Adversarial Search (Games)

COEN166

Artificial Intelligence

Adversarial Search - Games

- Multi-agent environment
- Impact of each agent on the others is significant

Typical Game Set Up

- Two players: MAX and MIN
- MAX moves first
- Then they take turns moving until the game is over
- At the end of the game
  - Points awarded to the winner
  - Penalties given to the loser

In other words

- The end-states have pay-off (utility function, or objective function)
  - MAX wants to maximize the pay-off
  - MIN wants to minimize the pay-off

Game as A Search Problem

- S_0: the **initial state**
  - Specifies how the game is set up at the start
- Player(s)
  - Defines which player has the move in the state
- Actions(s)
  - Returns the set of legal moves in a state
- Results(s, a): **transition model**
  - Defines the resulting state of taking action a at state s

Game as A Search Problem

- Terminal-Test(s): **terminal test**
  - True when the game is over, False otherwise
- Utility(s, p): **utility function**
  - Defines the pay-off for a game that ends in terminal state s for a player p
  - Chess
    - Outcome: win $(+1)$, loss (0), or draw $(+\frac{1}{2})$
  - Backgammon
    - Payoffs range from 0 to $+192$

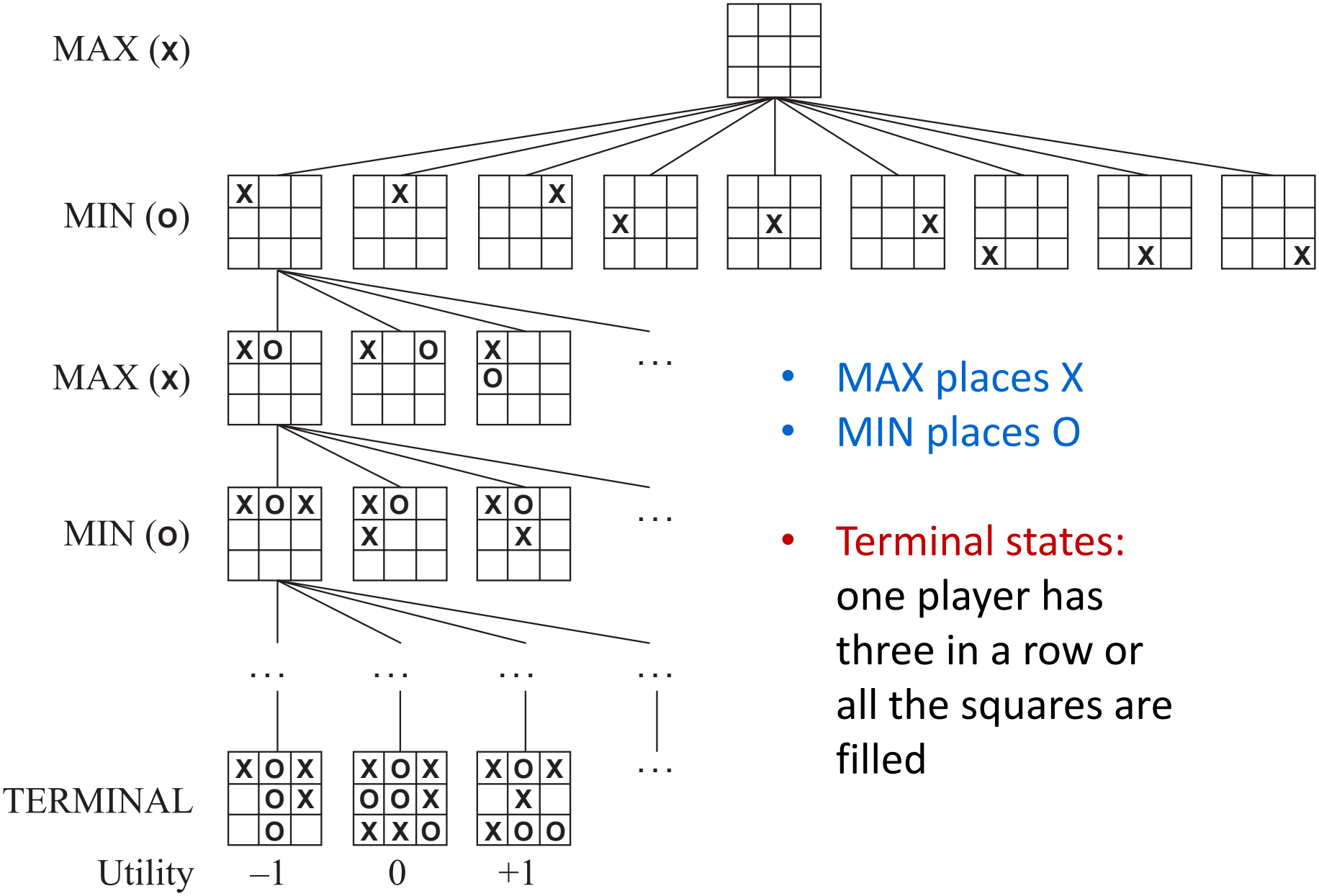
Game as A Search Problem

- Utility(s, p): **utility function**
  - Constant-sum game: the total payoff to all players is the same for every instance of the game
    - Chess: $0 + 1$, $1 + 0$, or $\frac{1}{2} + \frac{1}{2}$
  - Zero-sum game:
    - The utility values at the end of the game are always equal and opposite
    - One player wins a game in chess, the other player loses

Game Tree

- Defined by
  - S_0: the **initial state**
  - Actions(s)
  - Results(s, a): **transition model**
- A tree where the nodes are game states, and the edges are moves/actions

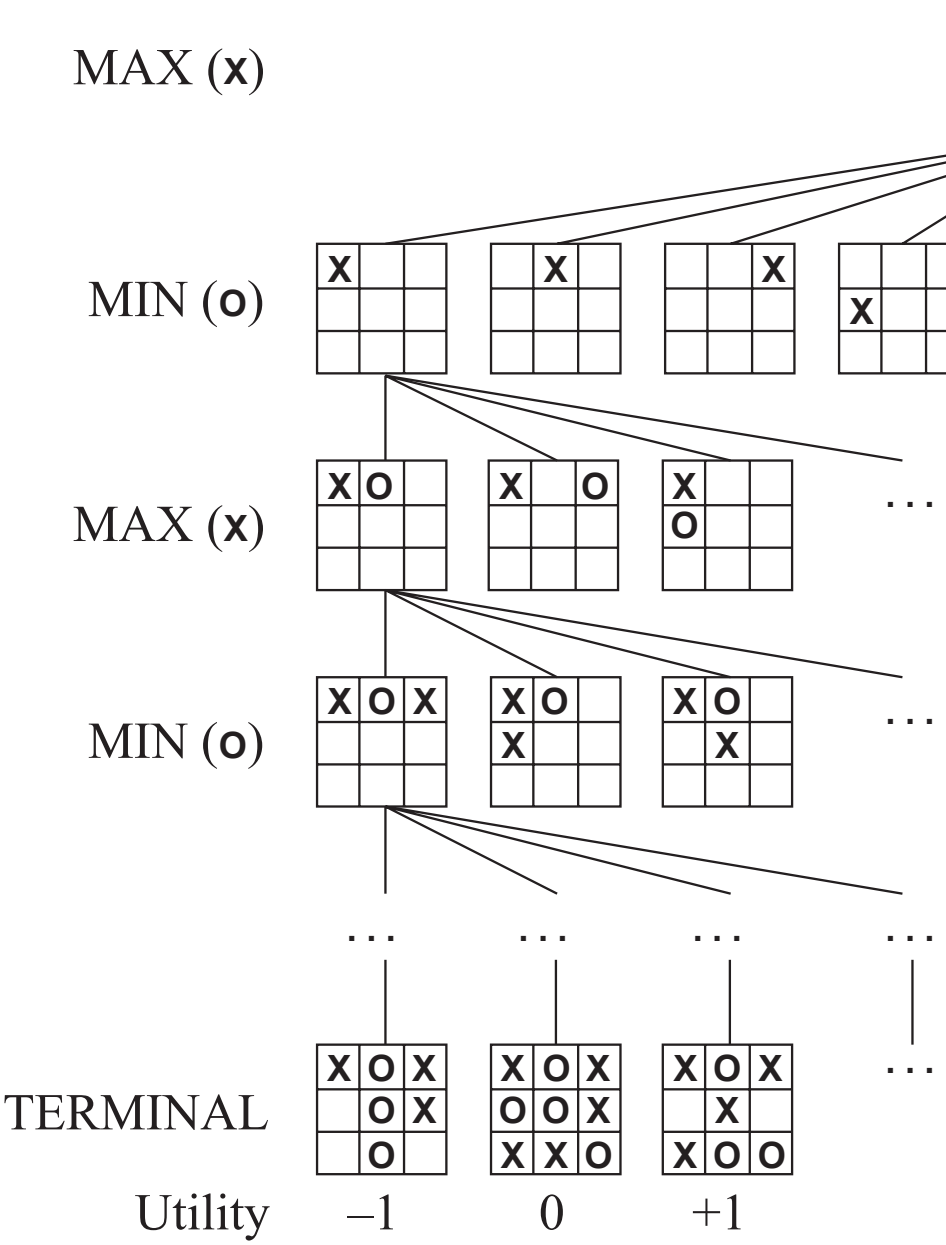
Tic-tac-toe



- MAX places X
- MIN places O

- Terminal states: one player has three in a row or all the squares are filled

Tic-tac-toe



- The number on each leaf node
  - the utility value of the terminal state
- High values: good for MAX, bad for MIN

Optimal Decisions in Games

- The optimal solution
  - A sequence of actions leading to a goal state
    - A terminal state that is a win
- Given a game tree, the optimal strategy can be determined from the **minimax values** of the nodes

- For node n, the minimax value is denoted as

$$\text{Minimax}(n)$$

Minimax(n)

- The utility function value of being in the corresponding state, assuming that both players play optimally from there to the end of the game
  - MAX prefers to move to a state of maximum value
  - MIN prefers to move to a state of minimum value

$$\text{Minimax}(s) = \begin{cases} \text{Utility}(s), & \text{if Terminal} - \text{Test}(s) = true \\ \max_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s,a)) & \text{if Player}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{Minimax}(\text{Result}(s,a)) & \text{if Player}(s) = \text{MIN} \end{cases}$$

s: current state

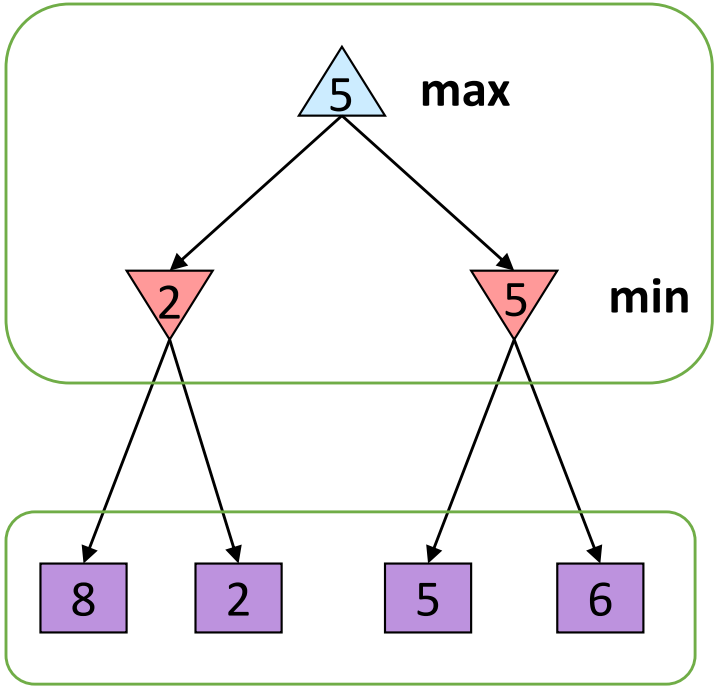
s': next state, obtained from **Result(s,a)**, a is any possible action that can be taken at s

If Player(s) = MAX, then Minimax(s) = max(Minimax(s'))

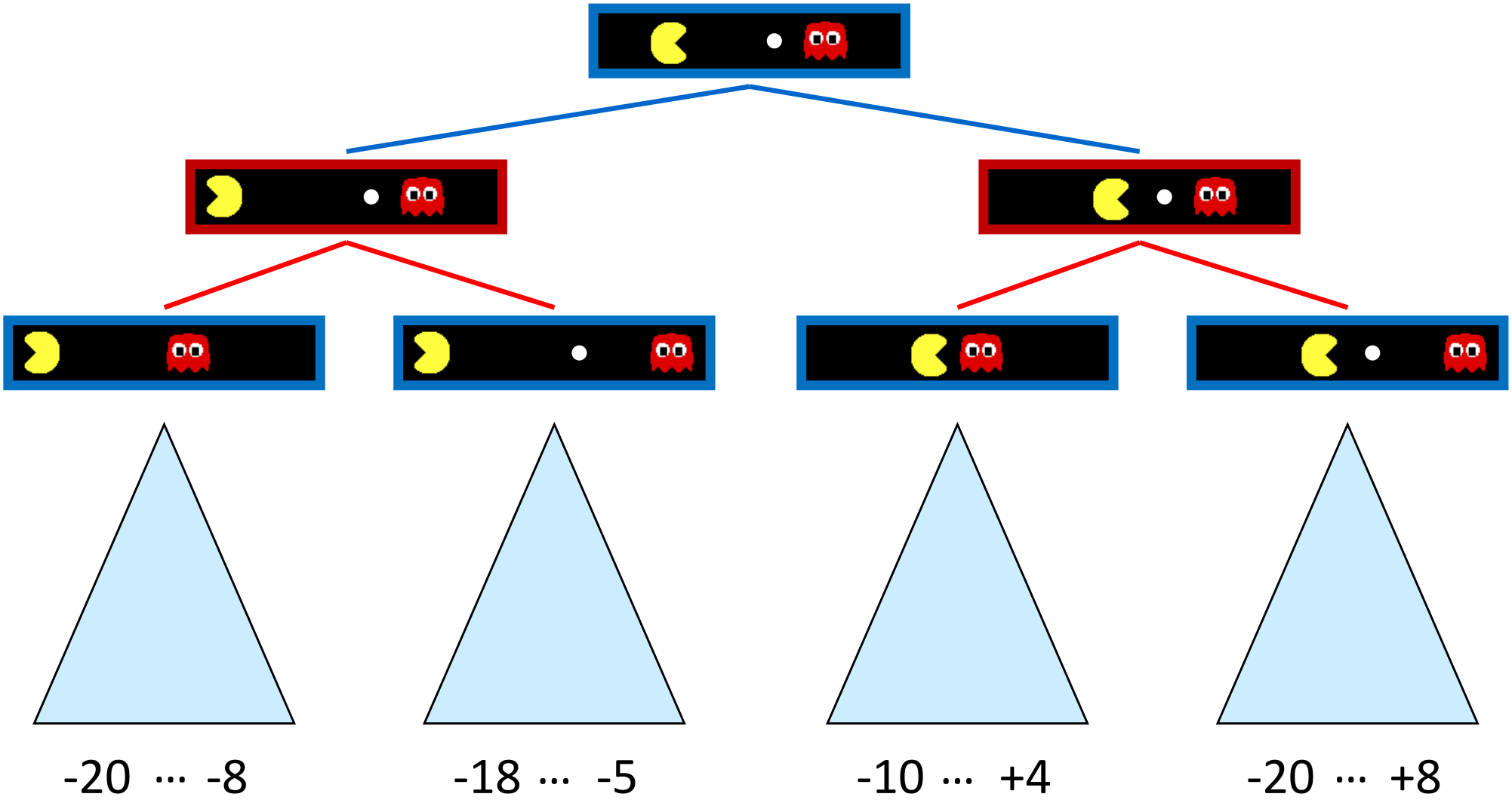
If Player(s) = MIN, then Minimax(s) = min(Minimax(s'))

Minimax Algorithm

- Players alternate turns
- The minimax values are computed recursively
  - Propagated from the leaf nodes to upper layers



Adversarial Game Trees



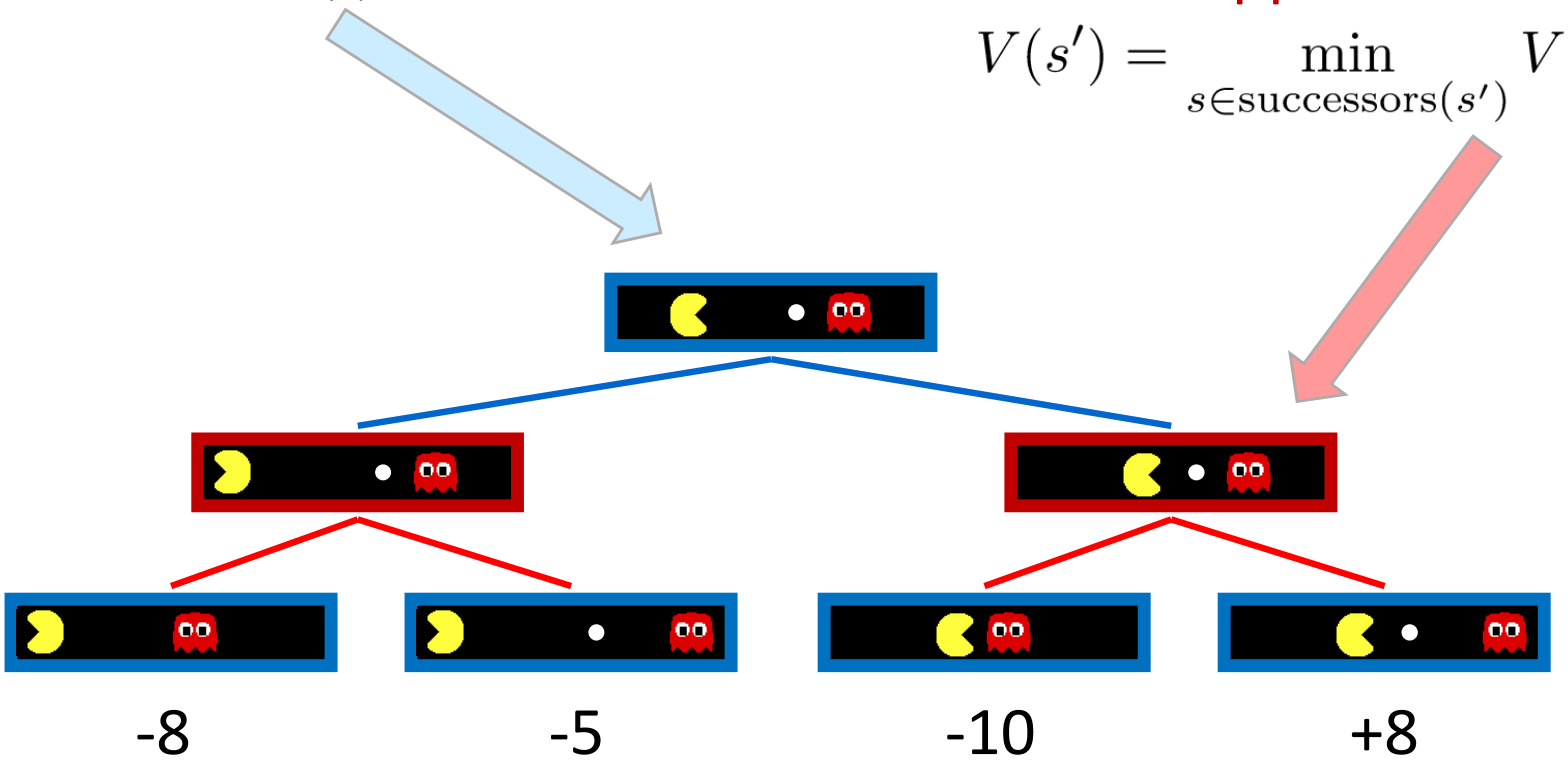
Minimax Values

States Under Agent's Control:

$$V(s) = \max_{s' \in \text{successors}(s)} V(s')$$

States Under Opponent's Control:

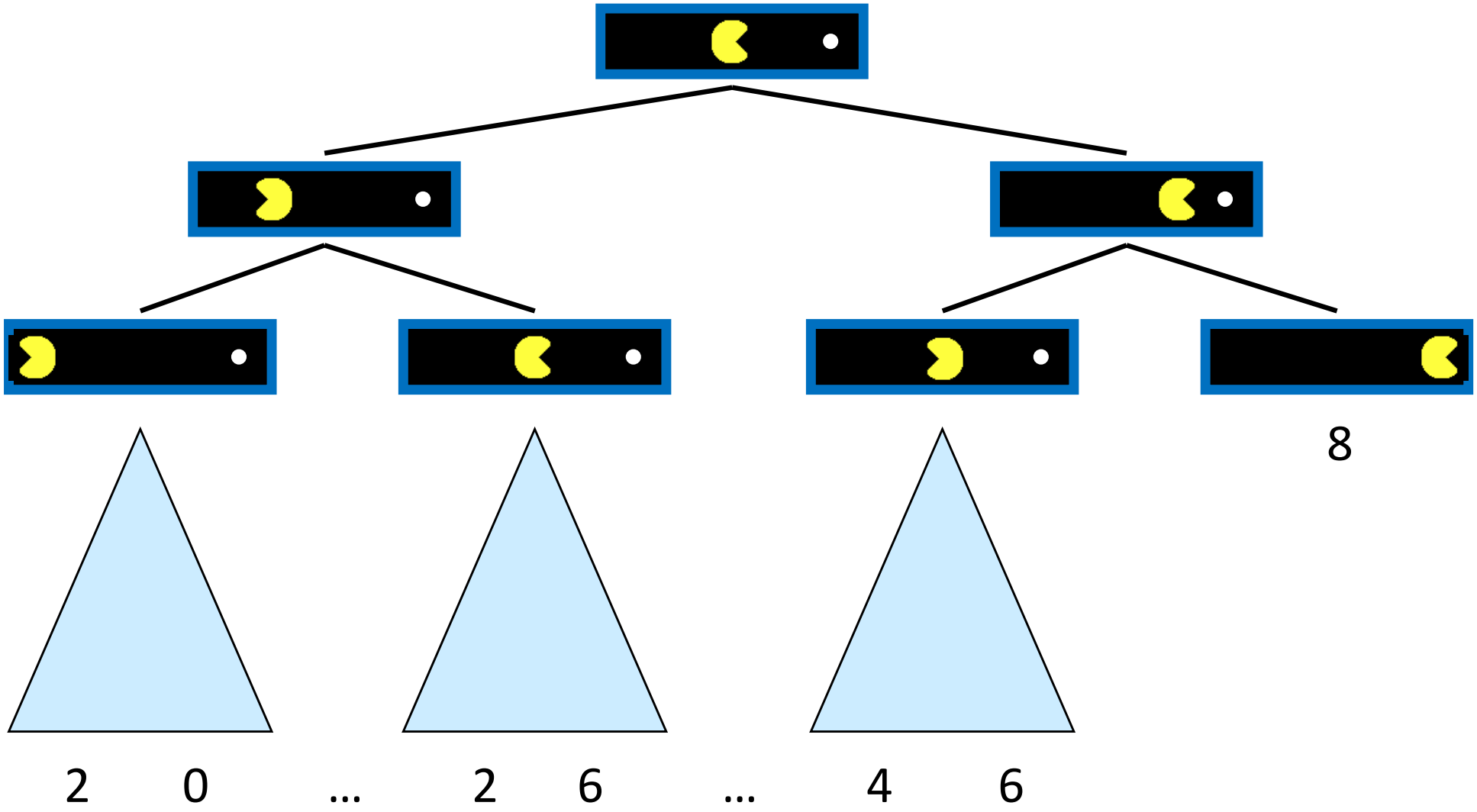
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

$$V(s) = \text{known}$$

Single-Agent Trees

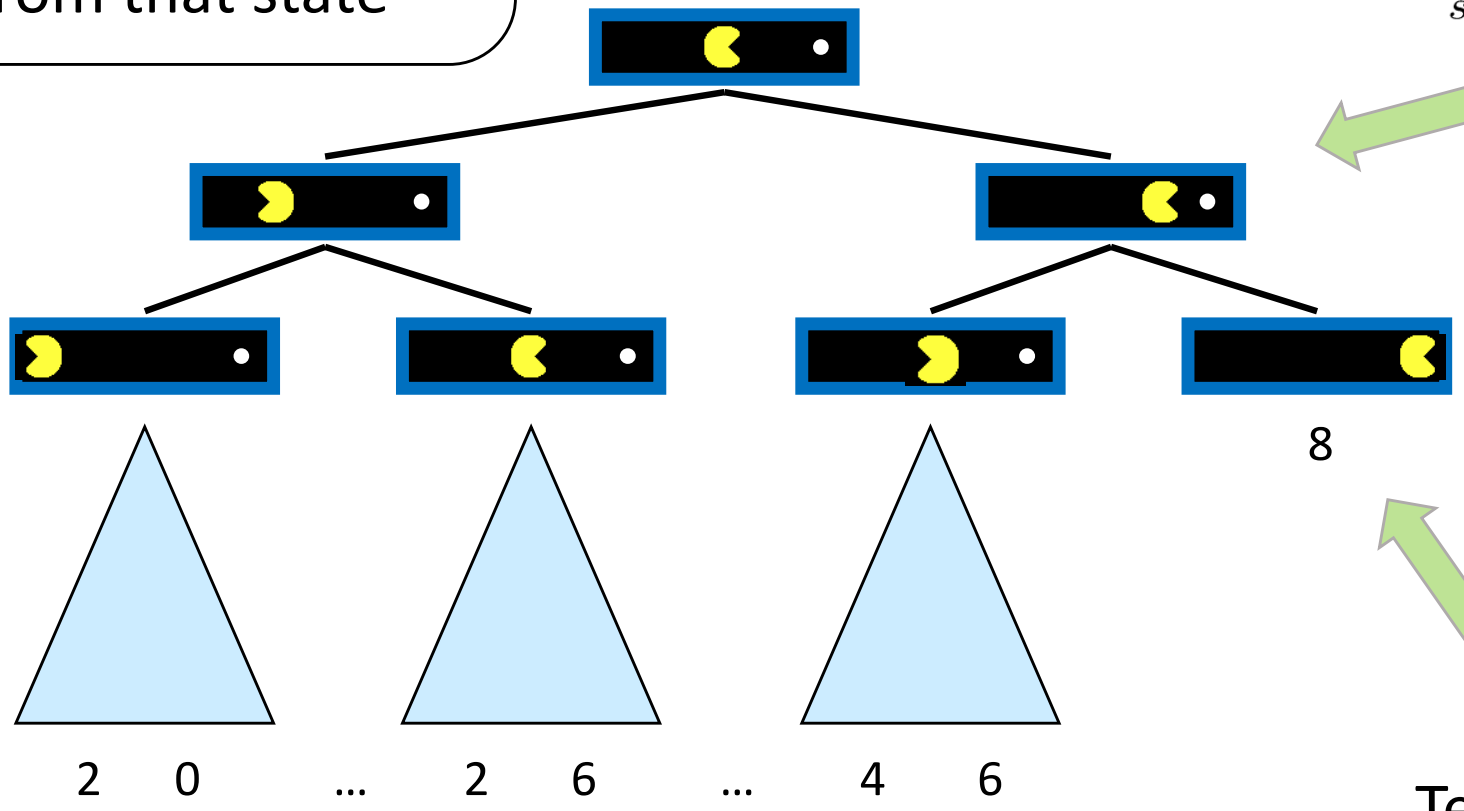


Value of a State

Value of a state:
The best achievable outcome (utility) from that state

Non-Terminal States:

$$V(s) = \max_{s' \in \text{children}(s)} V(s')$$



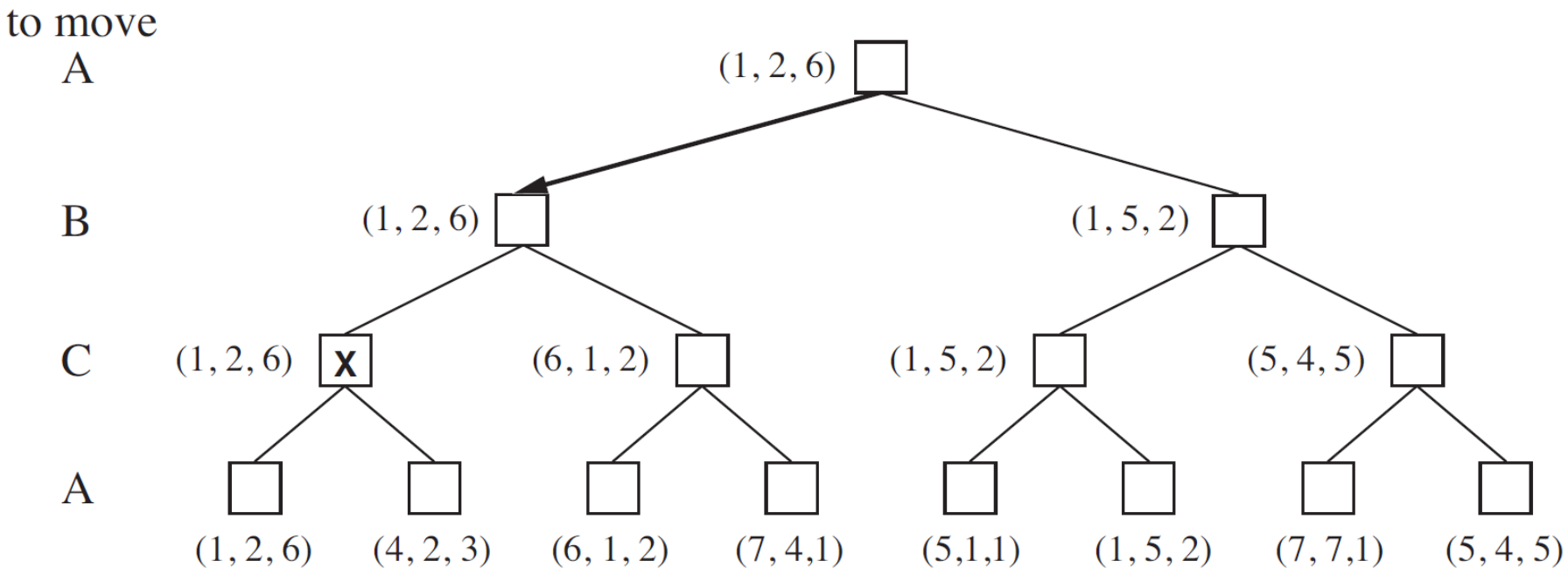
Terminal States:

$$V(s) = \text{known}$$

Multiplayer Games

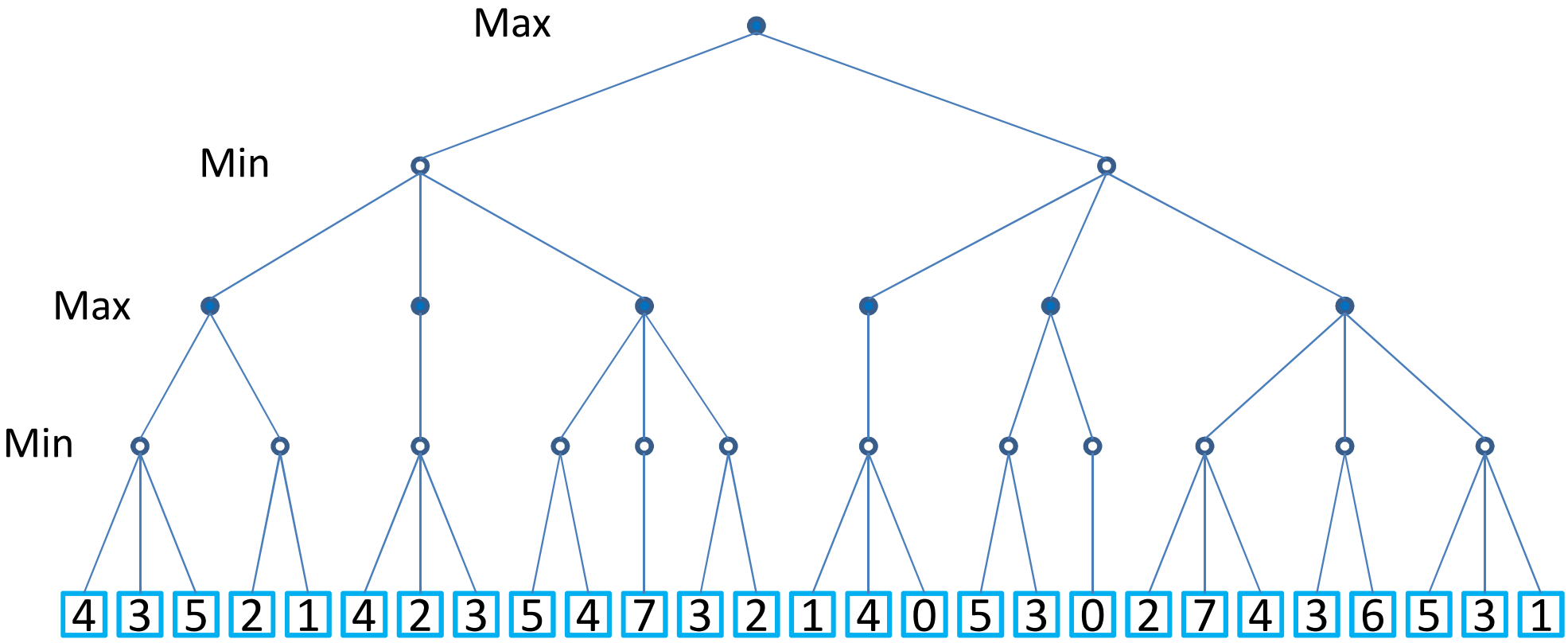
- Extend the minimax idea to multiplayer games
  - Three-player game
  - Players A, B, C
  - A vector of values <v_A, v_B, v_C>
  - For terminal states, each element in this vector gives the utility of the state from each player's viewpoint

Multiplayer Games



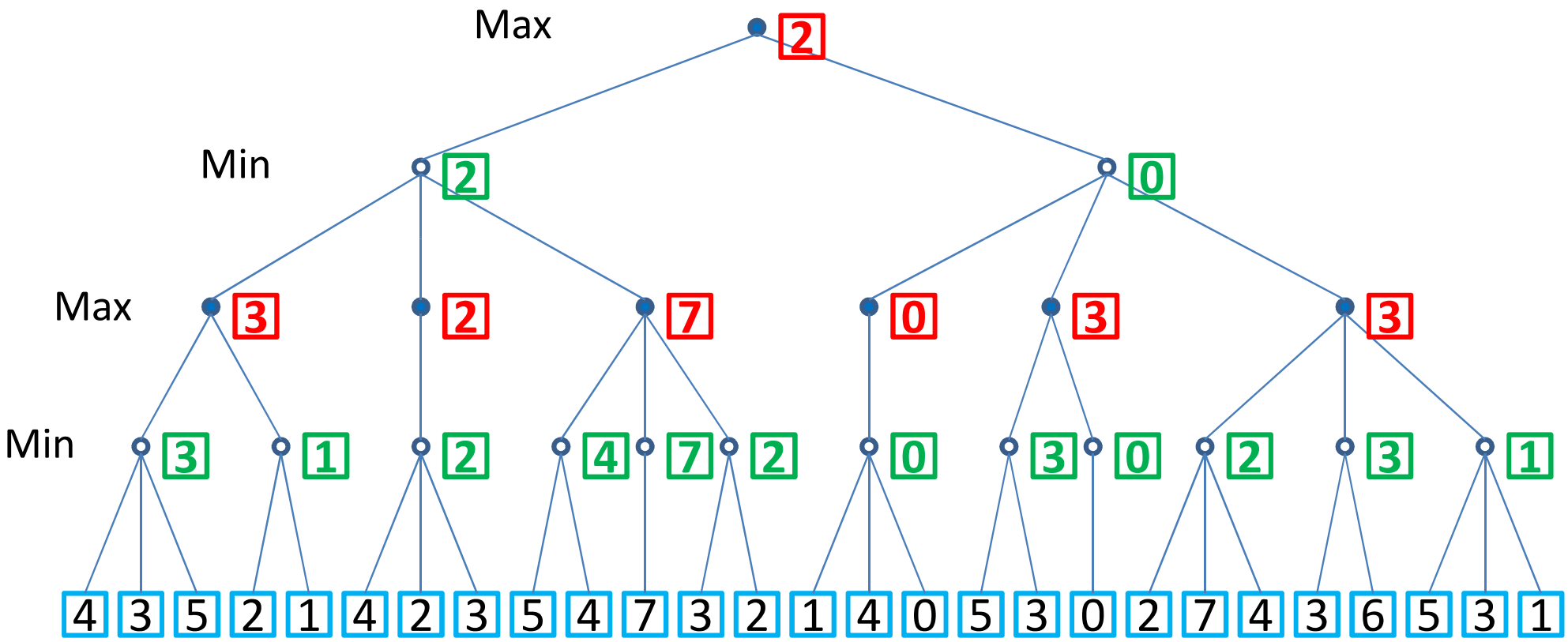
The minimax value of each node is a 3-tuple

Minimax Algorithm Example



The minimax value of each node: determined by passing the values recursively from the terminal-states to the upper layers

Minimax Algorithm Example

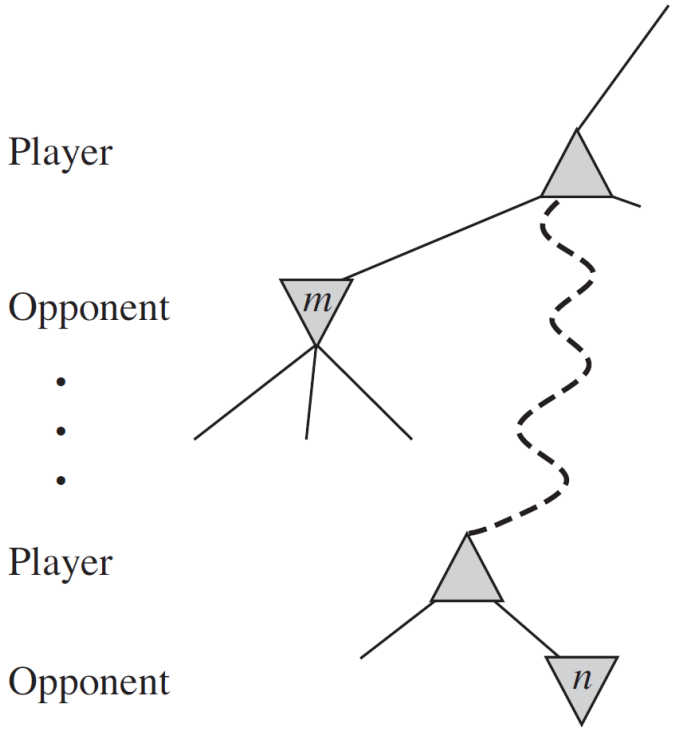


The minimax value of each node: determined by passing the values recursively from the terminal-states to the upper layers

Alpha-Beta Pruning

- Evaluating the whole game tree is cumbersome

- Pruning: No need to evaluate values that cannot change the maximum/minimum score the player/opponent can get



If m is better than n for Player, we will never get to n in play.

The General Case

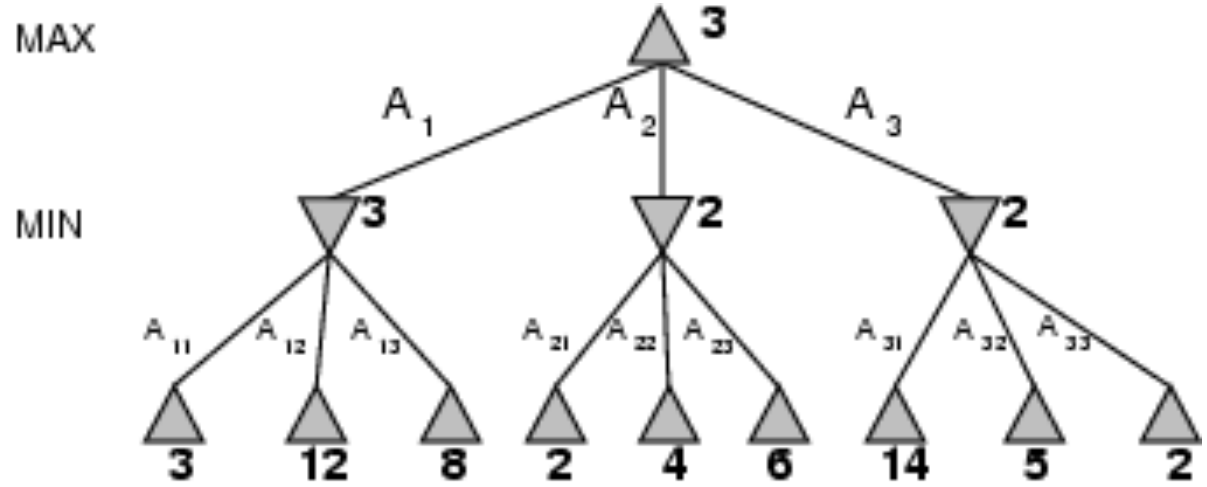
- For each node, two values are assigned

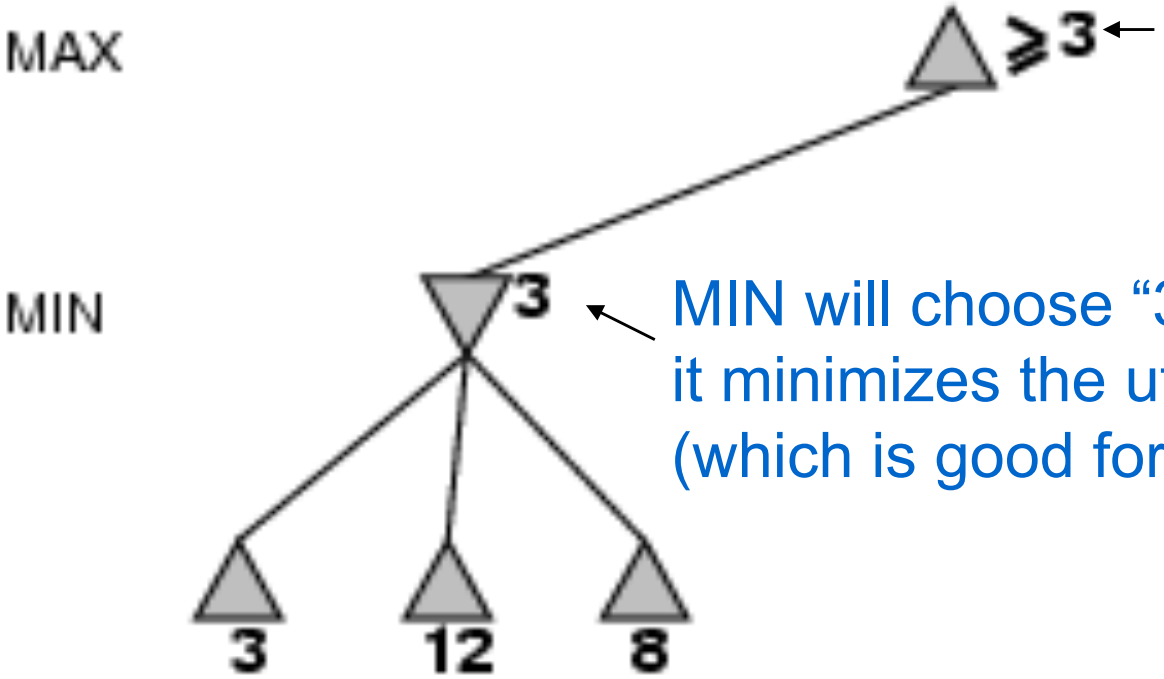
$$[\alpha, \beta]$$

- α = the lower bound of the minimax value for that node
- β = the upper bound of the minimax value for that node

- The α-β pruning algorithm
  - Keeps updating the lower and/or upper bound of a node, until no further inference is needed

α-β Pruning Example

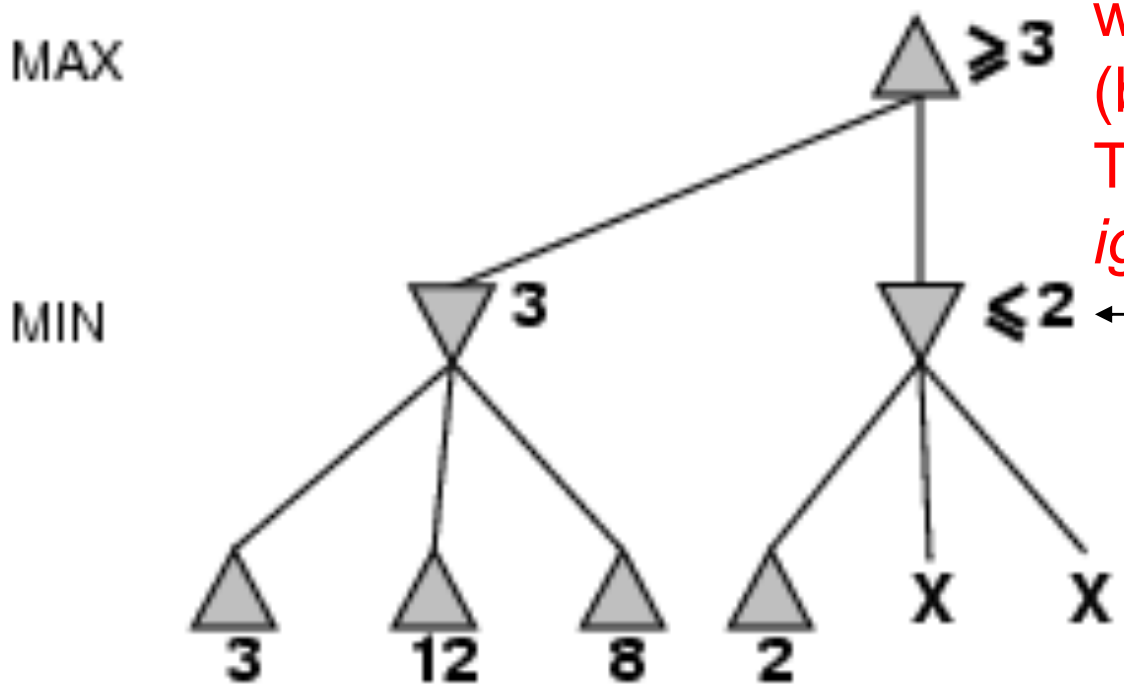




MAX knows that it can at least get "3" by playing this branch

MIN will choose "3", because it minimizes the utility (which is good for MIN)

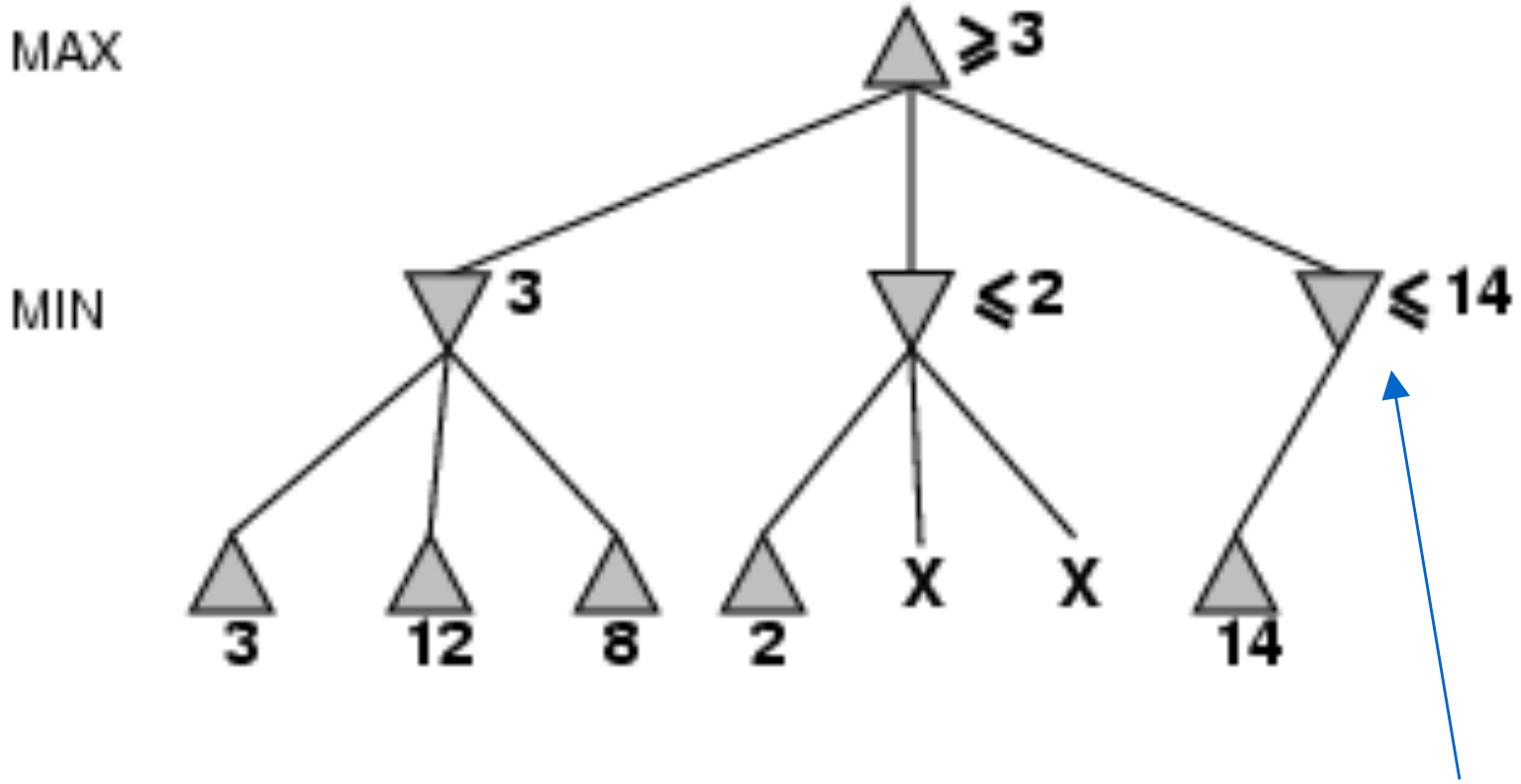
α-β Pruning Example



MAX knows that the new branch will never be better (bigger) than 2. This branch can be *ignore*d.

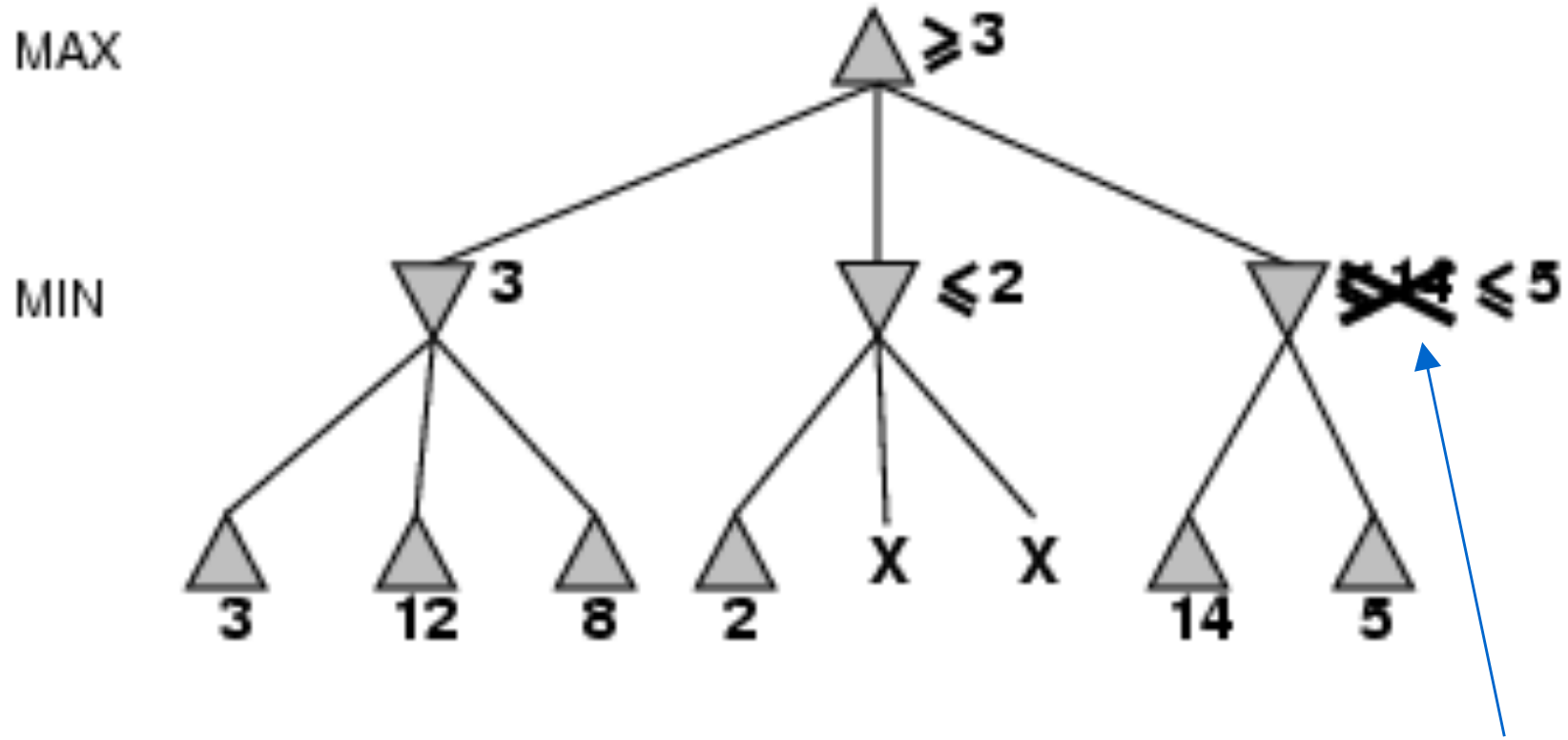
← MIN can certainly do as good as 2, but maybe better (i.e., smaller)

α-β Pruning Example



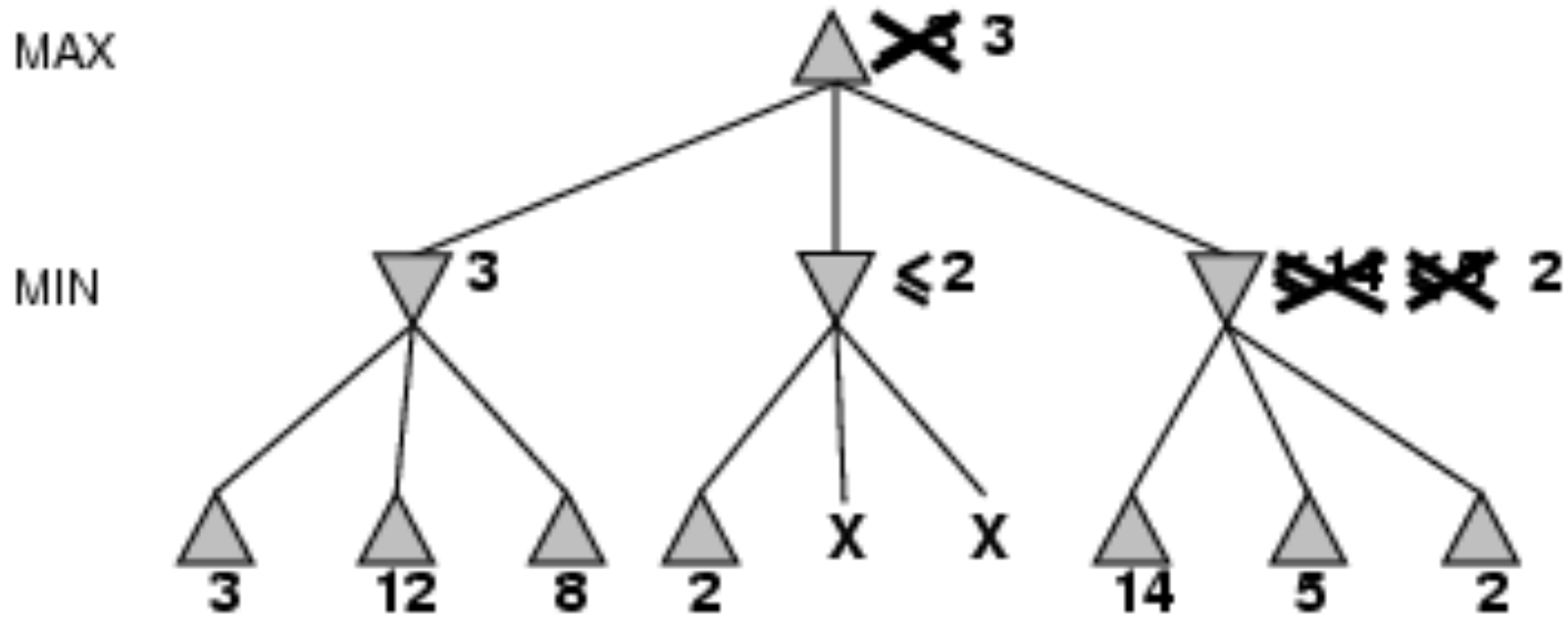
MIN will do at least as good as 14 in this branch (which is very good for MAX!) so MAX will want to explore this branch more.

α-β Pruning Example



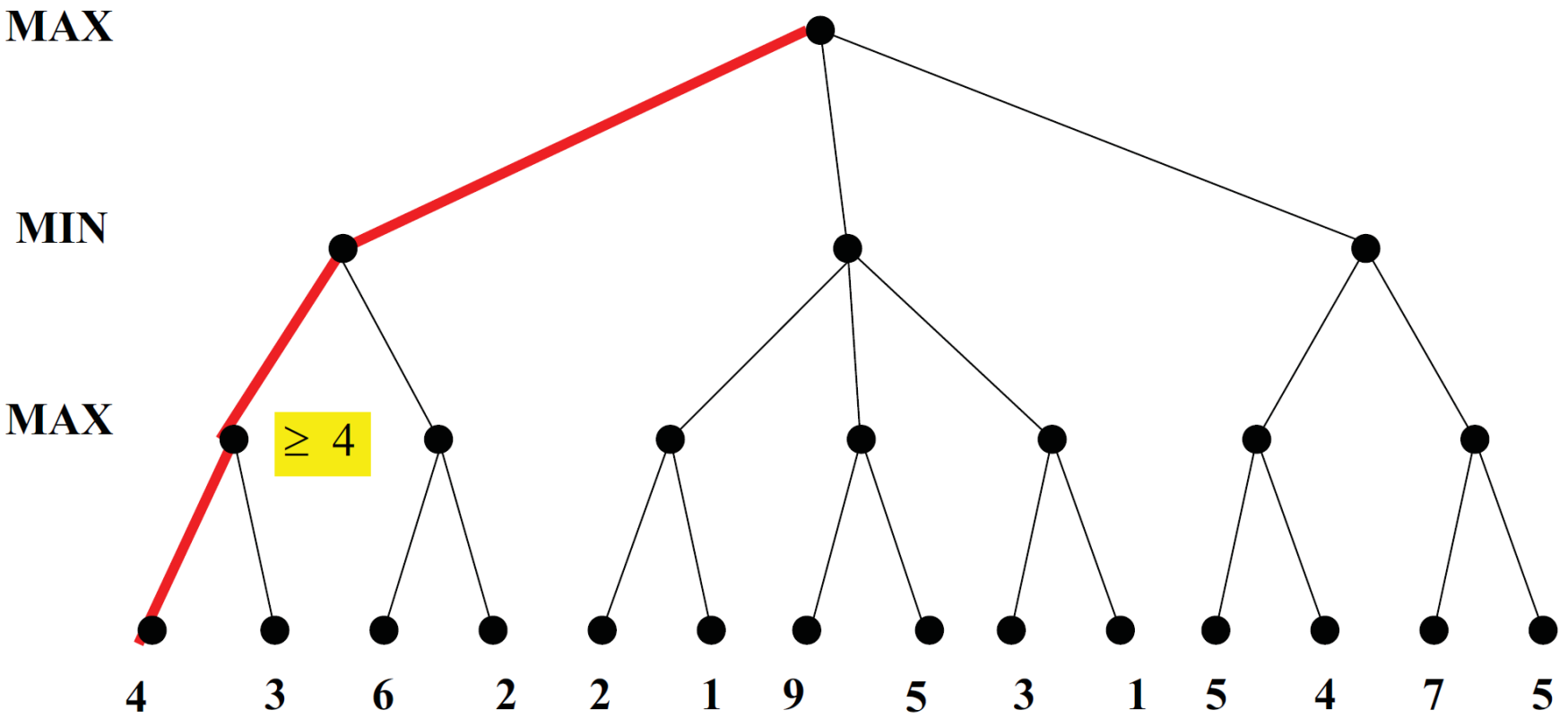
MIN will do at least as good as 5 in this branch (which is still good for MAX) so MAX will want to explore this branch more.

α-β Pruning Example

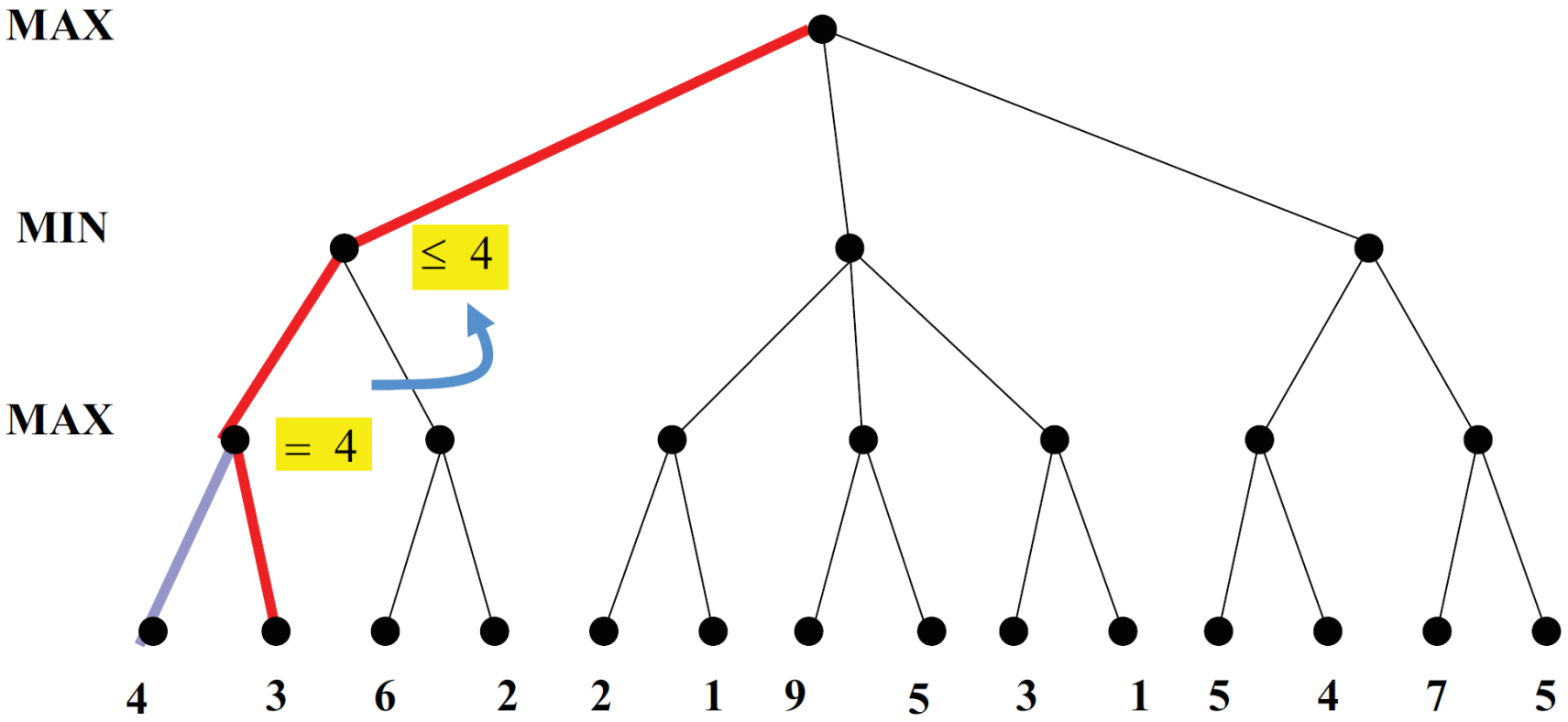


Bummer (for MAX): MIN will be able to play this last branch and get 2. This is worse than 3, so MAX will play 3.

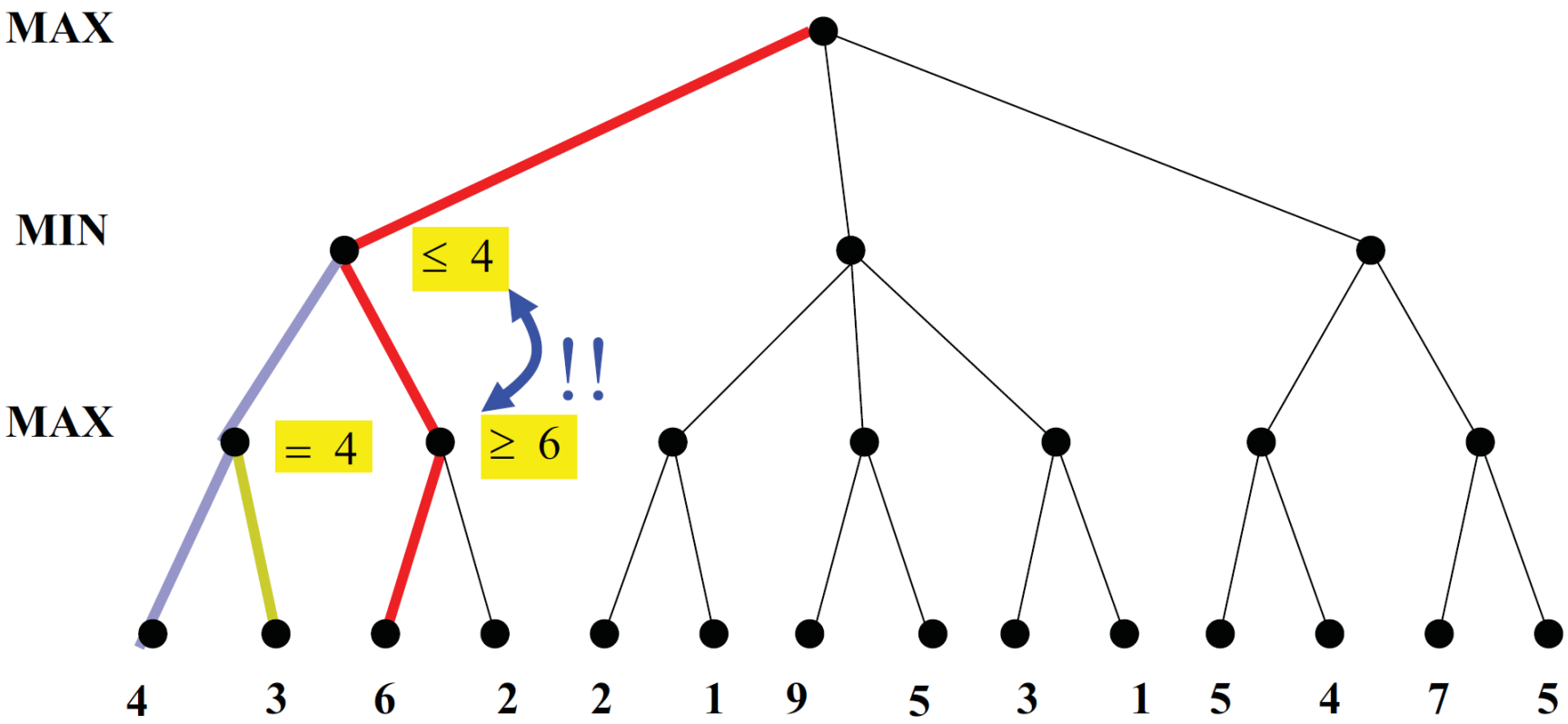
α-β Pruning Example



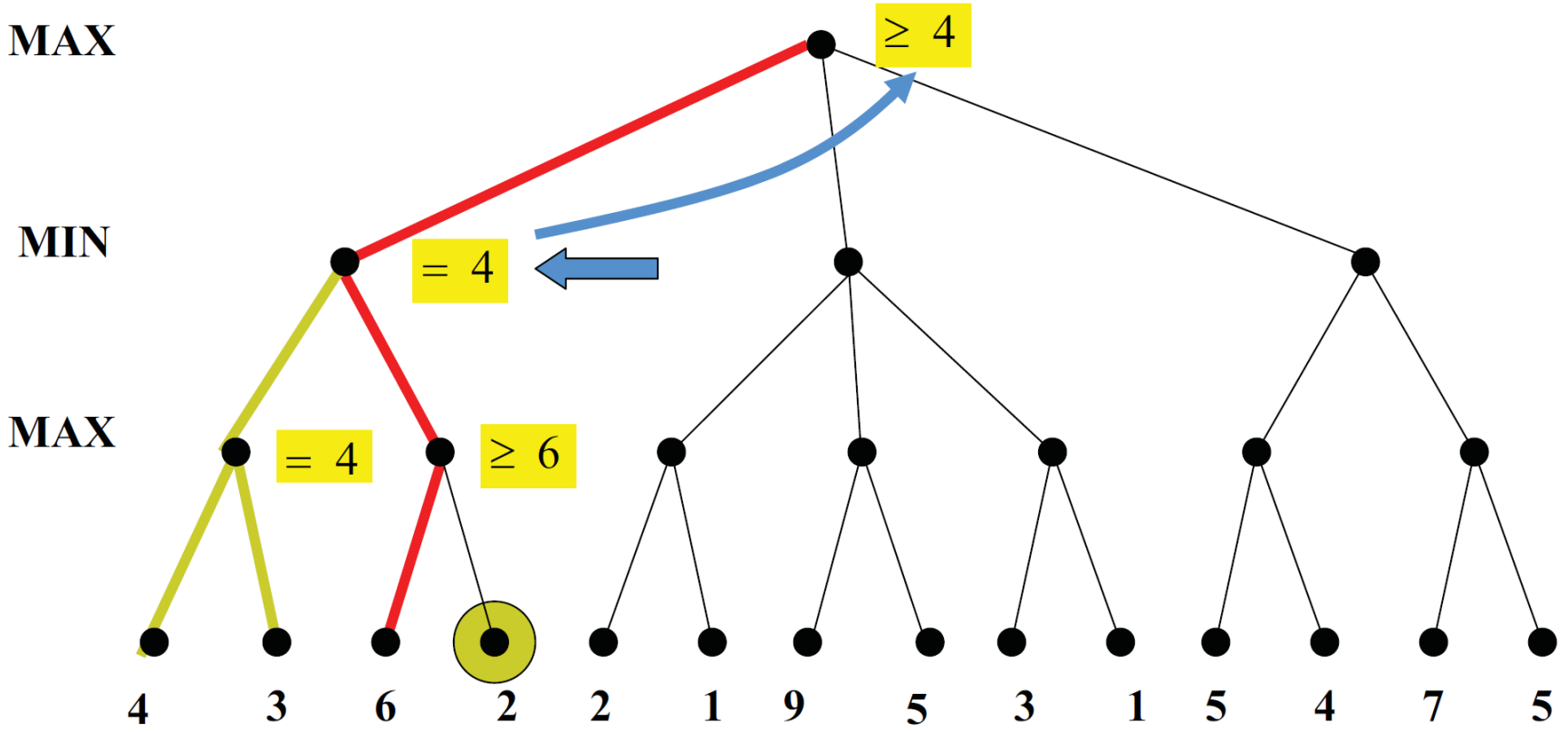
α-β Pruning Example



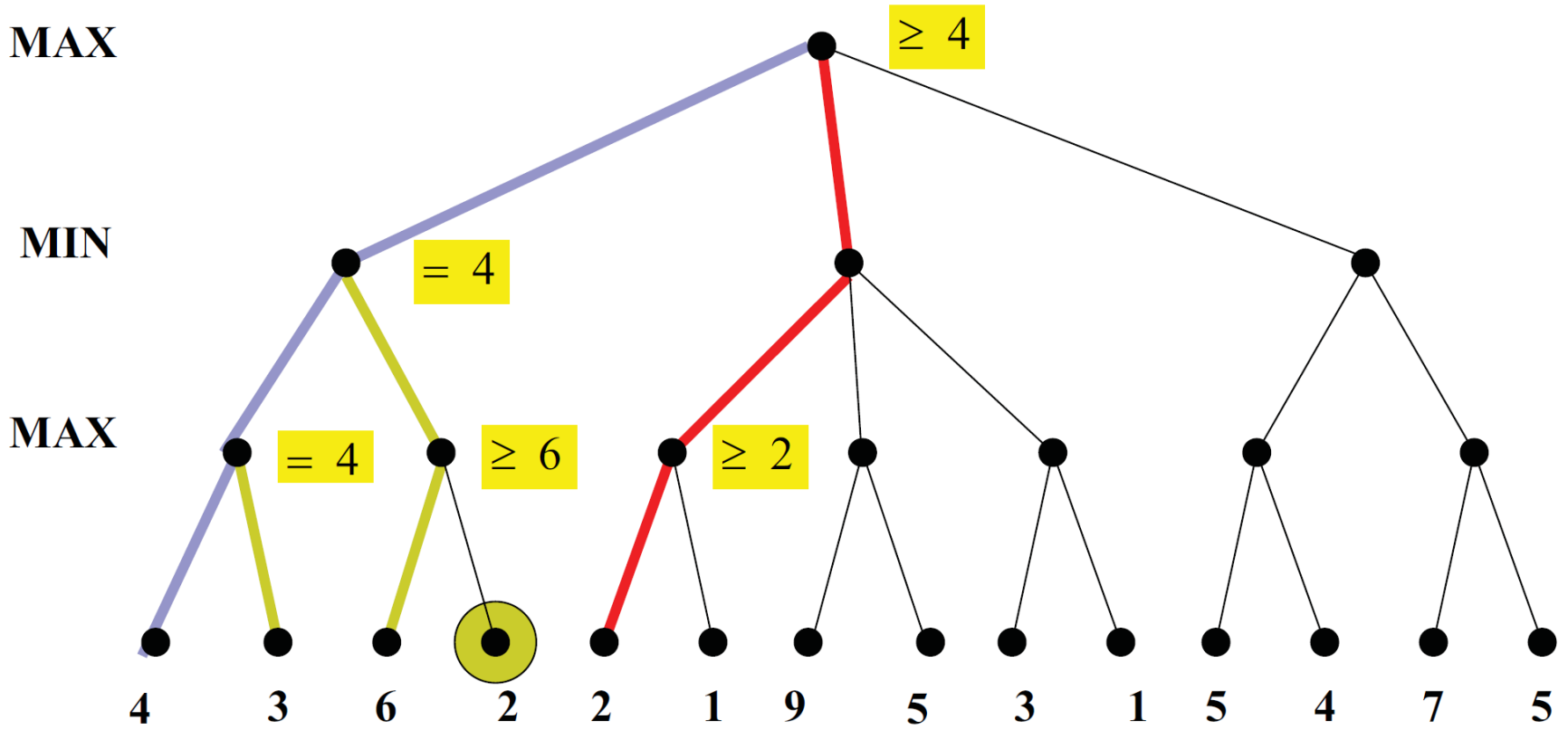
α-β Pruning Example



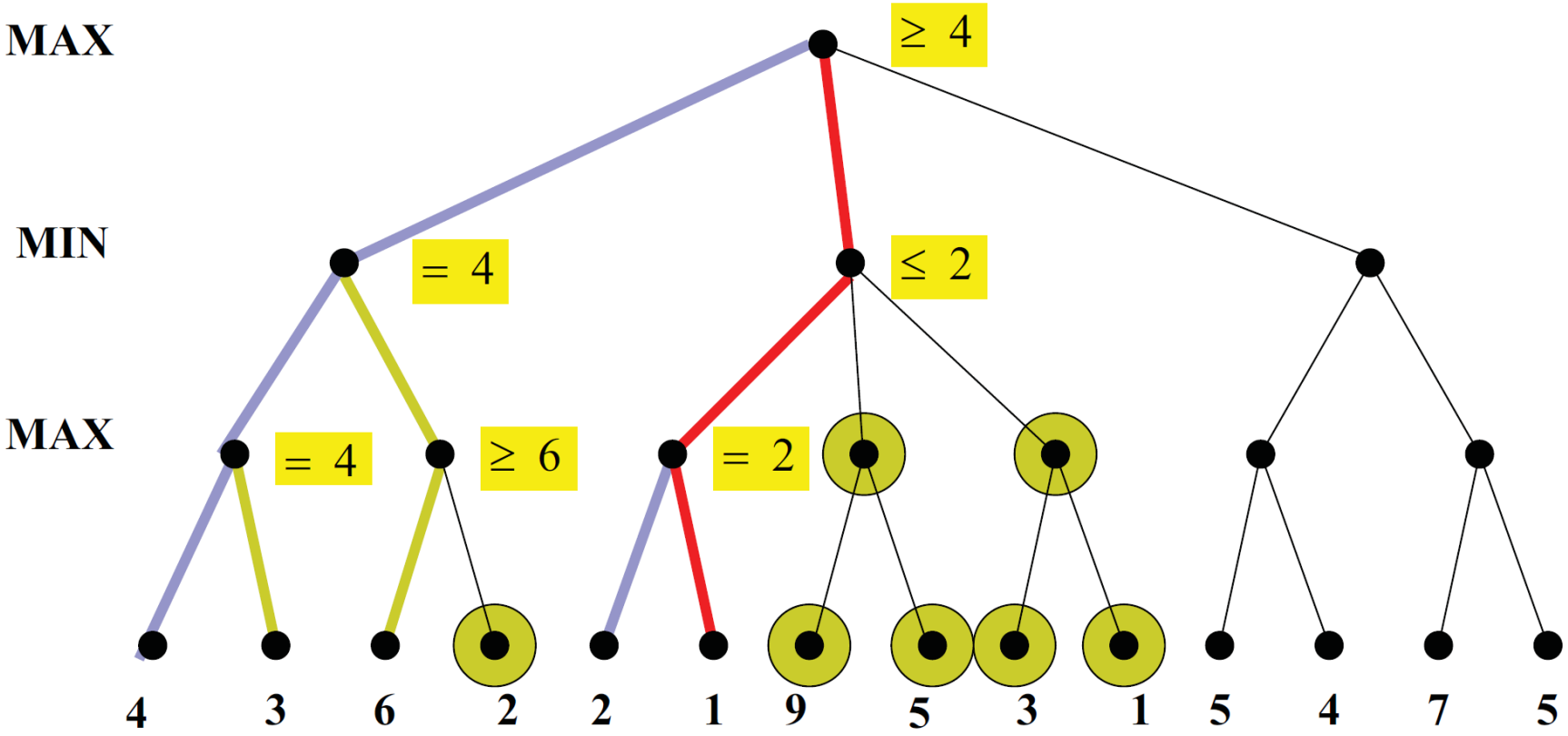
α-β Pruning Example



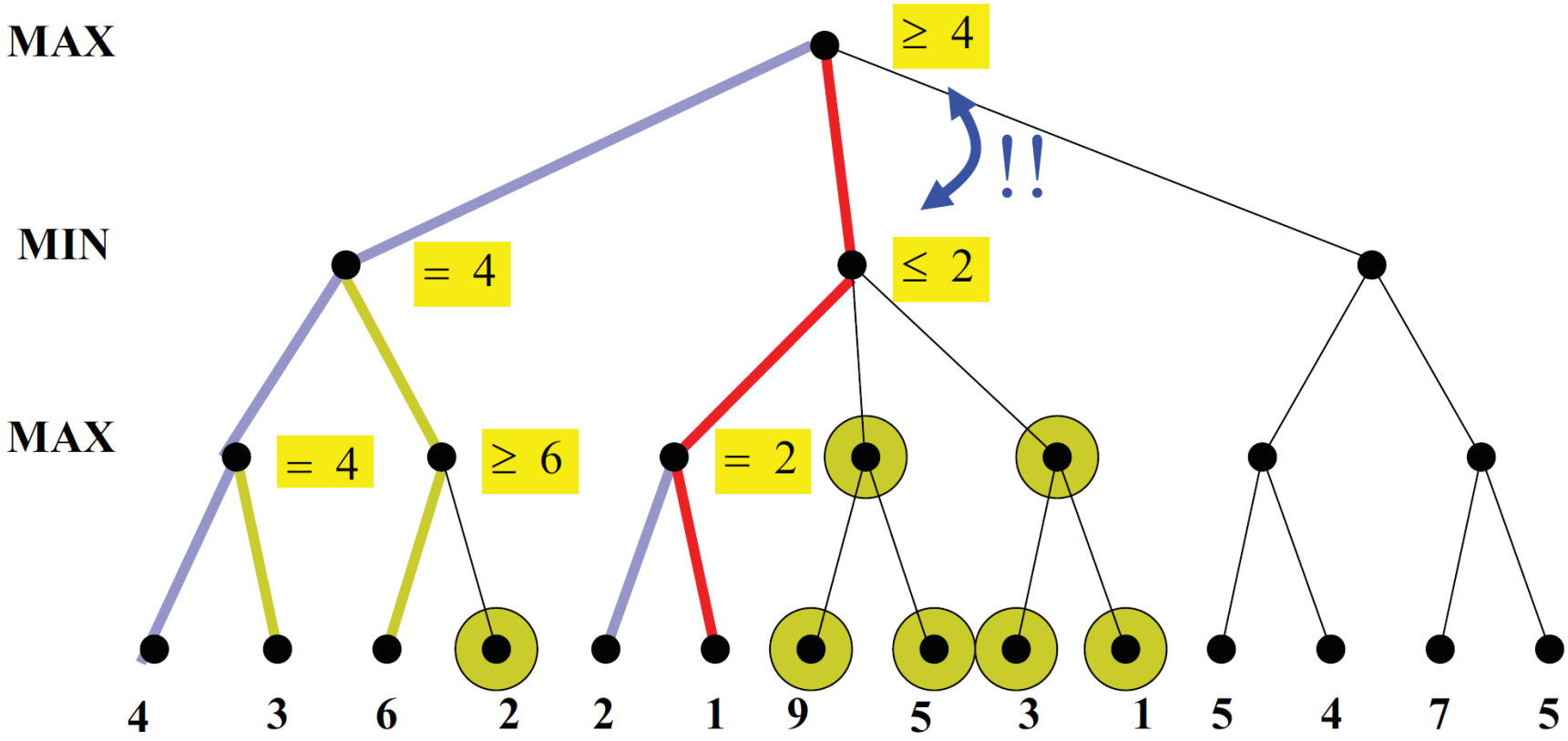
α-β Pruning Example



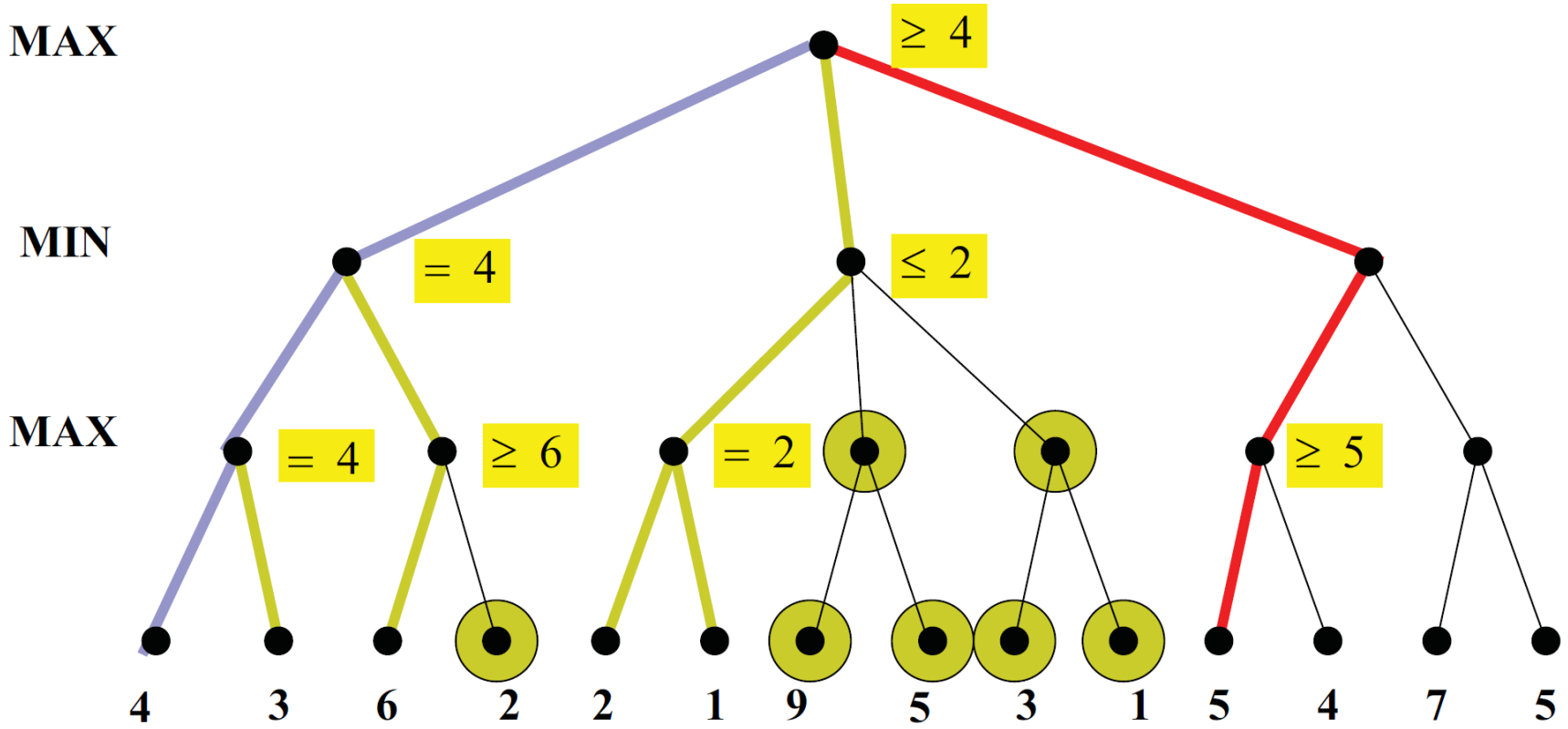
α-β Pruning Example



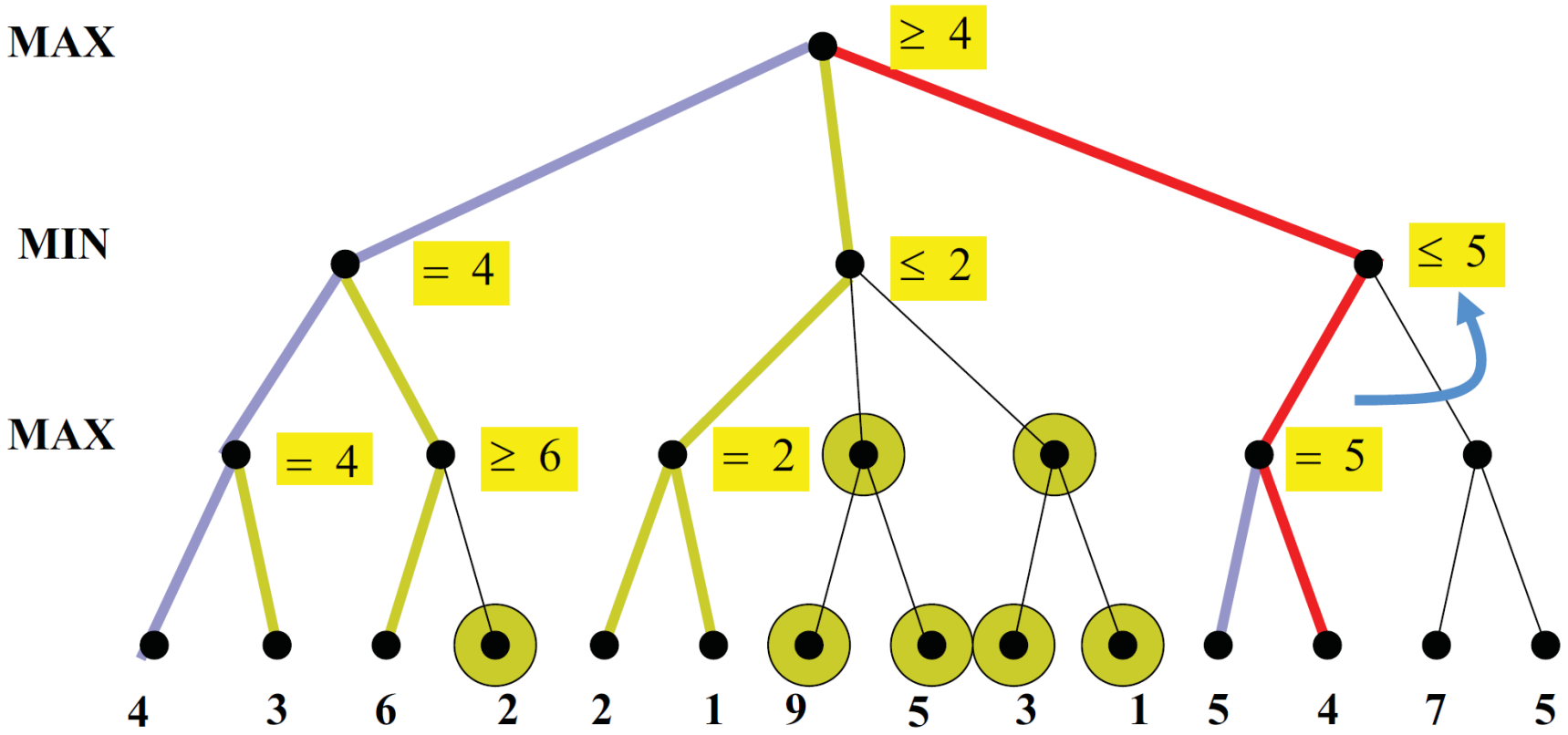
α-β Pruning Example



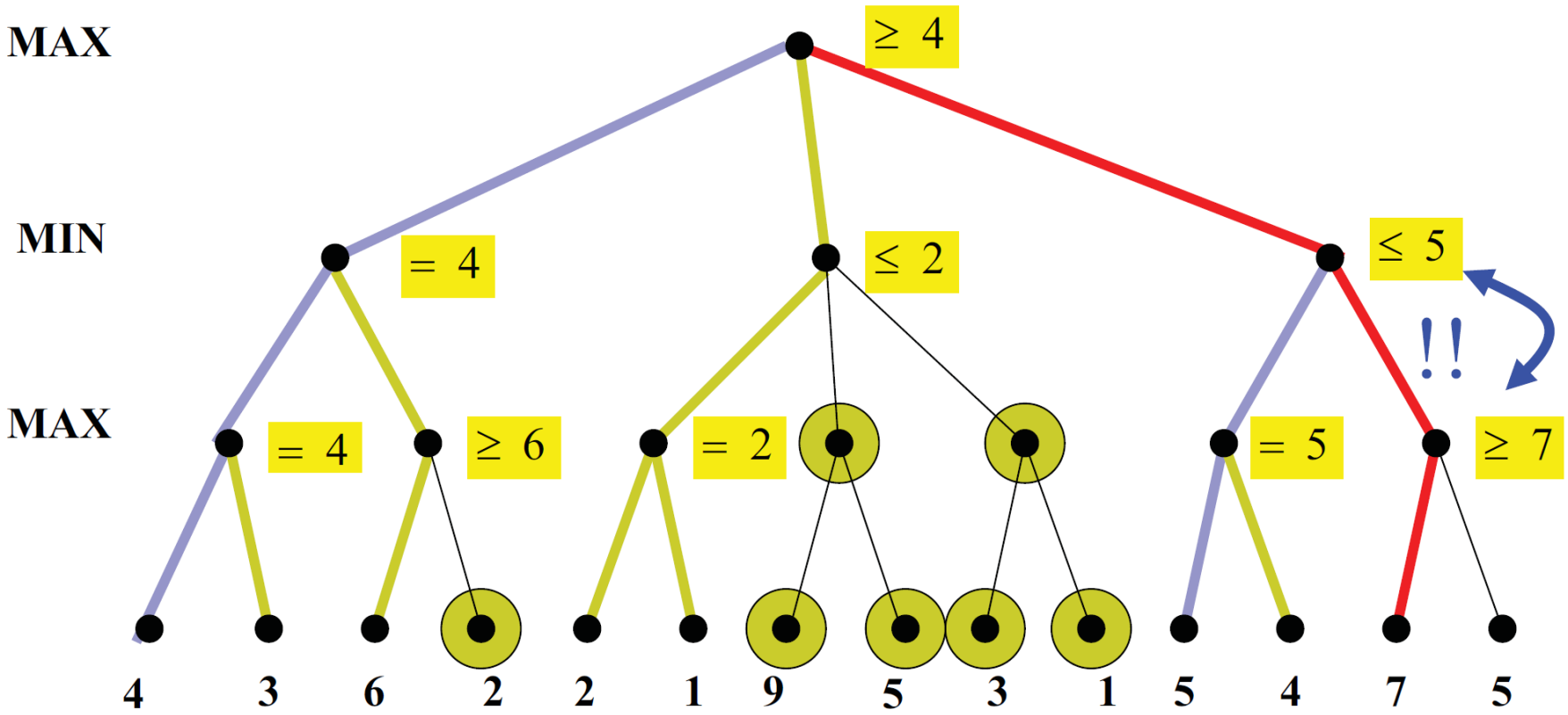
α-β Pruning Example



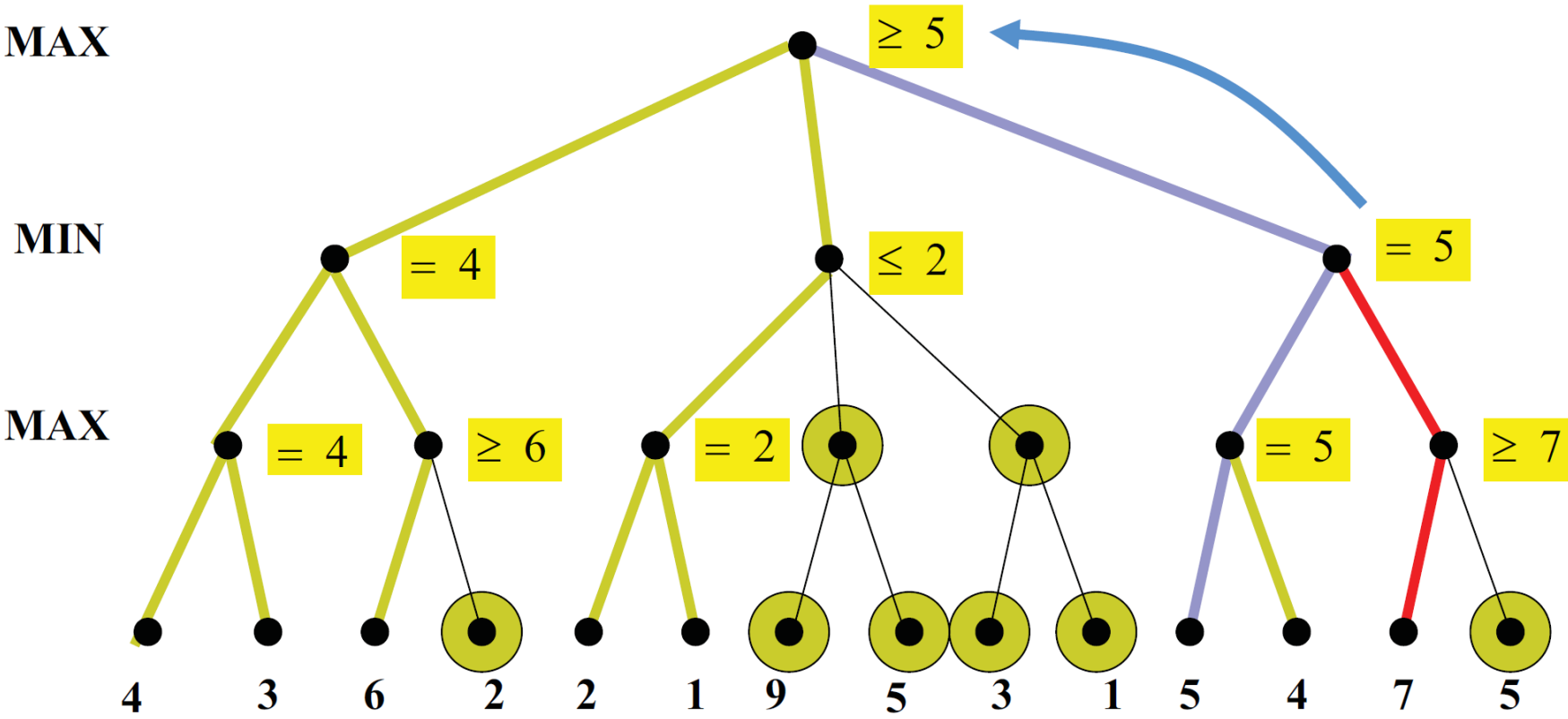
α-β Pruning Example



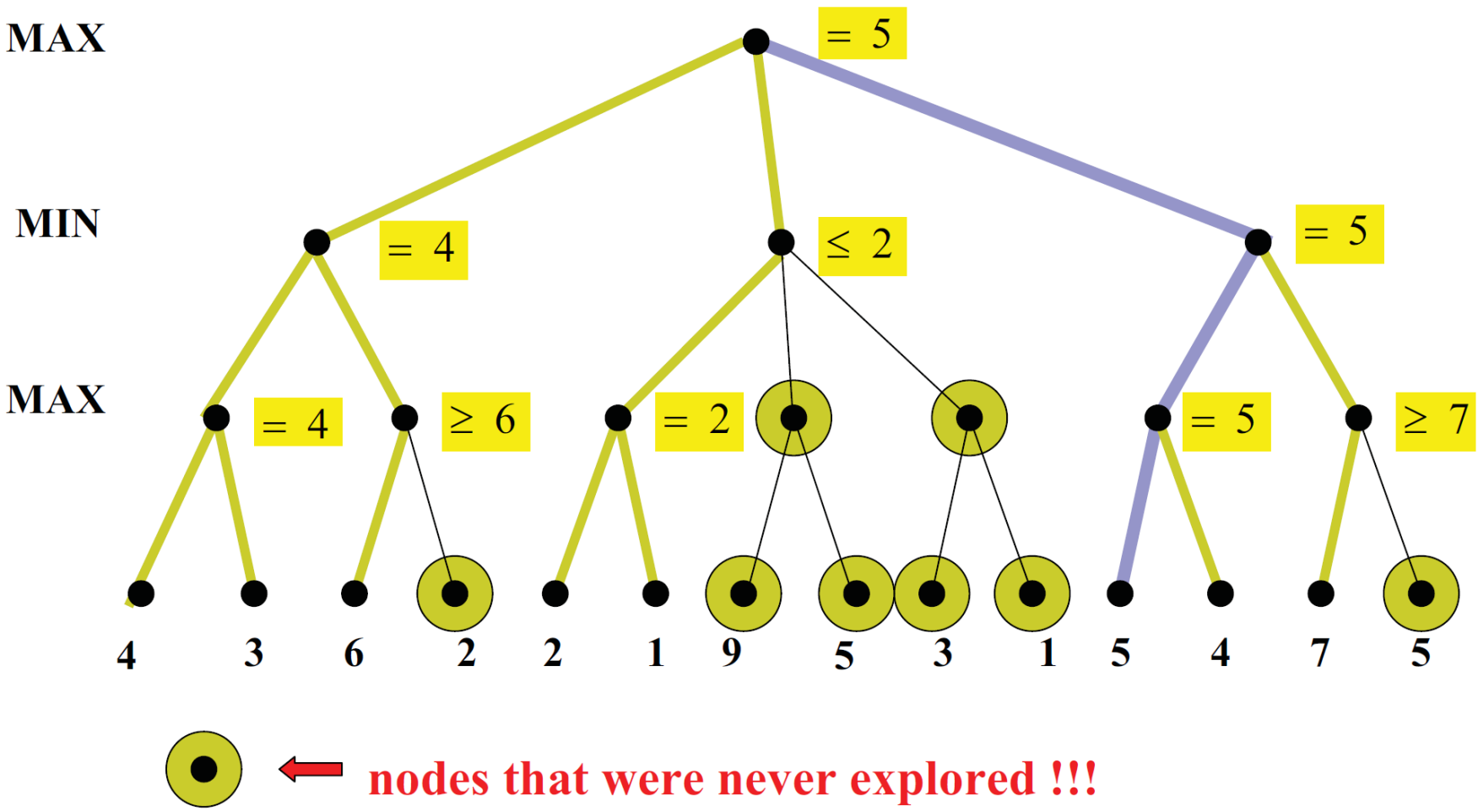
α-β Pruning Example



α-β Pruning Example



α-β Pruning Example

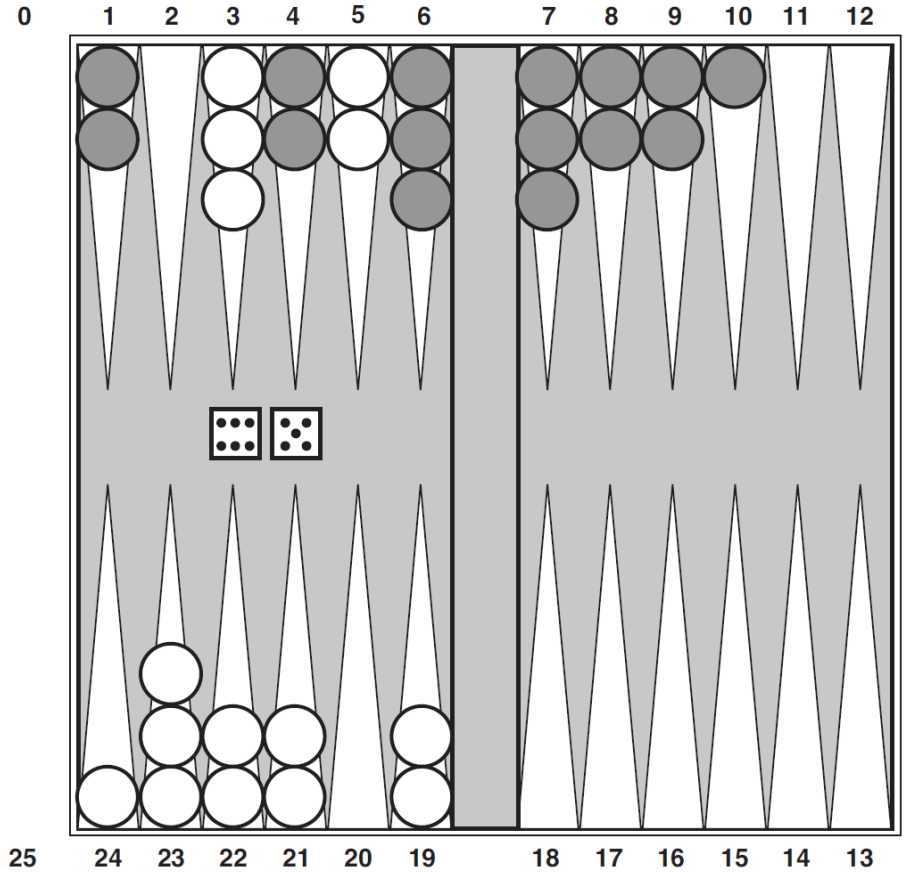


Properties of α-β Pruning

- Pruning does not affect final result (the minimax value of the root is the same as that found without pruning).

Stochastic Games

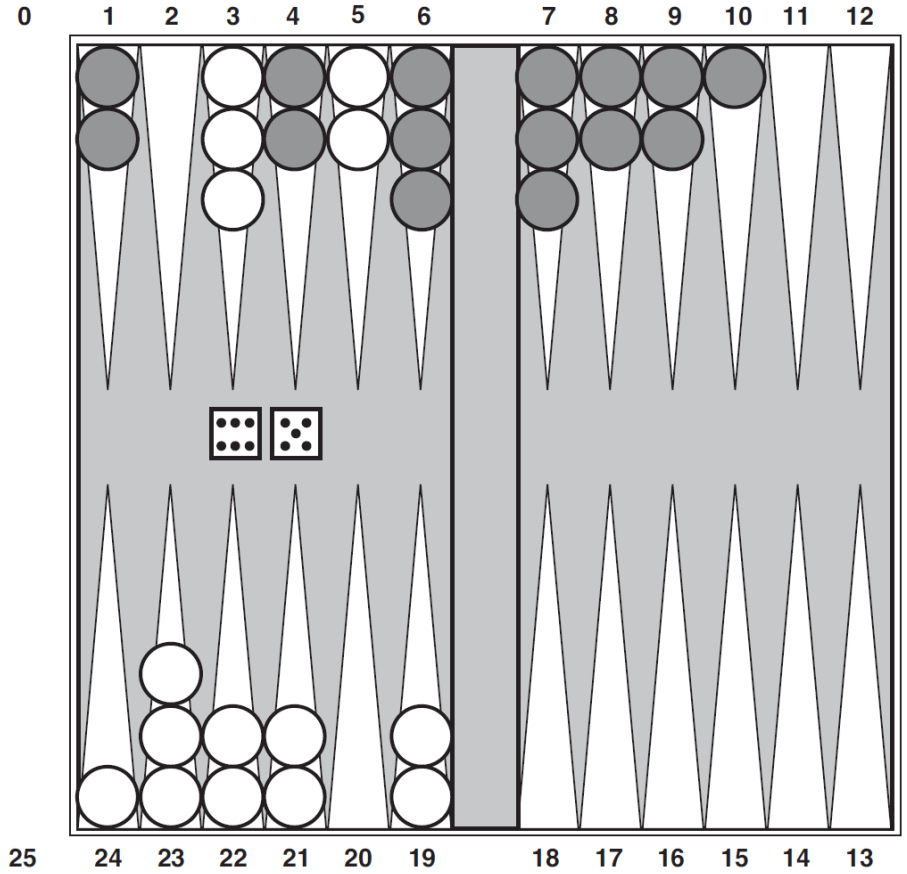
- Backgammon
  - Each player rolls dice to determine the legal moves



- Goal: move all one's checkers off the board
- White: moves clockwise toward 25
- Black: moves counterclockwise toward 0
- One can start moving checkers off the board only when all his/her checkers are in his/her home area

Stochastic Games

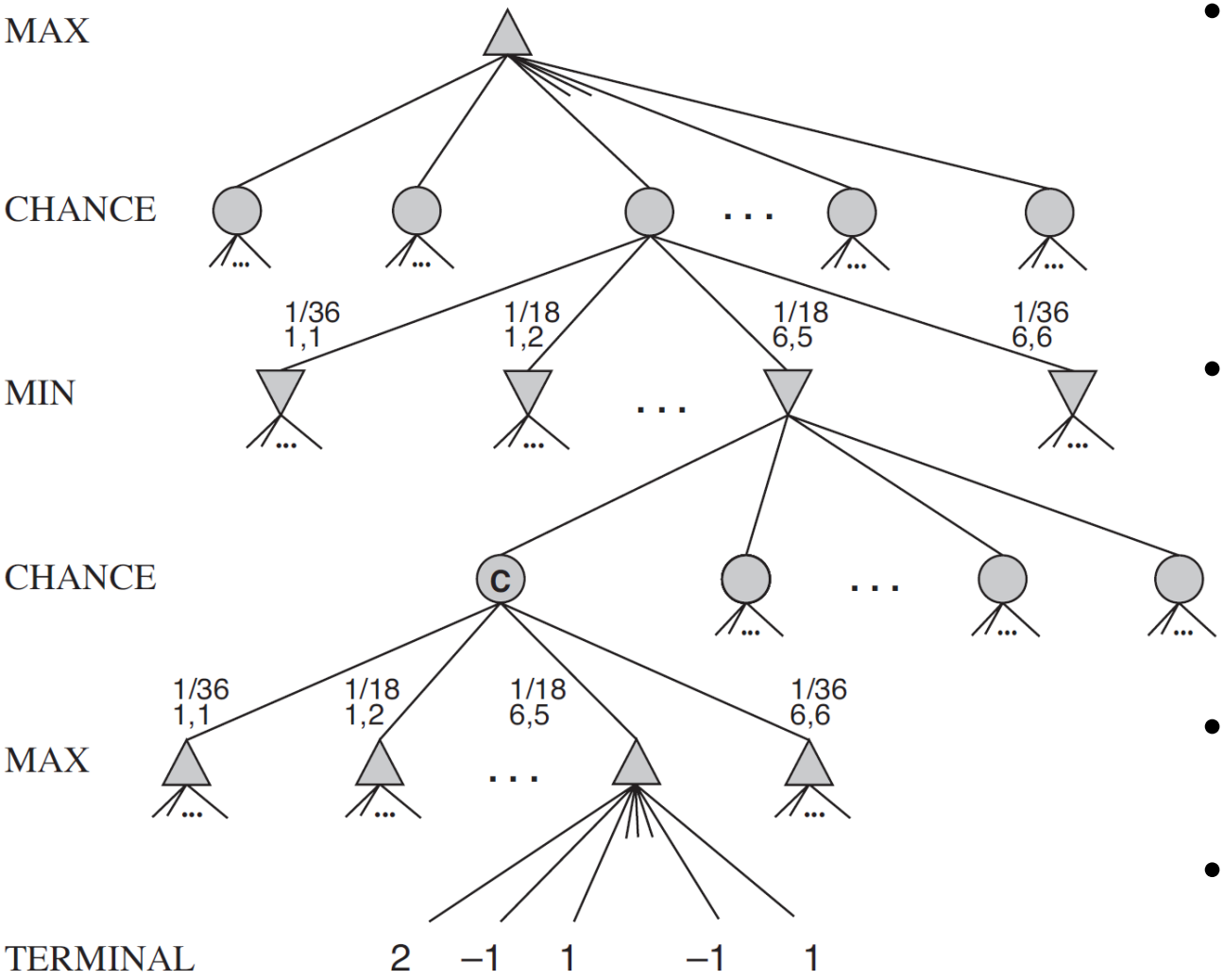
- Backgammon
  - Each player rolls dice to determine the legal moves



- White knows what his/her own legal moves are
- White does not know what Black is going to roll, and does not know what Black's legal moves will be
- White cannot construct a standard game tree
- A game tree in backgammon: must include **chance nodes**

Stochastic Games

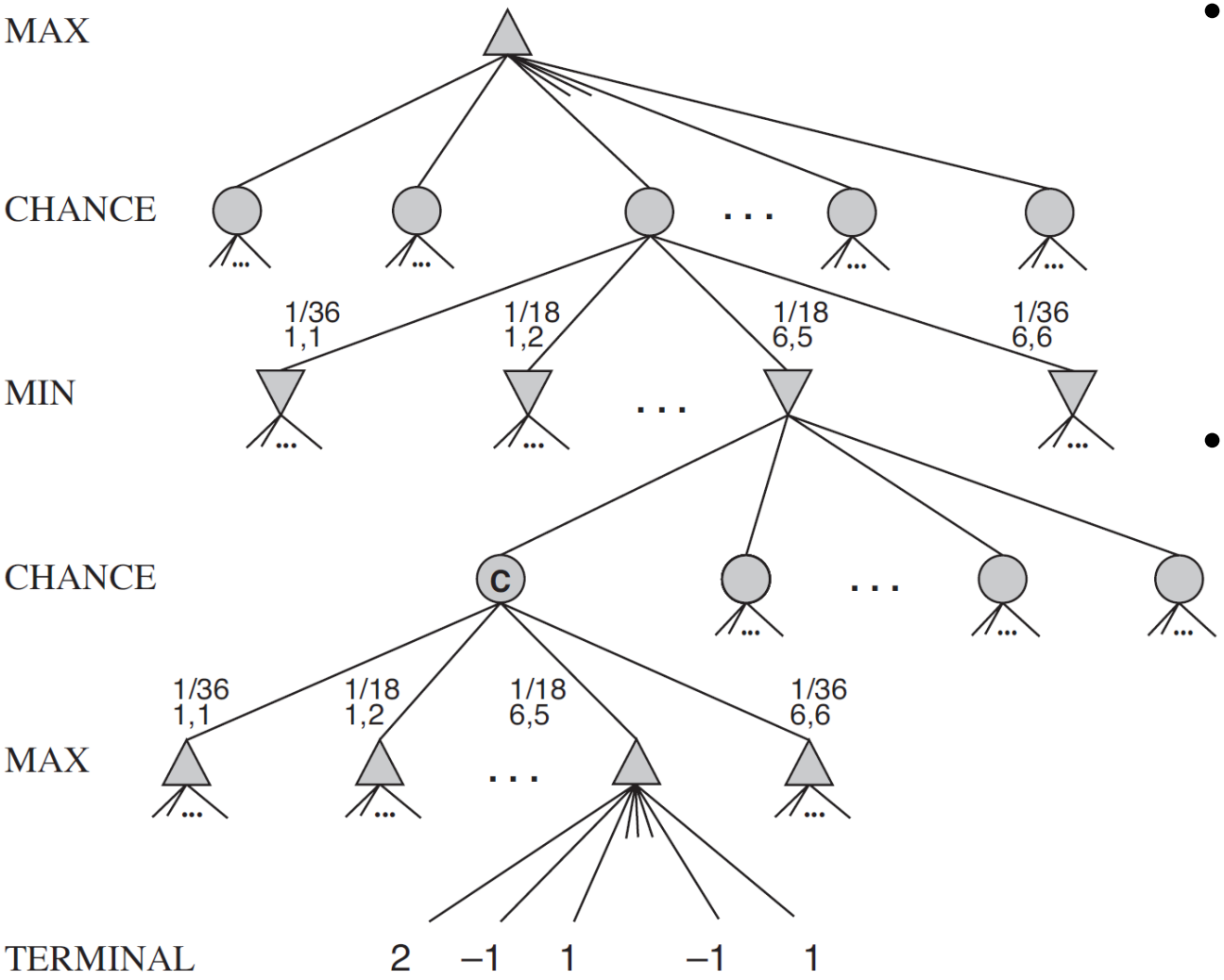
- Chance Nodes: Circles in this figure



- Branches leading from each chance node denote the possible dice rolls
- Each branch is labeled with the roll and its probability
- 36 ways to roll two dice
- Only 21 distinct rolls

Stochastic Games

- Chance Nodes: Circles in this figure



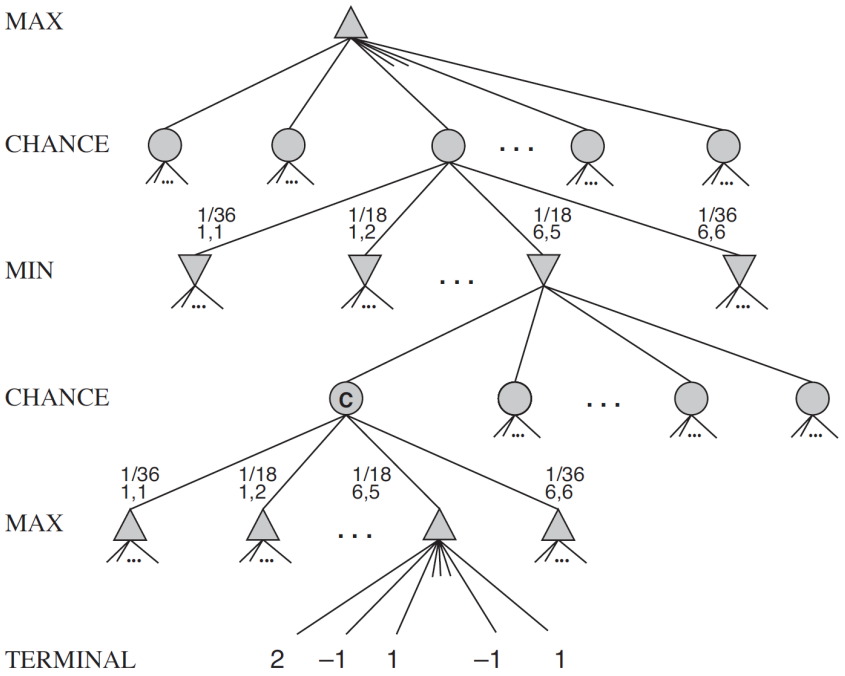
- The six doubles (1-1 through 6-6): each has a probability of 1/36
- The other 15 distinct rolls each has a 1/18 probability

Stochastic Games

- Expecti-minimax values
  - The average over all possible outcomes of the chance nodes
- Terminal nodes and MAX and MIN nodes (for which the dice roll is known) work the same way as before
- For chance nodes: compute the expected value, which is the sum of the value over all outcomes, weighted by the probability of each chance action

Expecti-minimax values

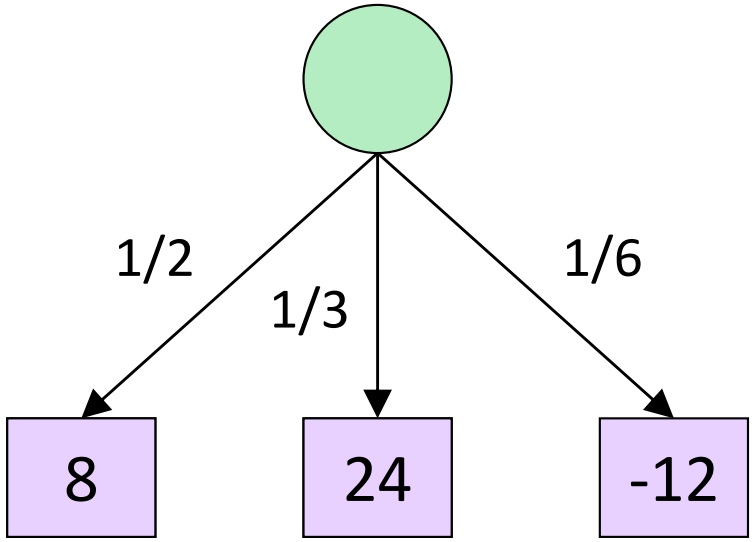
- The average over all possible outcomes of the chance nodes



$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$

Example 1

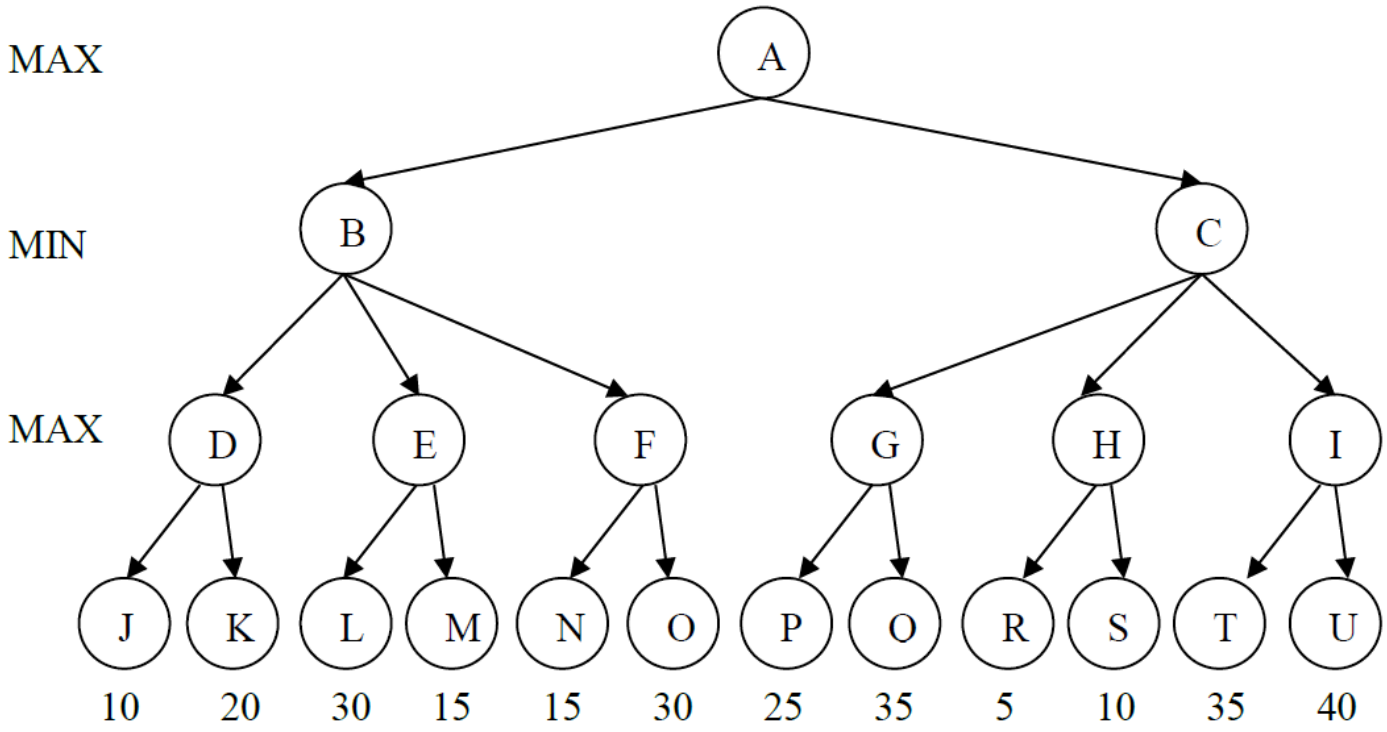
- Calculate the expectiminimax value of the chance node



v = (1/2) (8) + (1/3) (24) + (1/6) (-12) = 10

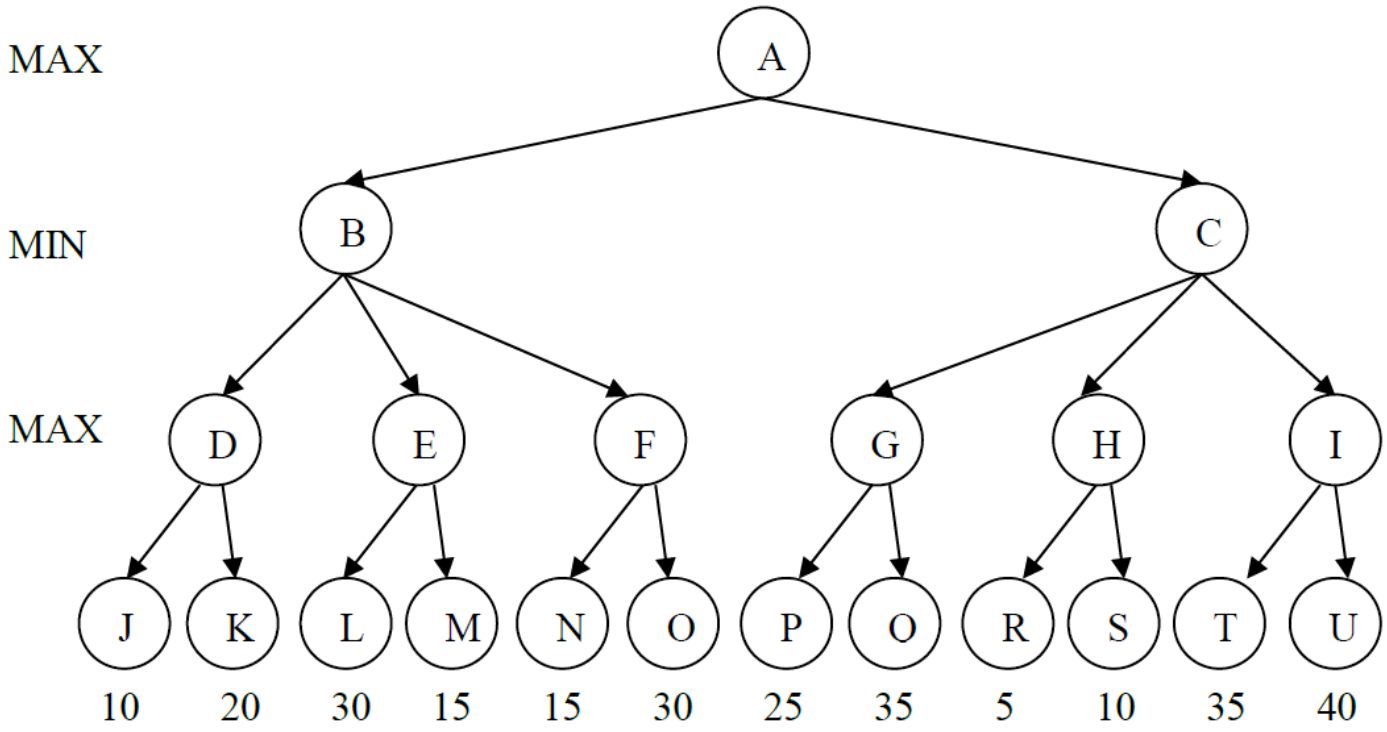
Example 2

- In the tree below assume that the nodes D, …, I are ***not*** MAX nodes but instead are positions where a fair coin is flipped (so going to each child has probability 0.5). What is the *Expectiminimax* value at node A?



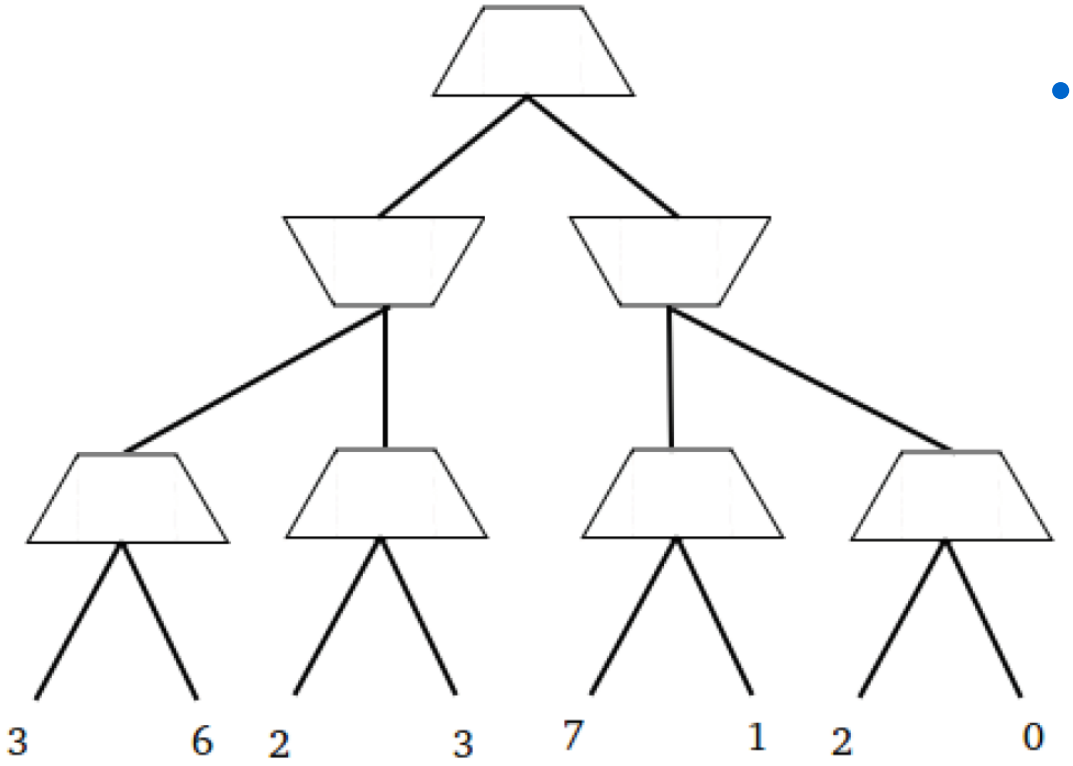
Example 2

- Answer: 15



Example 3: Standard Minimax

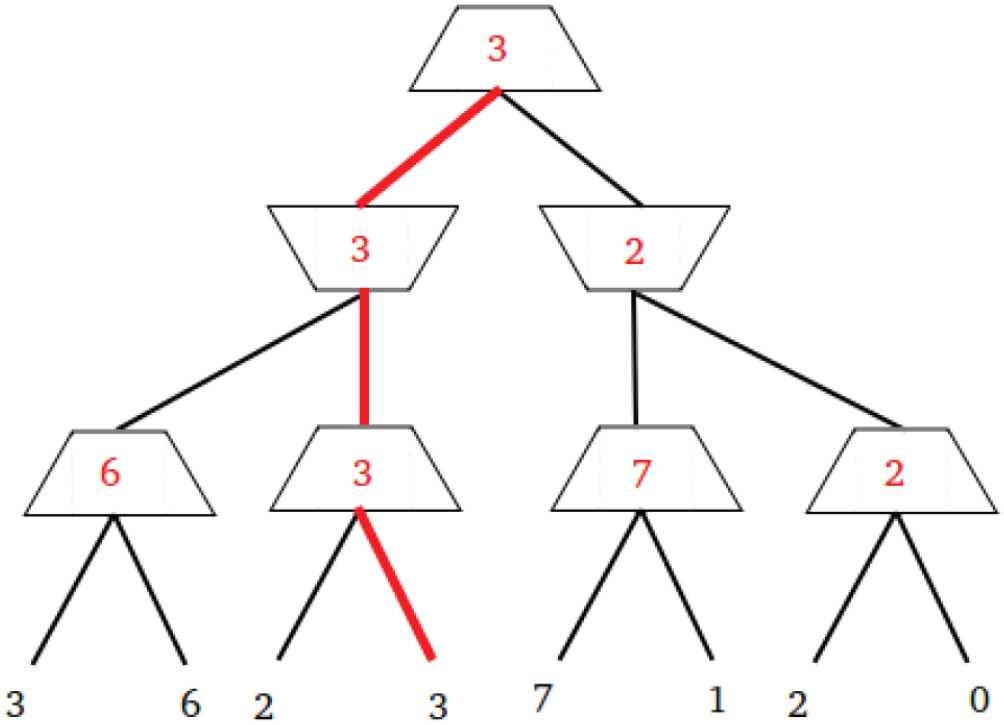
- Fill in the values of each of the nodes in the following Minimax tree. The upward pointing trapezoids correspond to maximizer nodes (layer 1 and 3), and the downward pointing trapezoids correspond to minimizer nodes (layer 2). Each node has two actions available, Left and Right.



- Mark the sequence of actions that correspond to Minimax play.

Example 3: Standard Minimax

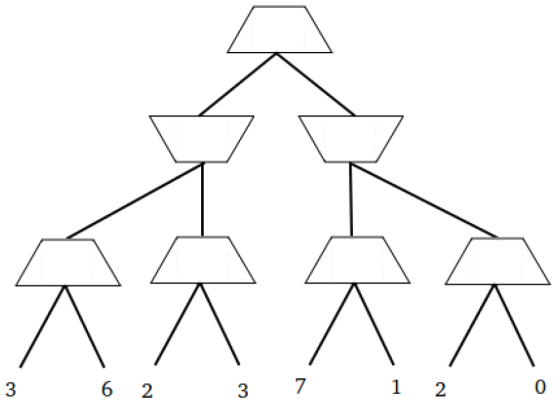
- Fill in the values of each of the nodes in the following Minimax tree. The upward pointing trapezoids correspond to maximizer nodes (layer 1 and 3), and the downward pointing trapezoids correspond to minimizer nodes (layer 2). Each node has two actions available, Left and Right.



- Mark the sequence of actions that correspond to Minimax play.

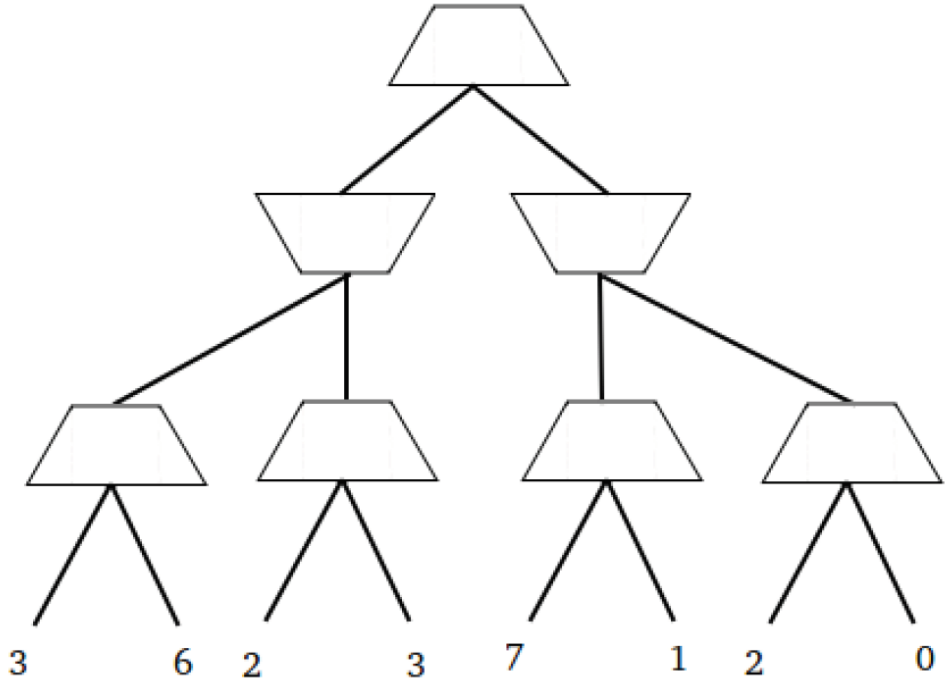
Example 4: Dark Magic

- Pacman (= maximizer) has mastered some dark magic. With his dark magic skills Pacman can take control over his opponent's muscles while they execute their move - and in doing so be fully in charge of the opponent's move. But the magic comes at a price: every time Pacman uses his magic, he pays a price of c - which is measured in the same units as the values at the bottom of the tree.
- Note: For each of his opponent's actions, Pacman has the choice to either let his opponent act (optimally according to minimax), or to take control over his opponent's move at a cost of c.



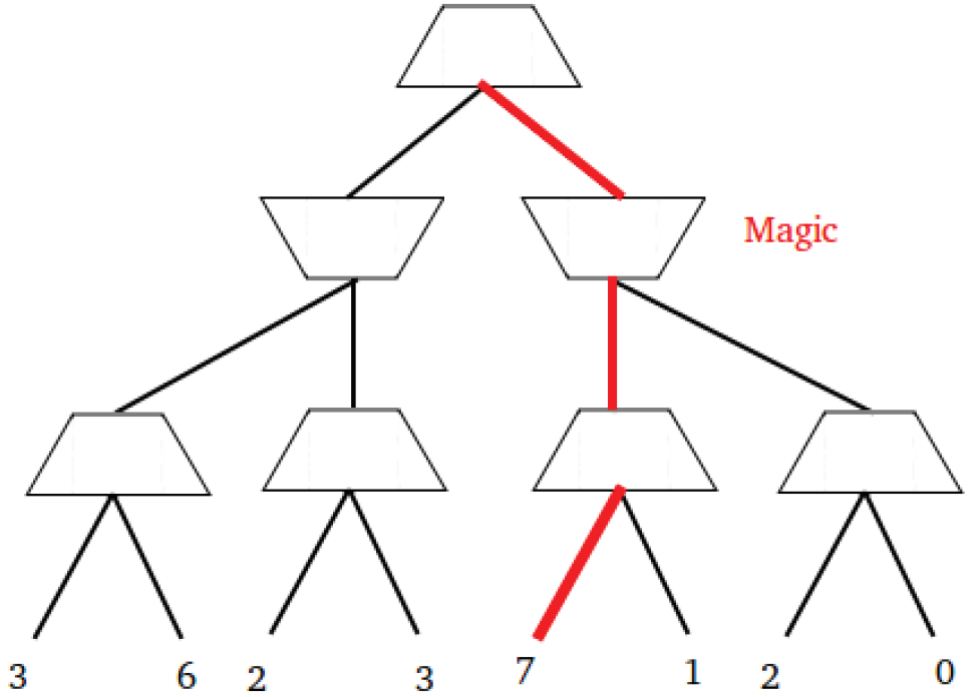
Example 4: Dark Magic

- (i) Dark Magic at Cost $c = 2$
- Consider the same game as before but now Pacman has access to his magic at cost $c = 2$. Is it optimal for Pacman to use his dark magic? If so, mark in the tree below where he will use it. Mark what the outcome of the game will be and the sequence of actions that lead to that outcome.



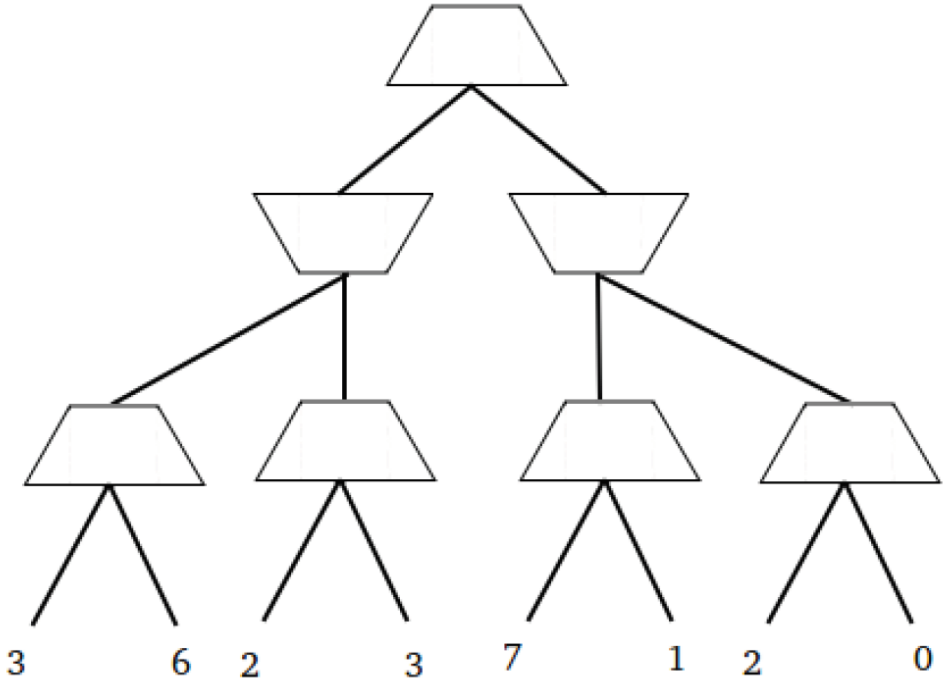
Example 4: Dark Magic

- (i) Dark Magic at Cost $c = 2$
- Consider the same game as before but now Pacman has access to his magic at cost $c = 2$. Is it optimal for Pacman to use his dark magic? If so, mark in the tree below where he will use it. Mark what the outcome of the game will be and the sequence of actions that lead to that outcome.



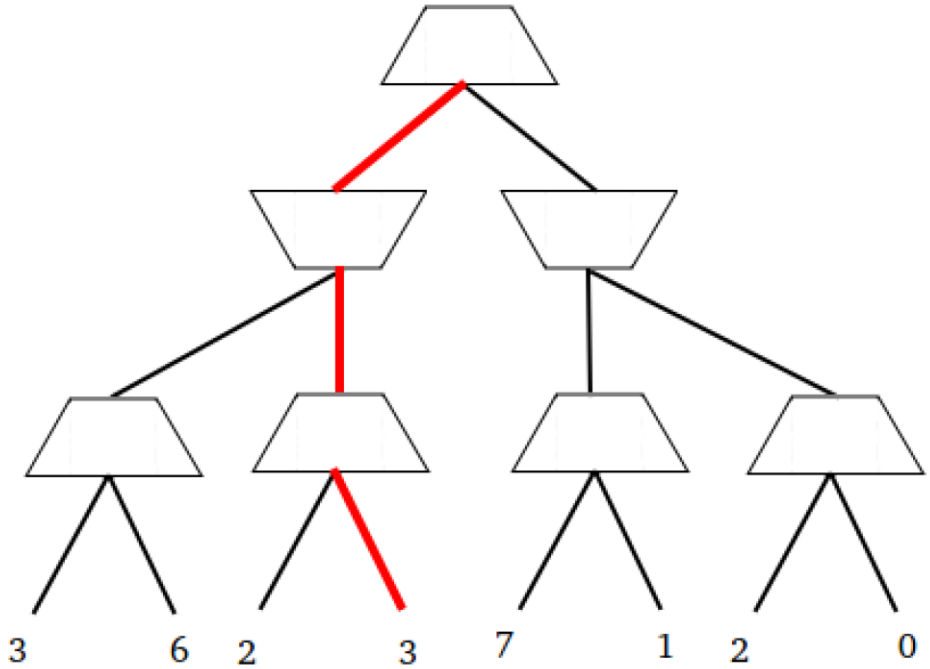
Example 4: Dark Magic

- (ii) Dark Magic at Cost $c = 5$
- Consider the same game as before but now Pacman has access to his magic at cost $c = 5$. Is it optimal for Pacman to use his dark magic? If so, mark in the tree below where he will use it. Mark what the outcome of the game will be and the sequence of actions that lead to that outcome.



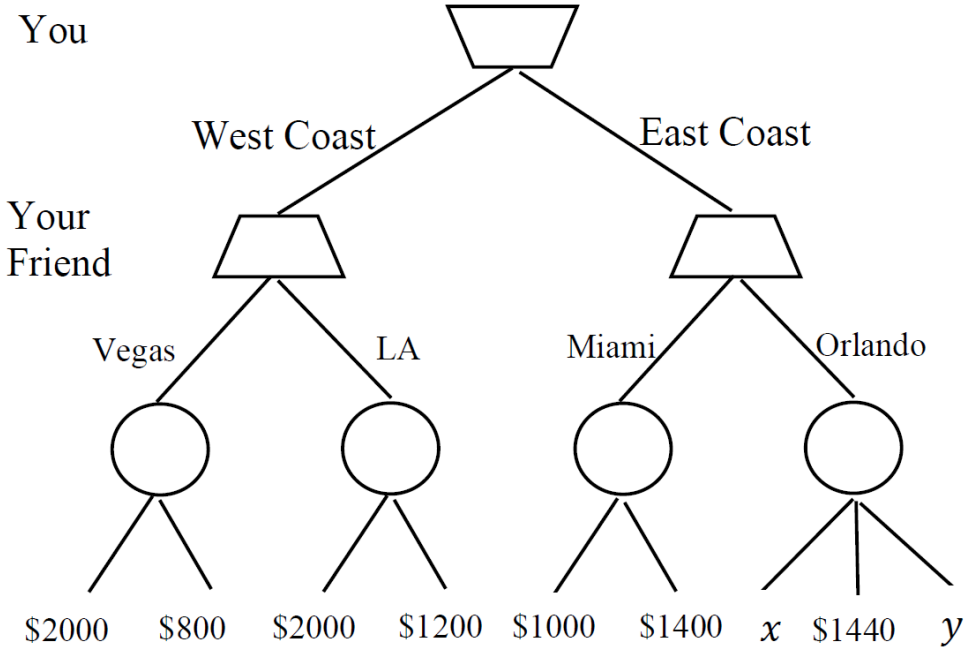
Example 4: Dark Magic

- (ii) Dark Magic at Cost $c = 5$
- Consider the same game as before but now Pacman has access to his magic at cost $c = 5$. Is it optimal for Pacman to use his dark magic? Mark in the tree below where he will use it. Mark what the outcome of the game will be and the sequence of actions that lead to that outcome.



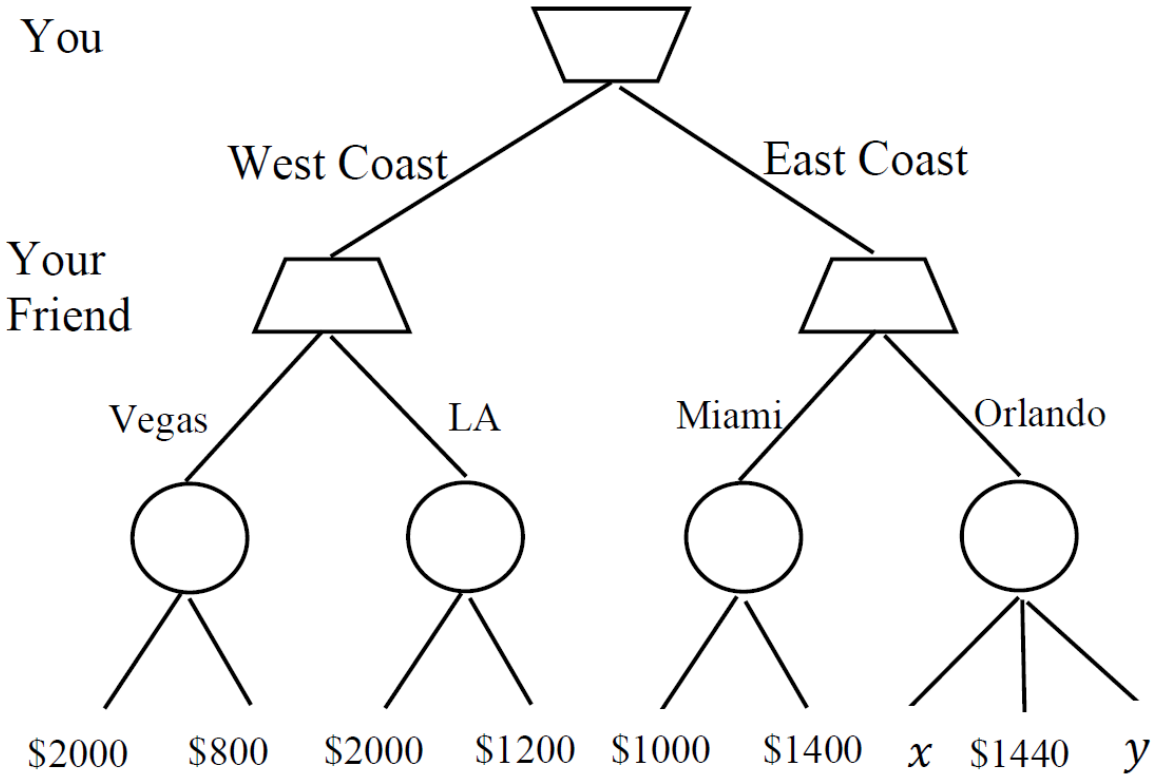
Example 5:

- You and your friend are planning for a trip. You want to minimize the cost, but your friend wants to choose a higher-cost trip to enjoy it. Finally, you decided to select the Coast, your friend will select the city to travel to, and each city has 2 or 3 trip plans available at random with equal probability. The prices are labeled on the leaf nodes and they are nonnegative.



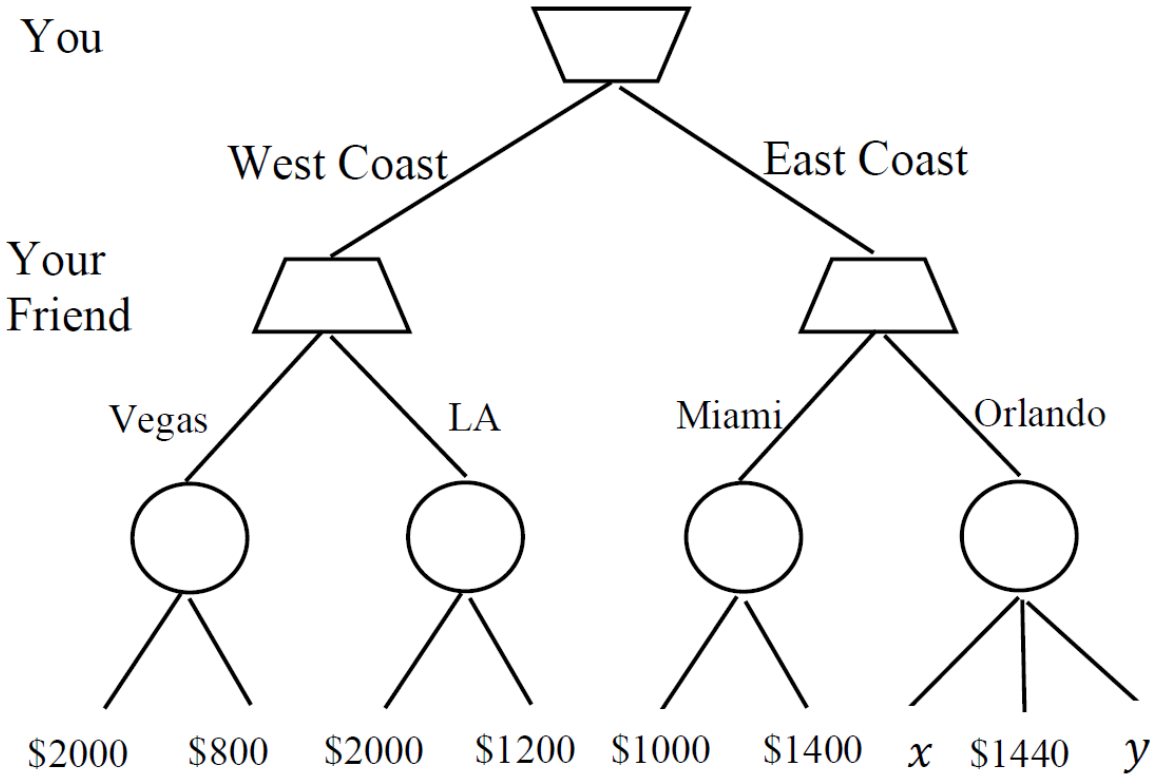
Example 5:

- A. Fill in the values of all the nodes that do not depend on the unknowns x and y.



Example 5:

- B. What values of x will make you select West Coast regardless of the price of y?



Example 5:

- C. We know that y is at most \$1200. What values of x will result in a trip to Miami regardless of the exact price of y?