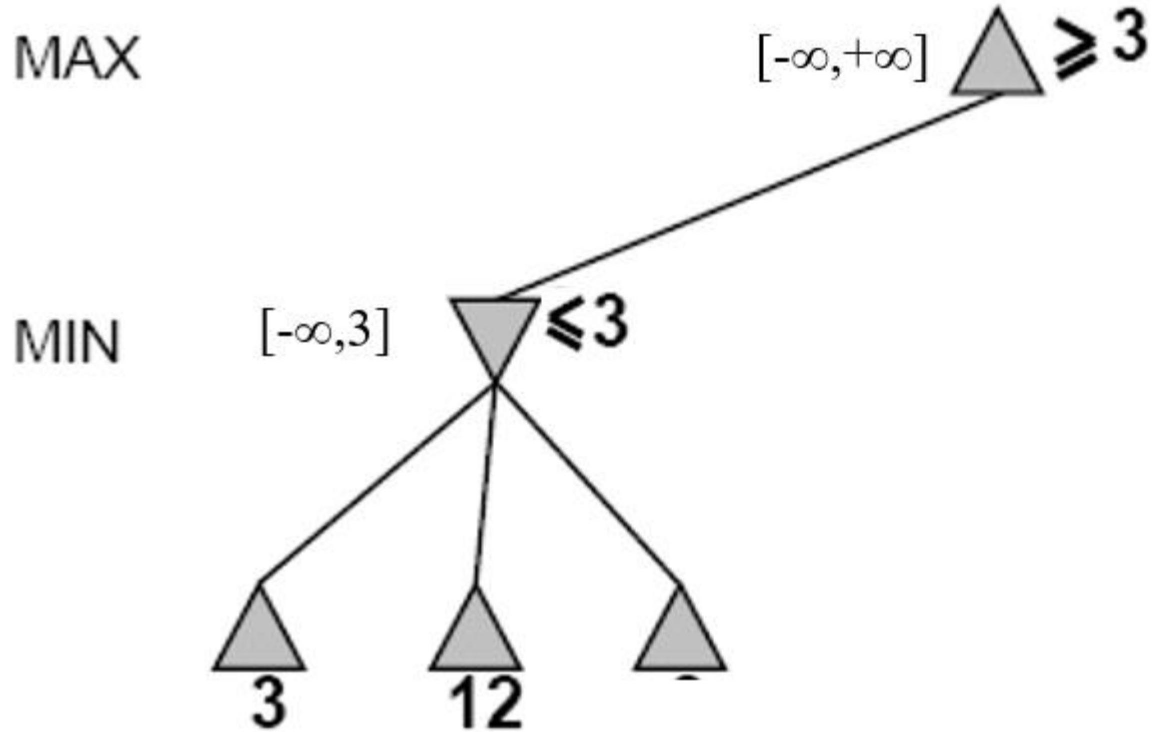
Alpha-beta pruning:

- It is possible to compute the exact minimax decision without expanding every node in the the game tree



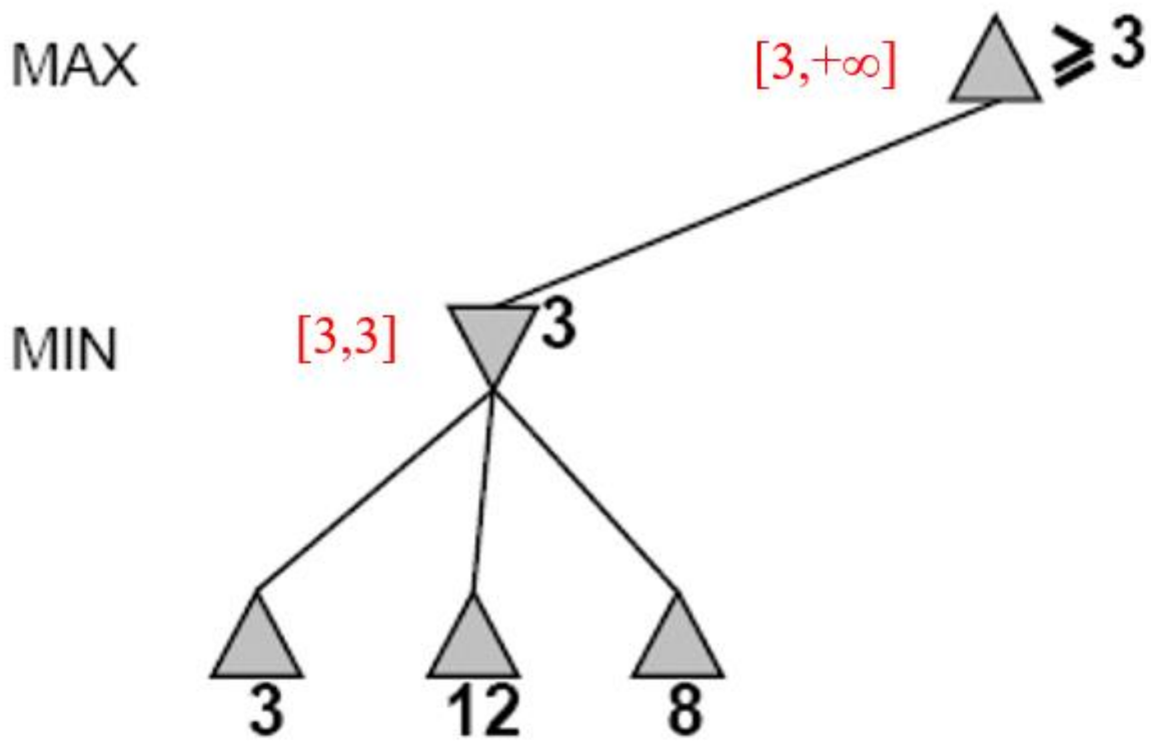
Alpha-beta pruning:

- It is possible to compute the exact minimax decision without expanding every node in the game tree



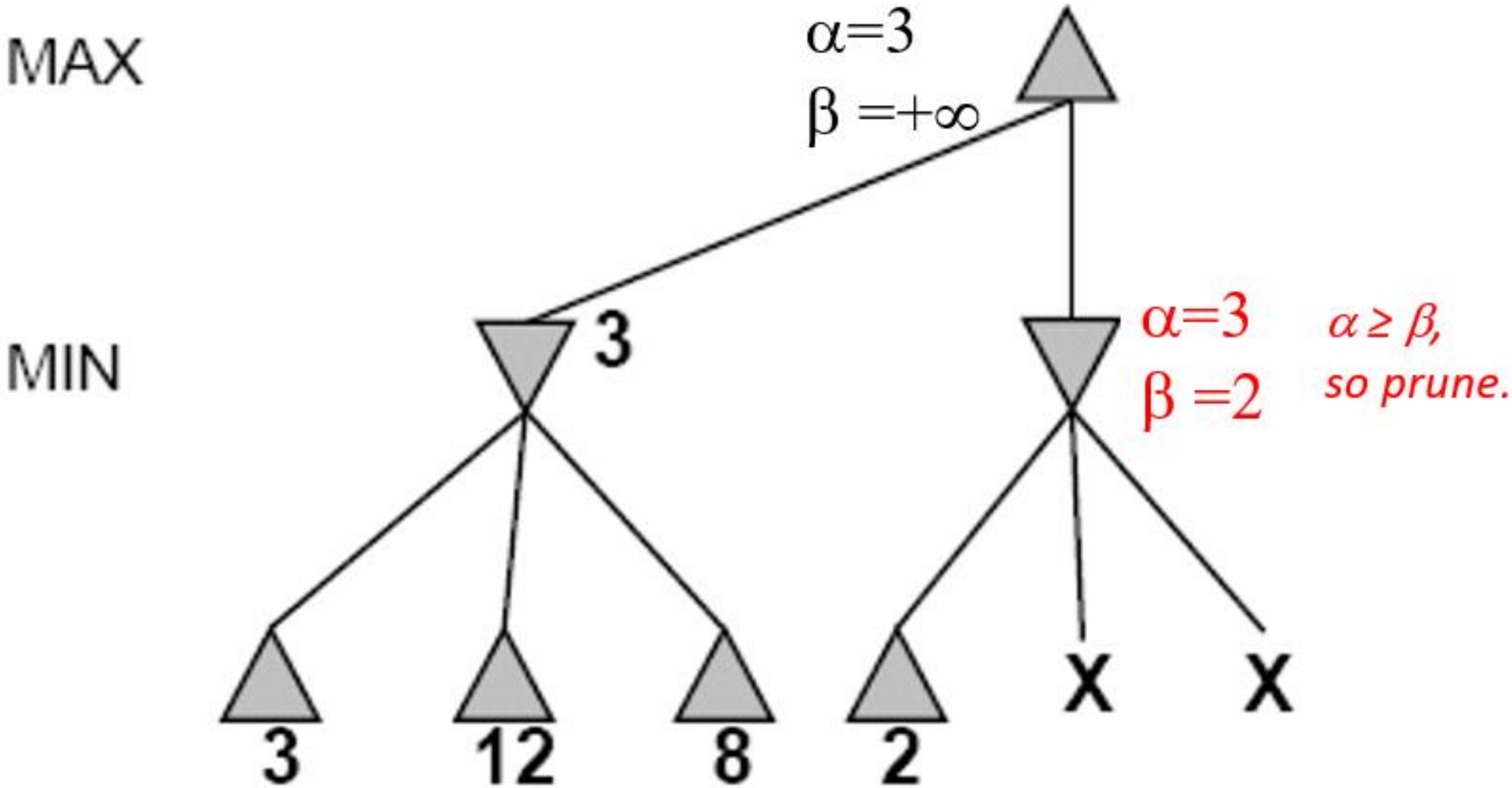
Alpha-beta pruning:

- It is possible to compute the exact minimax decision without expanding every node in the game tree



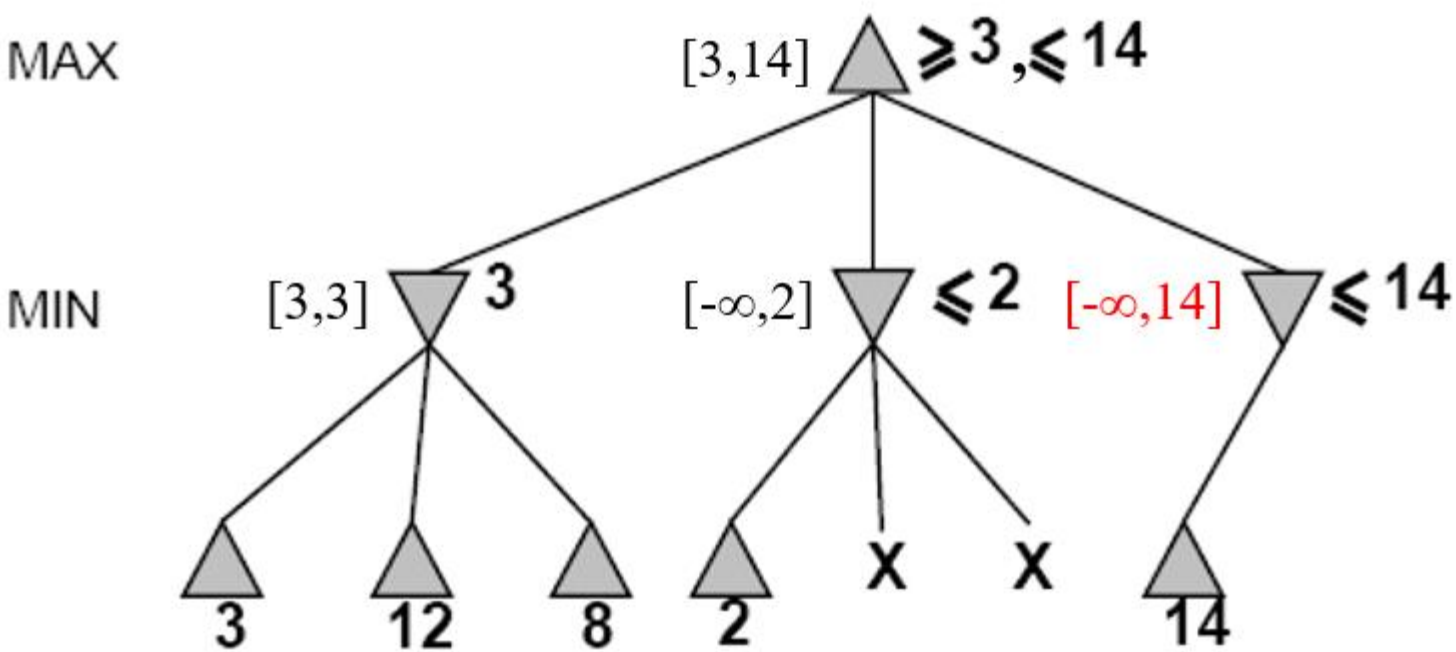
Alpha-beta pruning:

- It is possible to compute the exact minimax decision without expanding every node in the game tree



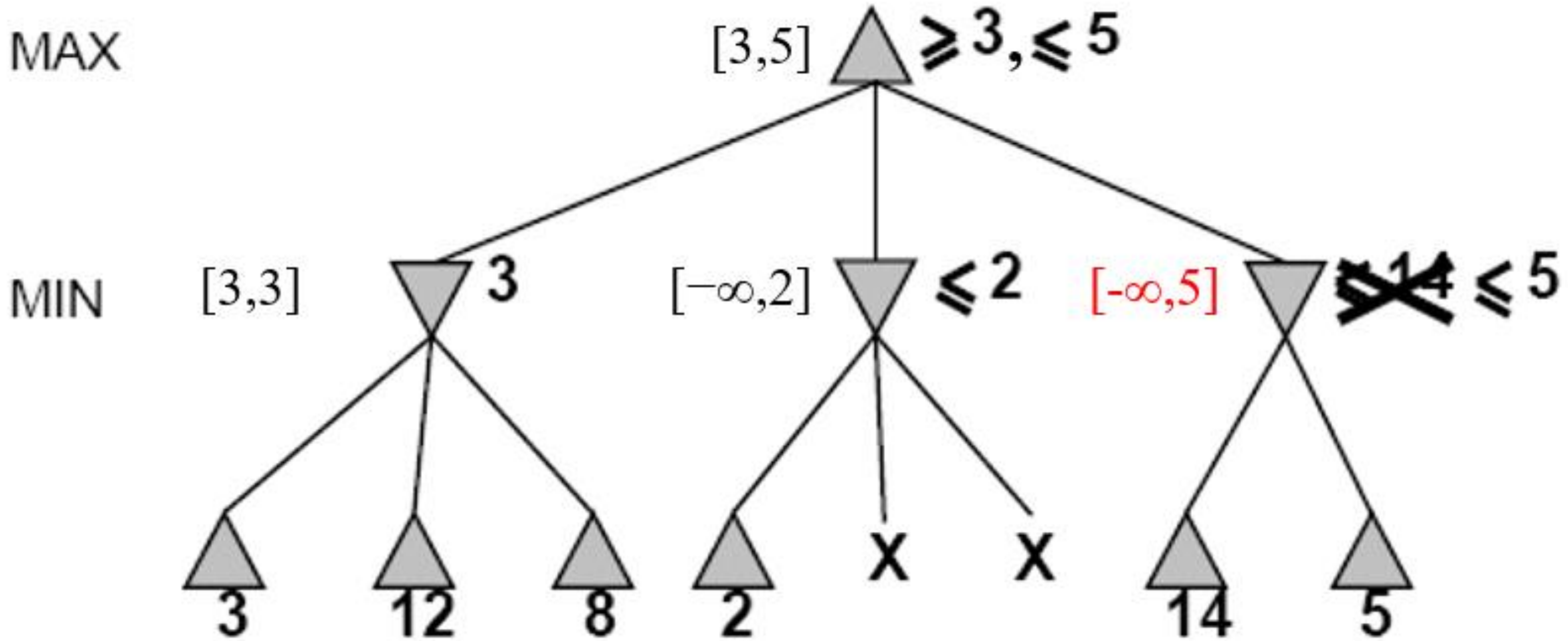
Alpha-beta pruning:

- It is possible to compute the exact minimax decision without expanding every node in the game tree



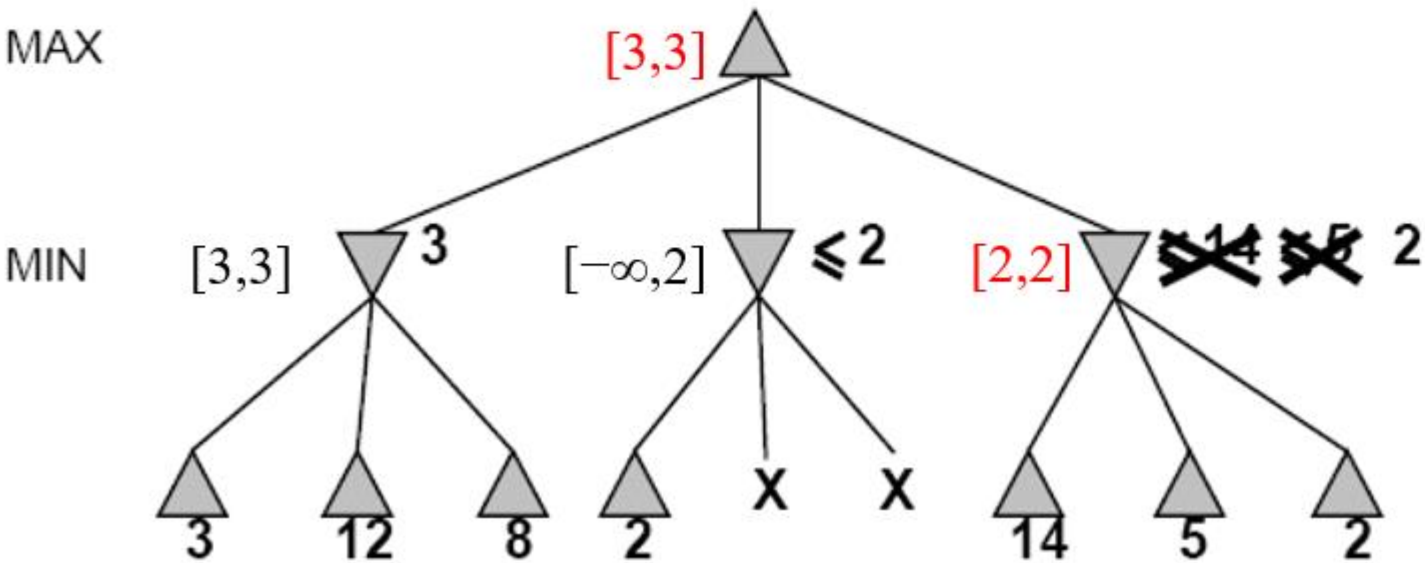
Alpha-beta pruning:

- It is possible to compute the exact minimax decision without expanding every node in the game tree



Alpha-beta pruning:

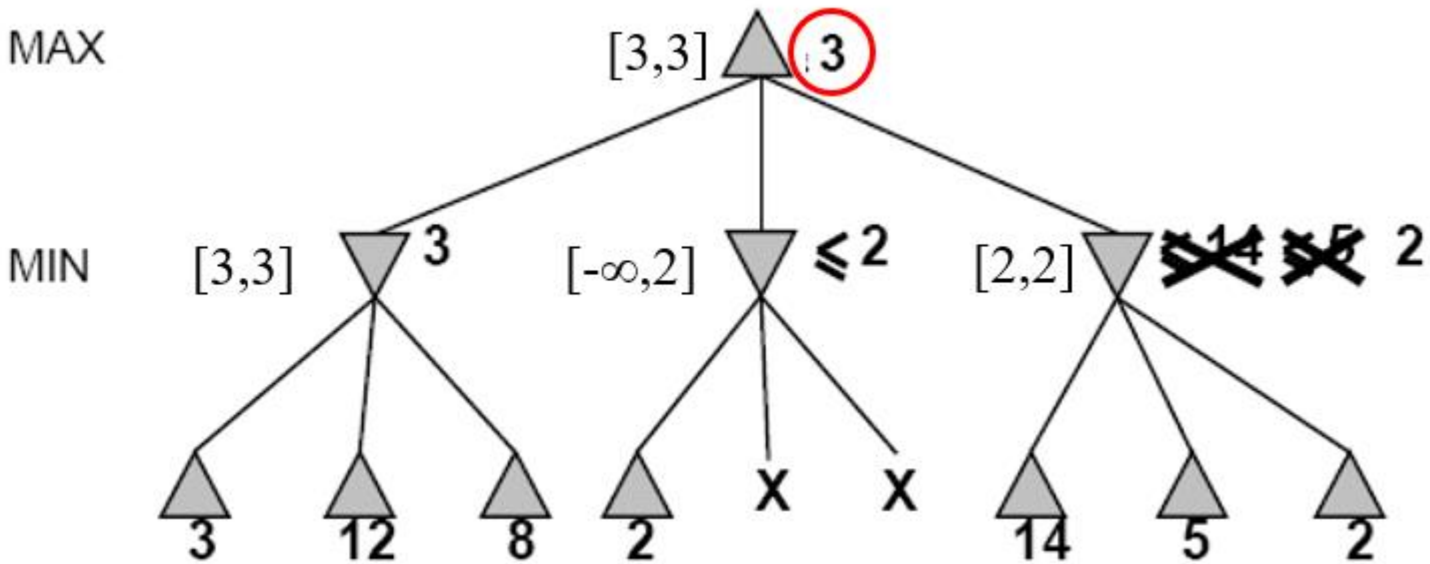
- It is possible to compute the exact minimax decision without expanding every node in the game tree



Credit: Khola Naseem

Alpha-beta pruning:

- It is possible to compute the exact minimax decision without expanding every node in the game tree

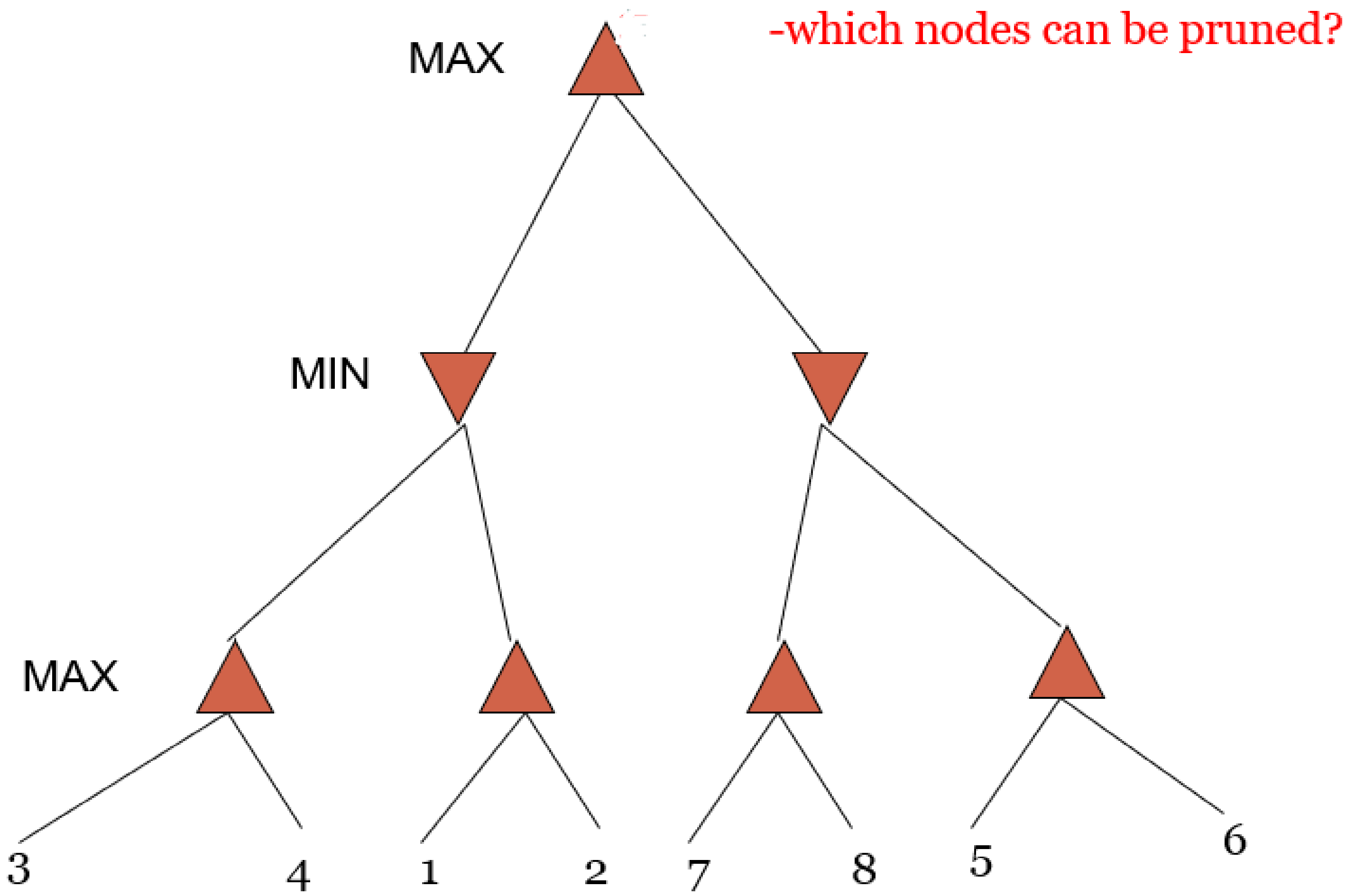


Alpha-beta pruning:

- In practice often get O(b^(m/2)) rather than O(b^m)
- Worst-Case
  - branches are ordered so that no pruning takes place. In this case alpha-beta gives no improvement over exhaustive search

Alpha-beta pruning:

- Example:



Alpha-beta pruning:

- Example:

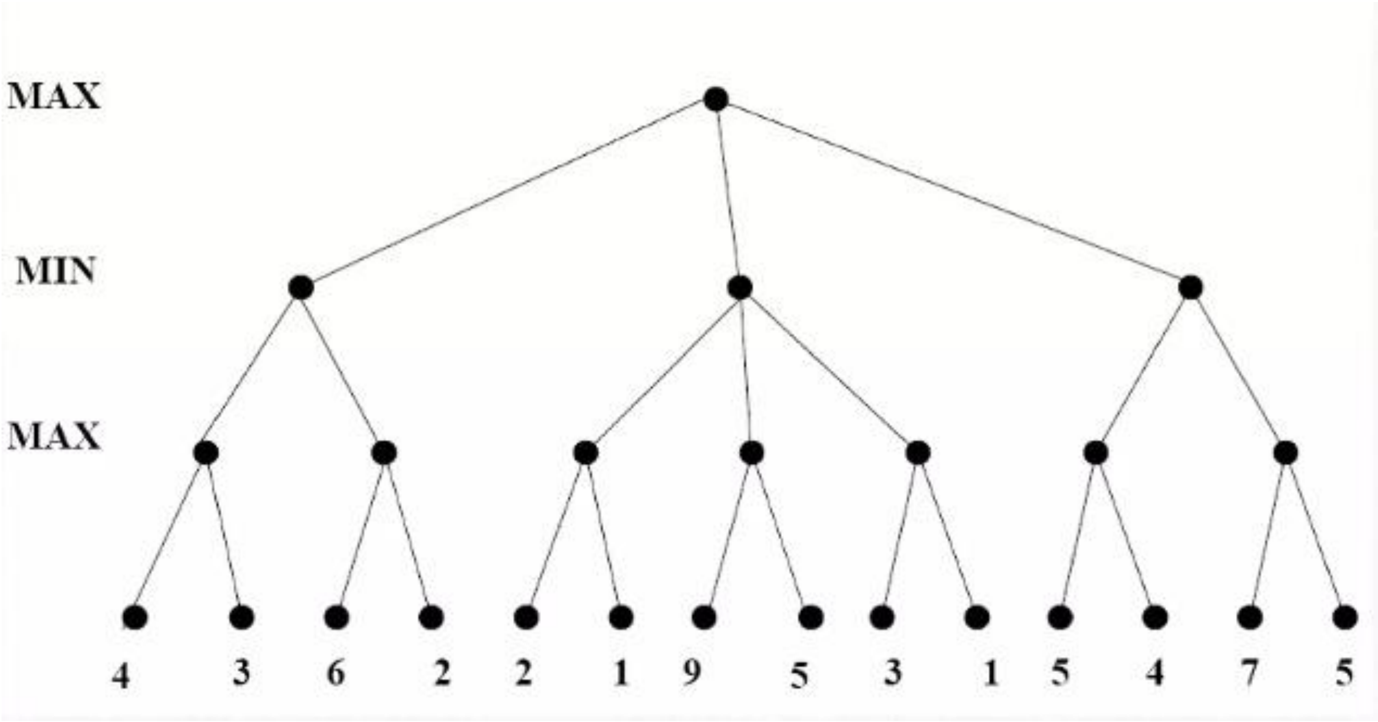
```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← −∞
  for each a in ACTIONS(state) do
    v ← MAX(v, MIN-VALUE(RESULT(s,a), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```

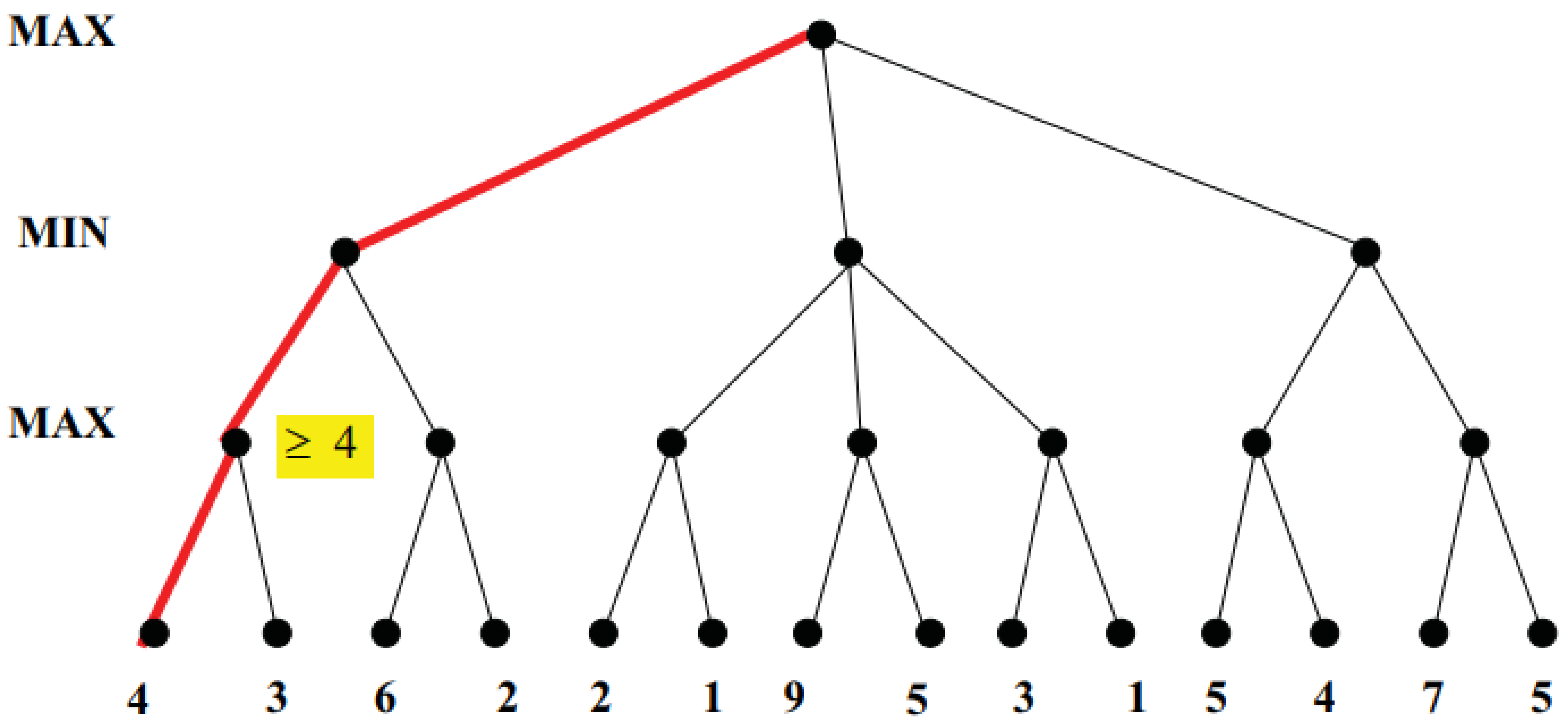
```
function MIN-VALUE(state, α, β) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
  v ← +∞
  for each a in ACTIONS(state) do
    v ← MIN(v, MAX-VALUE(RESULT(s,a) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

Alpha-beta pruning:

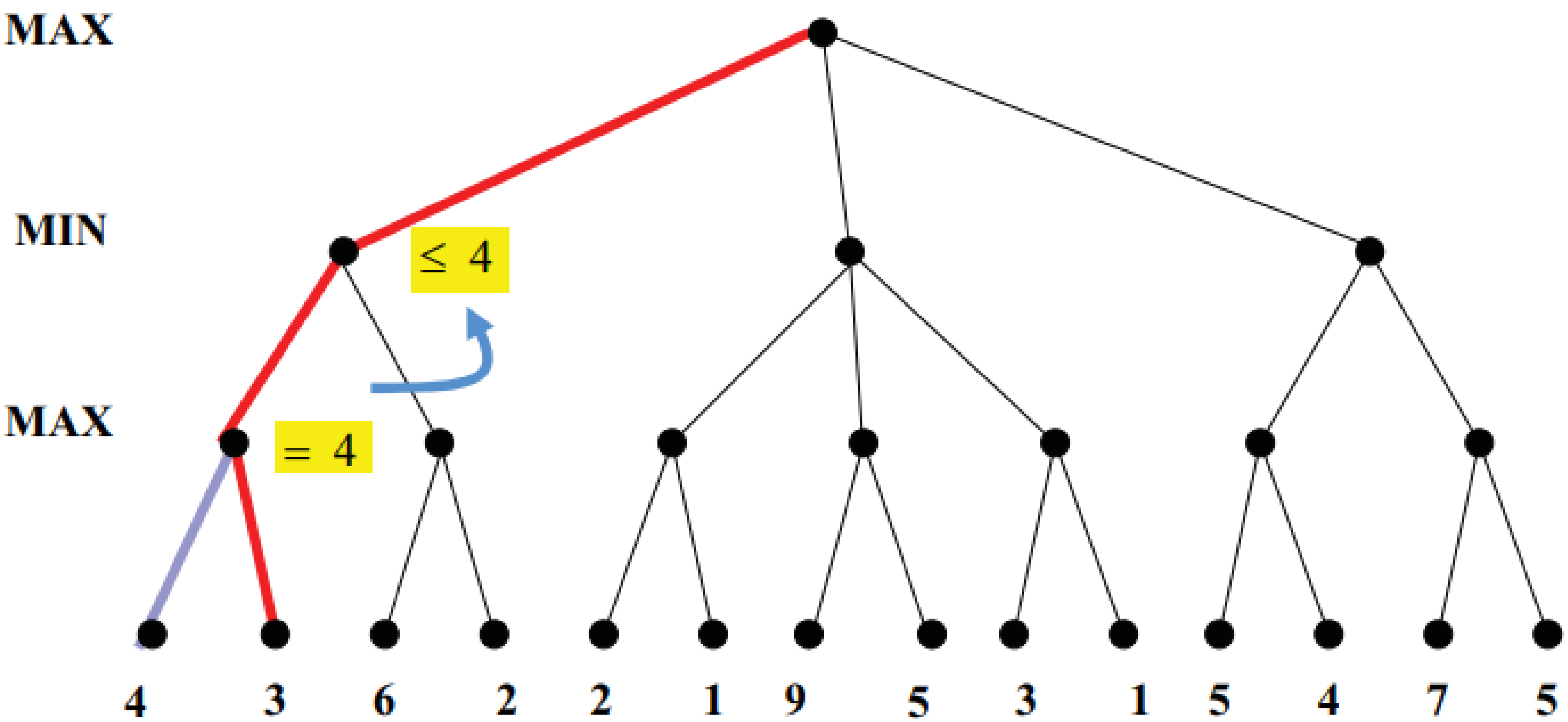
- Example:



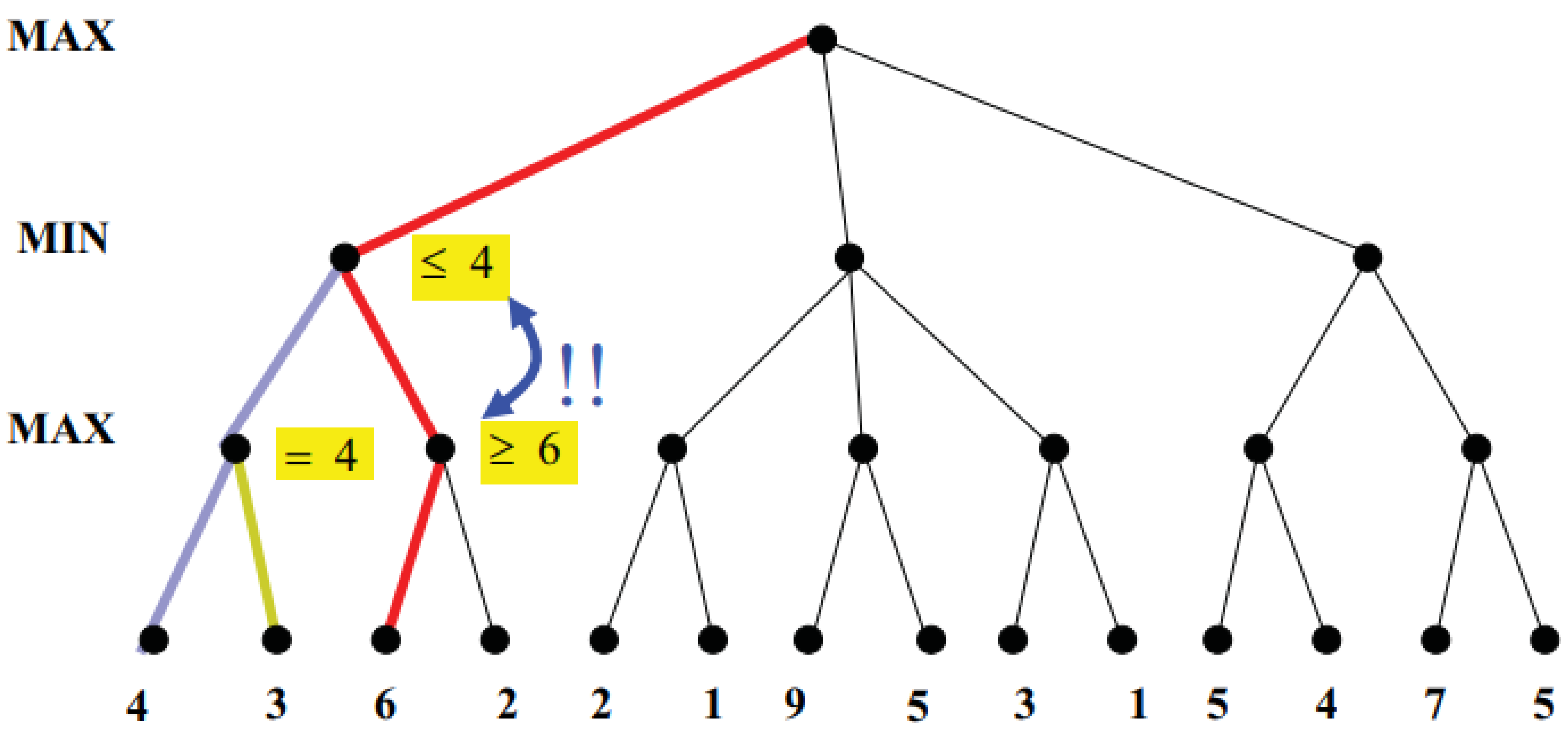
Alpha beta pruning. Example



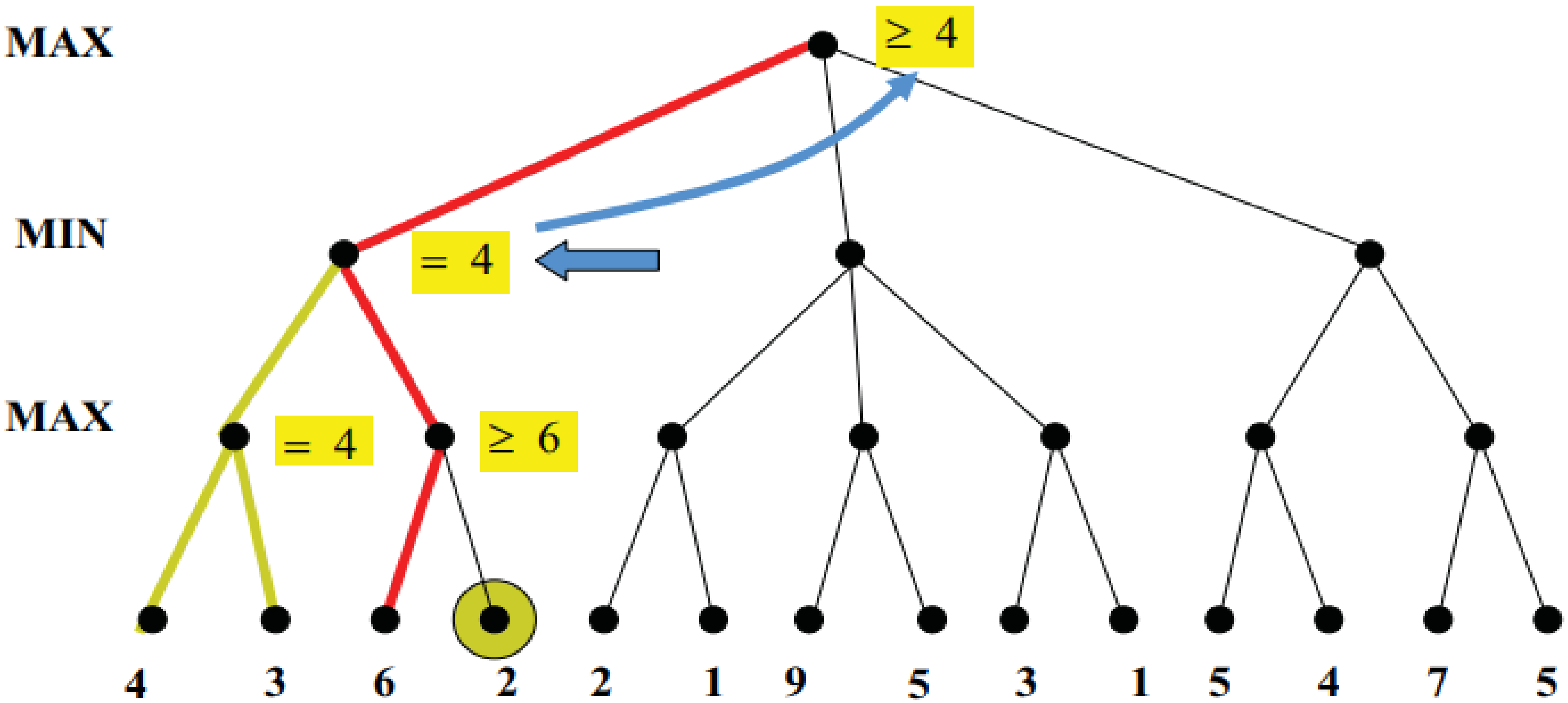
Alpha beta pruning. Example



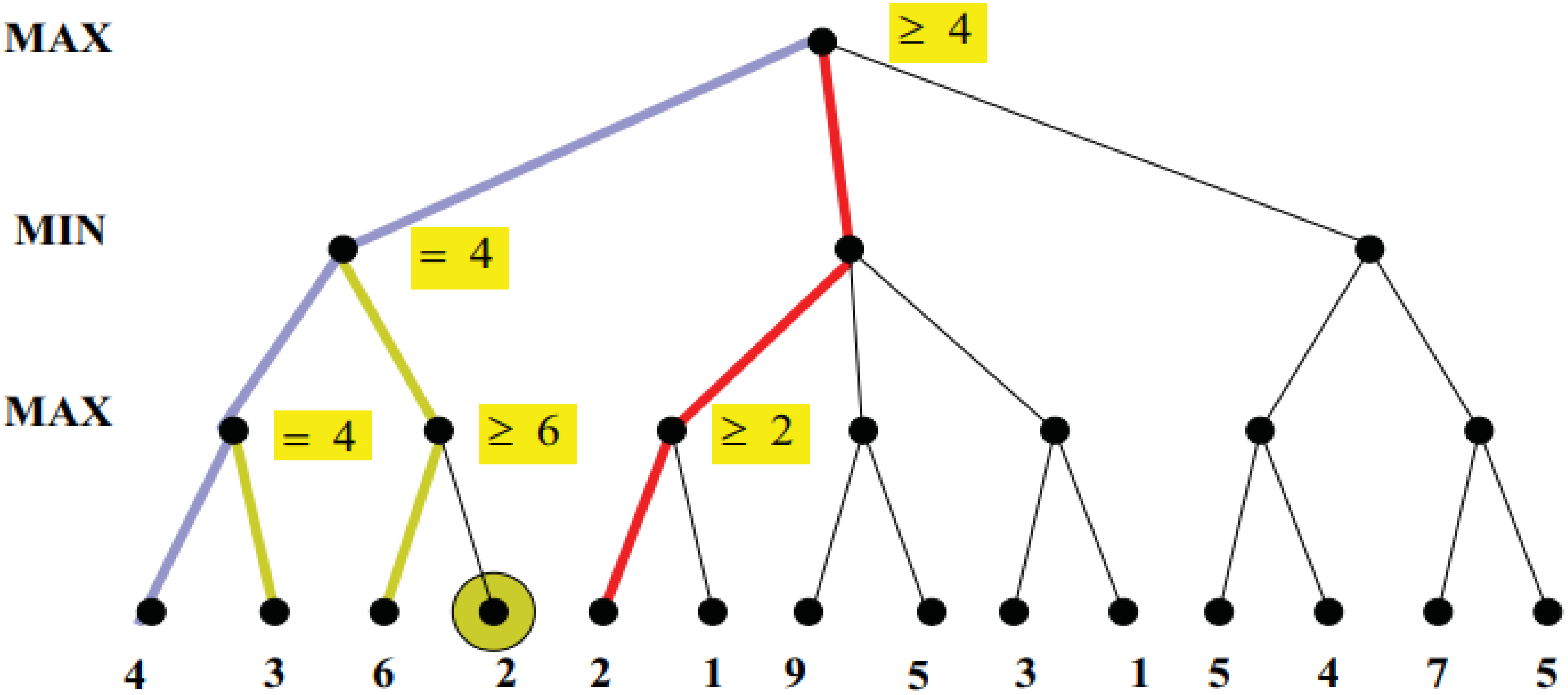
Alpha beta pruning. Example



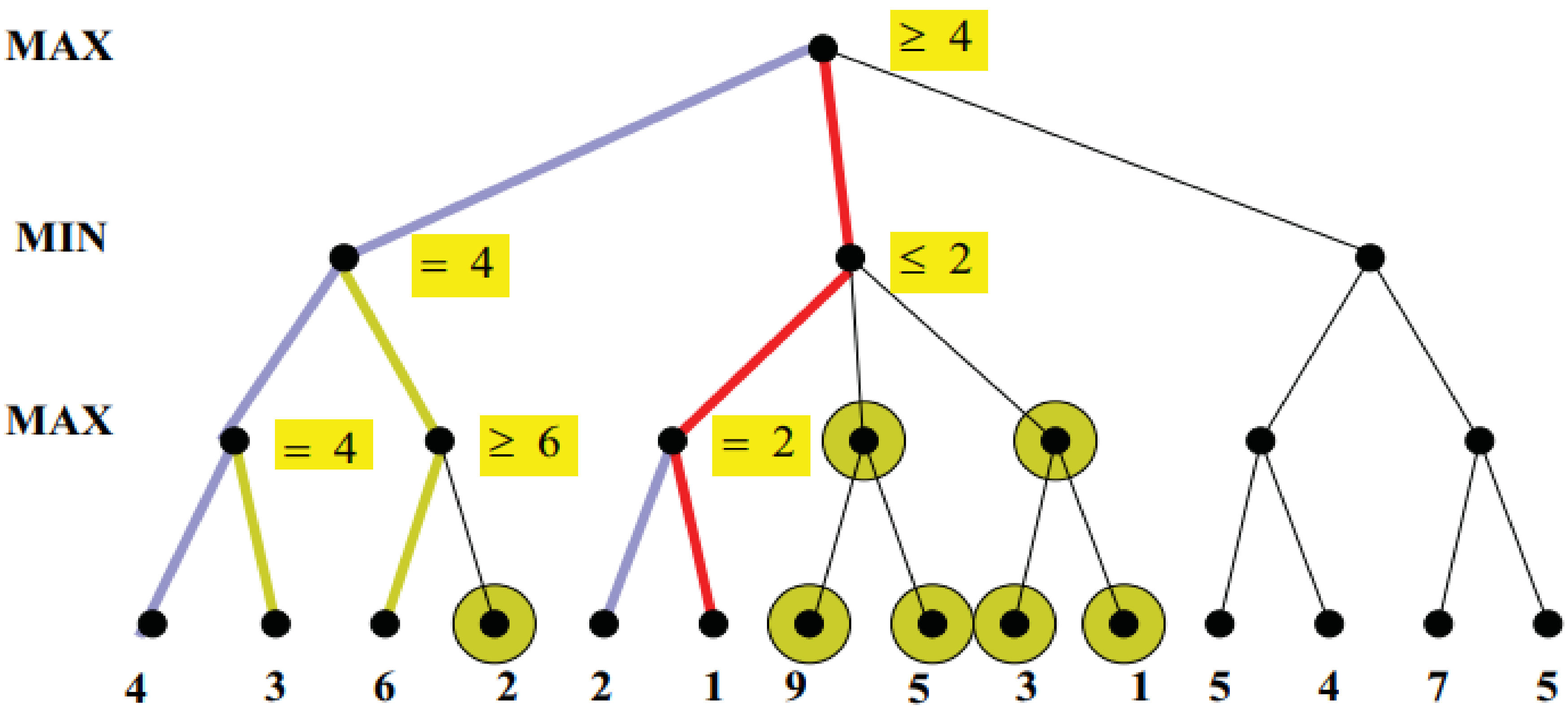
Alpha beta pruning. Example



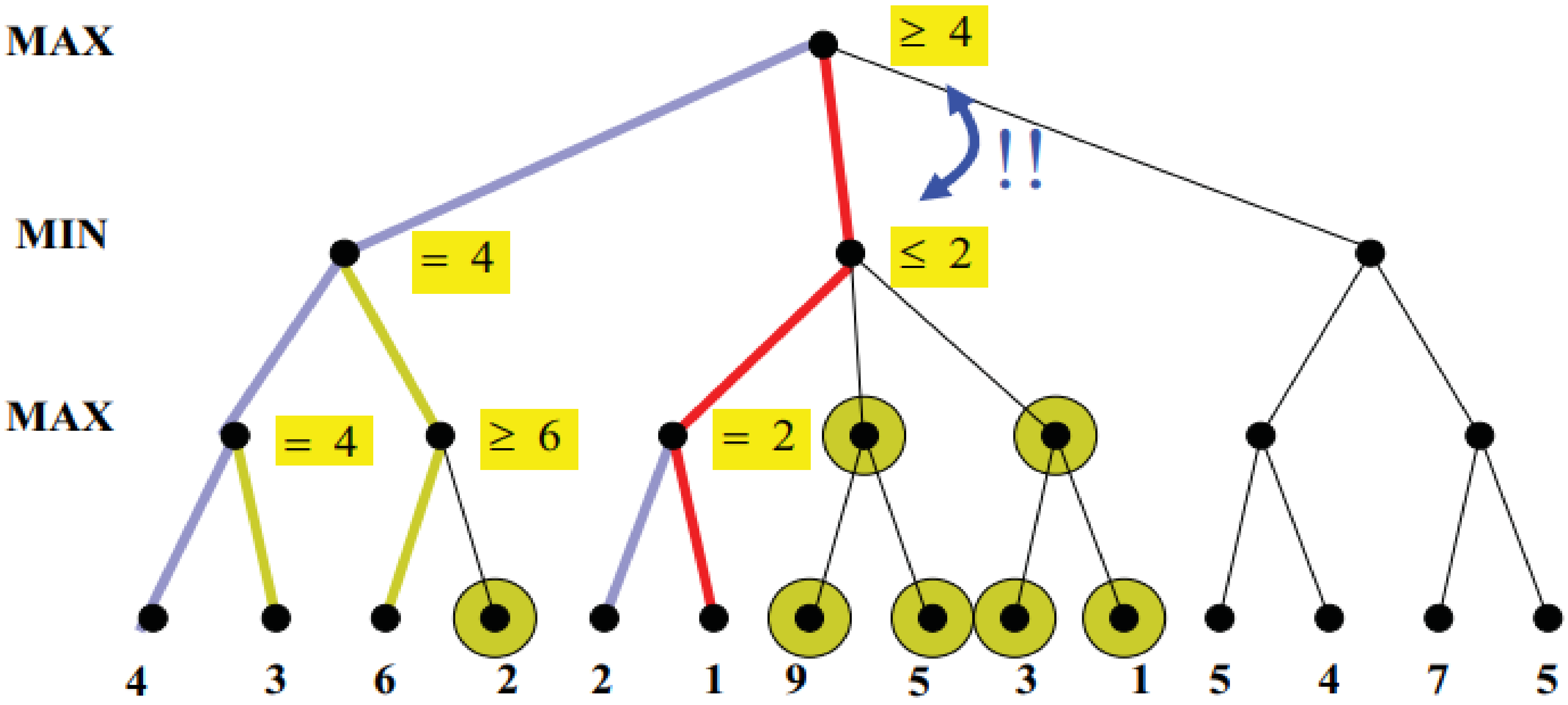
Alpha beta pruning. Example



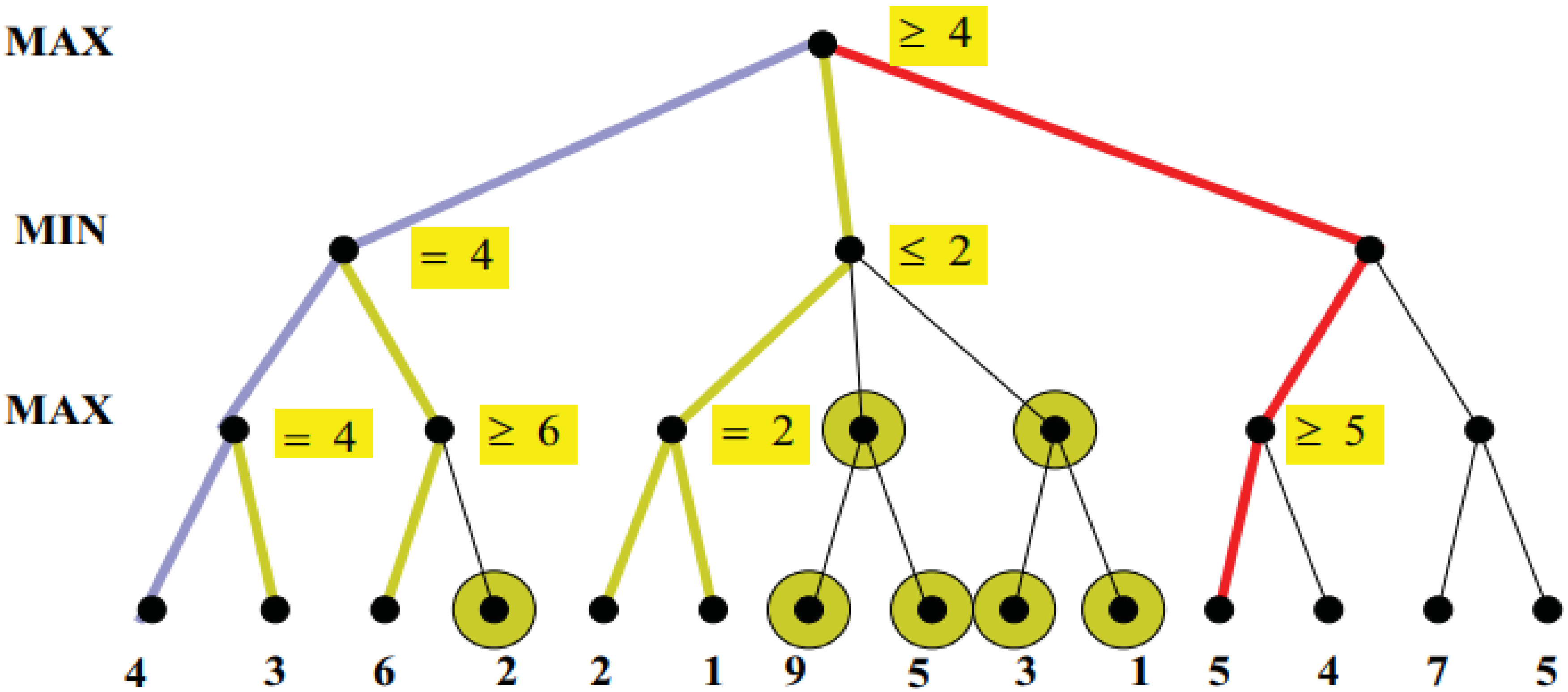
Alpha beta pruning. Example



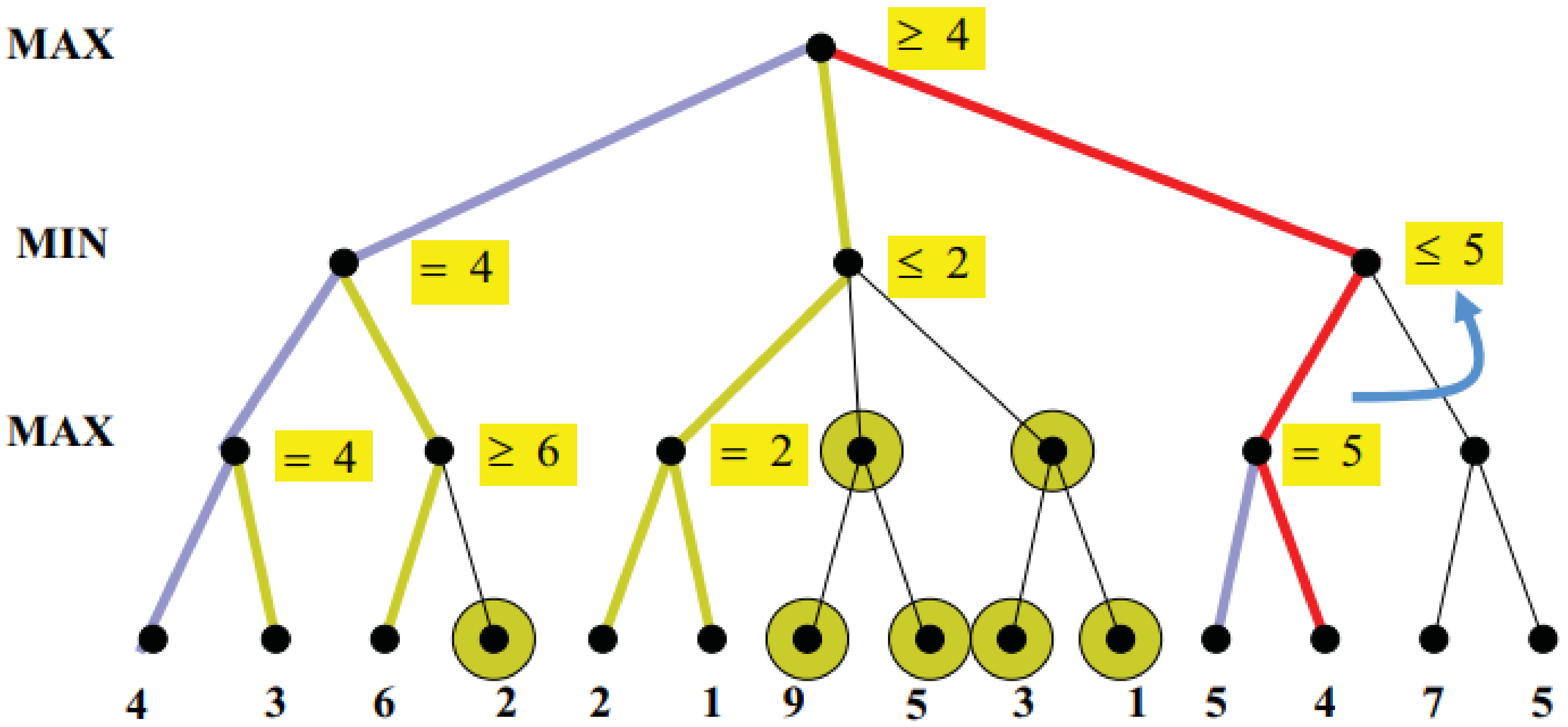
Alpha beta pruning. Example



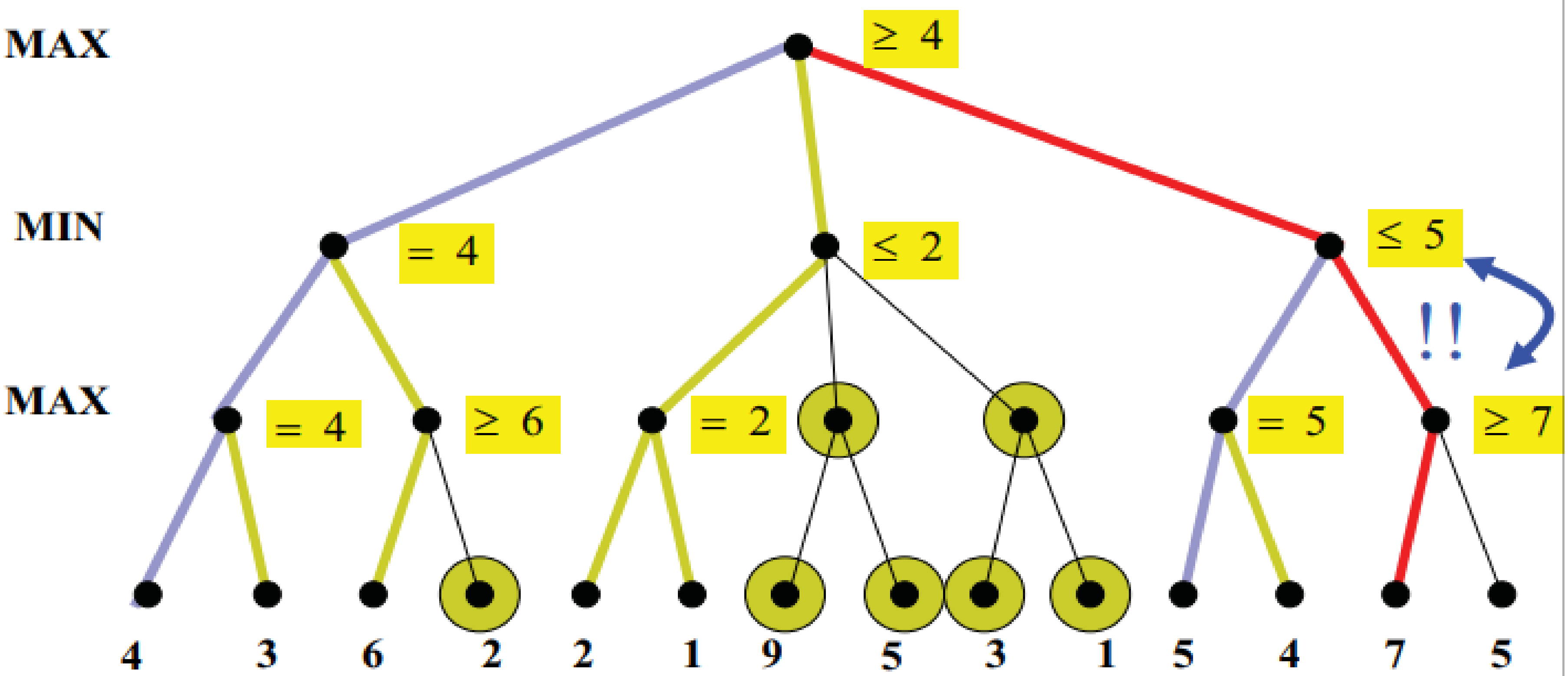
Alpha beta pruning. Example



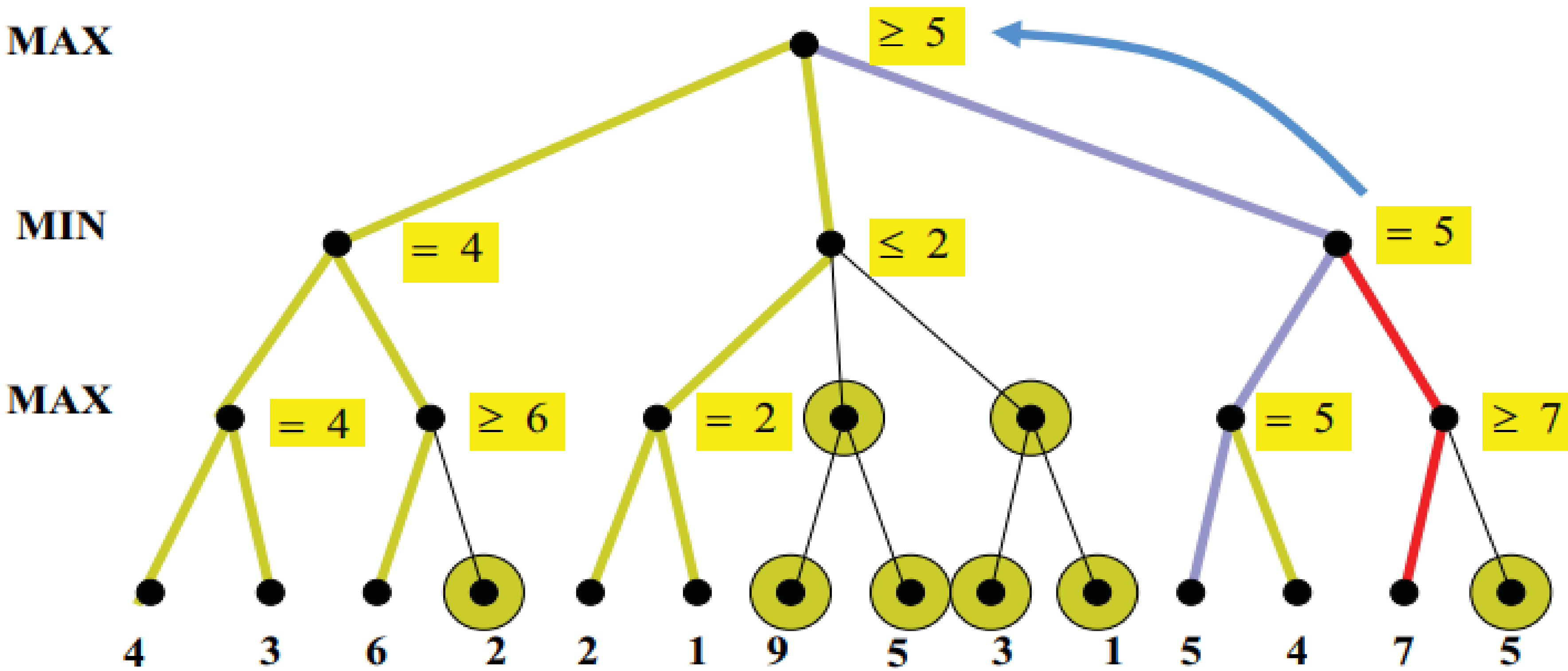
Alpha beta pruning. Example



Alpha beta pruning. Example



Alpha beta pruning. Example



Alpha beta pruning. Example