ADVERSARIAL SEARCH

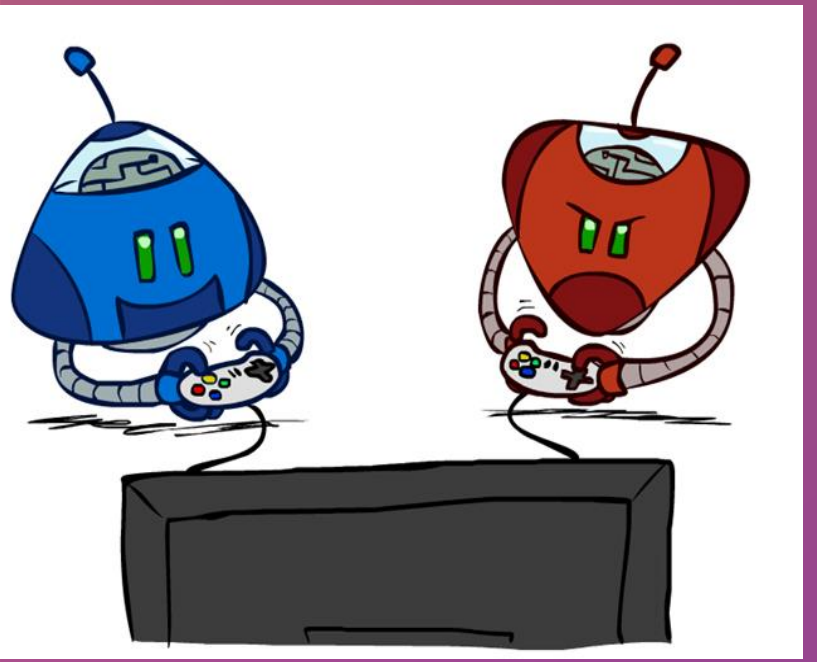
Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us

GAMES AND TYPES

- **AI researchers** study game playing as it's a **good reasoning problem** which is **formal(well defined rules), unbiased** and **nontrivial** as it require **intelligence** in direct comparison with humans. This can be utilized in development strategies of other computer programs too.

GAME TYPES

1. **Deterministic**(follow a strict pattern and set of rules for the games, and there is no randomness associated.Eg: chess, Checkers,) vs **stochastic**(have various unpredictable events and has a factor of chance or luck Eg: monopoly, bridge, poker)
2. **Single Agent or Multi Agent**
3. **Zero sum**(Agents have opposite utilities (values on outcomes), one maximizes and the other minimizes)
4. **Turn Taking(Poker, monopoly)**
5. **Perfect information** (Gives info abt states, players, actions, transition, terminal test and utilities.Eg: Chess, checkers) vs **Imperfect Information**(agents do not have all information about the game and not aware with what's going on Eg: battleships, bridge, poker)



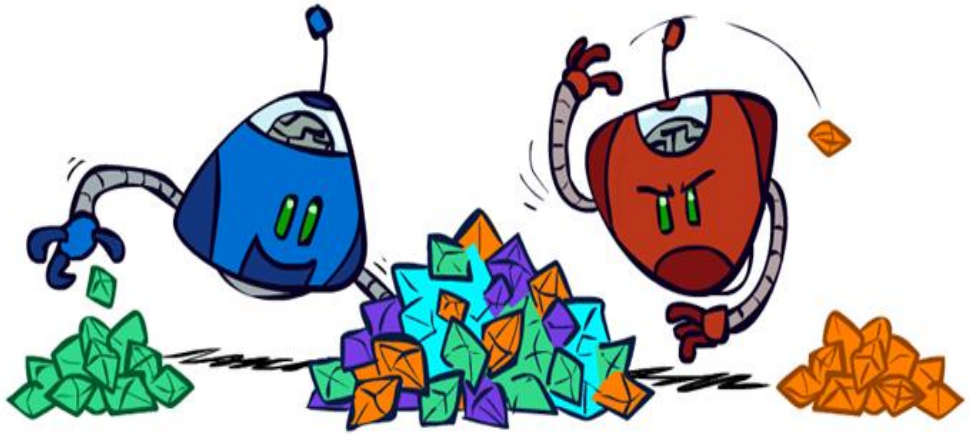
Game-playing programs like chess, checkers, GO, pacman etc were developed by AI researchers s (1950s)

Zero-Sum Games

- Agents have opposite utilities (values on outcomes)
- A single value that one maximizes and the other minimizes
- Adversarial game search is pure competition
- 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)

Eg:Chess and tic-tac-toe

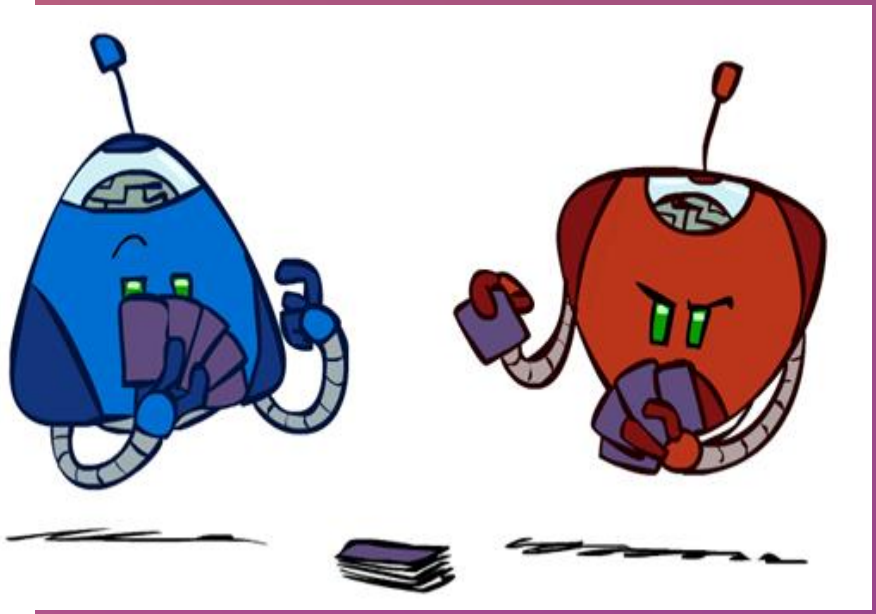
General Games



- Agents have independent utilities (values on outcomes)
- Cooperation, indifference, competition, and more are all possible
- More later on non-zero-sum games

GAME SEARCH

- Competitive environments, in which the agent's goals are in conflict require **adversarial search** are often known as **games.**
- Games are modeled as a Search problem and heuristic evaluation function
- Games require generally **multiagen**t (MA) **environments**
- **Specifics:**
  - Sequences of player's decisions we control
  - Decisions of other player(s) we do not control
- **Contingency problem:** many possible opponent's moves must be "covered" by the solution
- Contigency problem is because of **uncertaintyof Opponent's behavior.**
- **Rational opponent** – maximizes its own **utility(payoff) function**



GAMES VS SEARCH

- **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
  - The machine is “exploring” the search space by itself.

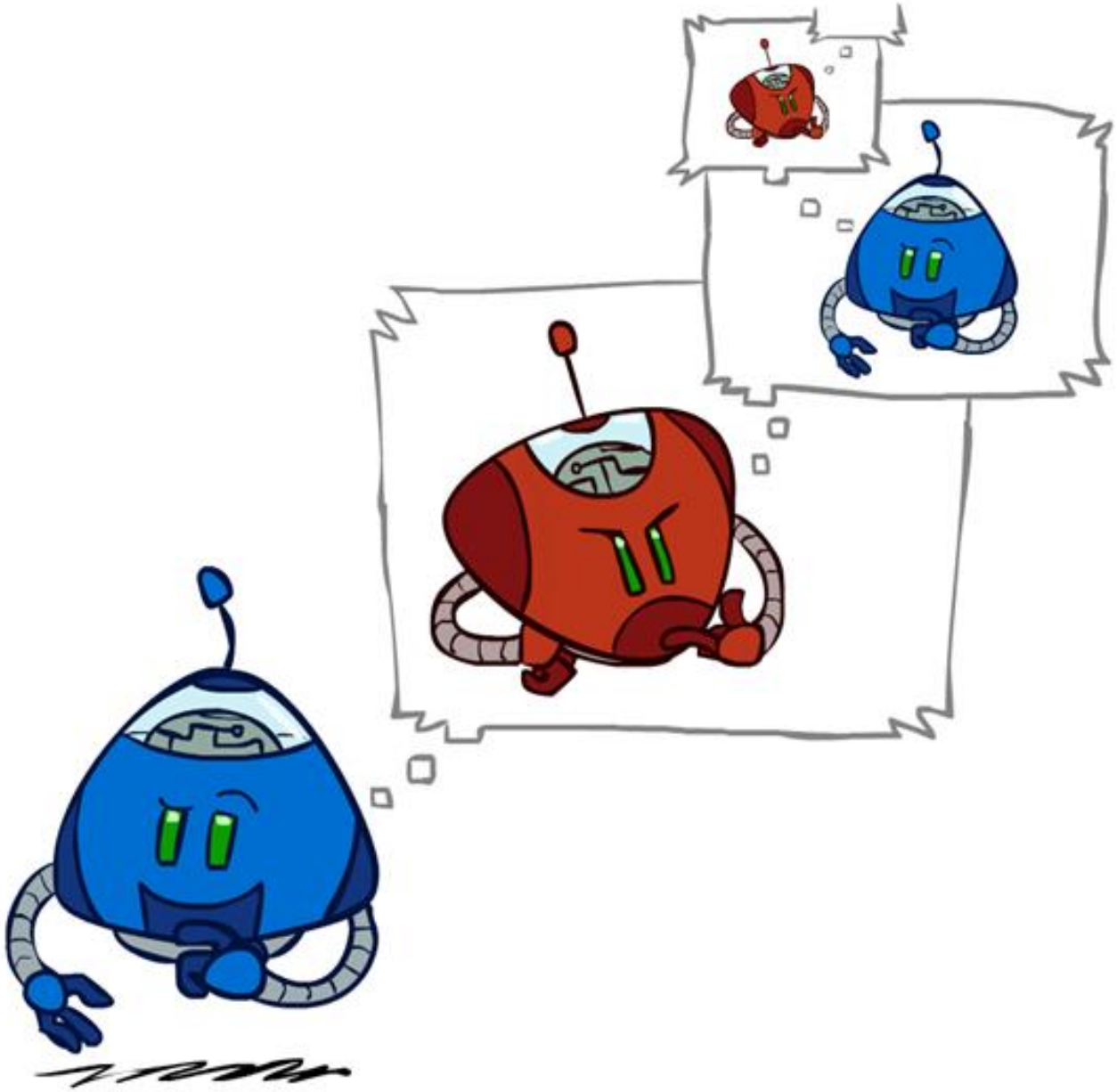
  Examples: path planning, scheduling activities
- **Games – adversary**
  - Solution is **strategy** (strategy specifies move for every possible opponent reply).
  - **Optimality depends on opponent.** Why?
  - Time limits force an *approximate* solution
  - **Evaluation function:** evaluate “goodness” of game position
  - **"Unpredictable" opponent** --> specifying a move for every possible opponent reply
  - **Time limits -->** unlikely to find goal, must approximate

  Examples: chess, checkers, Othello, backgammon

ADVERSARIAL SEARCH

•Adversarial search is a search, where we examine the problem which arises when we try to plan ahead of the world and other agents are planning against us.

• 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, often known as Games.



GAME SEARCH PROBLEM FORMULATION AS ADVERSARIAL SEARCH

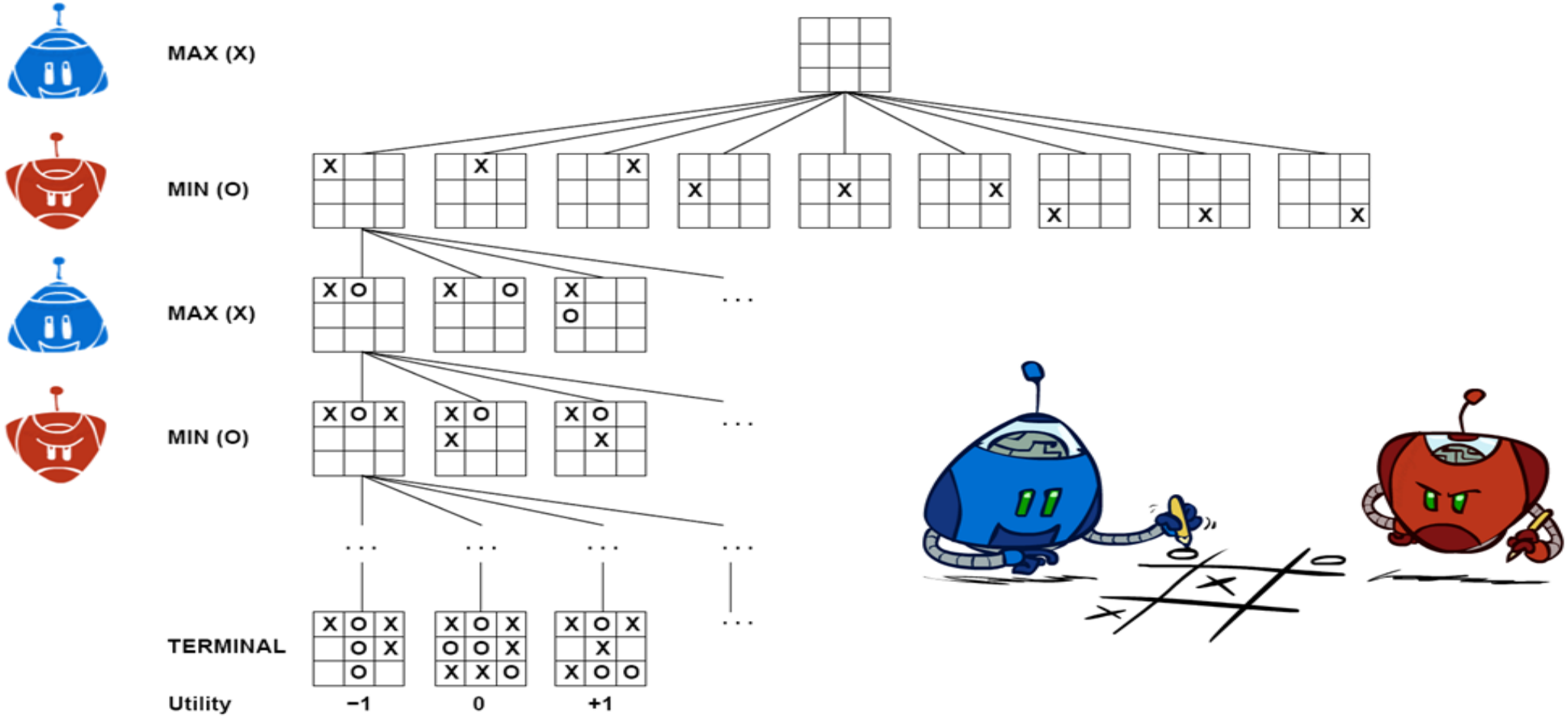
1. **Initial state(S_0):** initial board position + whose move it is. How game is set up at start.
2. **Operators:** legal moves a player can make. **Successor function** returns list of (move, state) pairs, each indicating a legal move and the resulting state.
3. **Goal (terminal test):** It reurns true when game is over.
4. **Utility (payoff) function:** measures the outcome of the game and its desirability. A utility function gives the final numeric value for a game that ends in terminal states s for player p. Utility can be thought of as a way to "score" each possible move based on its potential to result in a win.
5. **States**: board configurations.
6. **Result(s,a);** Transition model which defines result of moves.
7. **Player(S):** Defines which player has move in a state.
8. **Actions(S):** Returns the set of legal moves in a state.

Search objective:

- Find the sequence of player's decisions (moves) maximizing its utility (payoff)
- Consider the opponent's moves and their utility

GAME TREE

A game tree is a tree where nodes of the tree are the game states and Edges of the tree are the moves by players. Game tree involves initial state, actions function, and result Function.



- **At the leaf nodes, the utility function is employed. Big value means good, small is bad.**
- Each layer of game tree is called as **Ply**.

MIN MAX SEARCH

MIN MAX SEARCH

- **Min-max algorithm** is a **recursive or backtracking algorithm** which is used in decision-making and game theory
- A strategy/solution for **optimal** decisions
- Min-Max Algorithm uses the **game tree** where each level has the additional information as:
  - **Max** if player wants to **maximize utility/reward**(Agent) (**Best move for player 1)**
  - **Min** if player wants to **minimize utility**(opponent)(**worst move player 1**)
- **Best move strategy** is used by both the players. They fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Min max algorithm calculate values of leaf node and propagate to root node so as to decide which path should be opted for.
- Perfect play for **deterministic environments with perfect information**

MIN-MAX SEARCH

- **Basic idea:** choose move with highest minimax value = best achievable payoff against best play
- A complete depth-first, recursive exploration of the game tree
- Initialize **max** and **min** with worst values ie, **Max=∞ Min=- ∞**

Algorithm:

1. Generate game tree completely

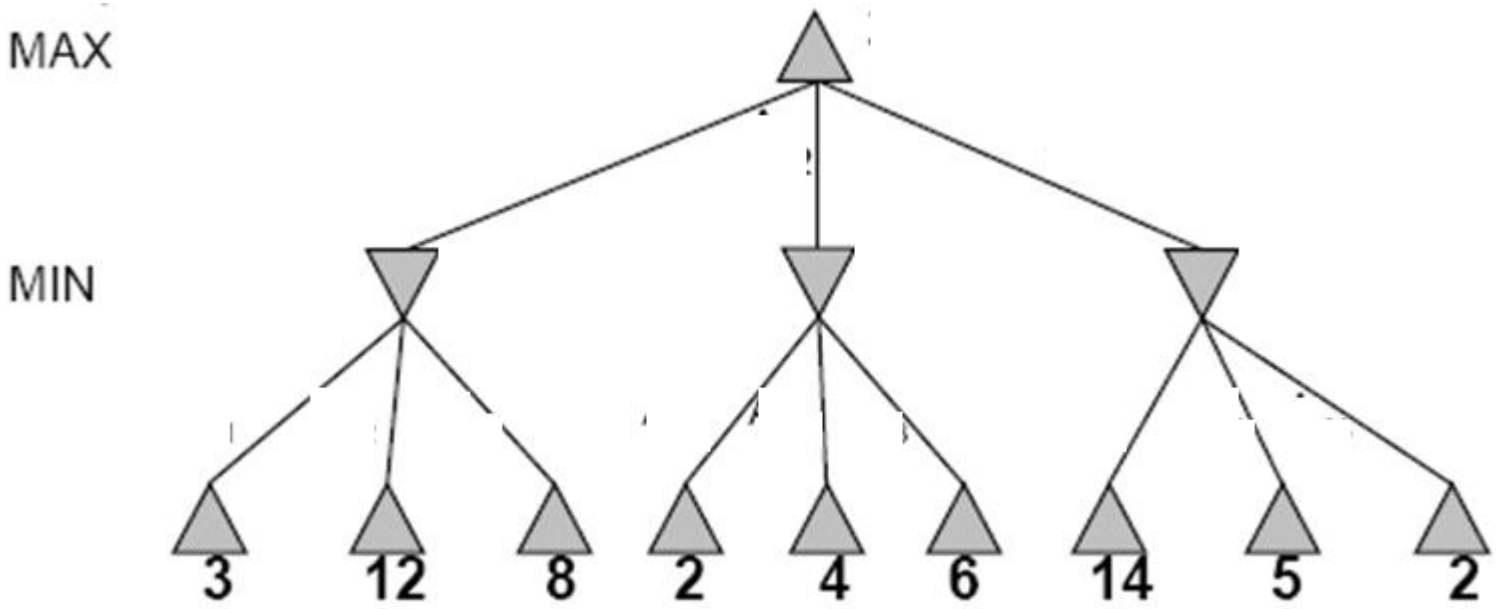
2. Determine utility of each terminal state

3. Propagate the utility values upward in the tree by applying MIN and MAX operators on the nodes in the current level

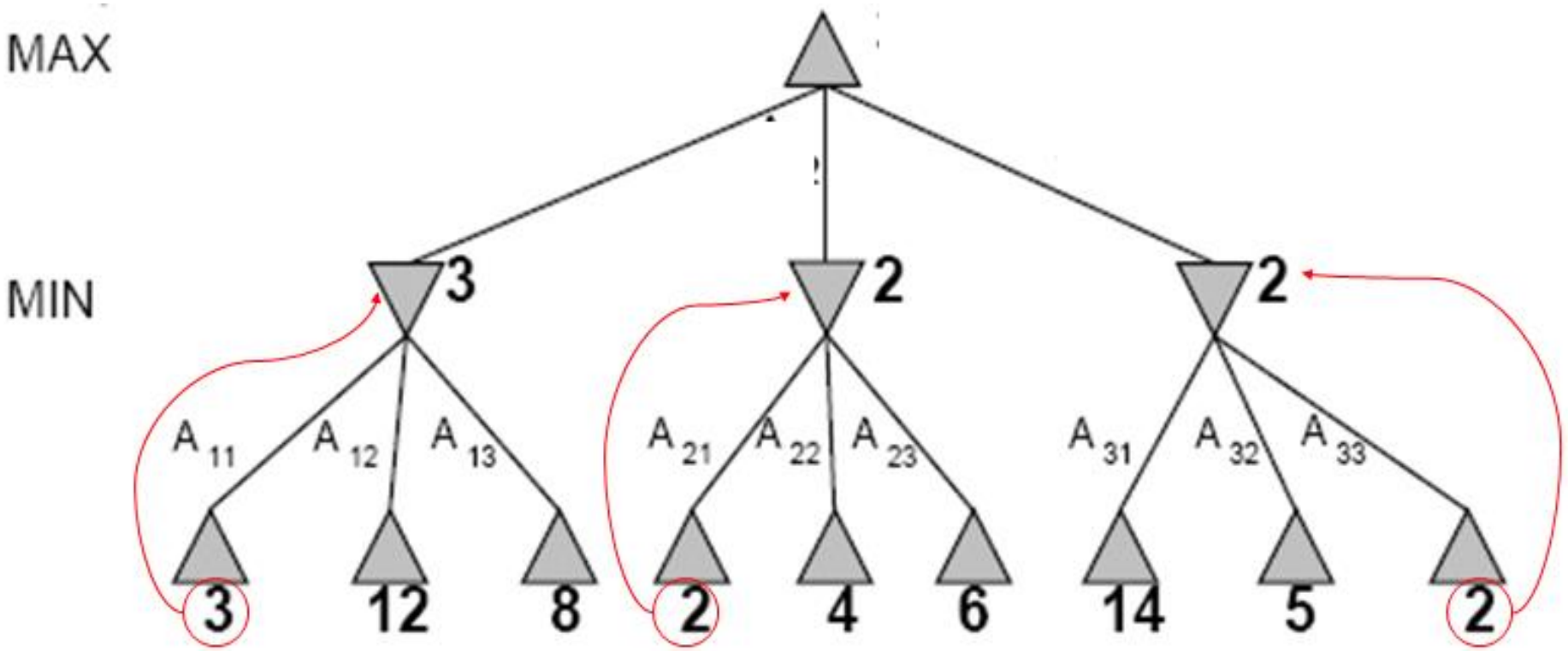
4. At the root node use minimax decision to select the move with the max (of the min) utility value

- Steps 2 and 3 in the algorithm assume that the opponent will play perfectly.

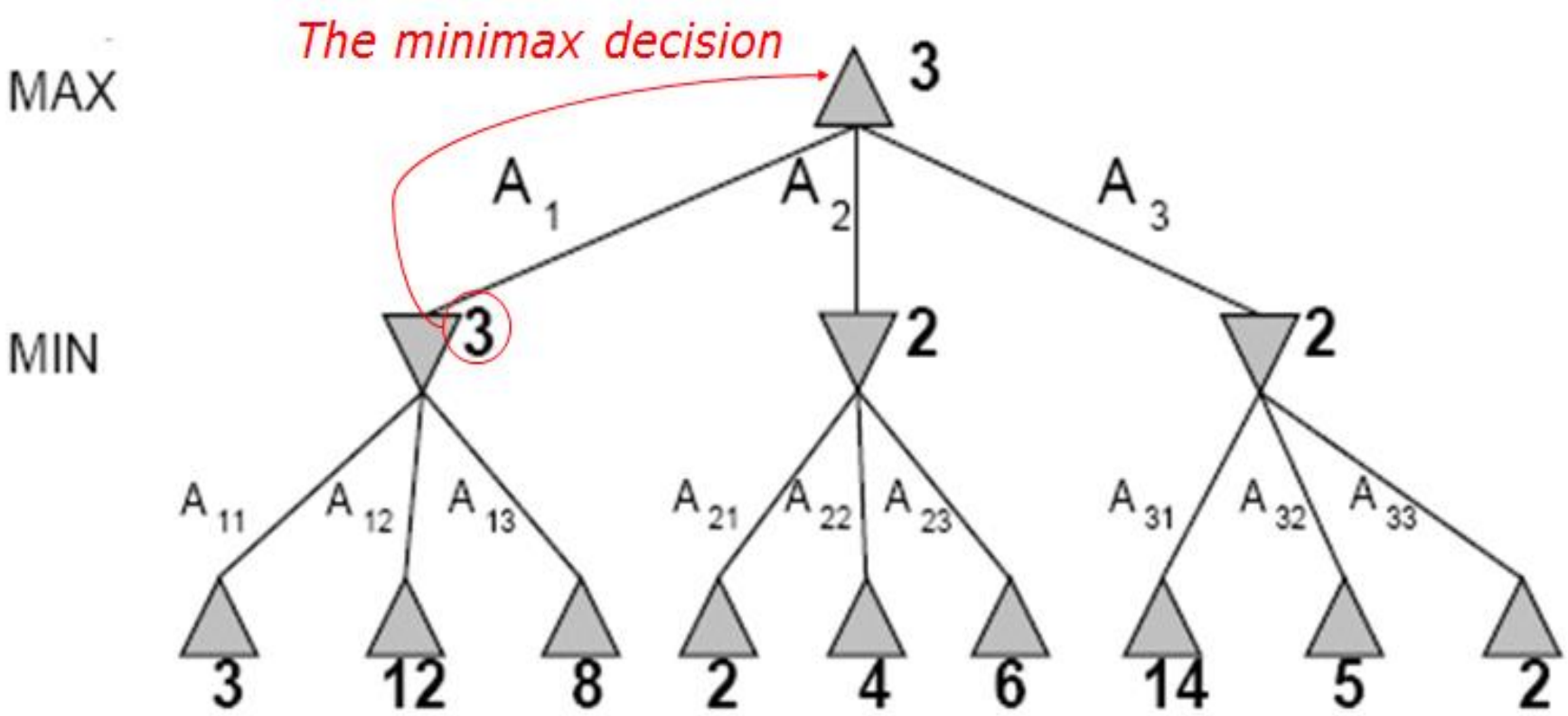
MIN-MAX SEARCH EXAMPLE



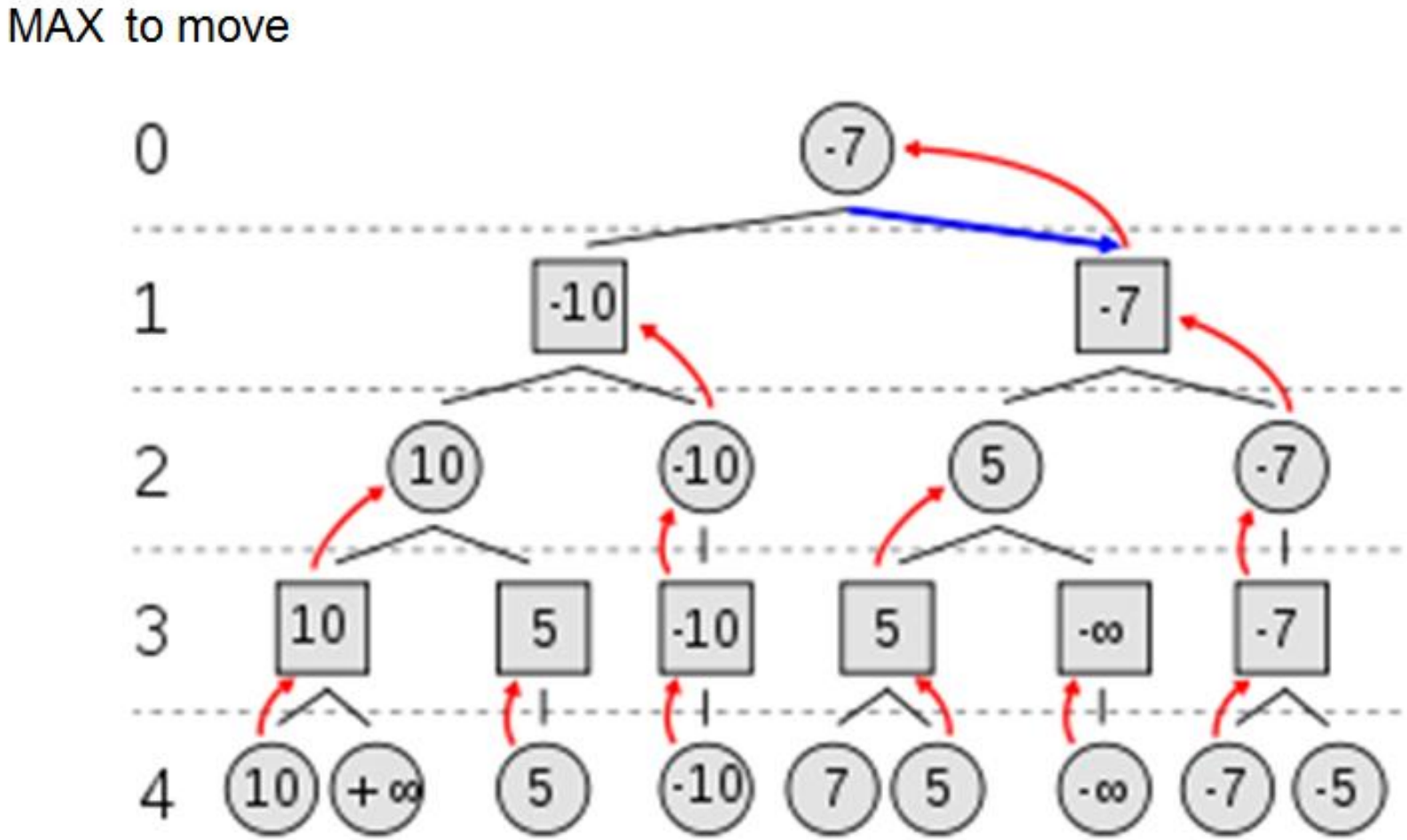
MIN-MAX SEARCH EXAMPLE



MIN-MAX SEARCH EXAMPLE



EXAMPLE

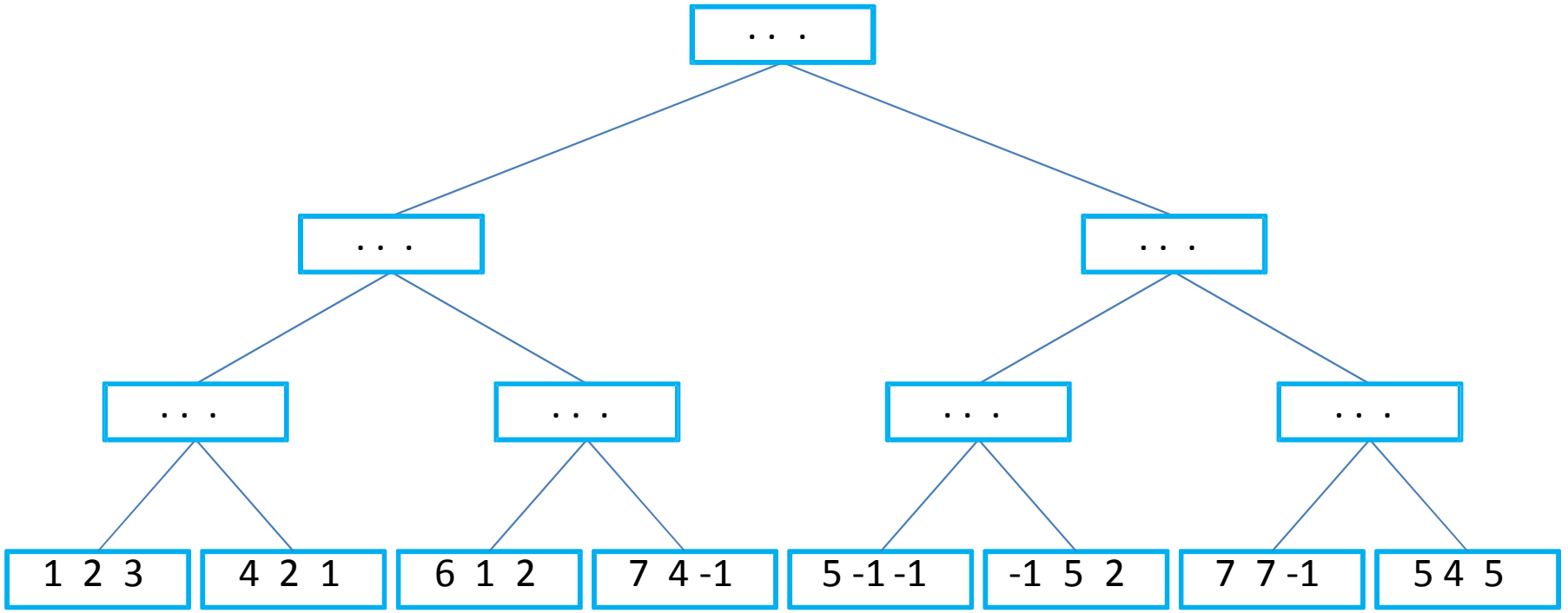


OPTIMALDECISION IN MULTIPLAYER GAME

- Extend the minimax idea to multiplayer games
- Replace the single value for each node with a vector of values (utility vector)
- Extend the minimax idea to multiplayer games
  - `Replace the single value for each node with a vector of values (utility vector)

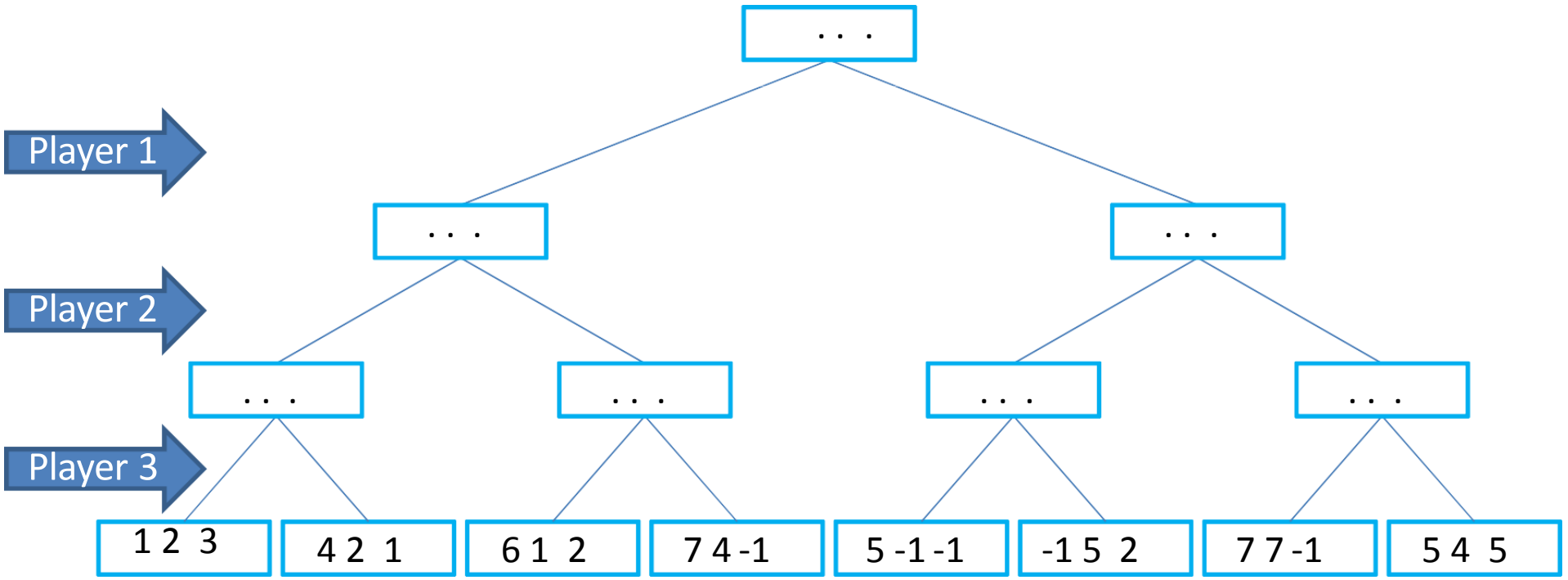
MiniMax For 3 Players

- All players are Max
- Evaluation function given by vector



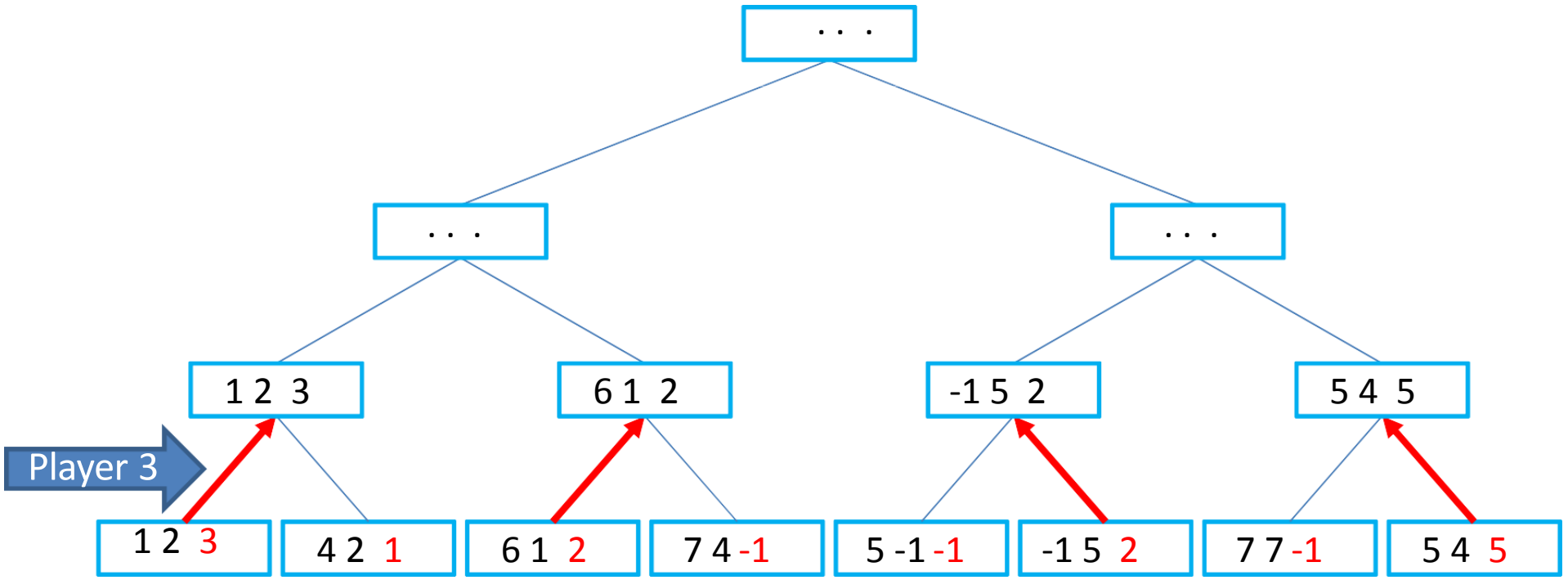
MiniMax For 3 Players

- Each layer assigned to 1 player
- Turn: every 3 layers



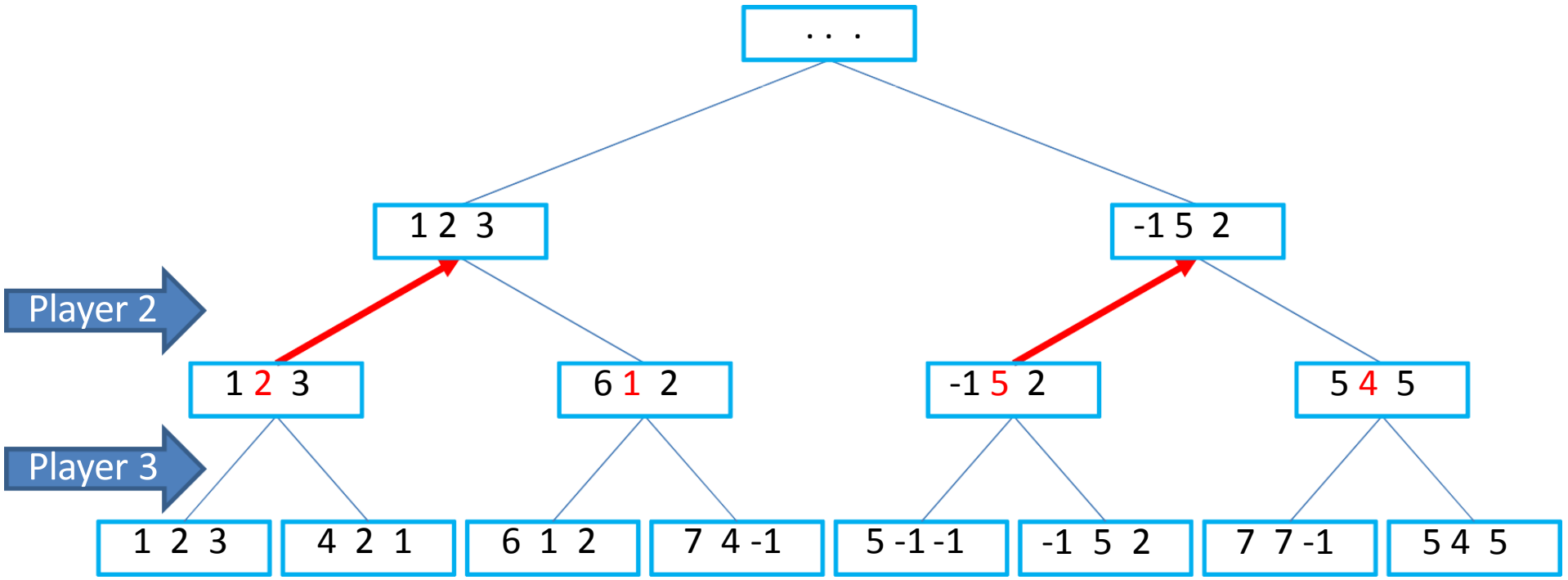
MiniMax For 3 Players

- MaxThirdPlayer([1,2,3],[4,2,1]) = [1,2,3]
- MaxThirdPlayer([6,1,2],[7,4,-1]) = [6,1,2]
- MaxThirdPlayer([5,-1,-1],[-1,5,2]) = [-1,5,2]
- MaxThirdPlayer([7,7,-1],[5,4,5]) = [5,4,5]



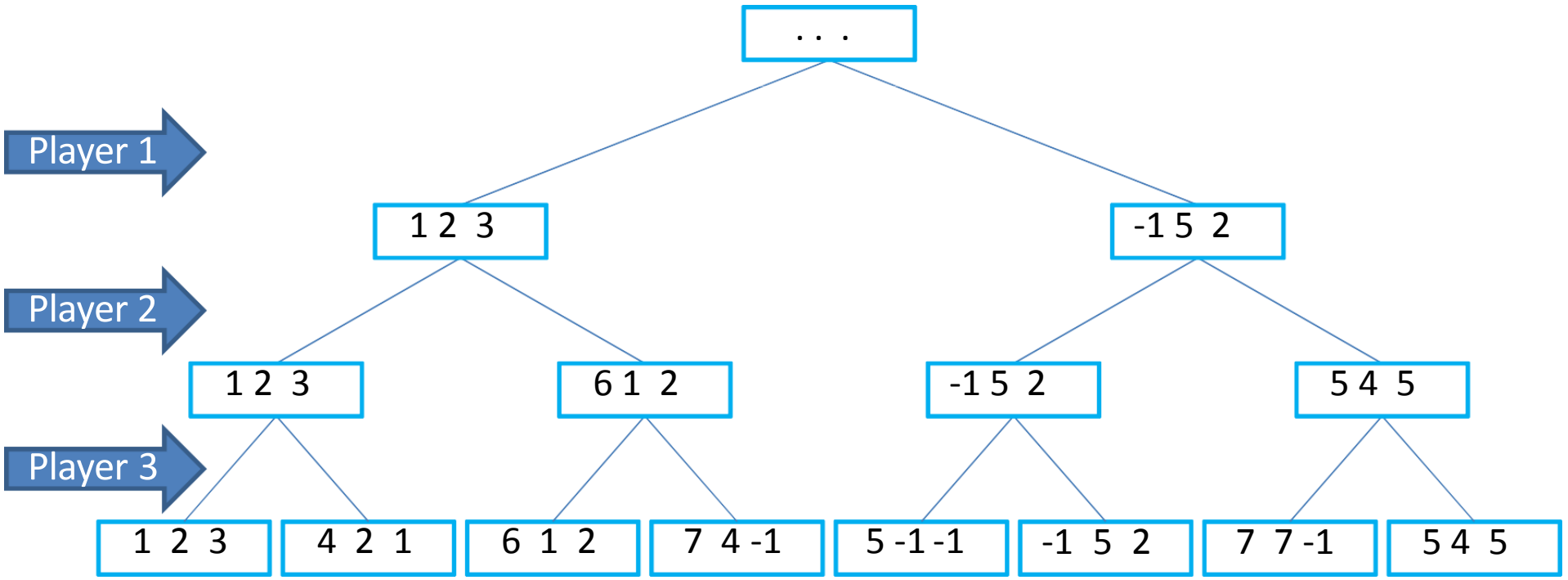
MiniMax For 3 Players

- MaxSecondPlayer([1,2,3],[6,1,2]) = [1,2,3]
- MaxSecondPlayer([-1,5,2],[5,4,5]) = [-1,5,2]



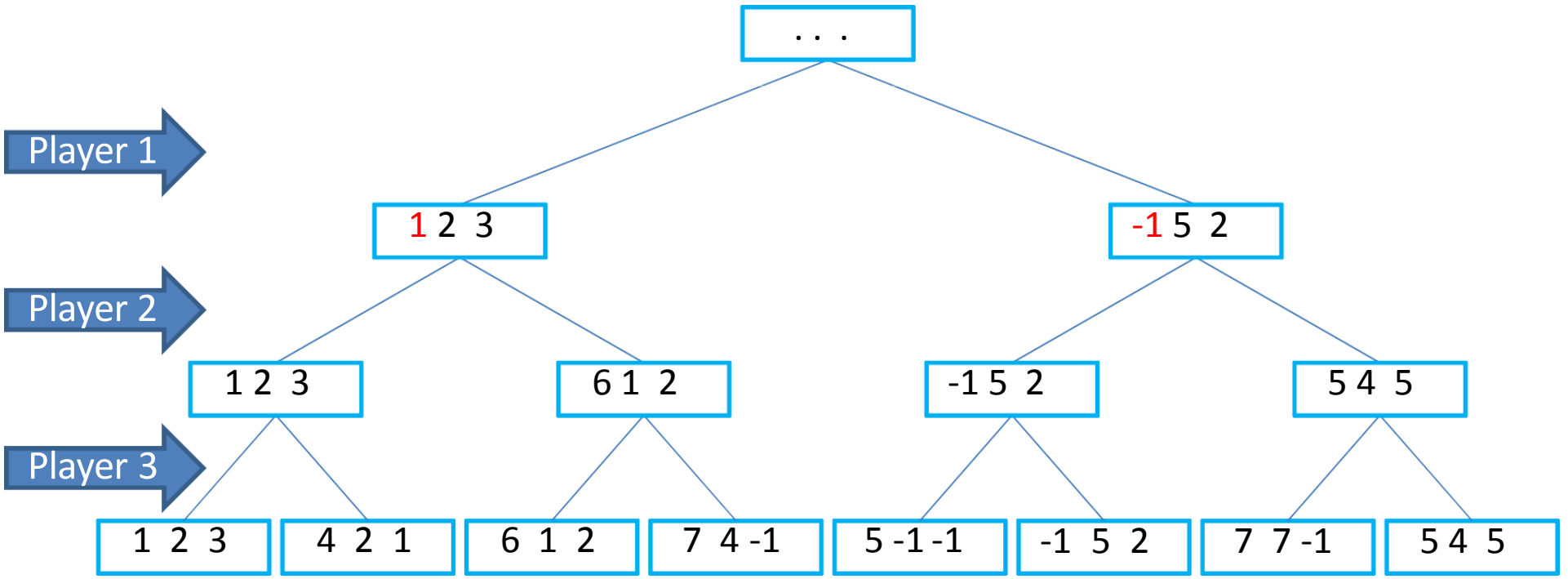
MiniMax For 3 Players

- First player's move



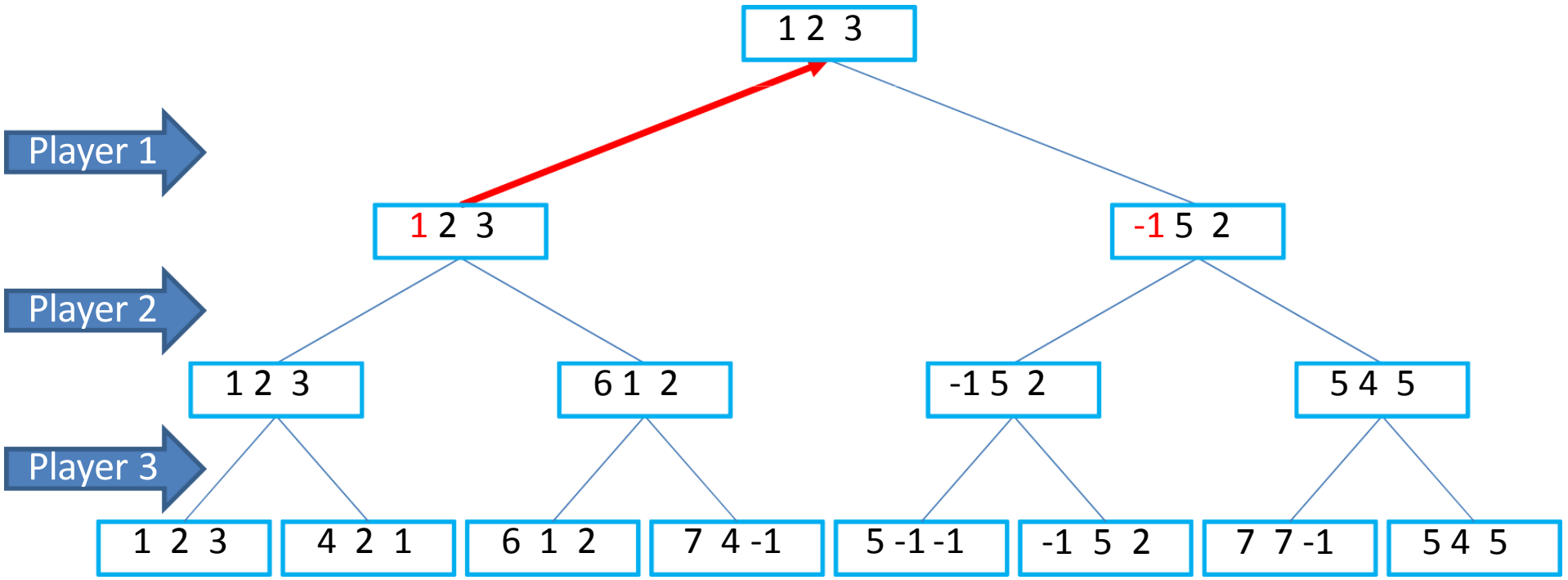
MiniMax For 3 Players

- Max first player: first position of vector



MiniMax For 3 Players

- MaxFirstPlayer([1,2,3],[-1,5,4]) = [1,2,3]



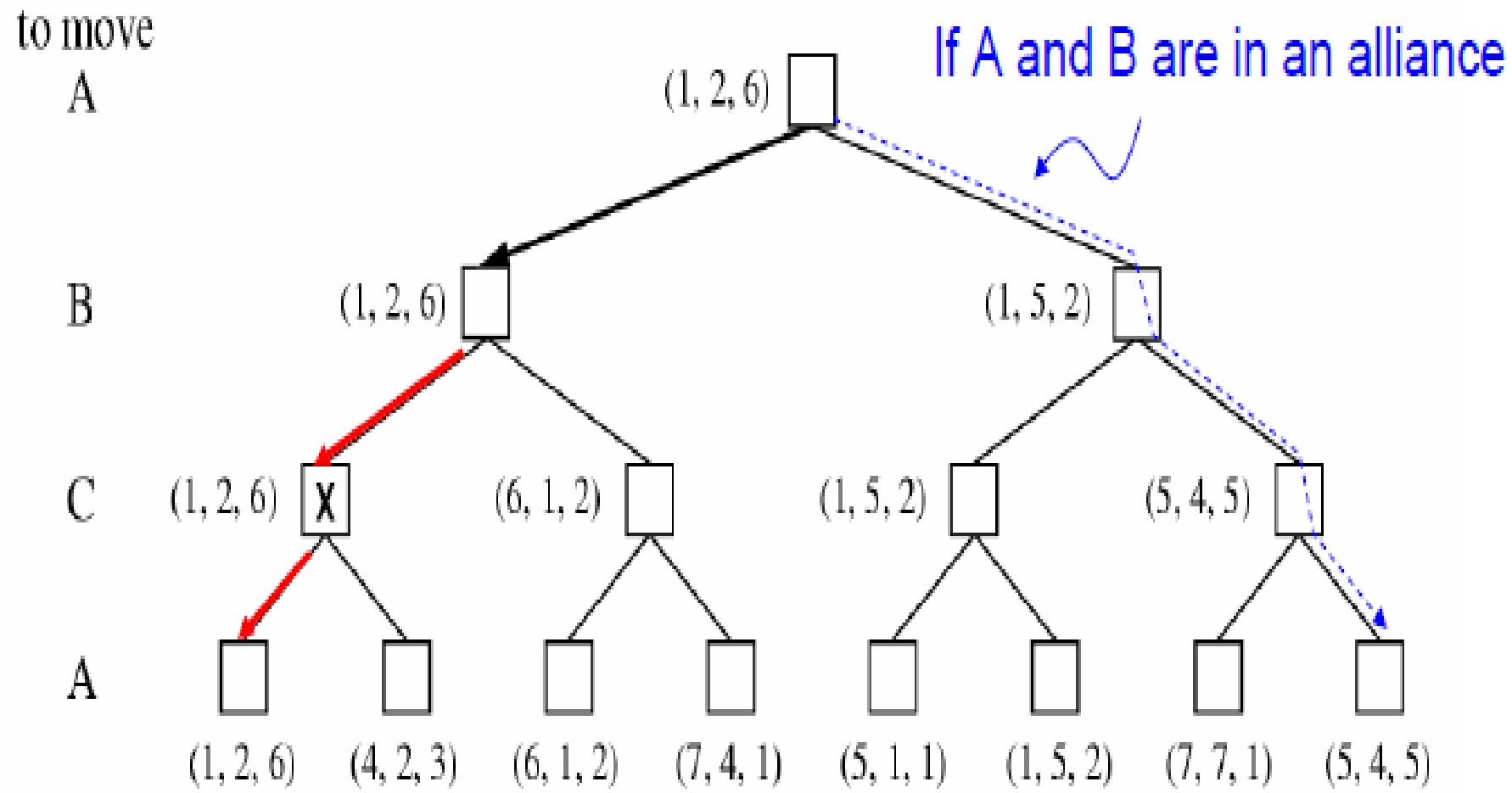
MULTIPLAYER GAMES

MAX

MIN

MAX

MIN



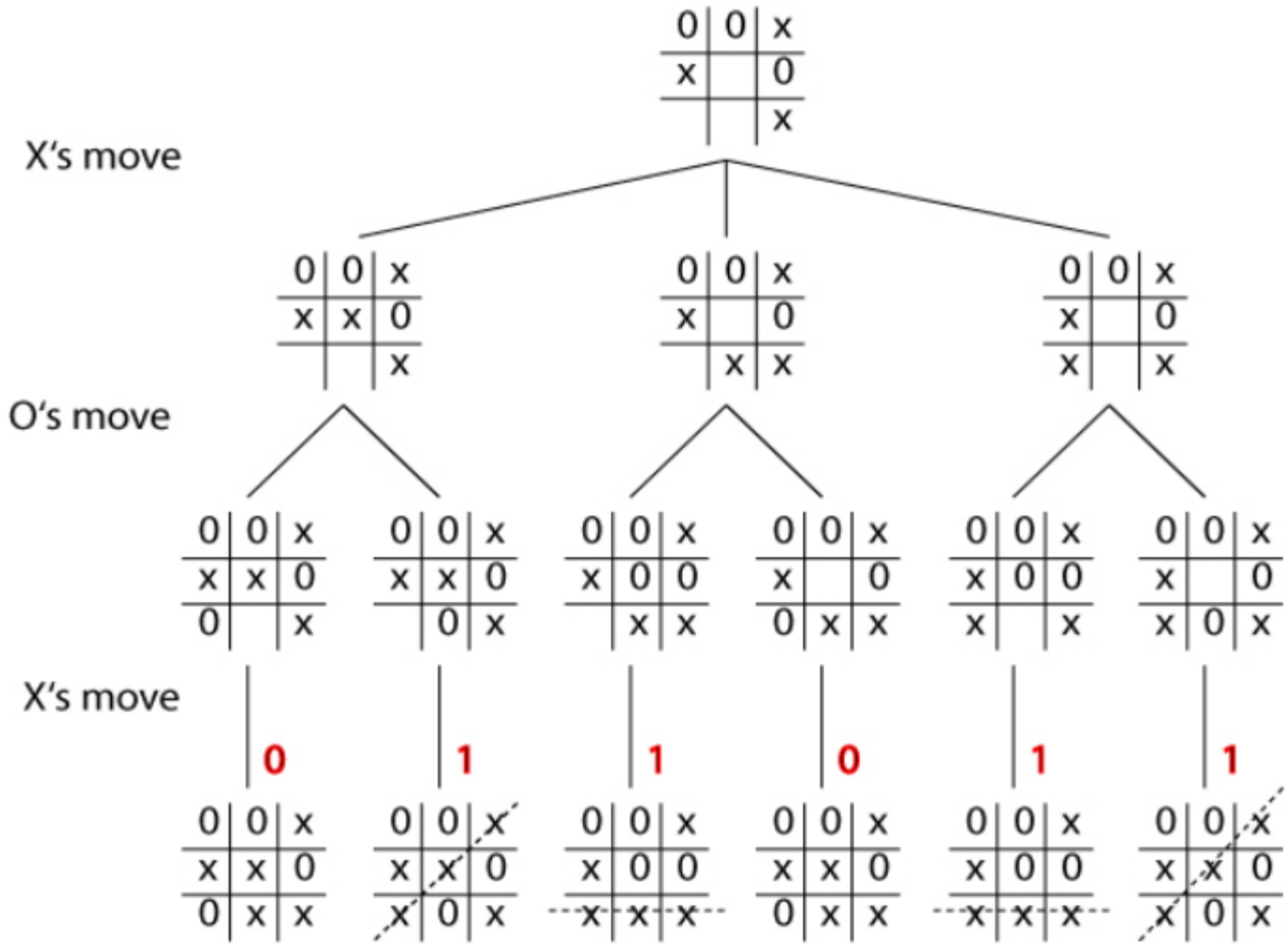
MIN MAX EFFICIENCY

- How efficient is minimax?
  - **Complete? Yes** (if tree is finite)
  - **Optimal? Yes** (against an optimal opponent)
  - **Time**: **$O(b^m)$** Just like (exhaustive) DFS
  - **Space: O(bm)**
- **Disadvantage:** Evaluate all nodes
- It 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**

Static Evaluation:

'+1' for a win, '0' for a draw

- **Criteria '+1' for a Win, '0' for a Draw**

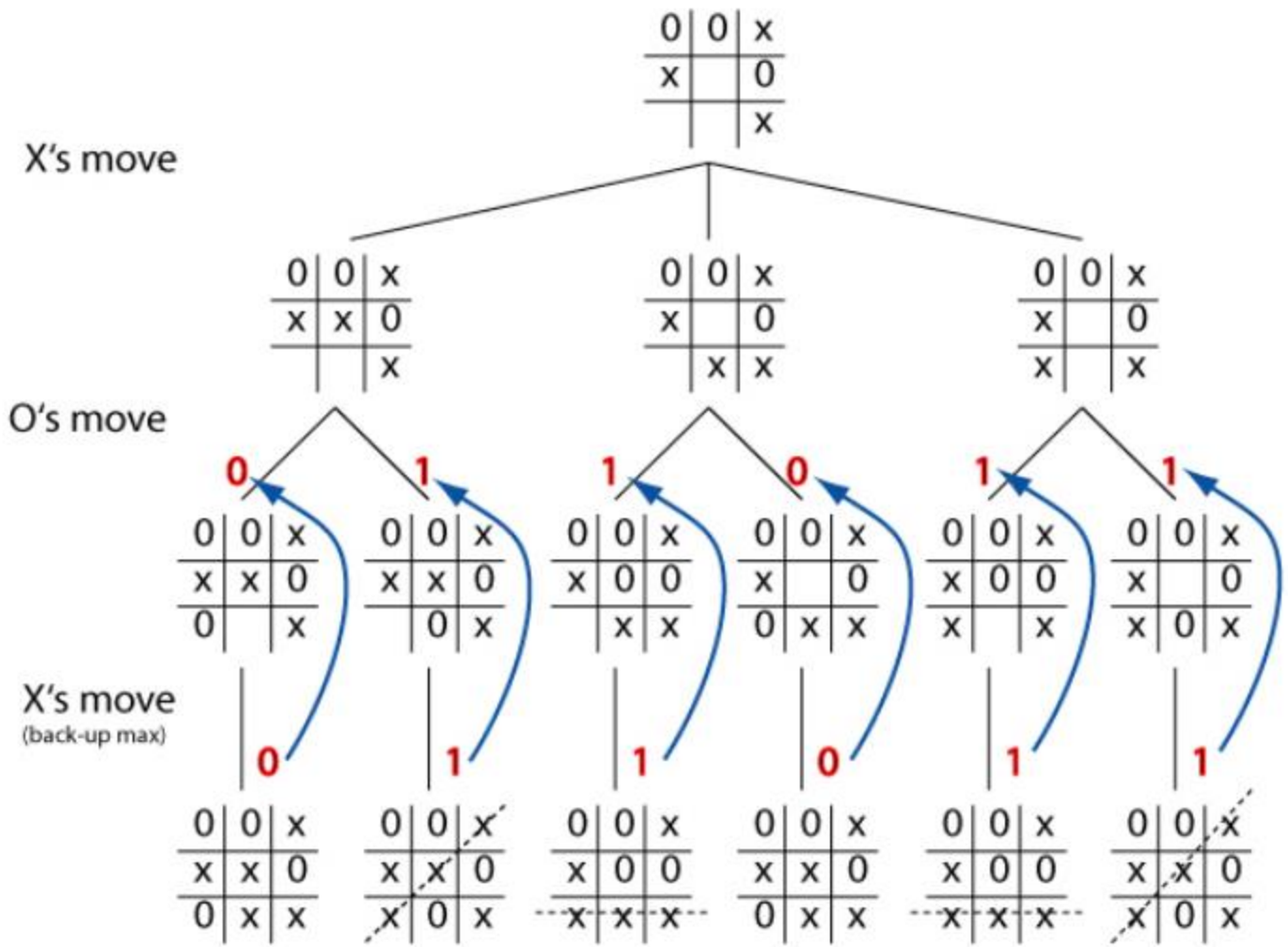


Back-up the Evaluations:

Game playing with Mini-Max

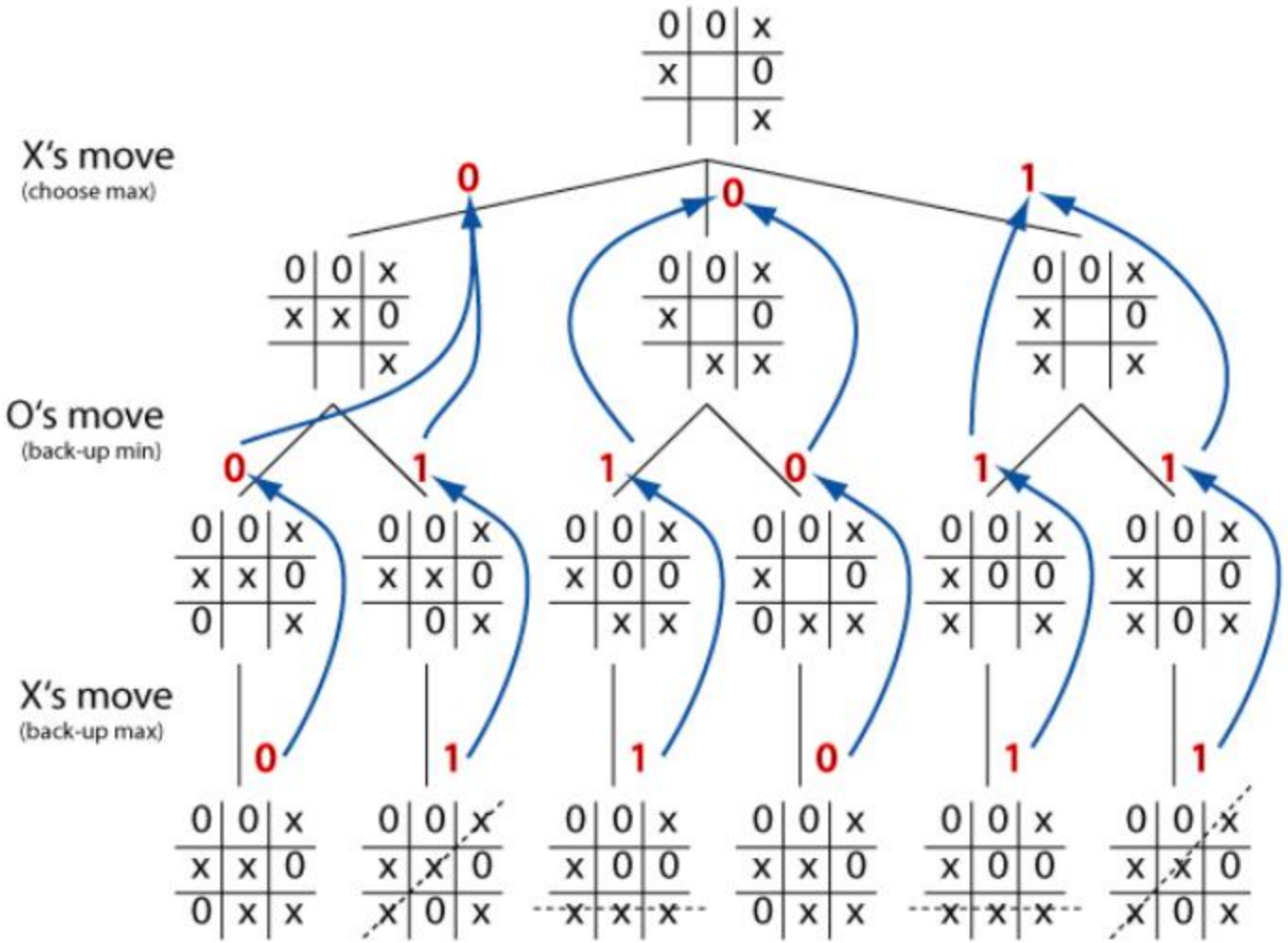
Level by level, on the basis of opponent's turn

- **Up : One Level**



- **Up : Two Levels**

Game playing with Mini-Max



MAX

MIN

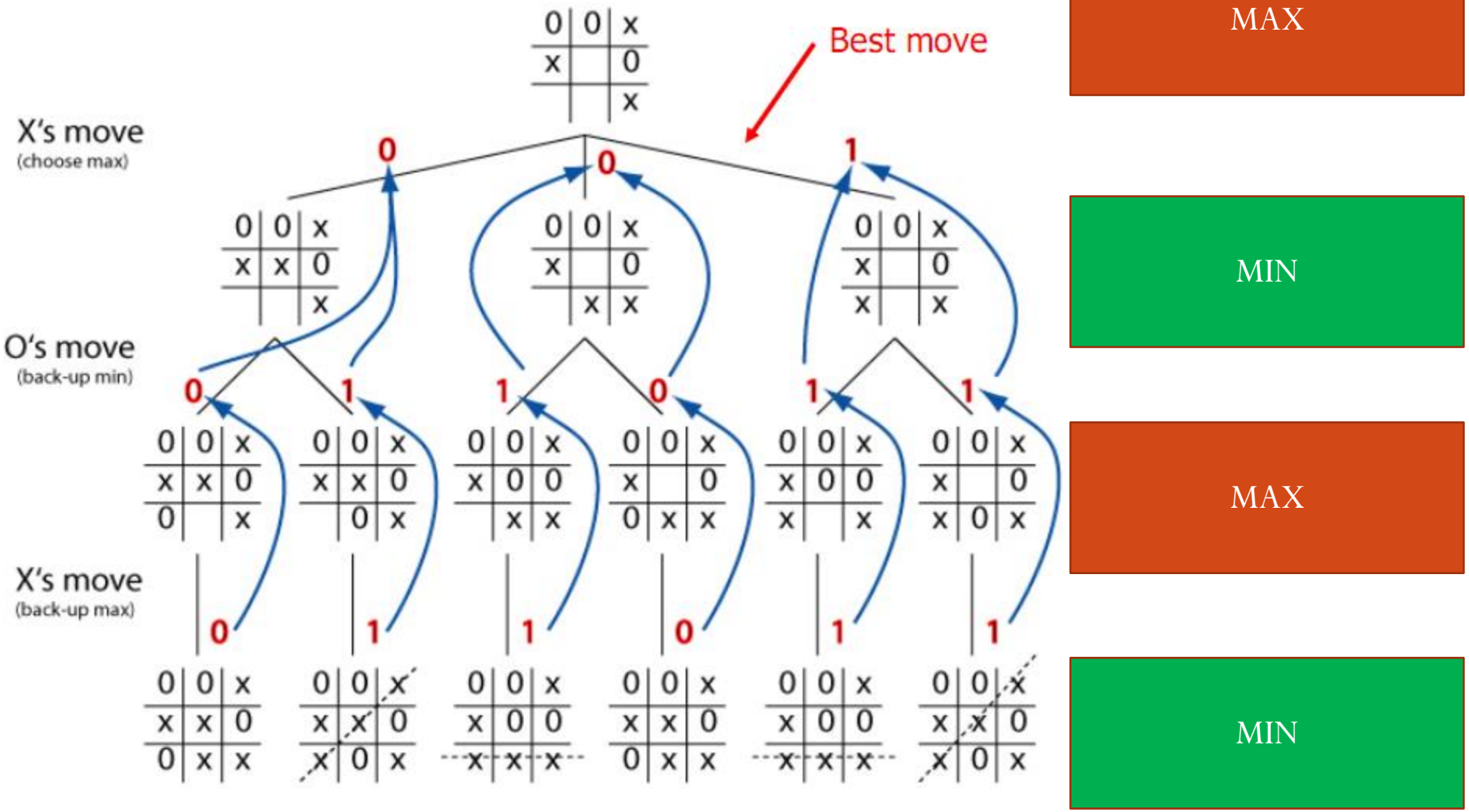
MAX

MIN

Evaluation obtained :

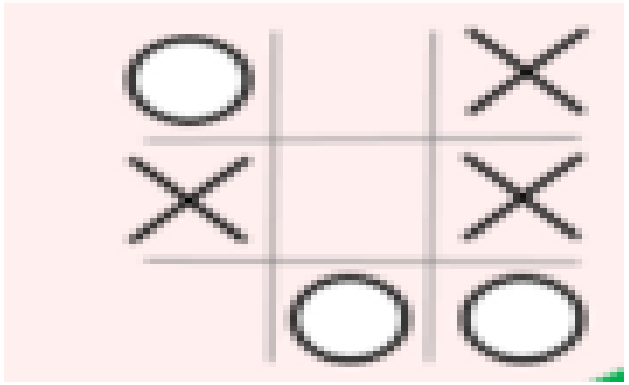
Game playing with Mini-Max

Choose best move which is maximum

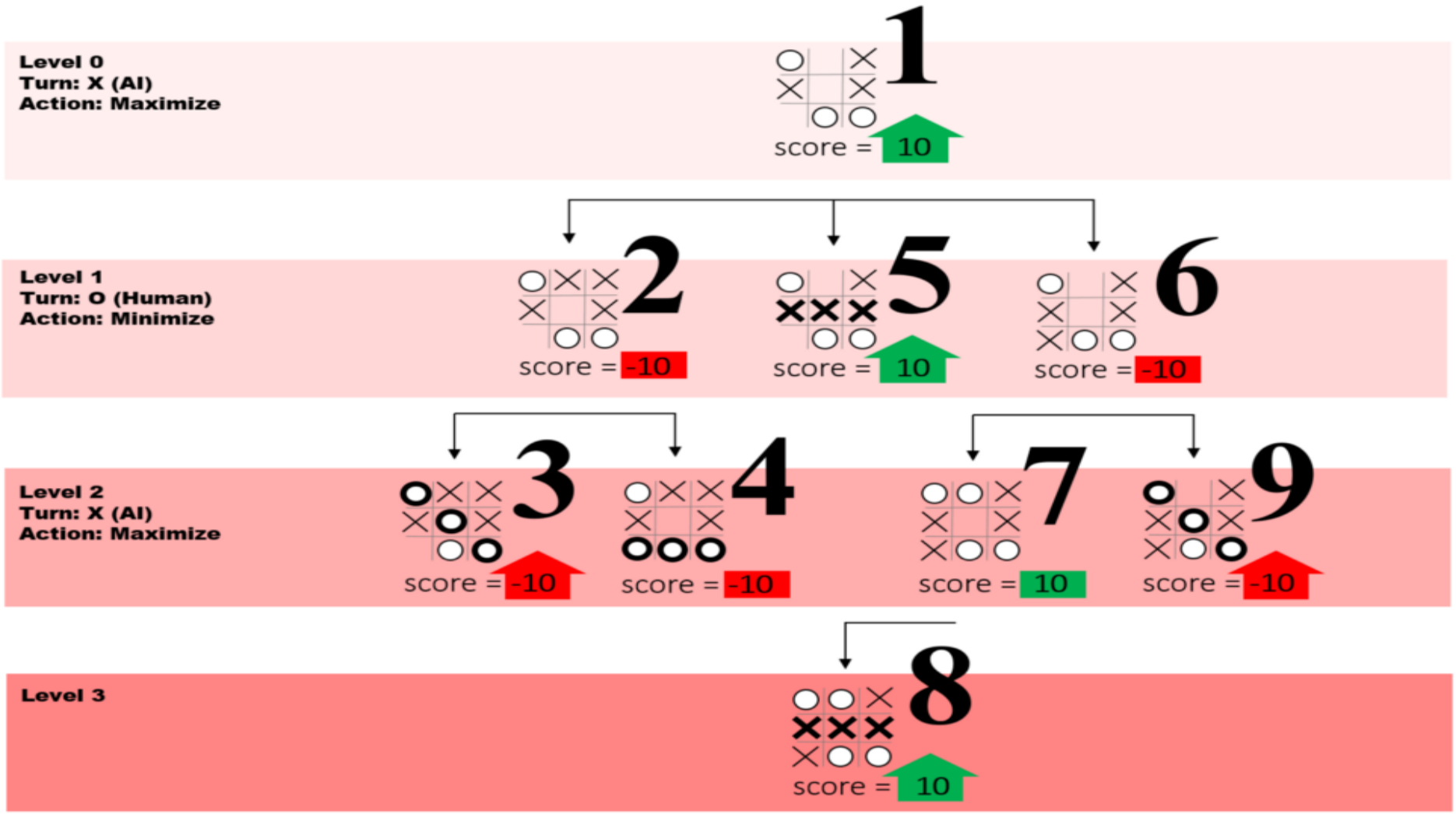


Challenge

- Let AI is the one who marks X and the human player marks O.Identify which step is to be taken further for the given state if AI player has next turn . Assume If O wins you should return -10, if X wins you should return +10.



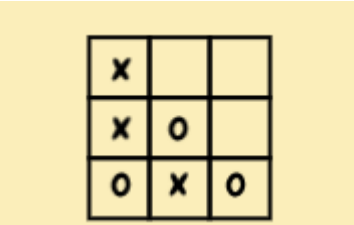
SOLUTION


Level 0
Turn: X (AI)
Action: Maximize
1
score = 10
Level 1
Turn: O (Human)
Action: Minimize
2
score = -10
5
score = 10
6
score = -10
Level 2
Turn: X (AI)
Action: Maximize
3
score = -10
4
score = -10
7
score = 10
9
score = -10
Level 3
8
score = 10

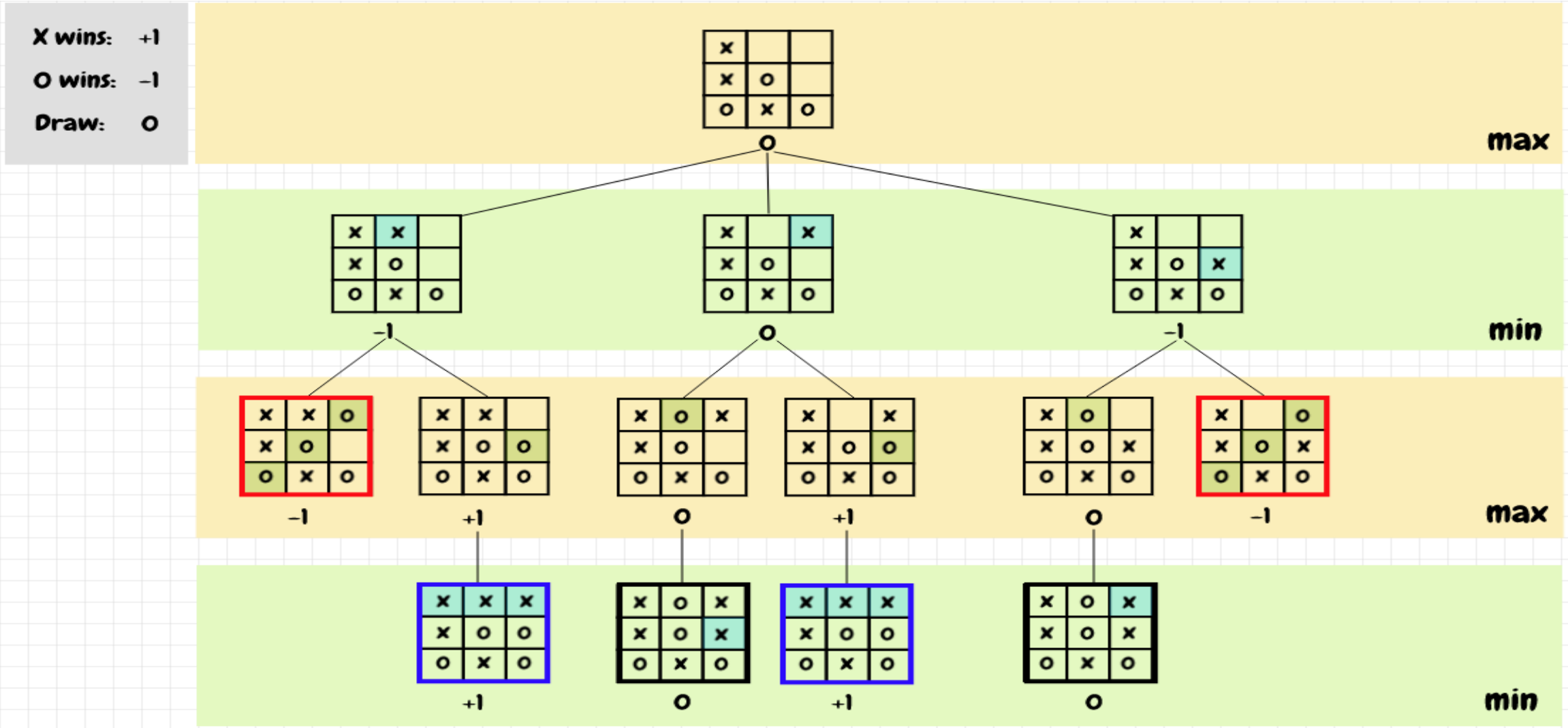
SOLUTION

- Identify which step is to be taken further for the given state if X player has next turn .

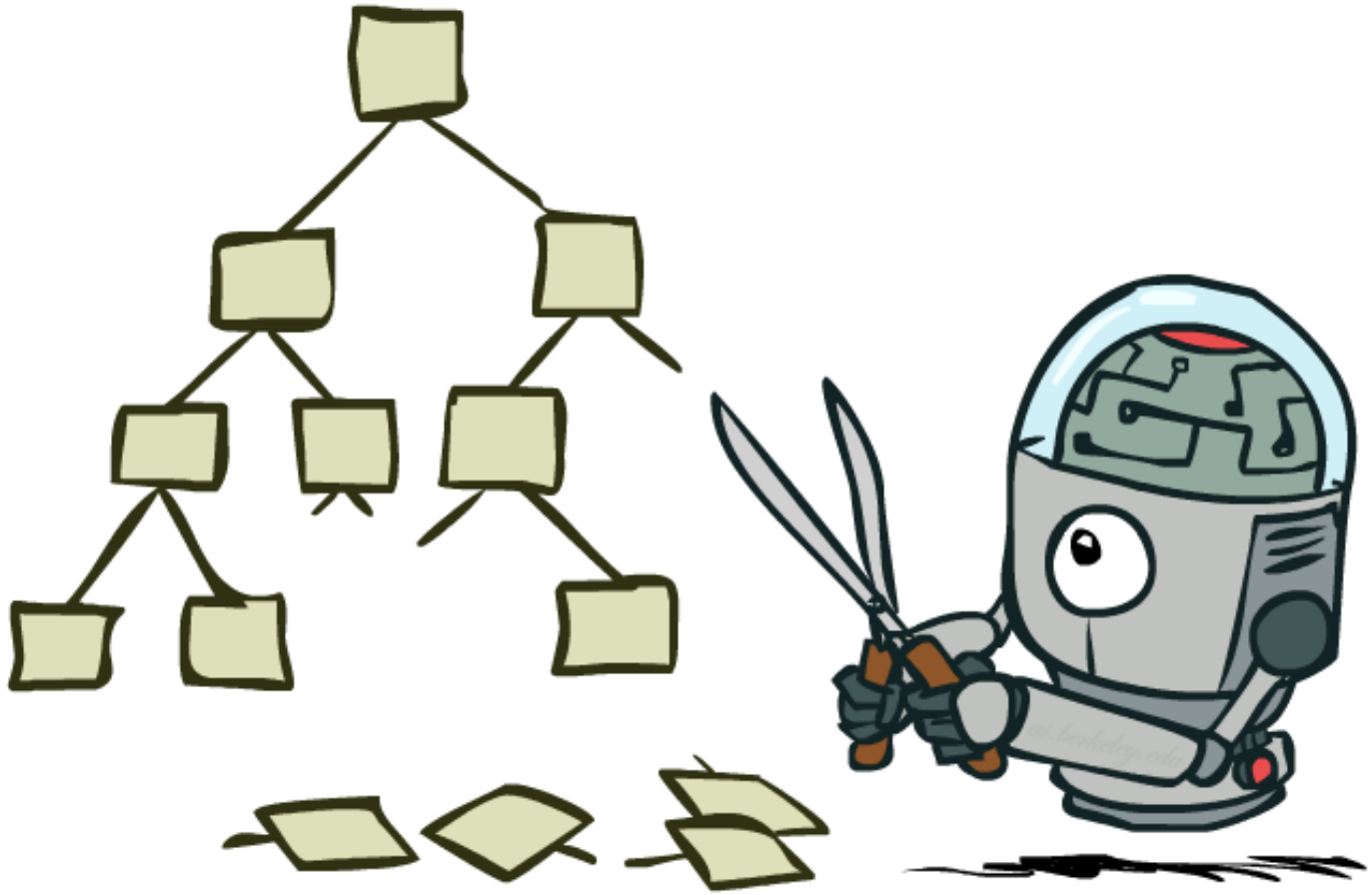
  Assume If X wins you should return +1, O wins returns –1 and wilbe 0 if it's a draw.



SOLUTION



GAME TREE PRUNING



MIN MAX CONCERN

- The **number of game nodes to examine** with minimax search is **exponential to number of moves.**
- Is it possible to **compute the correct minimax decision without looking at every node** in the game tree?

SOLUTION

- Alpha-beta pruning is a modified version of the minimax algorithm. It is an **optimization technique for the minimax algorithm.**
- Same as that of minmax but **prune** away branches that cannot **possibly influence the final decision.**

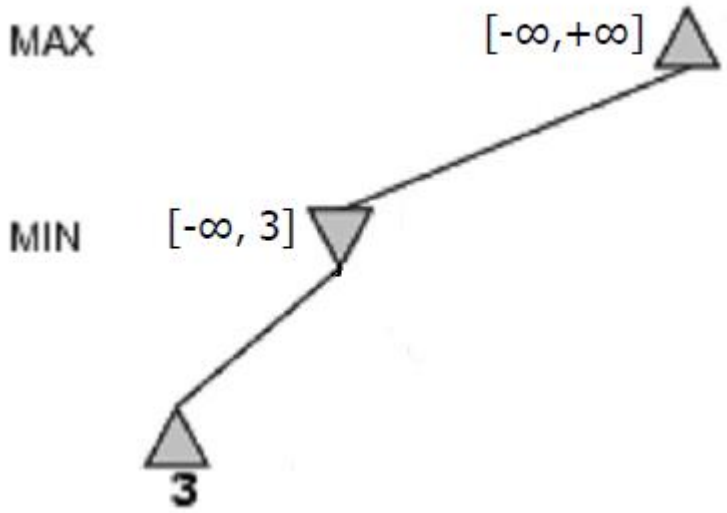
DEFINITION OF APLPHA BETA

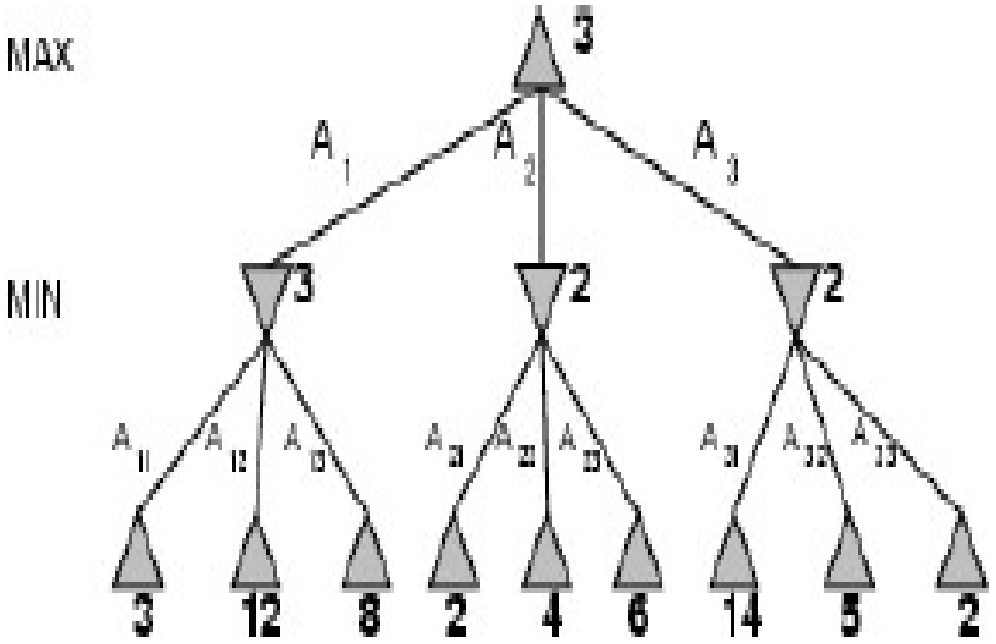
- **α** = the value of the **best (highest-value) choice we have found so far** at any choice point along the path for **MAX.** The initial value of alpha is **-∞**.
- **β** = the value of the **best (lowest-value) choice we have found so far** at any choice point along the path for **MIN.** The initial value of beta is **+∞**.
- If *v* is worse than α, *max* will avoid it-->prune that branch
- **α-β Search updates** the values of α and β as it goes along and **prunes** the remaining branches at a node as soon as the value of the current node is worse than the current α or β for MAX or MIN respectively

ALPHA BETA PRUNING

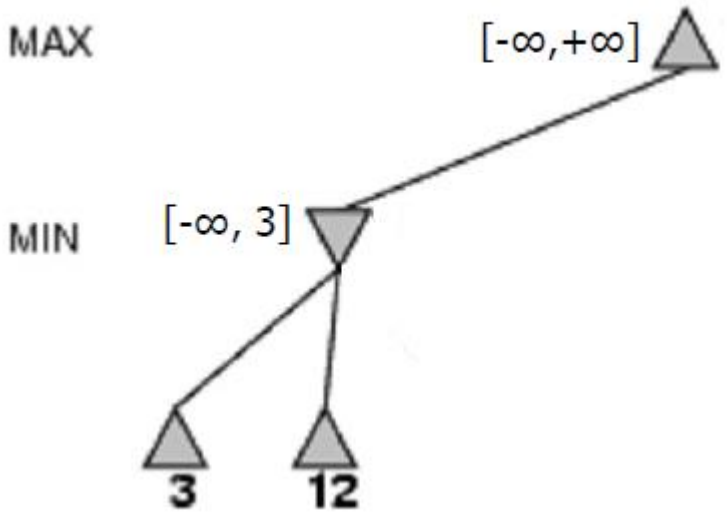
- **α-β pruning Search** cannot eliminate the exponent, but we can cut it to half. It cuts off search cost by exploring less no of nodes
- The **effectiveness** of alpha-beta pruning is highly dependent on the **order in which the successors are examined.**
- Alpha-beta pruning can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.
- 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.
- Always α>=β

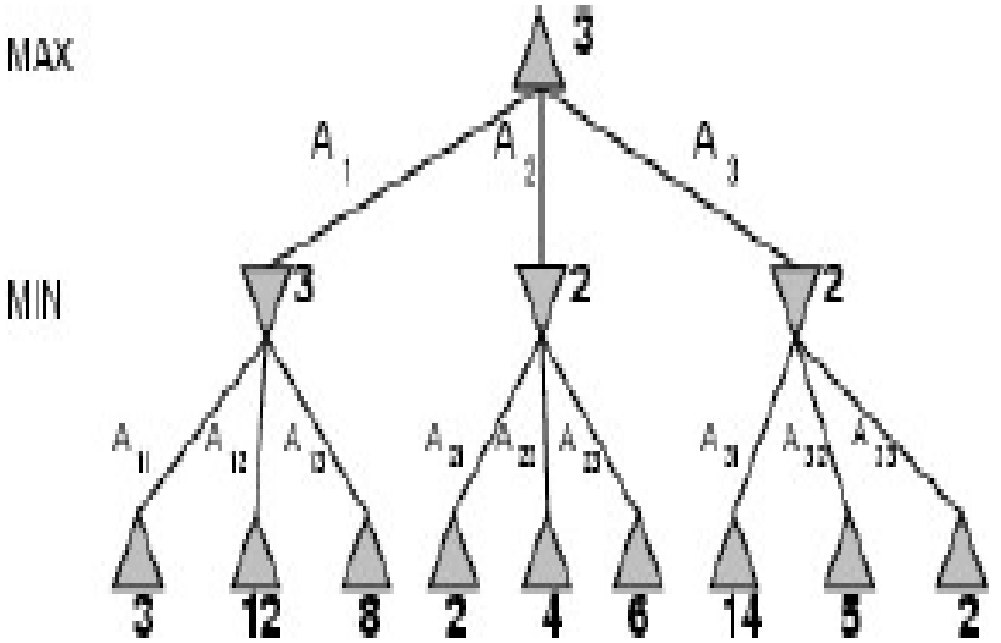
EXAMPLE



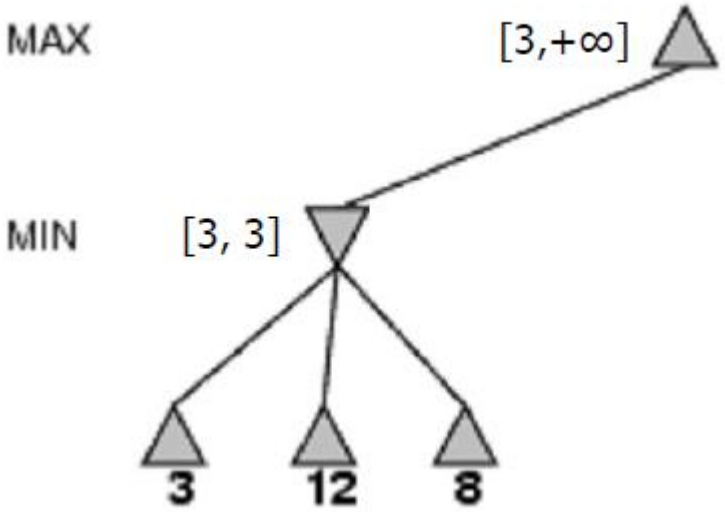


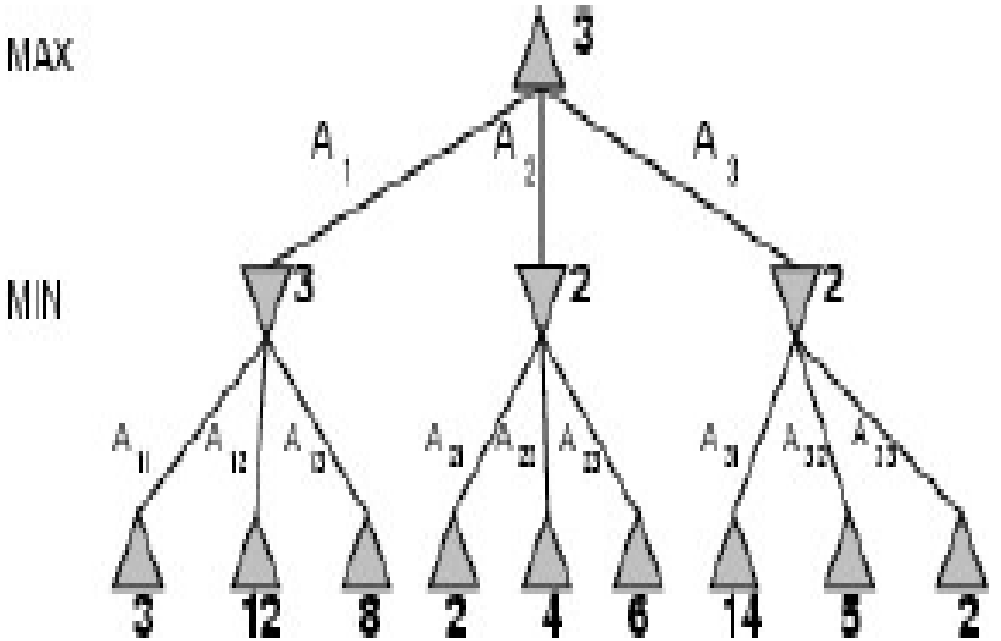
EXAMPLE



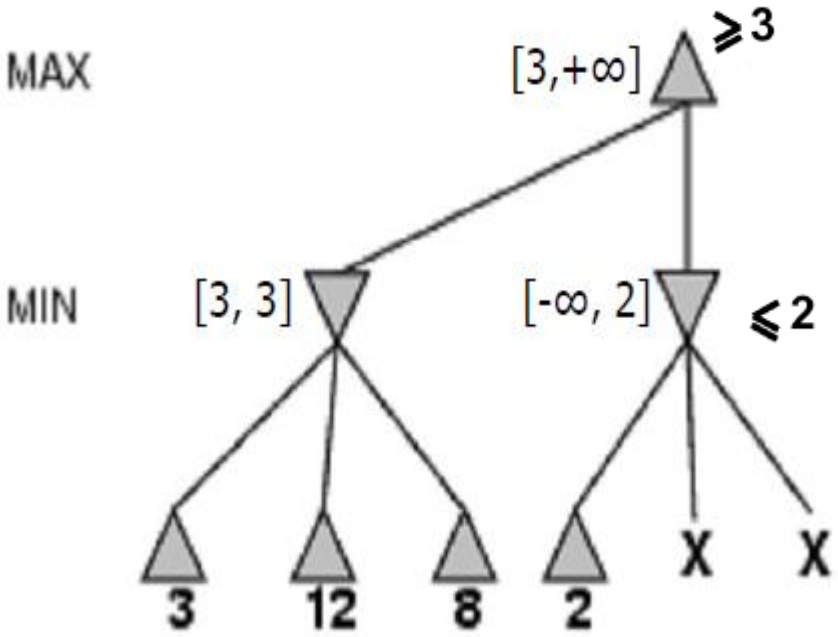


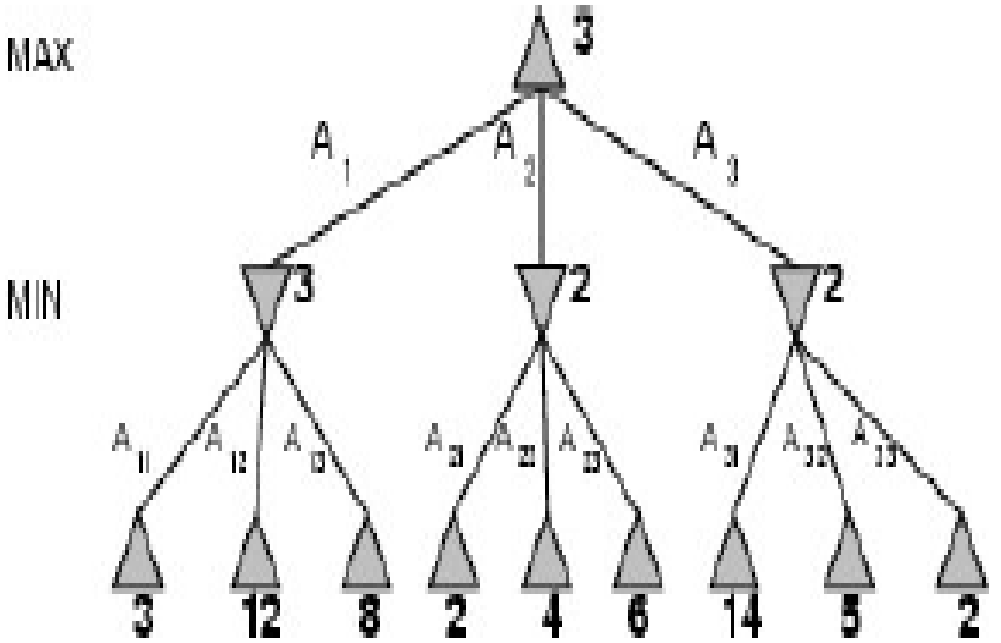
EXAMPLE



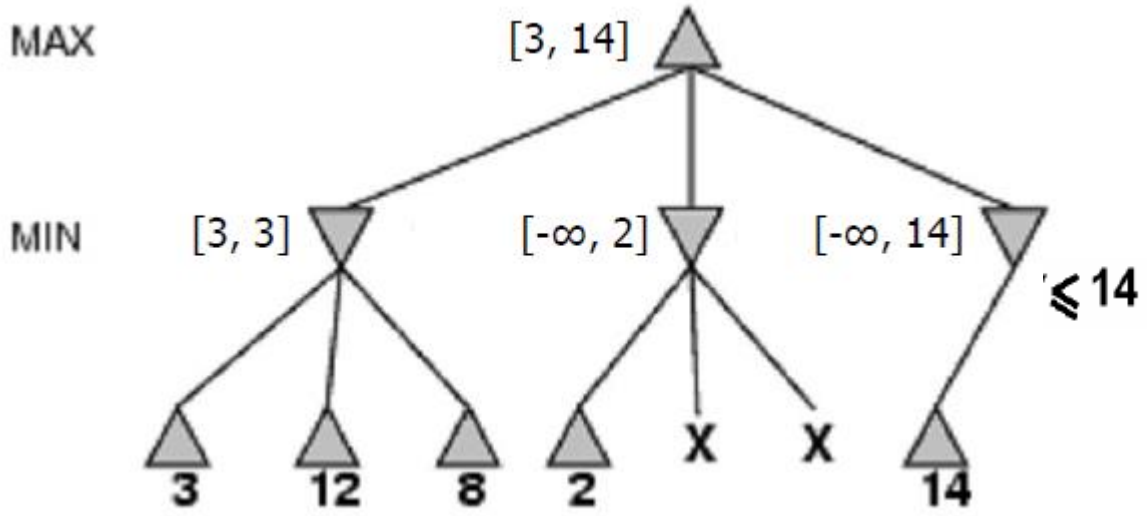


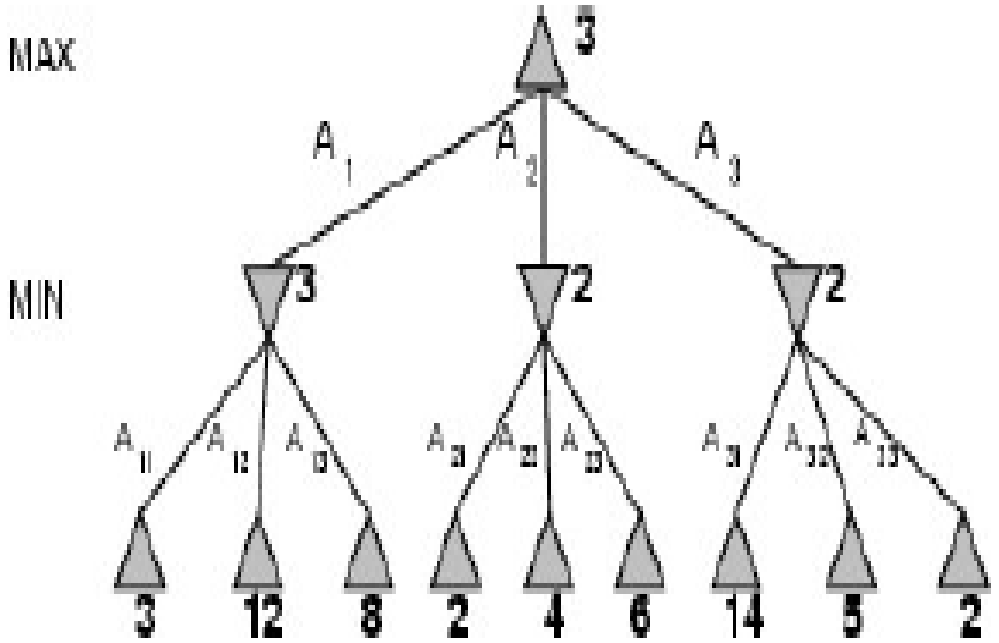
EXAMPLE



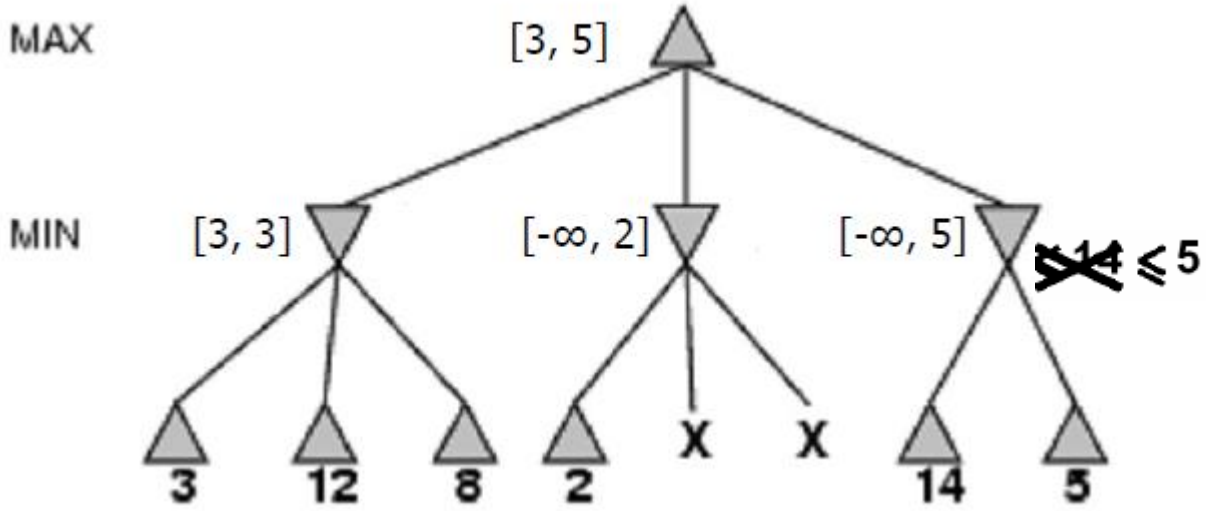


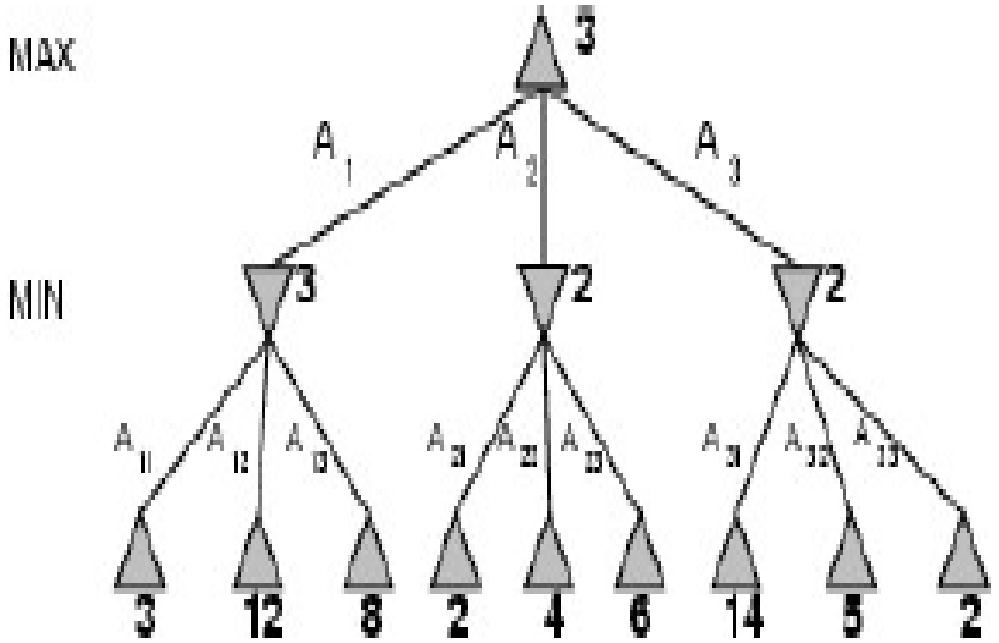
EXAMPLE



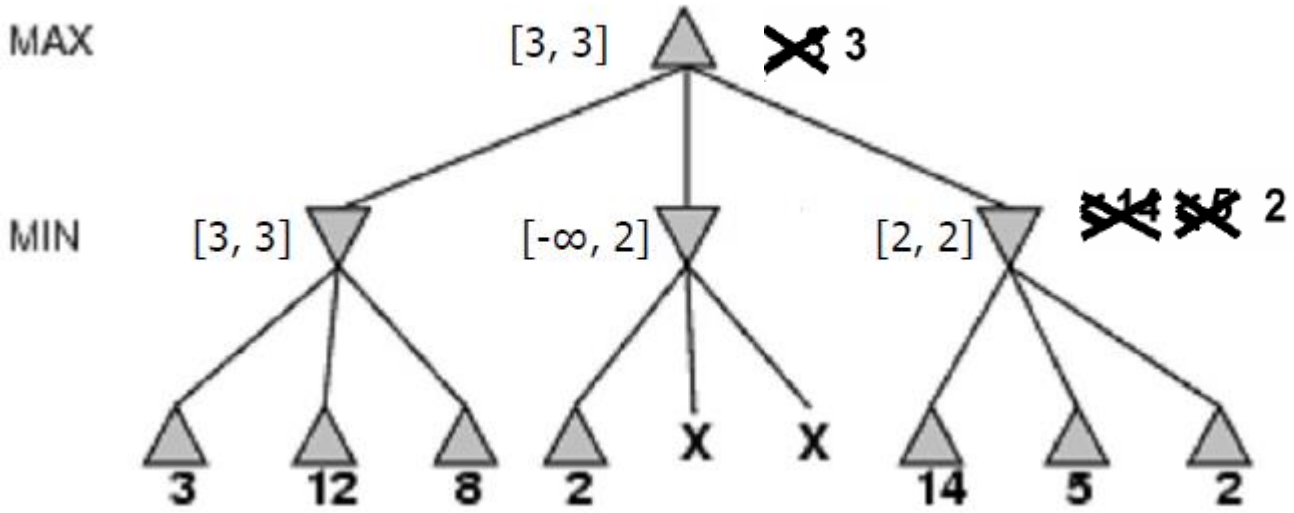


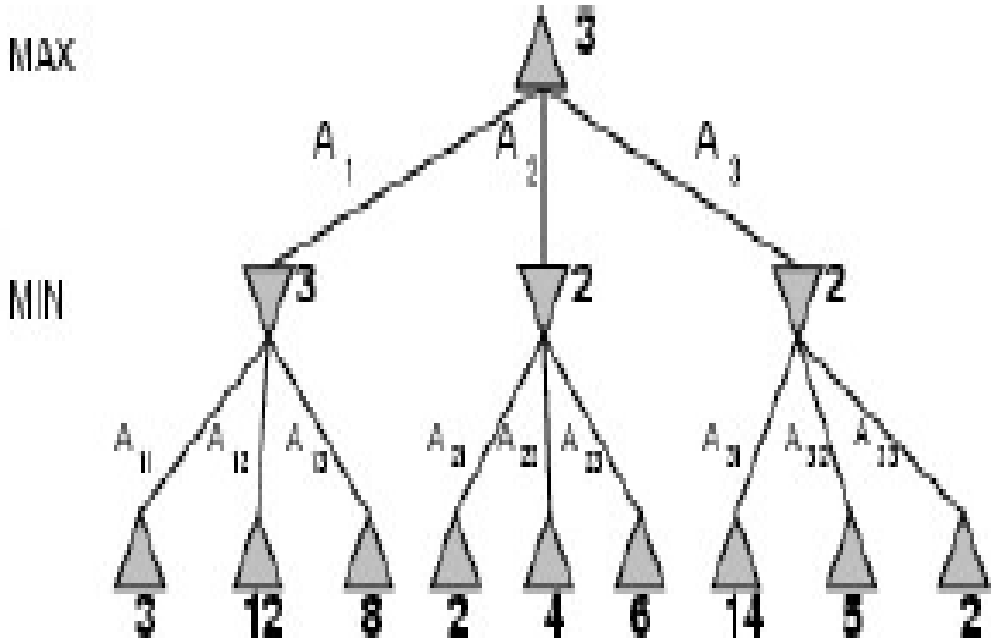
EXAMPLE



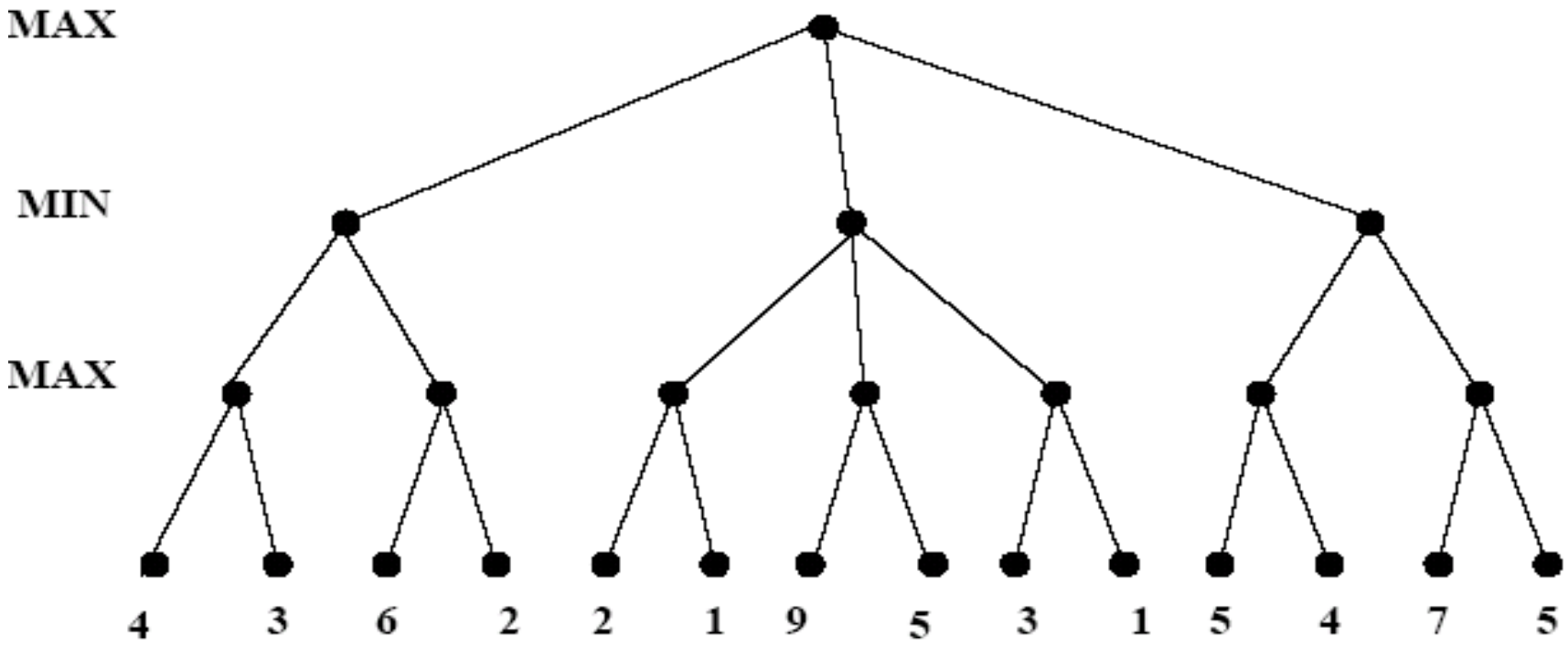


EXAMPLE

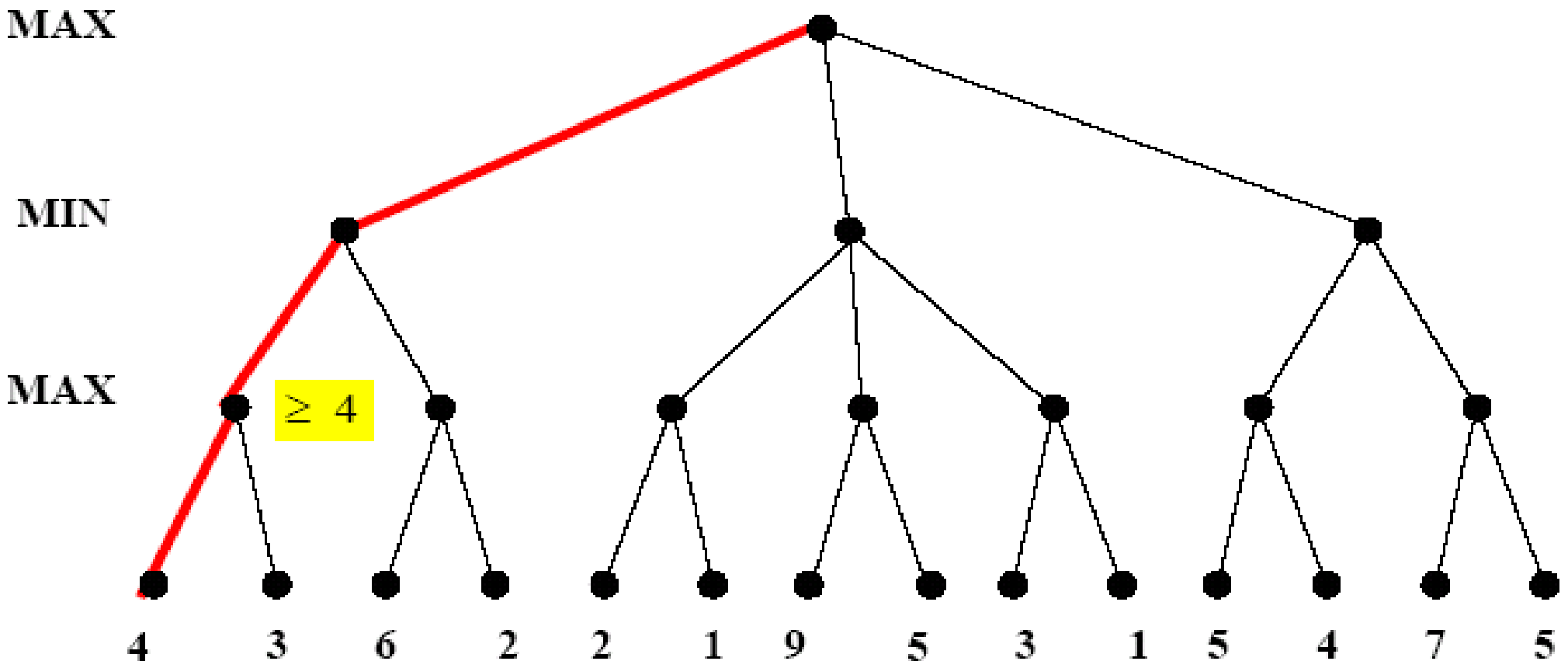

MAX
MIN
[3, 3]
3
[3, 3]
[-∞, 2]
[2, 2]
2
3
12
8
2
X
X
14
5
2


MAX
MIN
3
A1
A2
A3
3
2
2
3
12
8
2
4
6
14
5
2

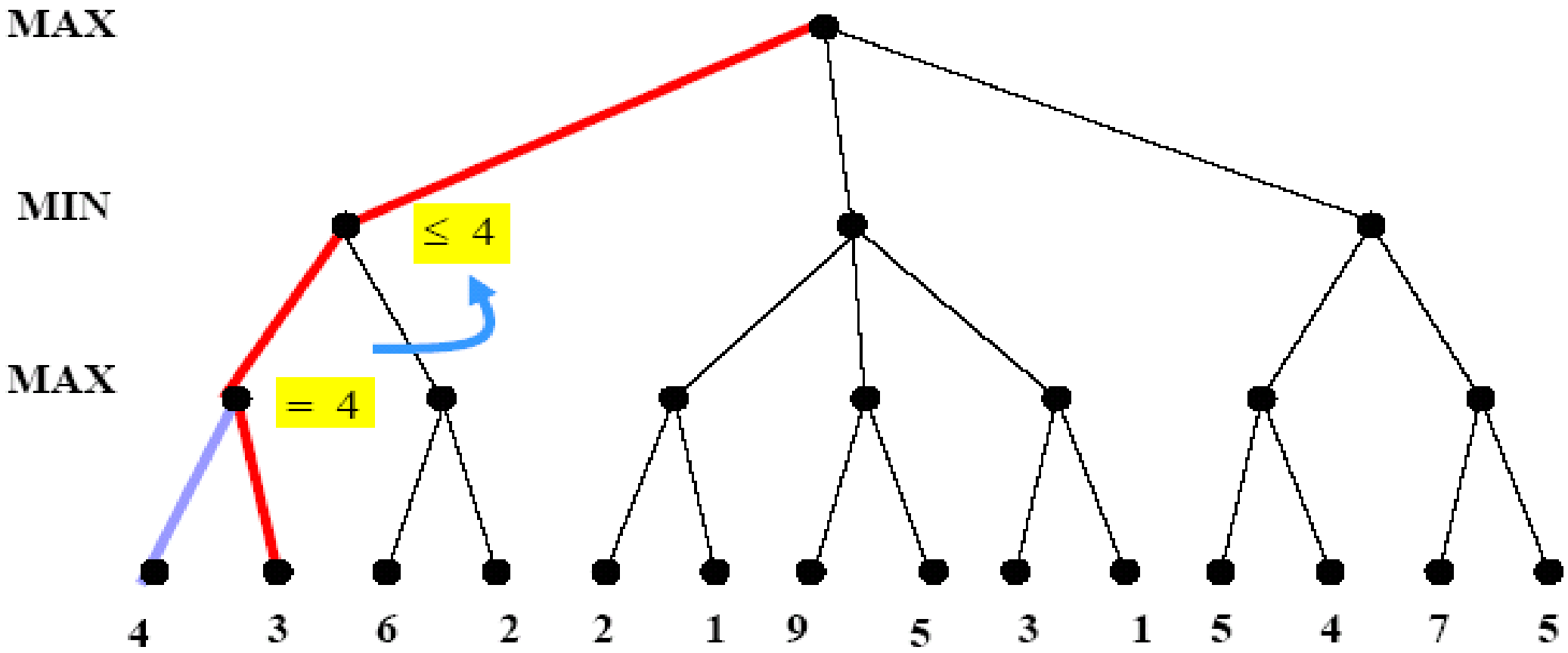
Alpha beta pruning. Example



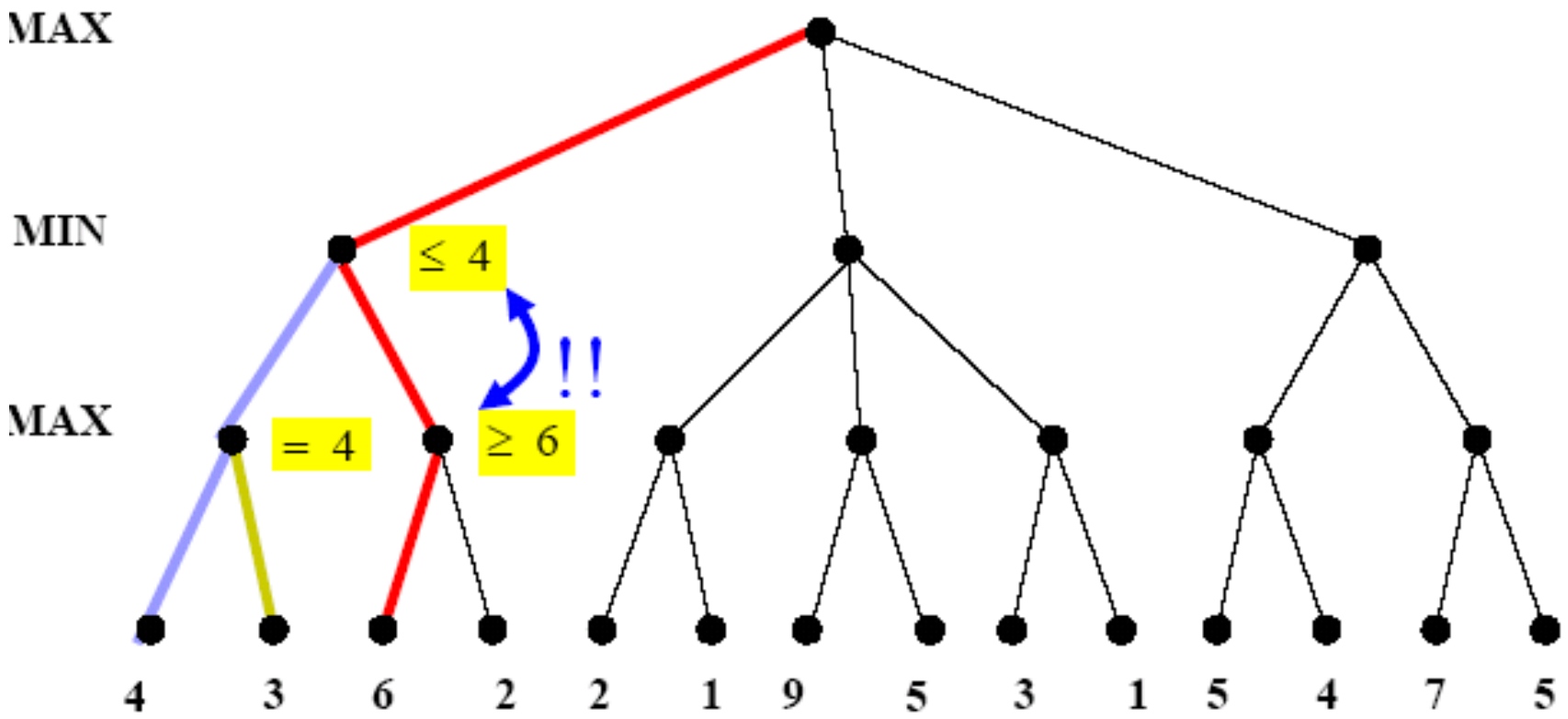
Alpha beta pruning. Example



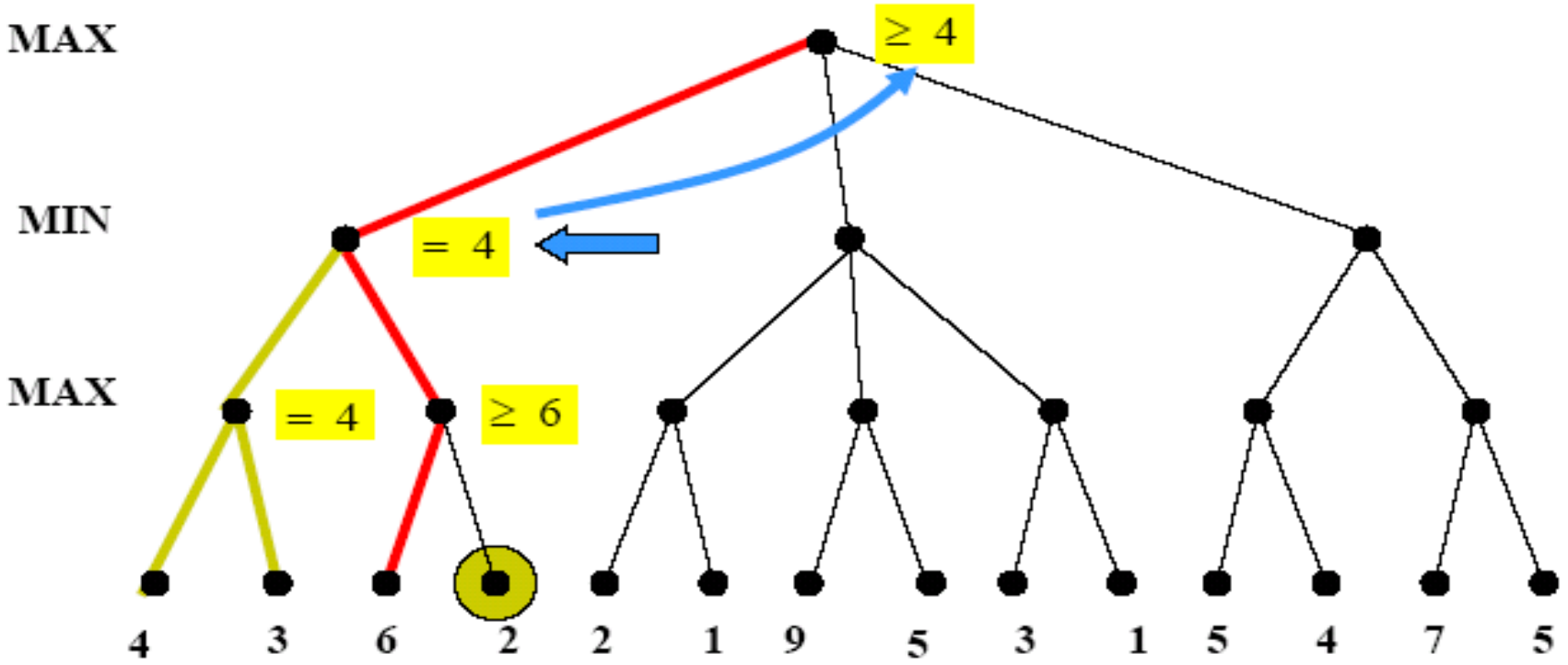
Alpha beta pruning. Example



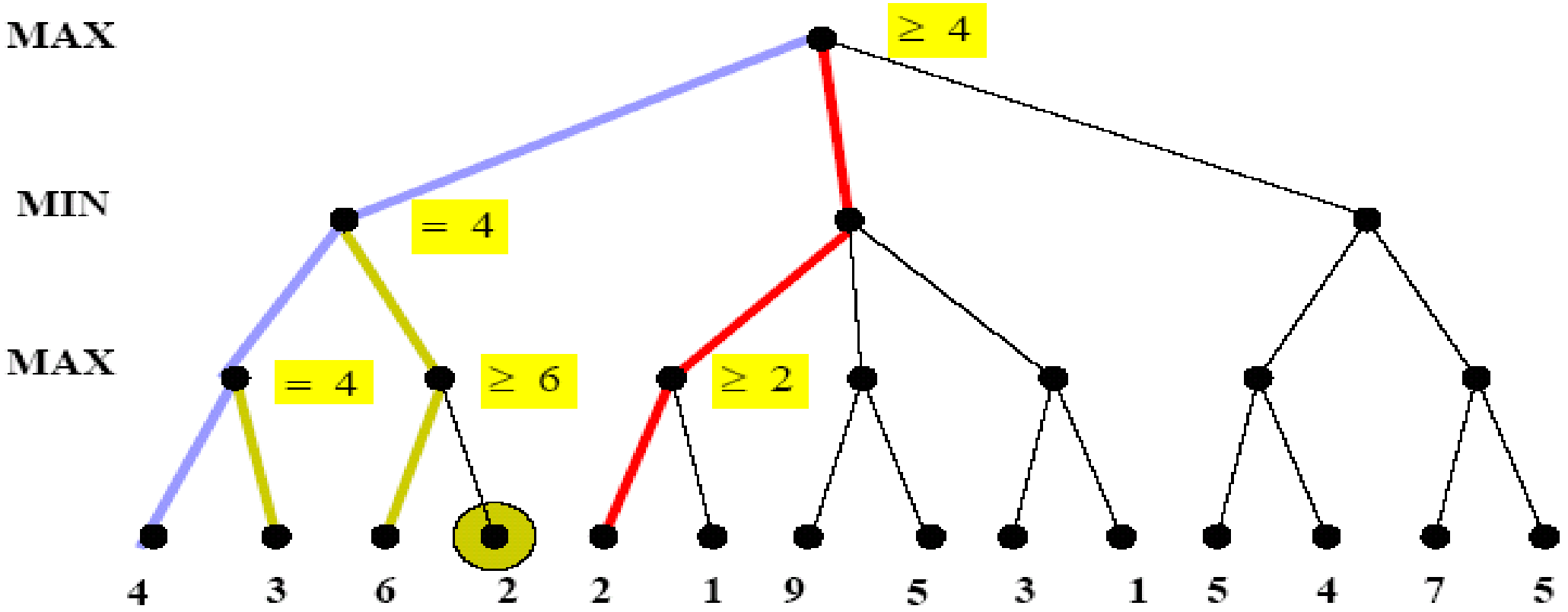
Alpha beta pruning. Example



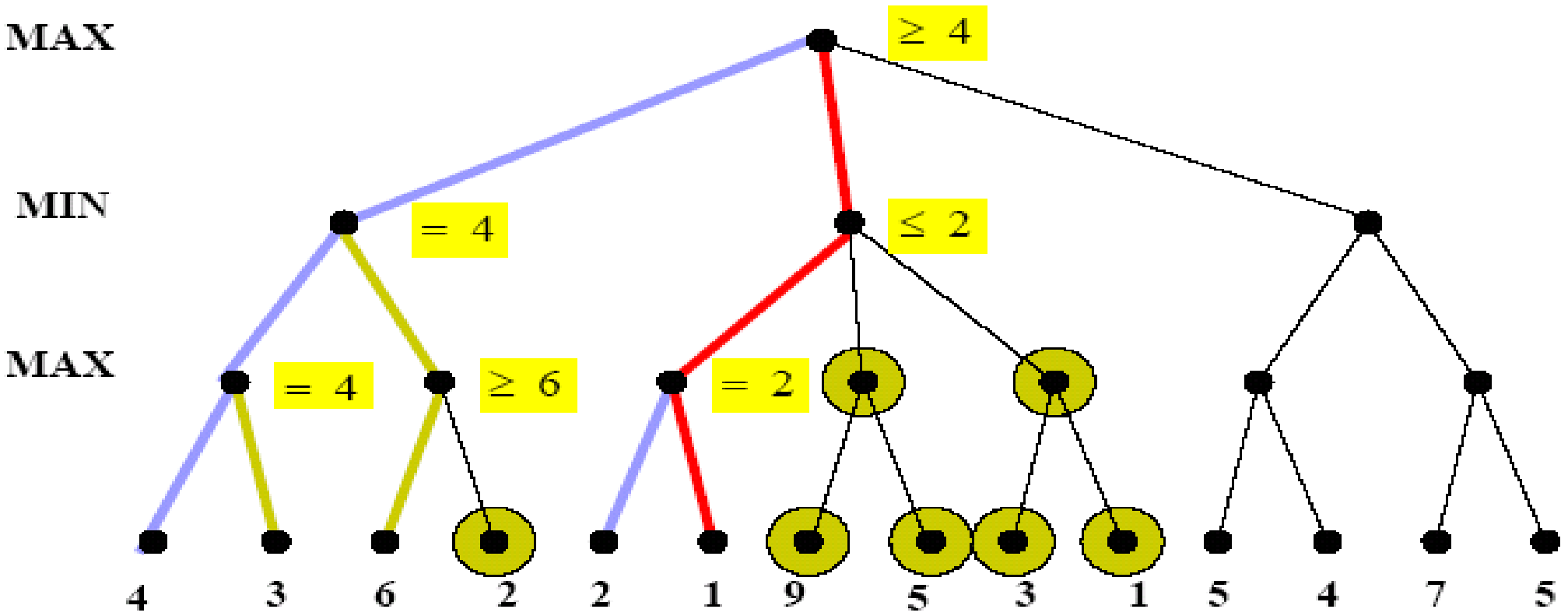
Alpha beta pruning. Example



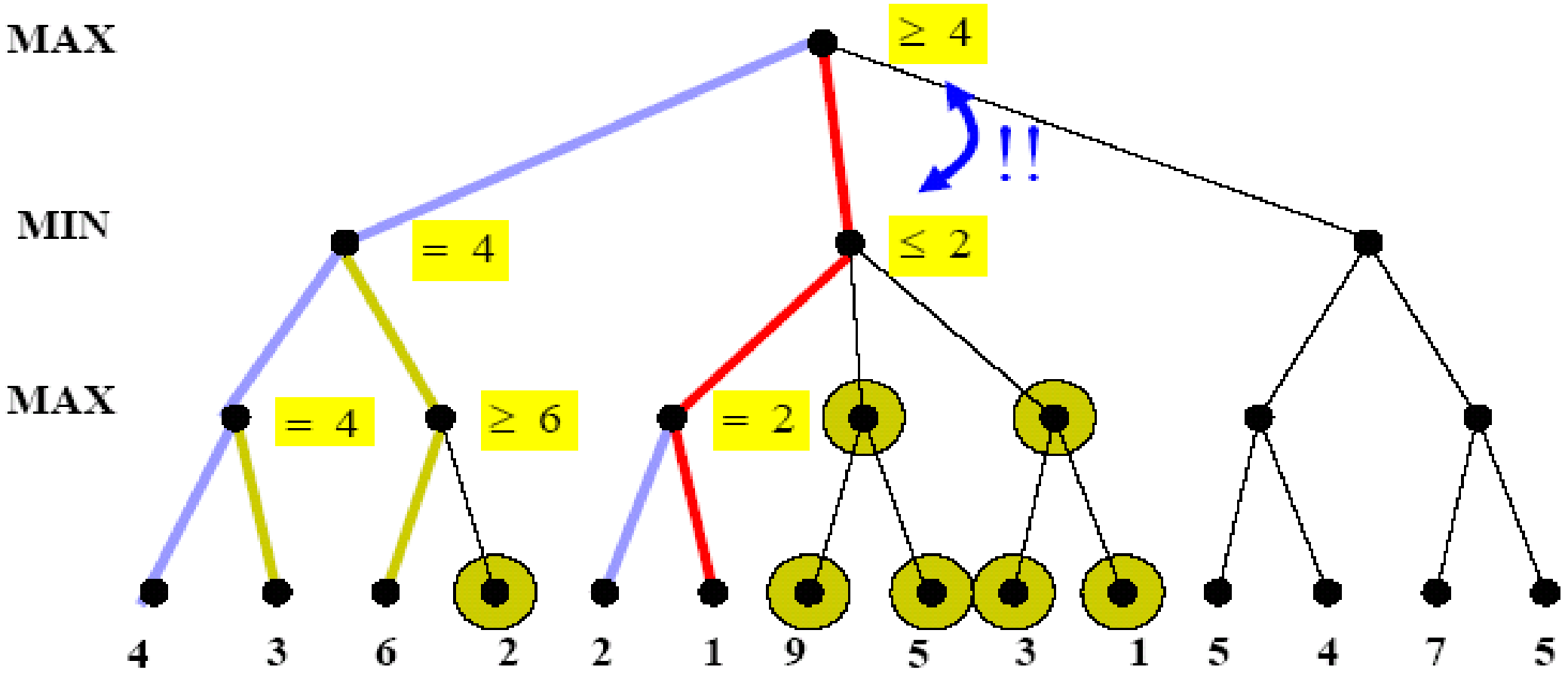
Alpha beta pruning. Example



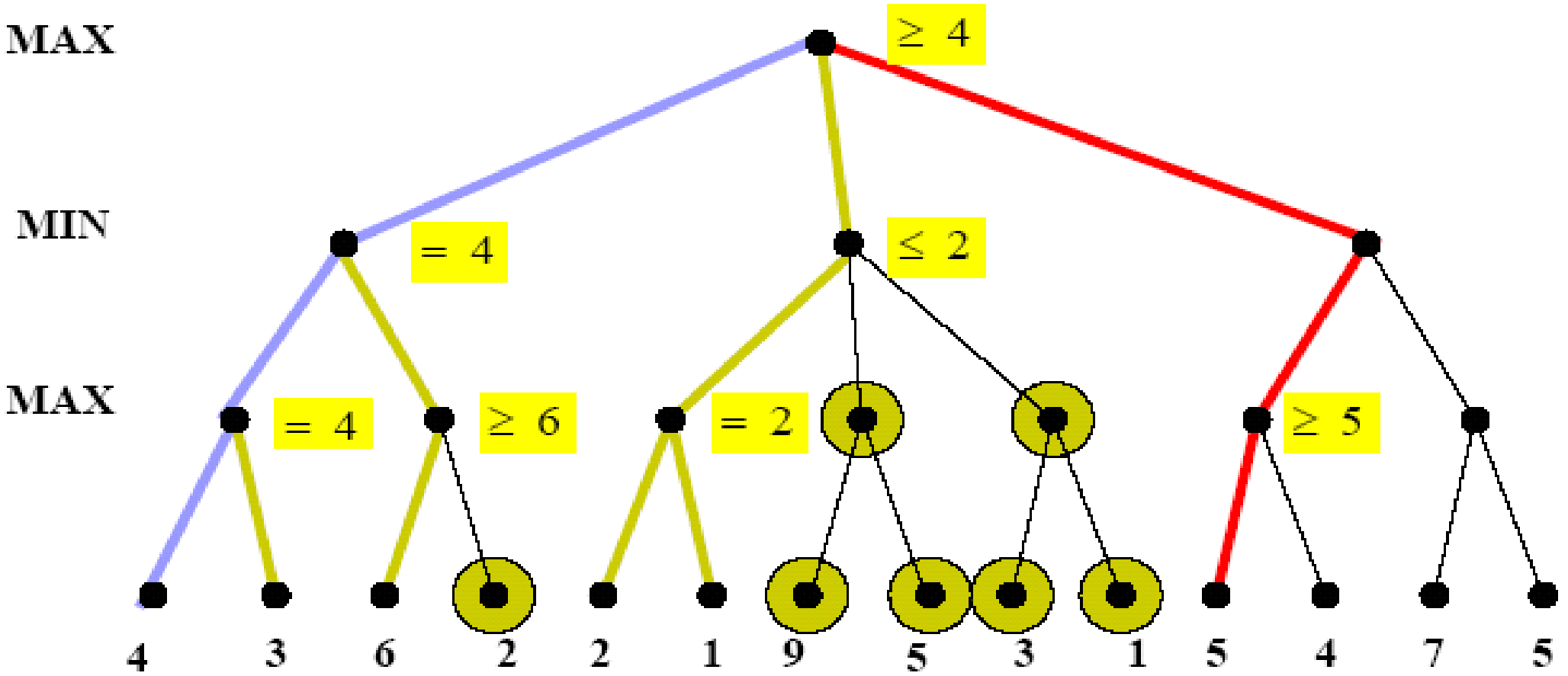
Alpha beta pruning. Example



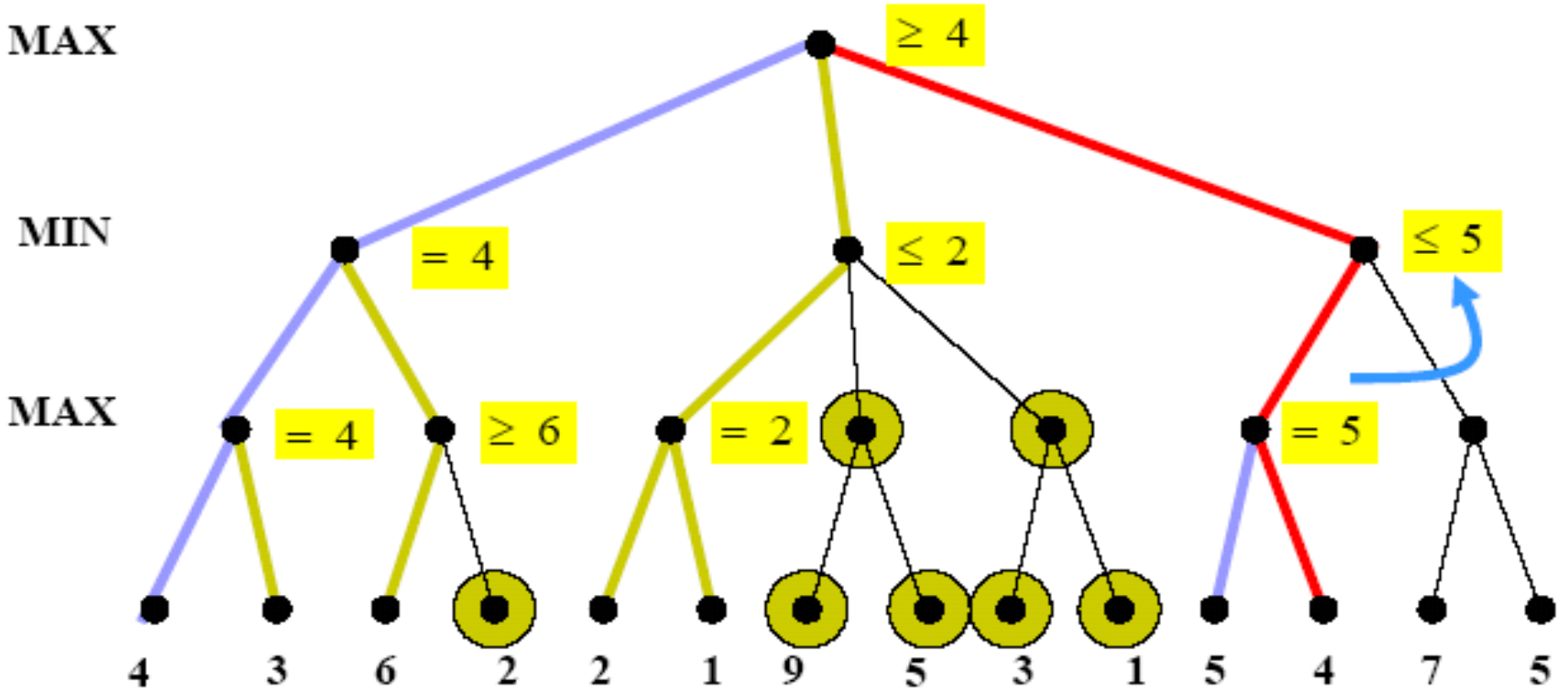
Alpha beta pruning. Example



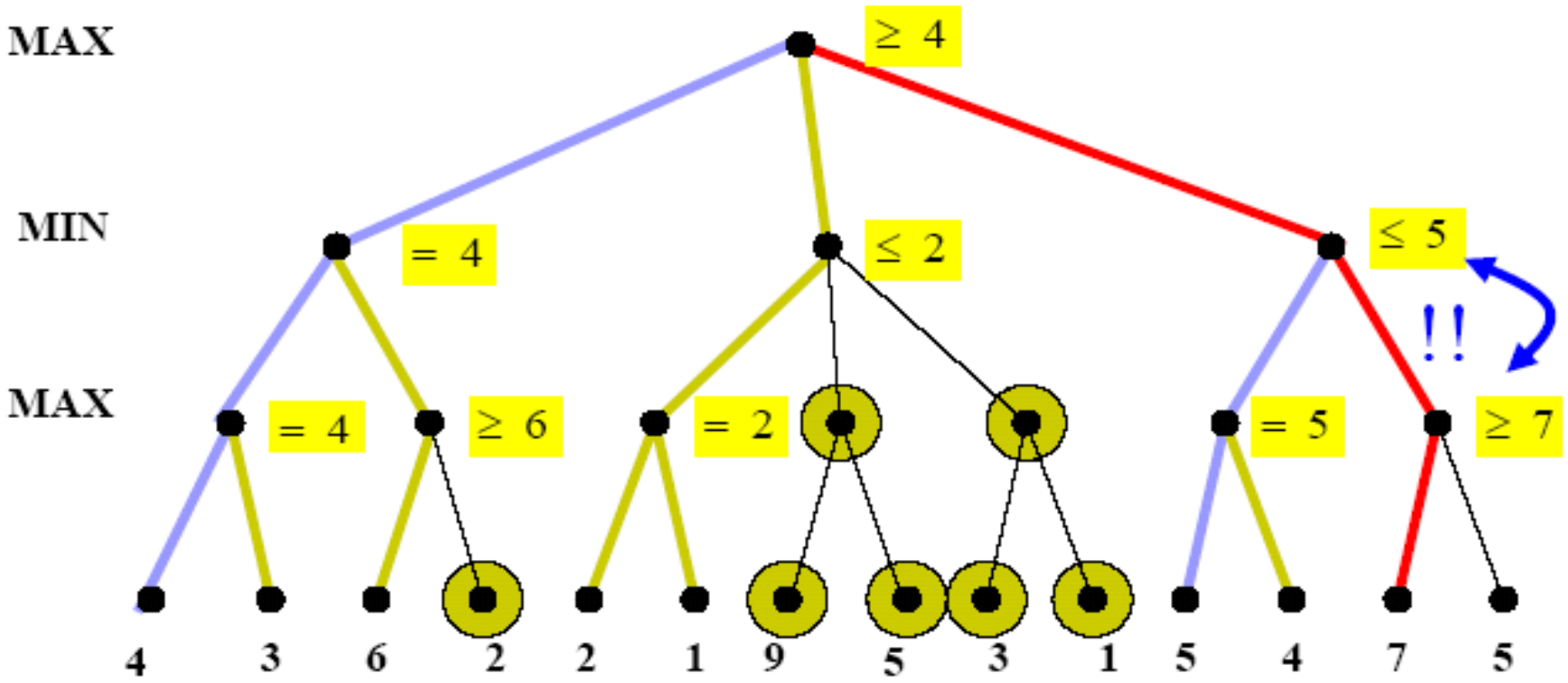
Alpha beta pruning. Example



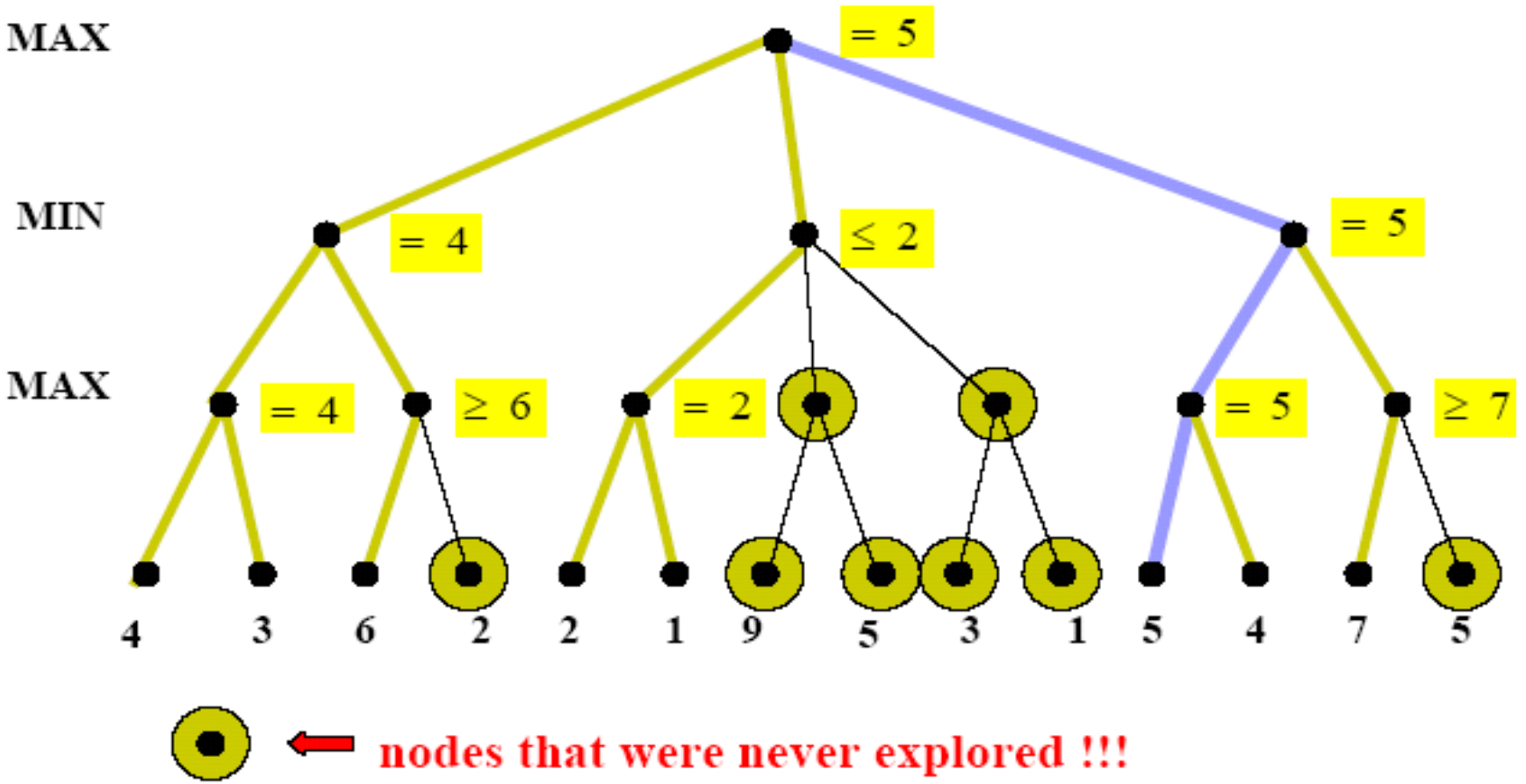
Alpha beta pruning. Example



Alpha beta pruning. Example



Alpha beta pruning. Example



CHALLENGE