Adversarial Search (Games)

CSEN266

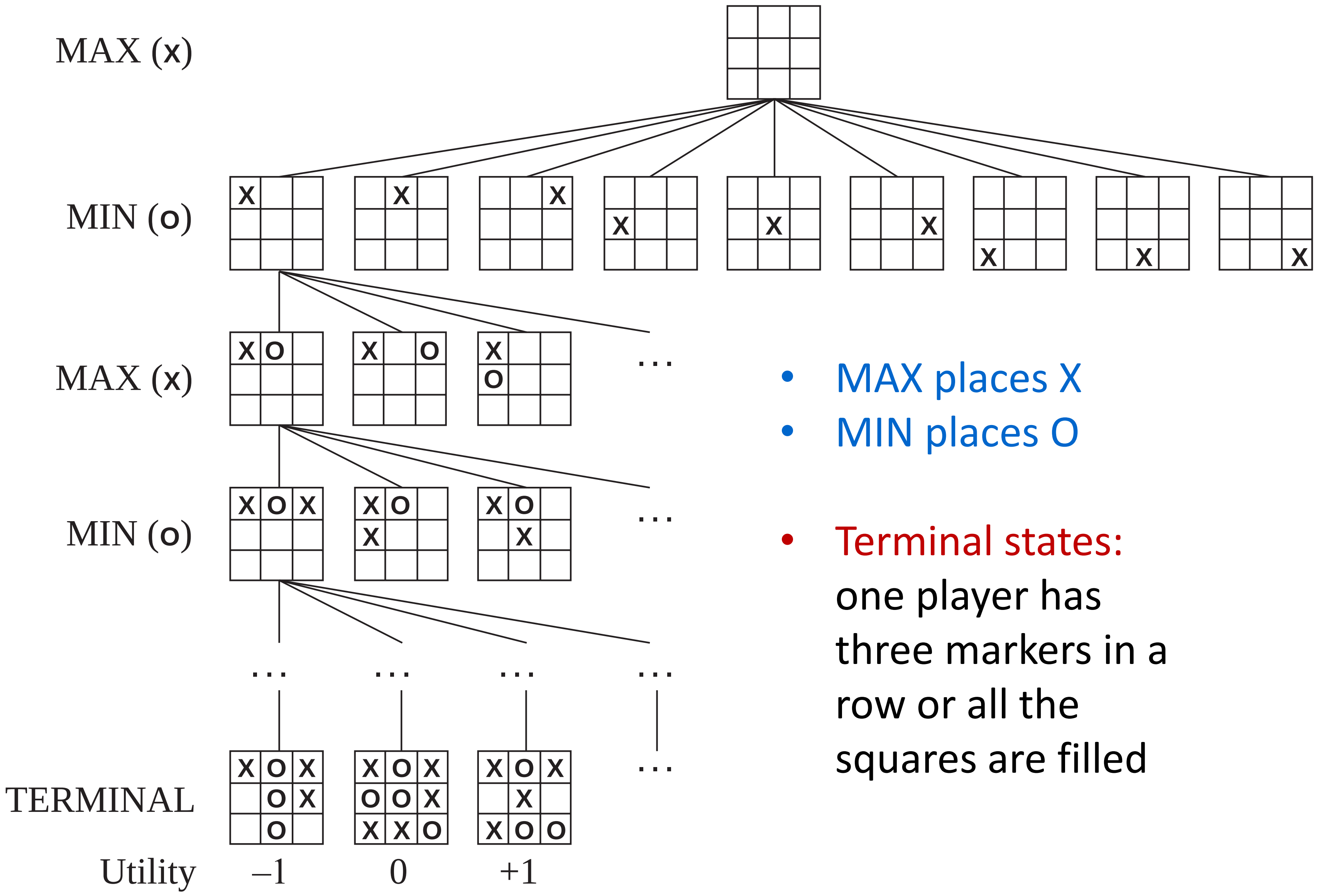
Artificial Intelligence

Adversarial Search - Games

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

- Objective:
  - Determine the best action that each player should take at each state s such that it can win the game

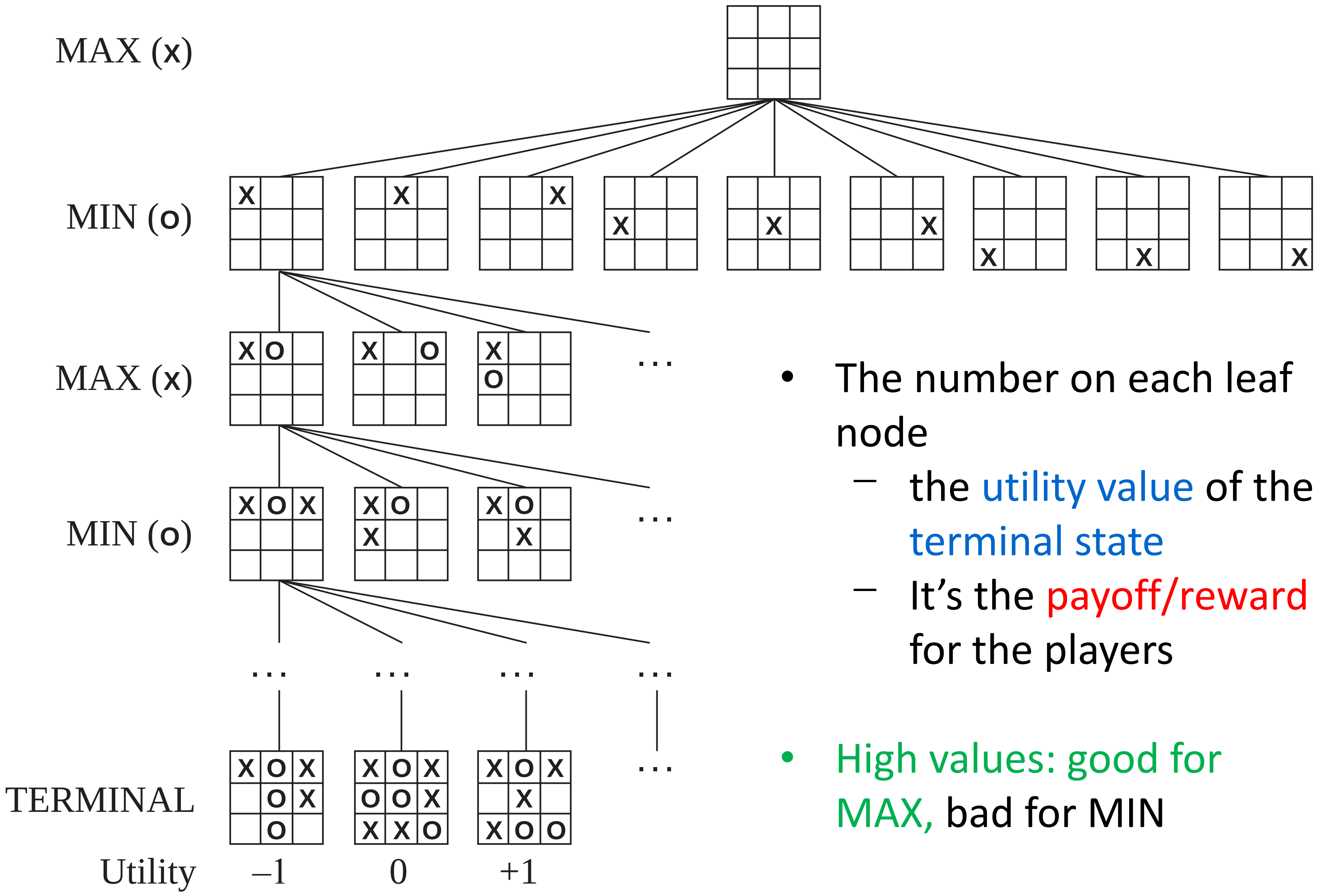
Tic-tac-toe



- MAX places X
- MIN places O

- Terminal states: one player has three markers 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
  - It's the payoff/reward for the players
- High values: good for MAX, bad for MIN

Typical Game Set Up

- Two players: MAX and MIN
- MAX moves first
- Then they take turns moving until the game is over
- 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
  - They have opposite goals
    - Compete with each other

Typical Game Set Up

- Objective: to determine what is the best action for each player at each state s?

- A decision-making problem
  - Formulate the problem as a search problem

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

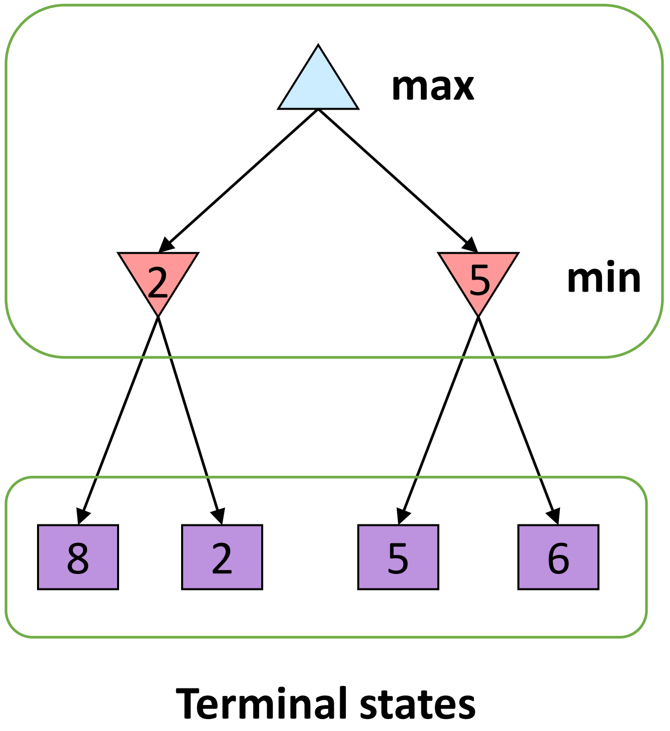
- Terminal-Test(s): **terminal test**
  - True when the game is over, False otherwise

- Utility(s): **utility function**
  - Defines the pay-off for a game that ends in terminal state s

- The optimal strategy for a certain player
  - Actions leading to a terminal state that is as good as possible for this player

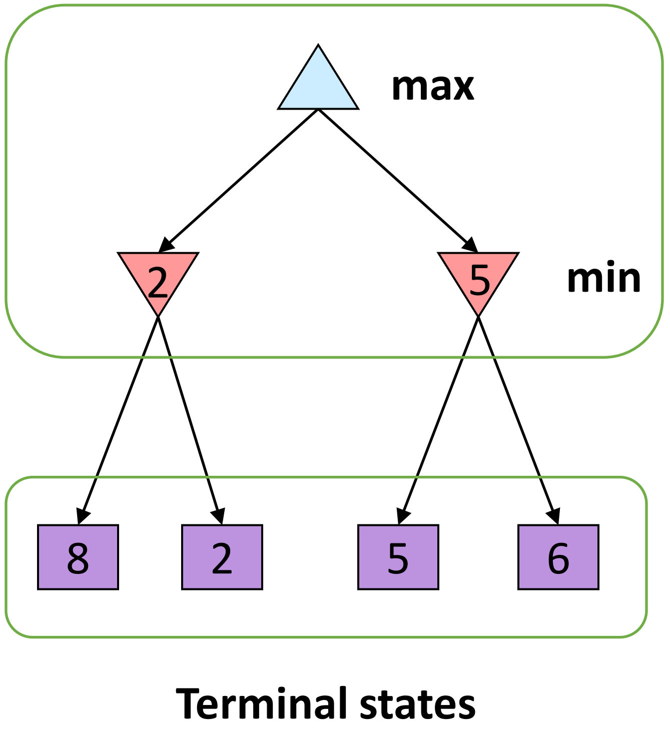
Minimax Algorithm

- Let’s consider player MIN
- What action should MIN take?
  - The one that leads to a smaller pay-off



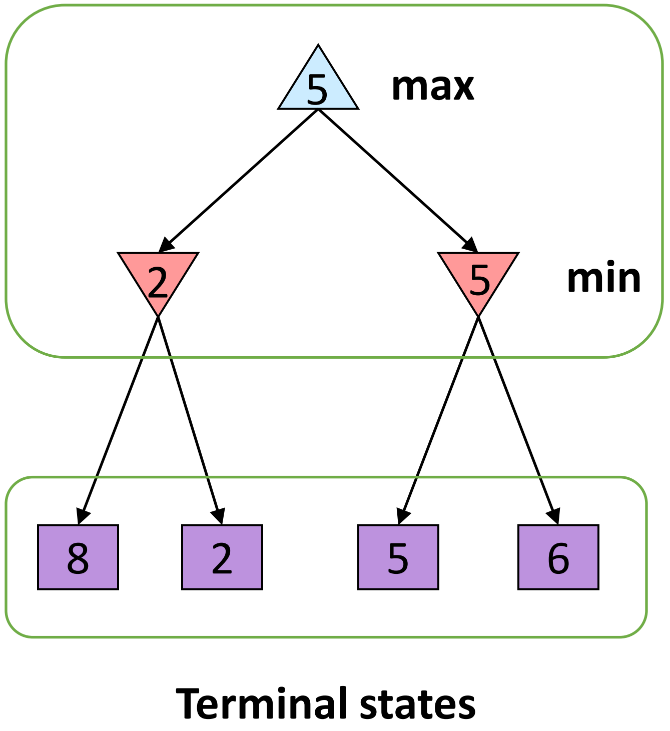
Minimax Algorithm

- Let’s consider player MIN
- 2 and 5 are the best that MIN can get



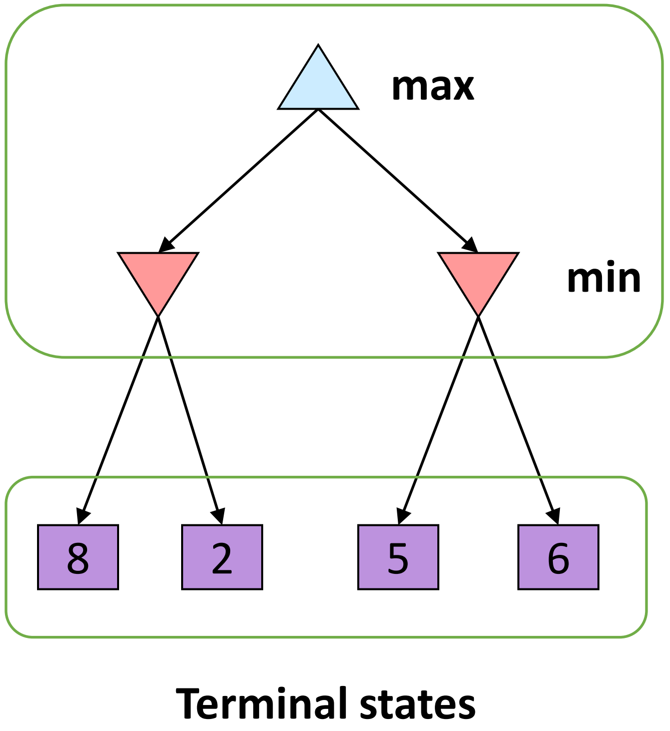
Minimax Algorithm

- Let's consider player MAX
- What action should MAX take?
  - The one that leads to a larger pay-off



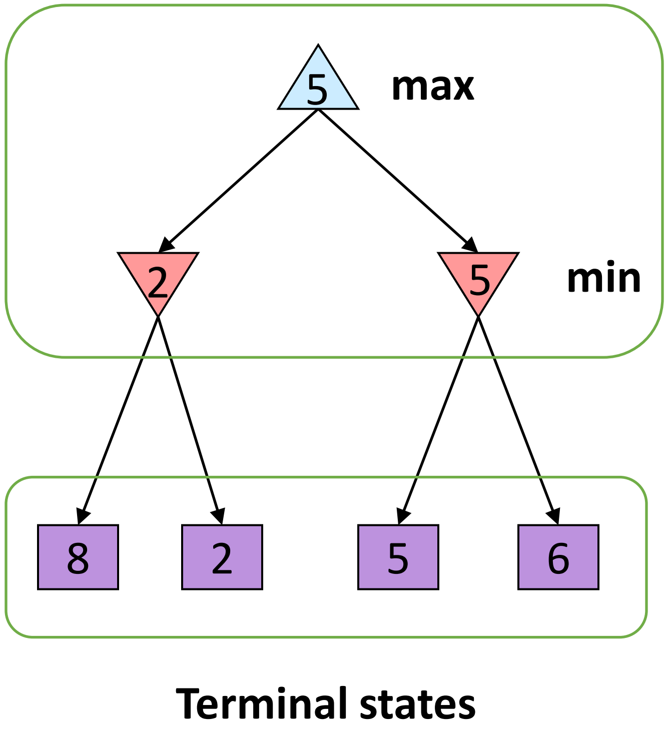
Minimax Algorithm

- Question: why doesn't MAX choose the left branch, given that there's a chance to get to the terminal state "8"?



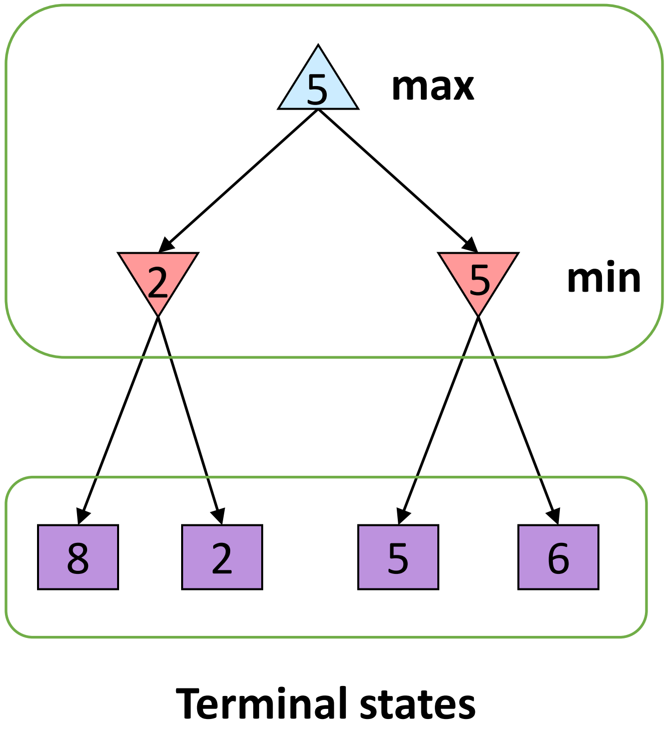
Minimax Algorithm

- Answer: We assume that MIN plays optimally
  - i.e. MIN is a rational agent



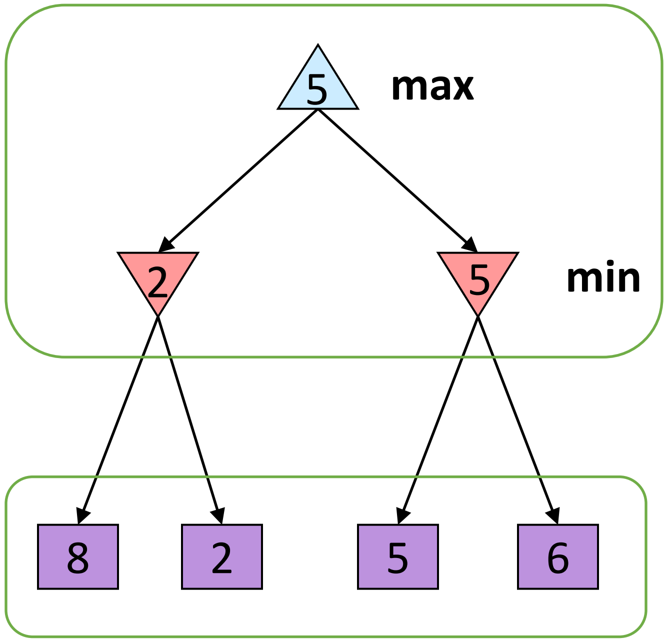
Minimax Algorithm

- Question: What if MIN does not play optimally?
- Answer: Then that's even better for MAX



Minimax Value

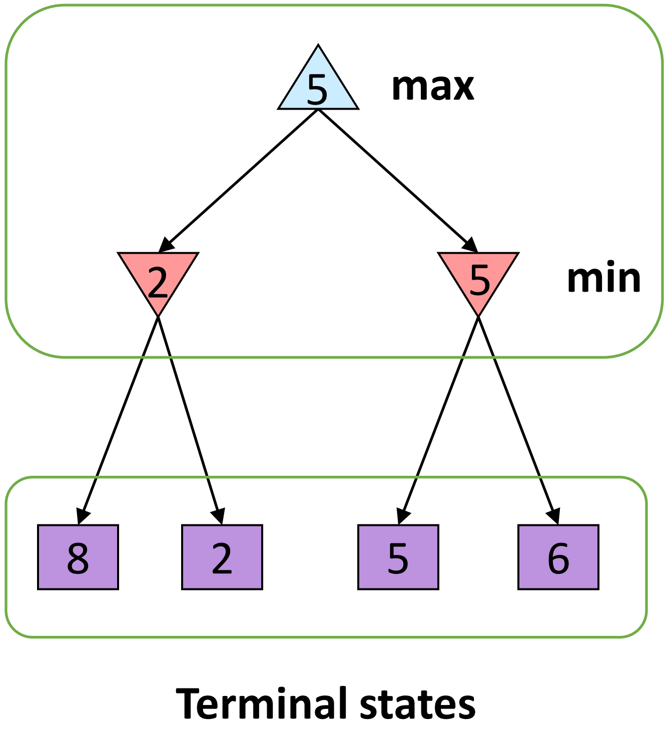
- Minimax value of state s: $\mathrm{minimax}(s)$
- $\mathrm{minimax}(s)$ values are computed recursively
  - Propagated from the leaf nodes to upper layers



Terminal states

Minimax Value

- $\text{minimax}(s)$: it is the pay-off value that the players will get when the game is over, assuming that both players play optimally from state-s to the end of the game



Minimax Value

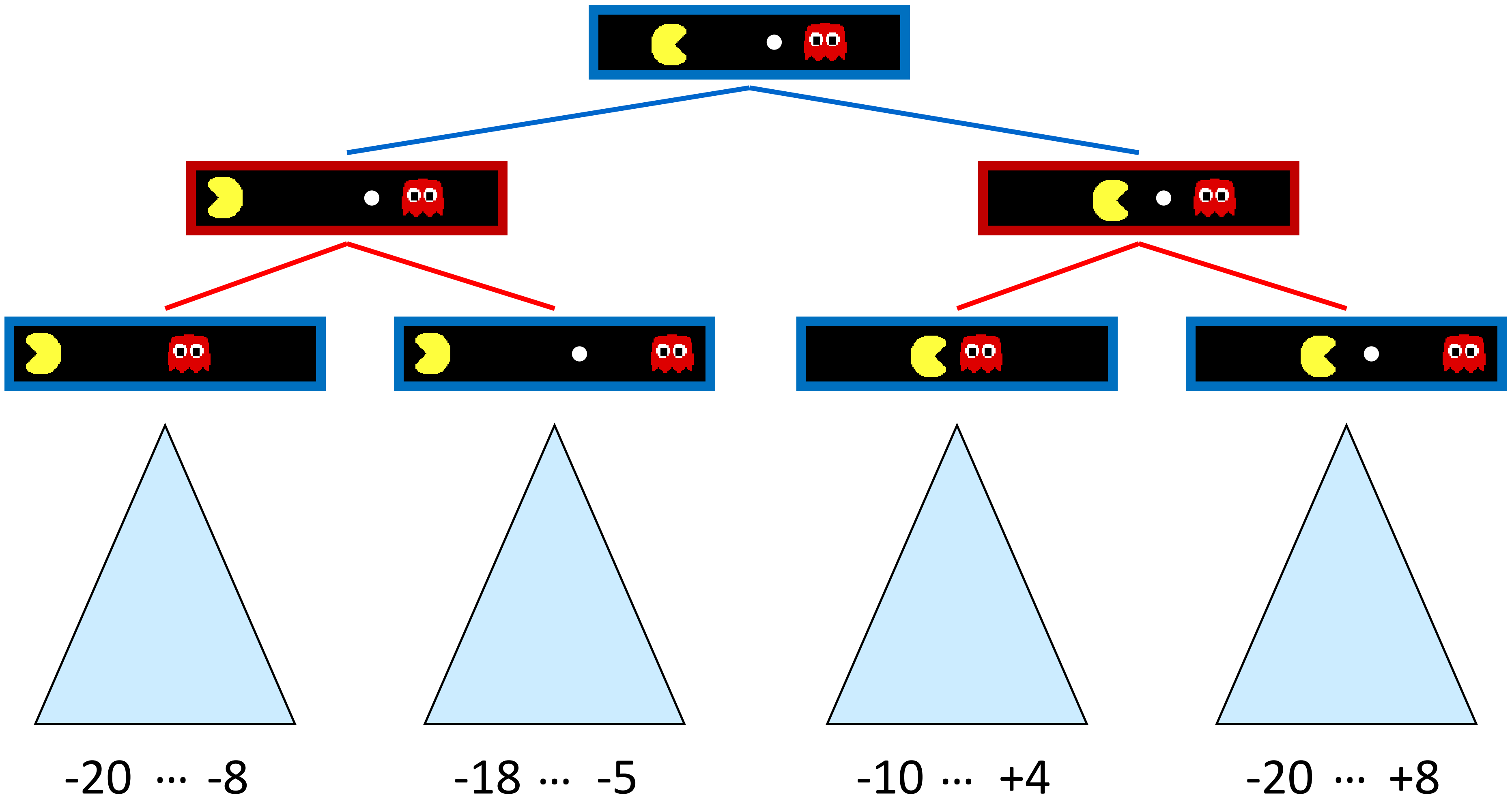
- s: current state
- s': next state, obtained from s'=Result(s, a), a is any possible action that can be taken at s

  - If s is a terminal state, then minimax(s)=Utility(s)
  - If Player(s) = MAX, then

    Minimax(s) = $\max_{s'}$ (Minimax(s'))
  - If Player(s) = MIN, then

    Minimax(s) = $\min_{s'}$ (Minimax(s'))

Adversarial Game Trees



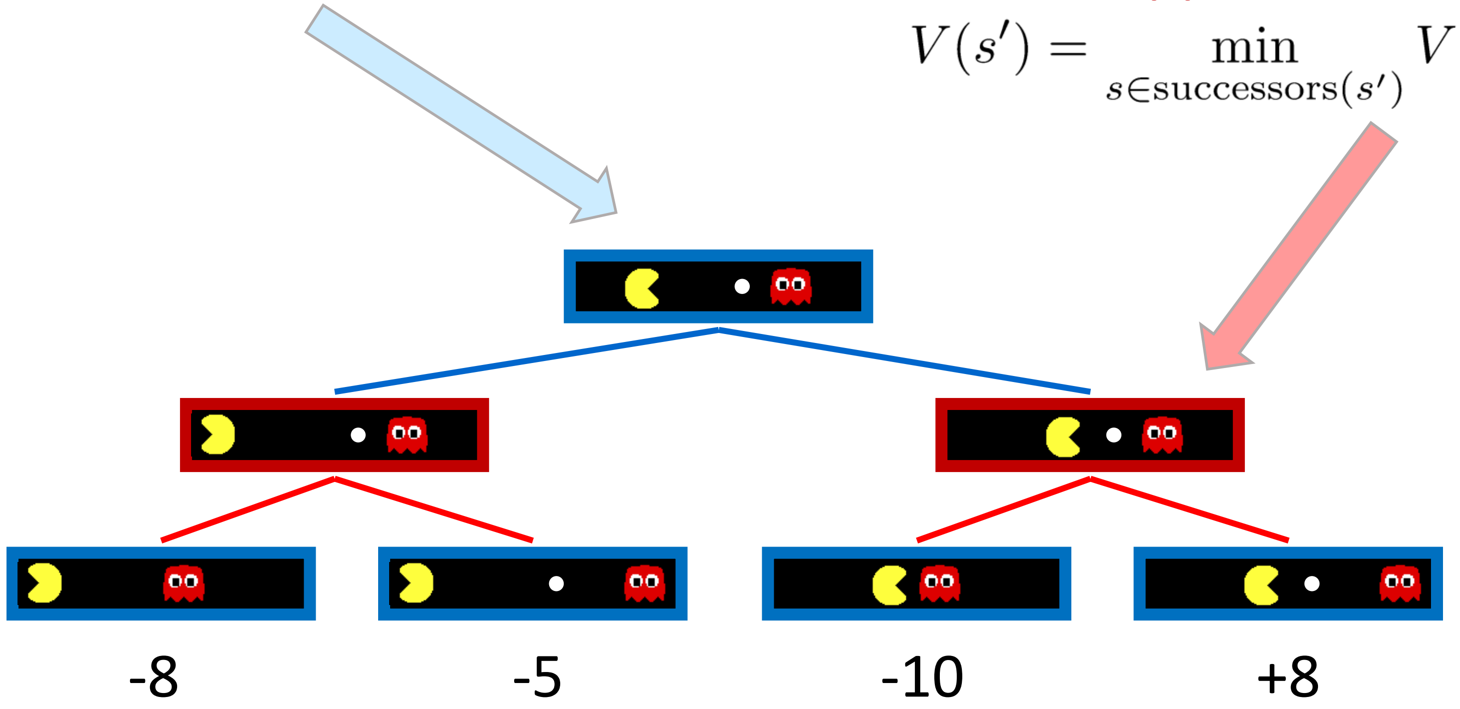
Minimax Values

States Under Agent's Control:

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

States Under Opponent's Control:

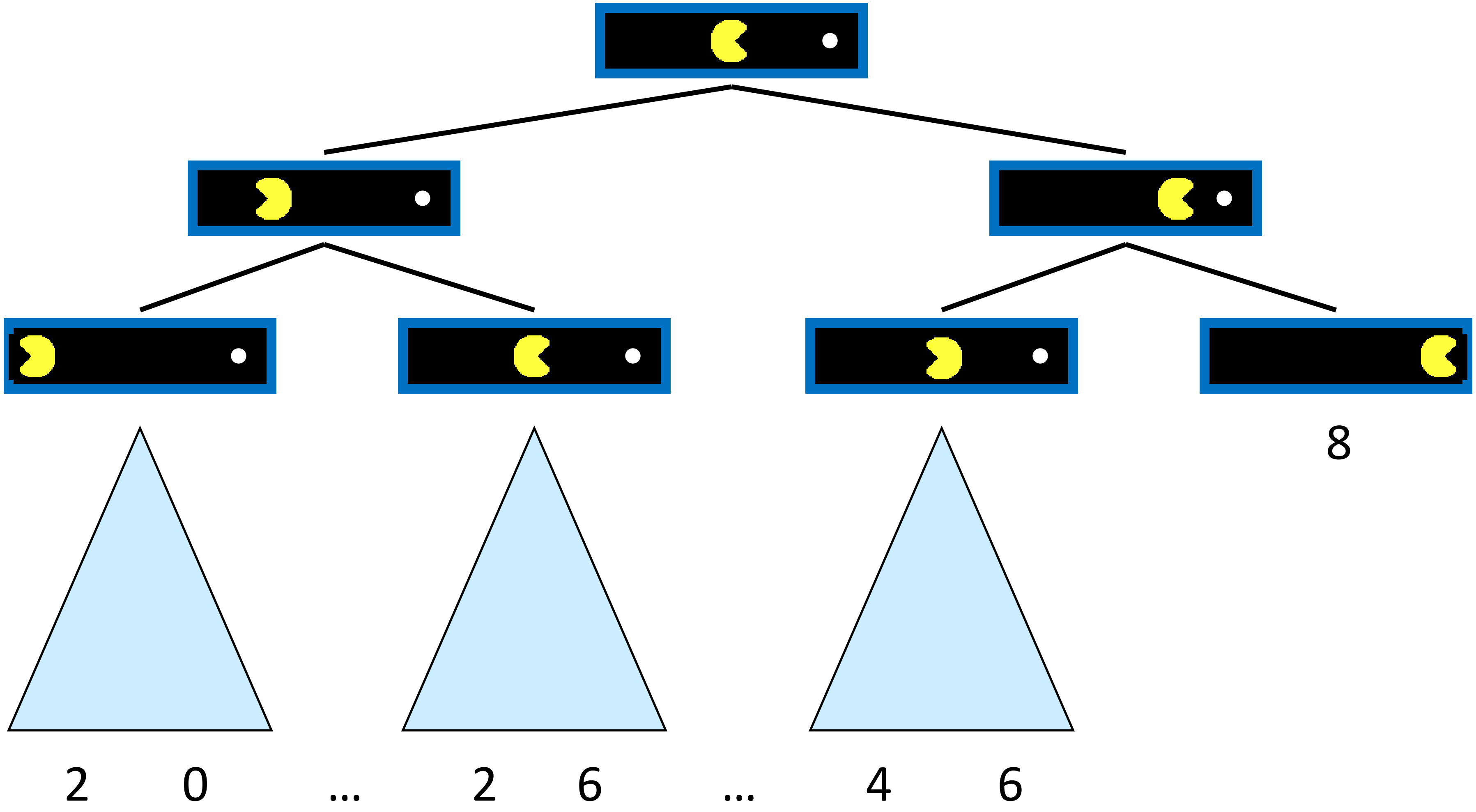
$$V(s') = \min_{s \in \text{successors}(s')} V(s)$$



Terminal States:

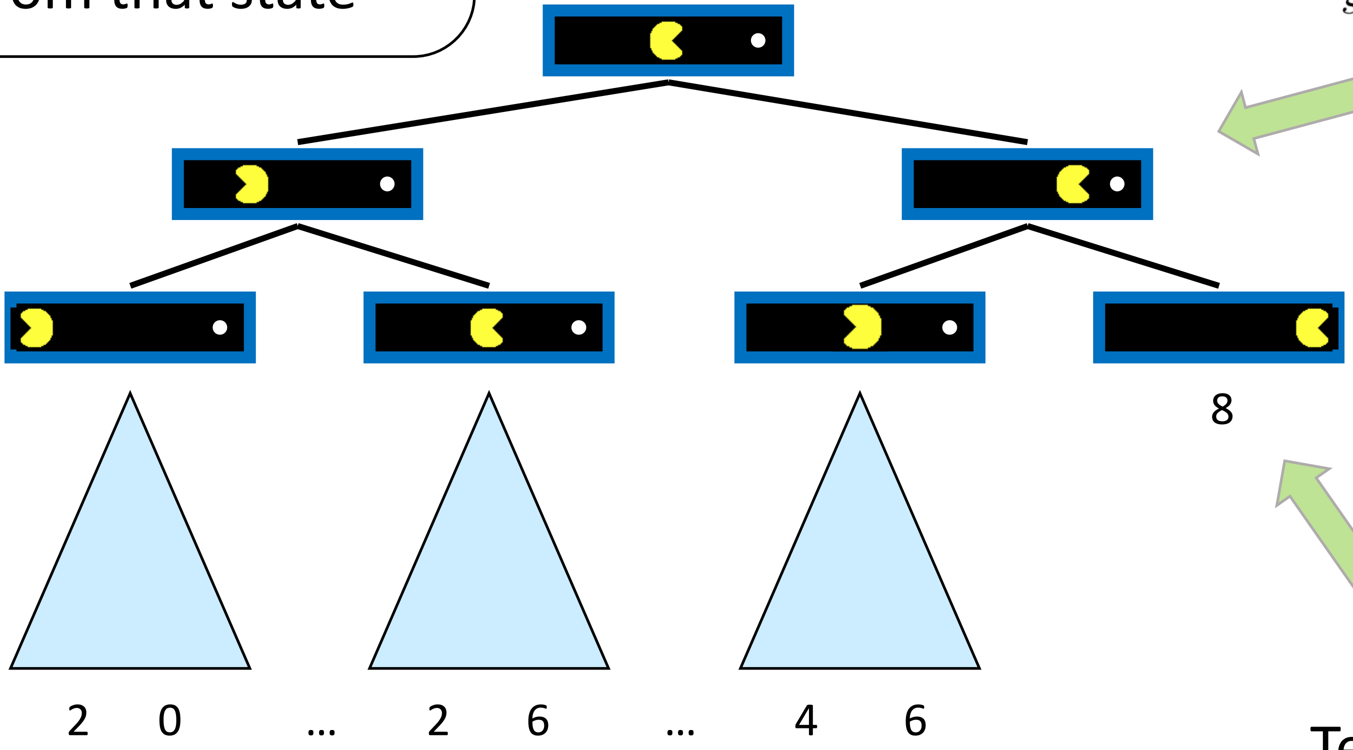
$$V(s) = \text{known}$$

Single-Agent Trees



Value of a State

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



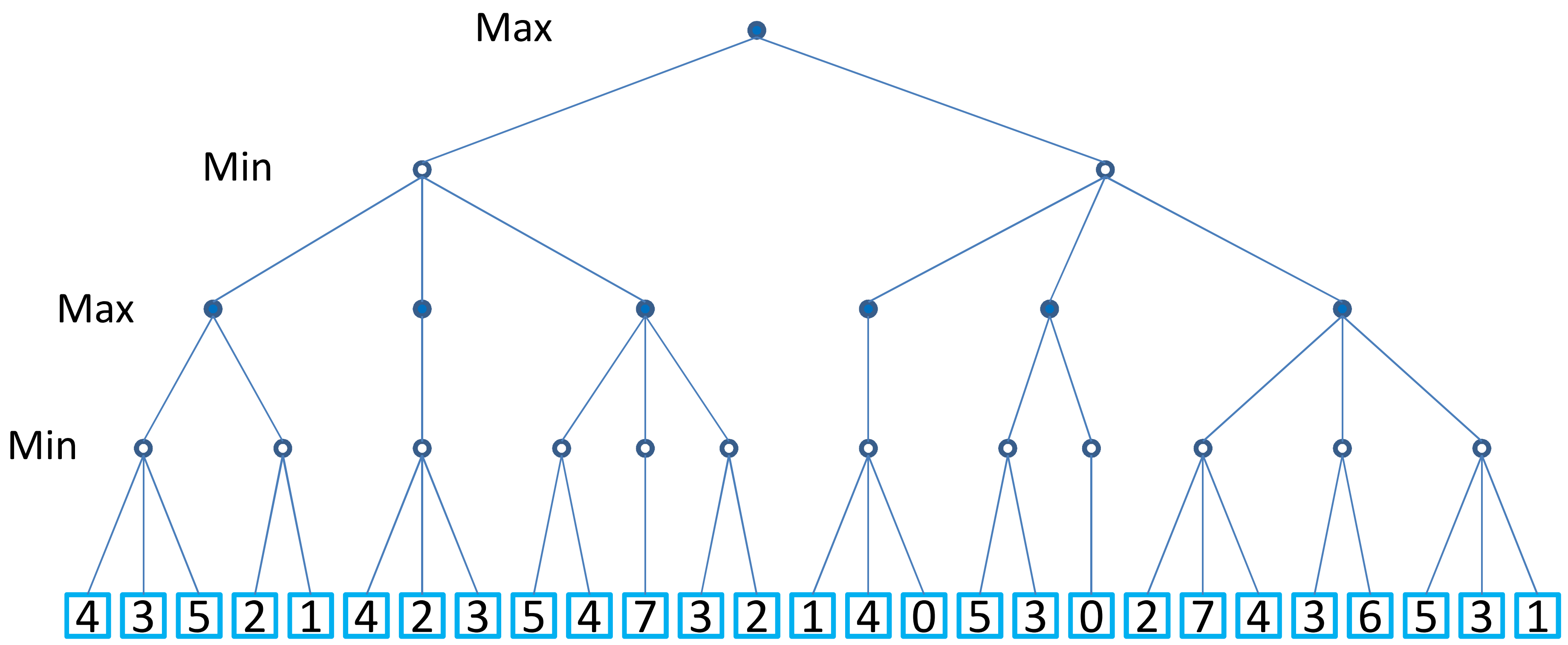
Non-Terminal States:

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

Terminal States:

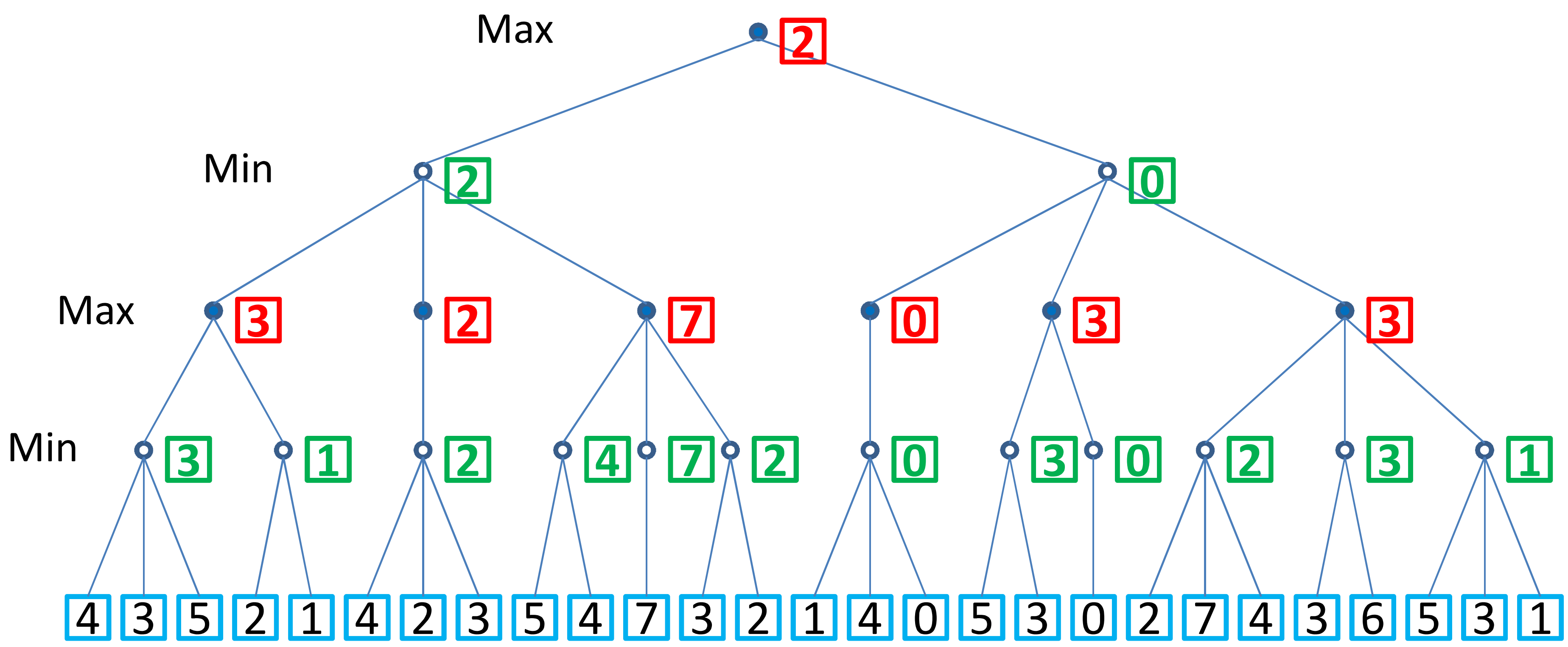
$$V(s) = \text{known}$$

Minimax Algorithm Example



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

Minimax Algorithm Example

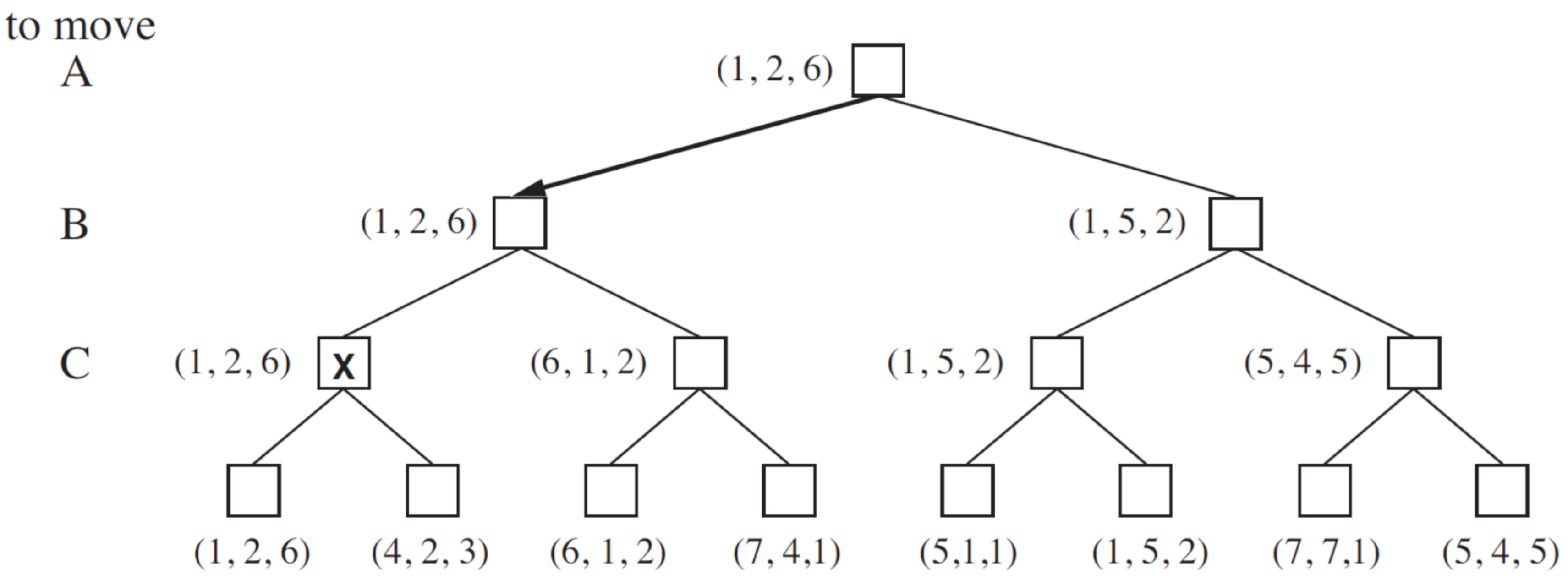


The minimax value of each node: determined by passing values recursively from terminal states to upper layers

Multiplayer Games

- Extend the minimax algorithm to multiplayer games
  - Three-player game
  - Players A, B, C
  - Minimax value: a vector (v_A, v_B, v_C)
  - For terminal states, each element in this vector gives the utility of the state from corresponding player’s viewpoint
    - e.g. (20, 5, 10) at a terminal state means player A’s payoff is 20, player B’s payoff is 5, and player C’s payoff is 10.
  - Each player takes actions to maximize its own value

Multiplayer Games



The minimax value of each node is a 3-tuple

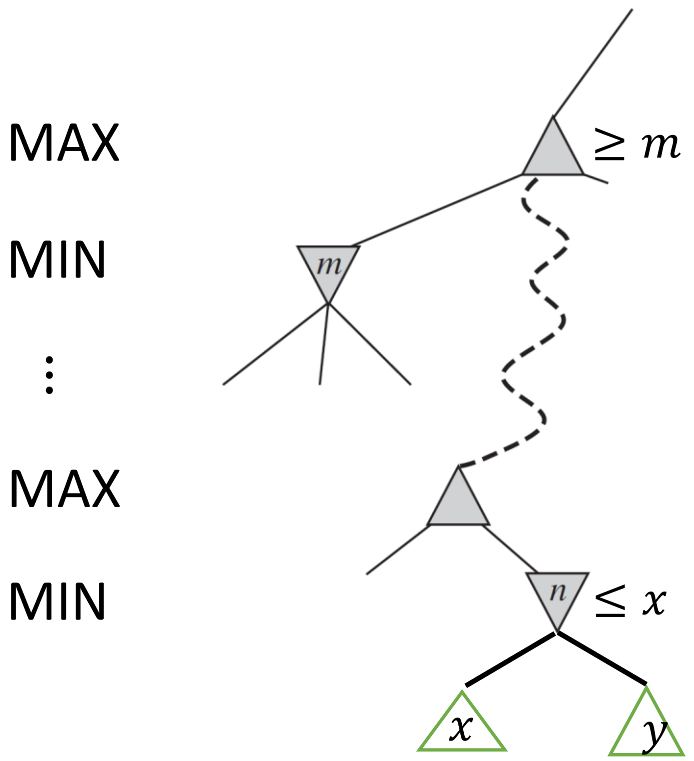
Alpha-Beta Pruning

- Evaluating the whole game tree is cumbersome

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

Alpha-Beta Pruning

- If we have found enough information to infer that $m > n$, then we don't need to evaluate the node with value n, because we will never get to the node with value n in play.



Assume m and x have been visited
If $m > x$,
then y and n don't need to be visited
That is, we don't need to know
the exact values of y and n

The General Case

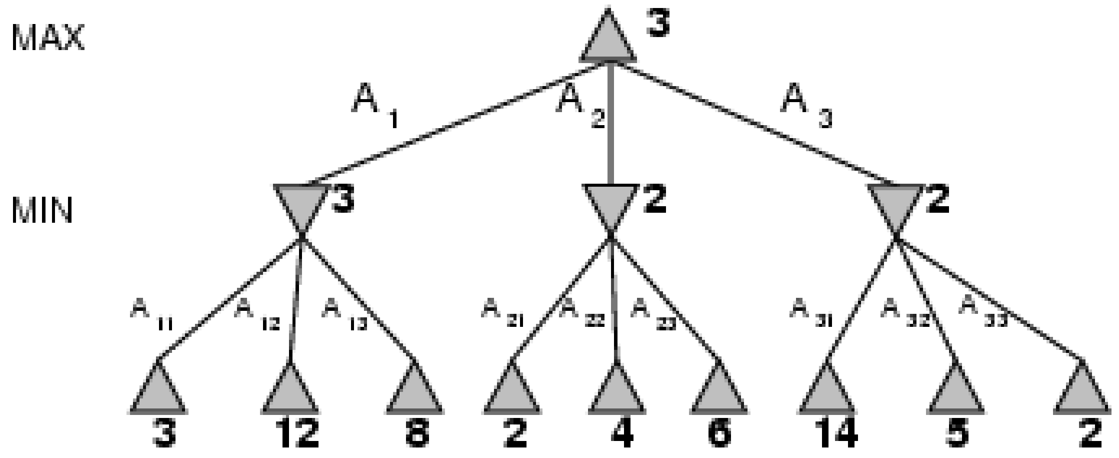
- For each node, two values are assigned

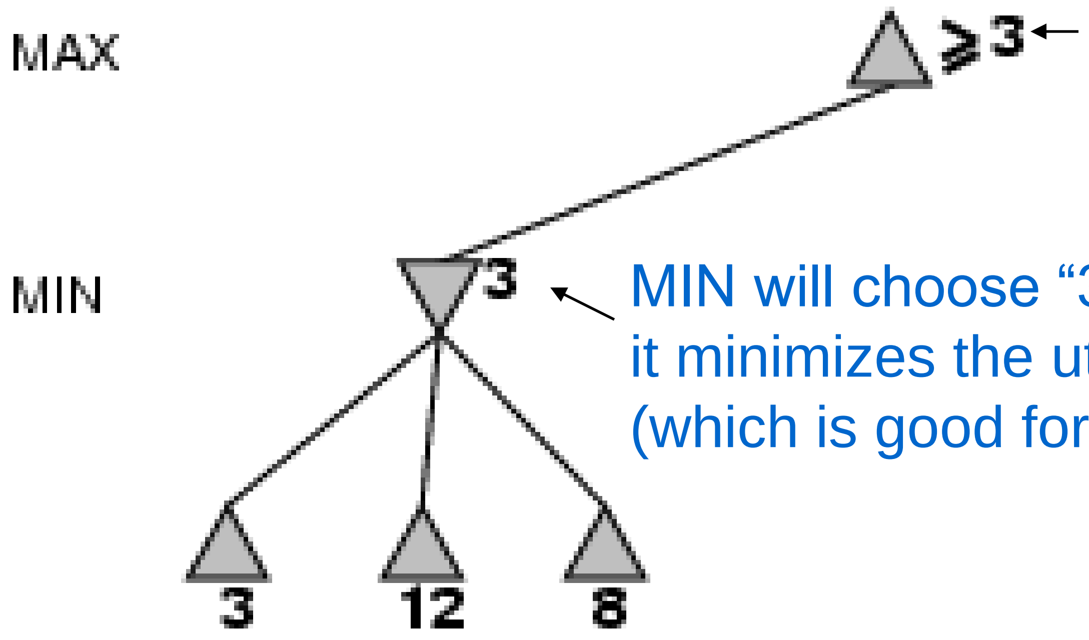
$$[\alpha, \beta]$$

- 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

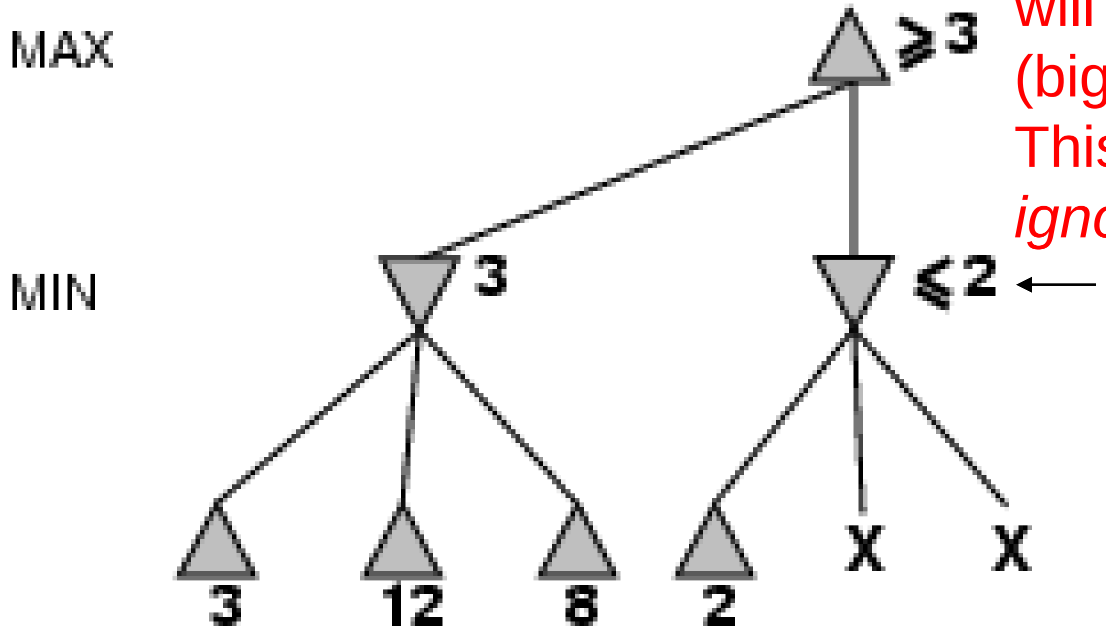




MAX knows that it can at least get “3” by playing this branch

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

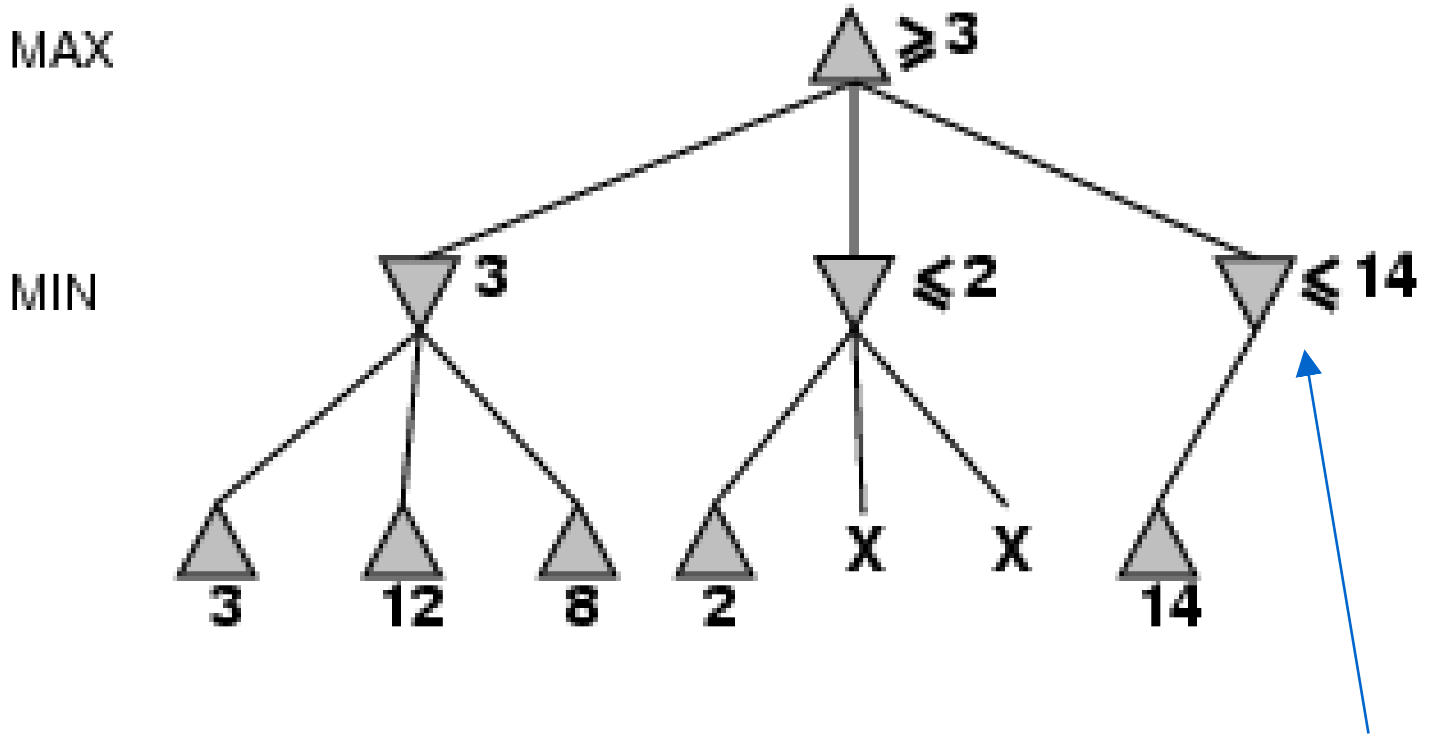
α-β Pruning Example



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

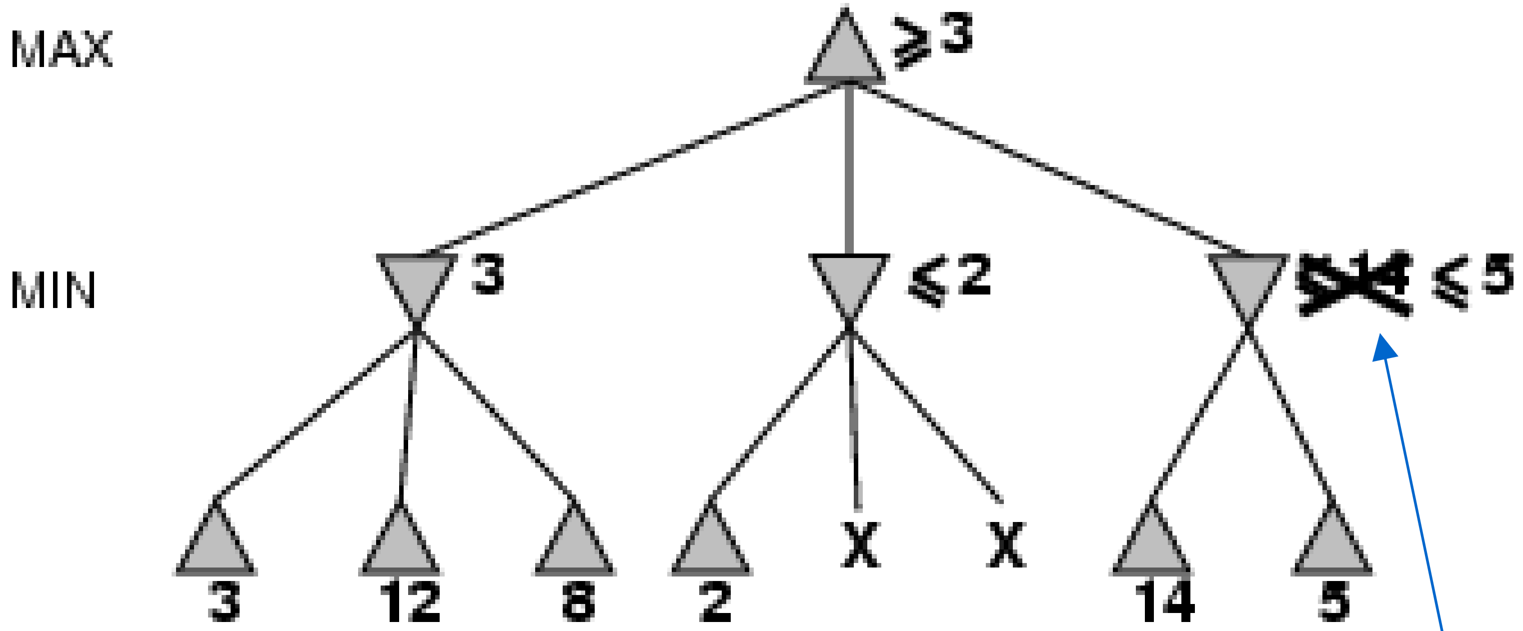
MIN can certainly do as good as 2, but maybe better (i.e., smaller)

α-β Pruning Example



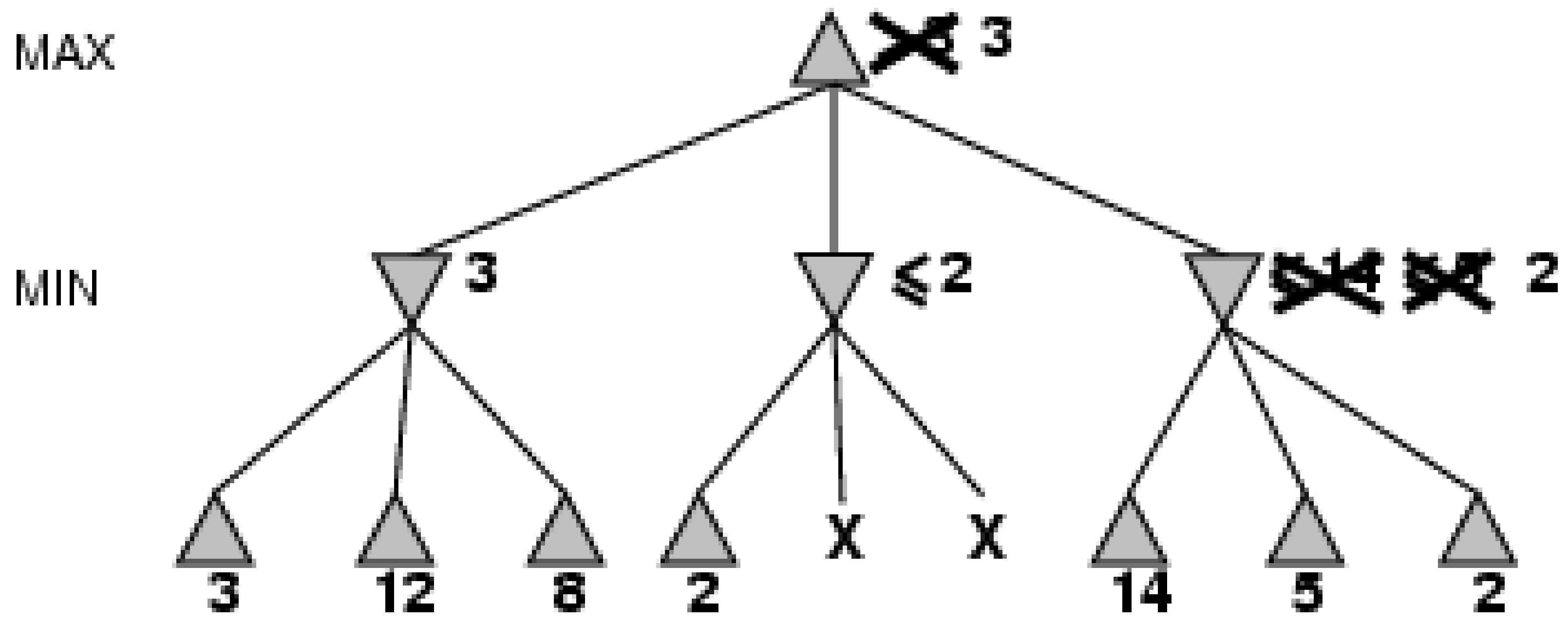
MIN will do at least as good as 14 in this branch (which is very good for MAX!) so MAX will want to explore this branch more.

α-β Pruning Example



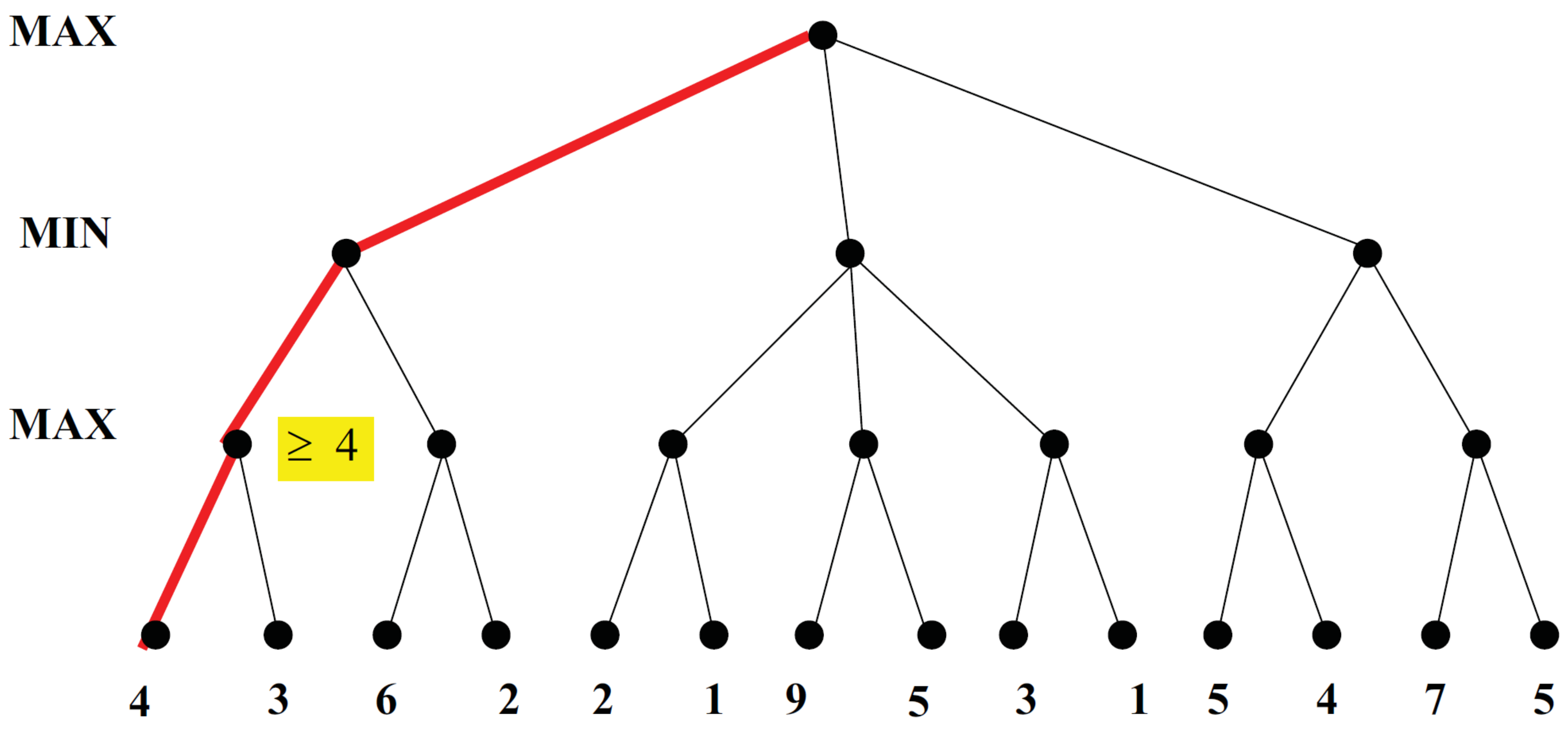
MIN will do at least as good as 5 in this branch (which is still good for MAX) so MAX will want to explore this branch more.

α-β Pruning Example

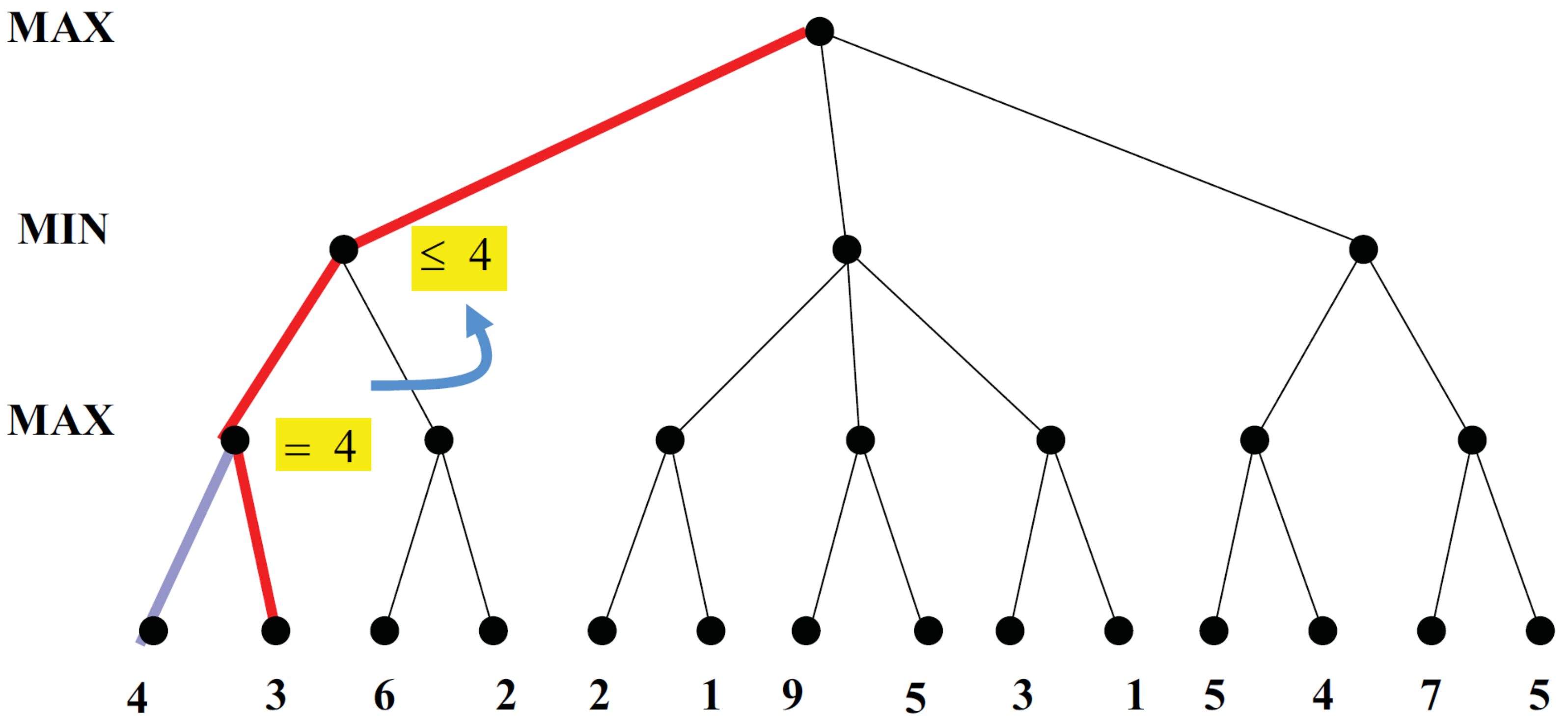


Bummer (for MAX): MIN will be able to play this last branch and get 2. This is worse than 3, so MAX will play 3.

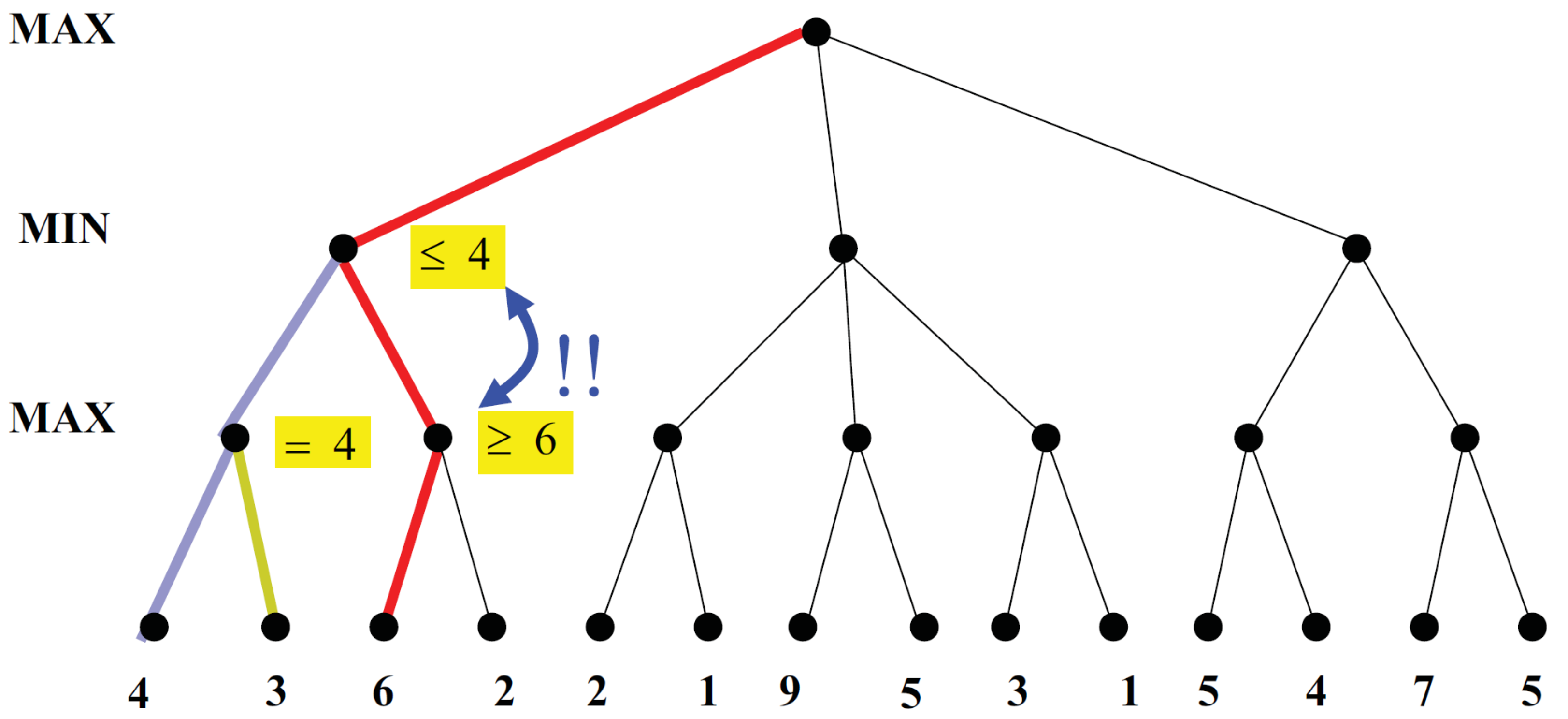
α-β Pruning Example



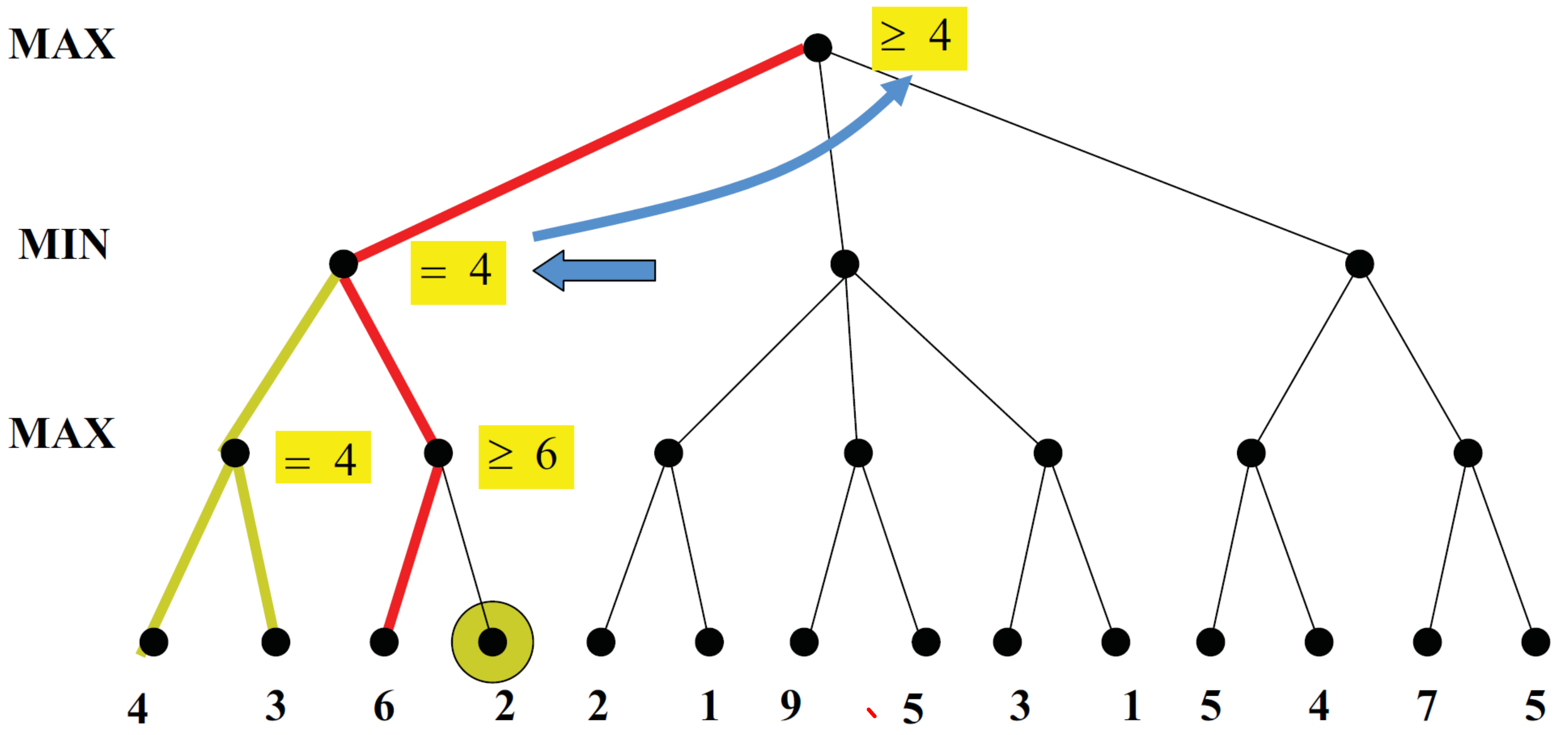
α-β Pruning Example



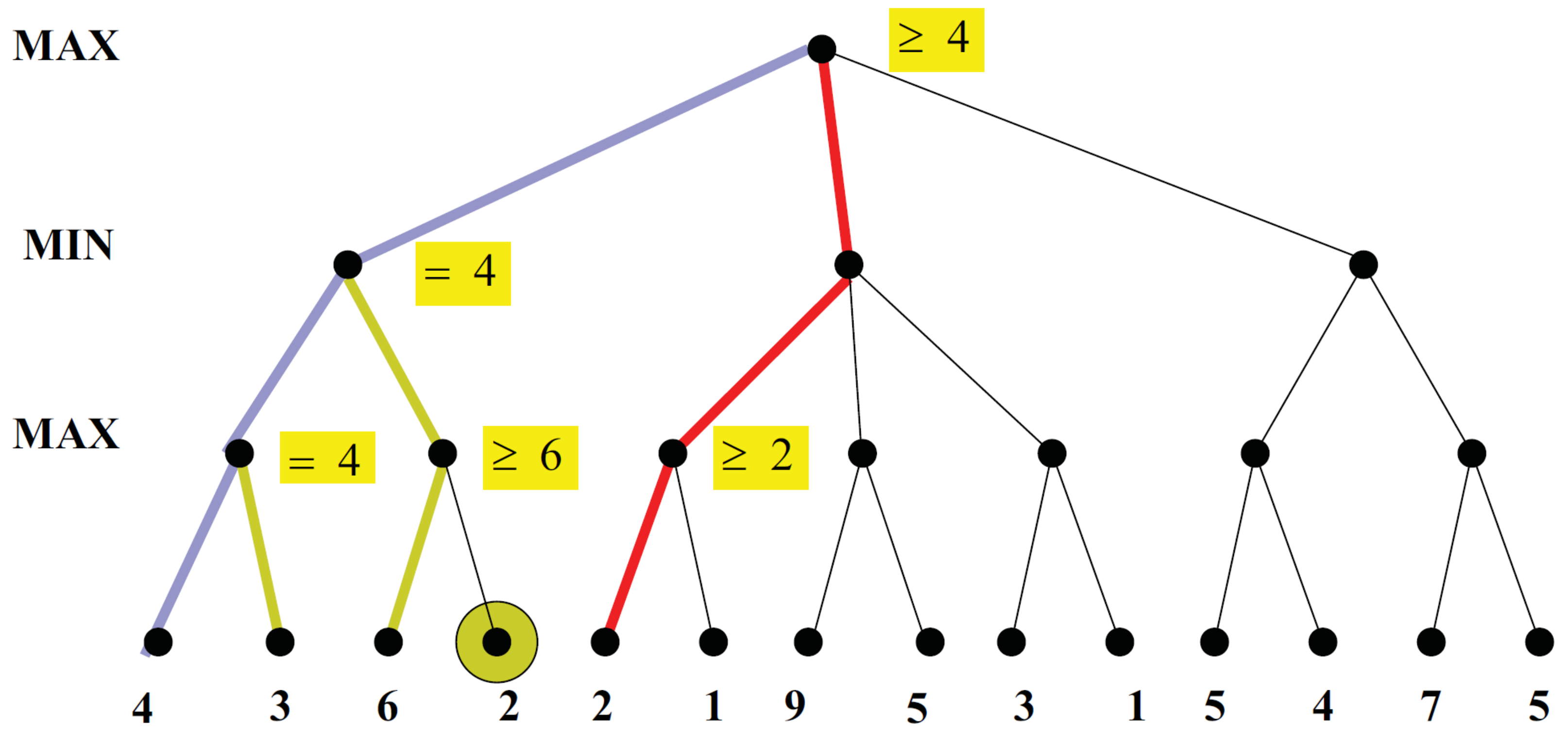
α-β Pruning Example



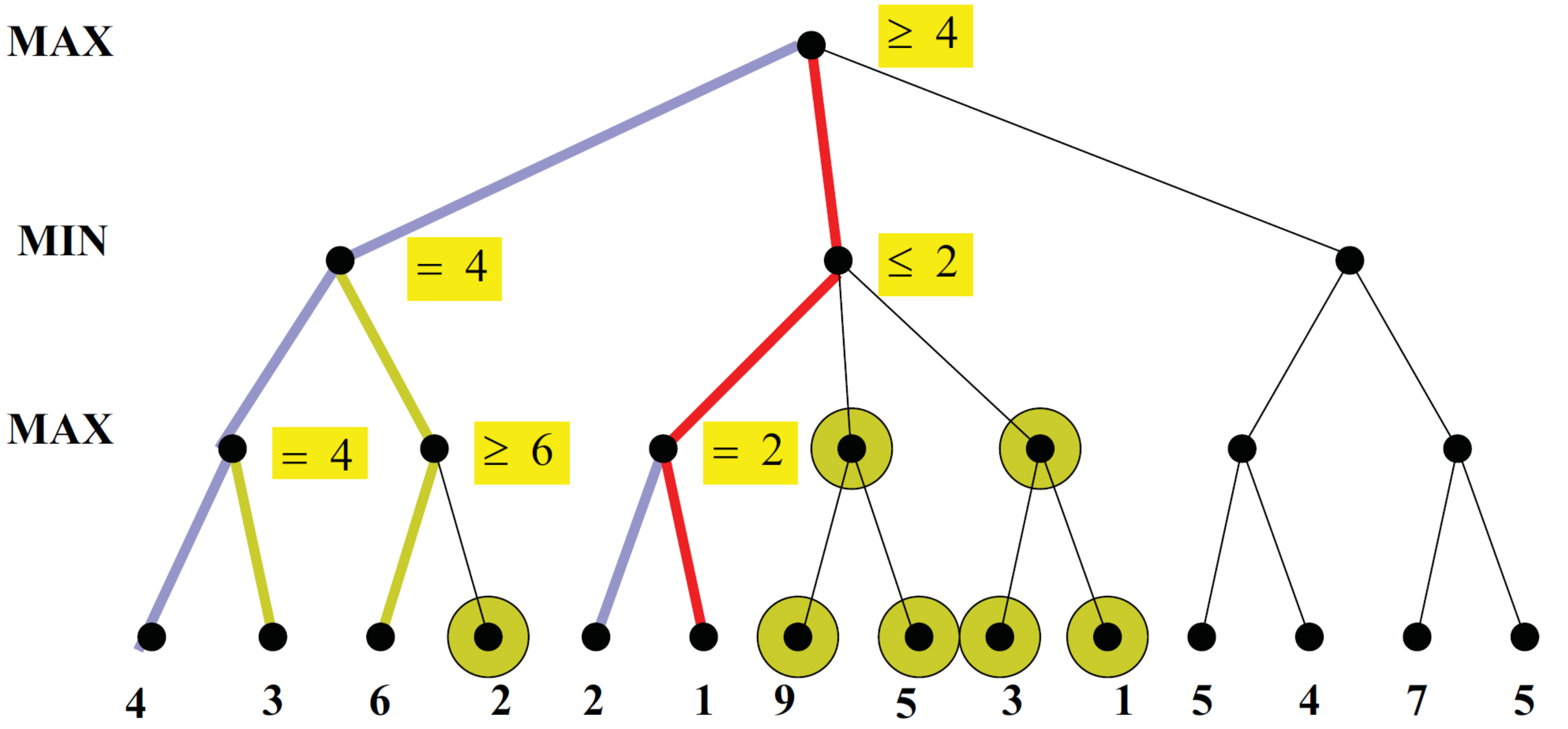
α-β Pruning Example



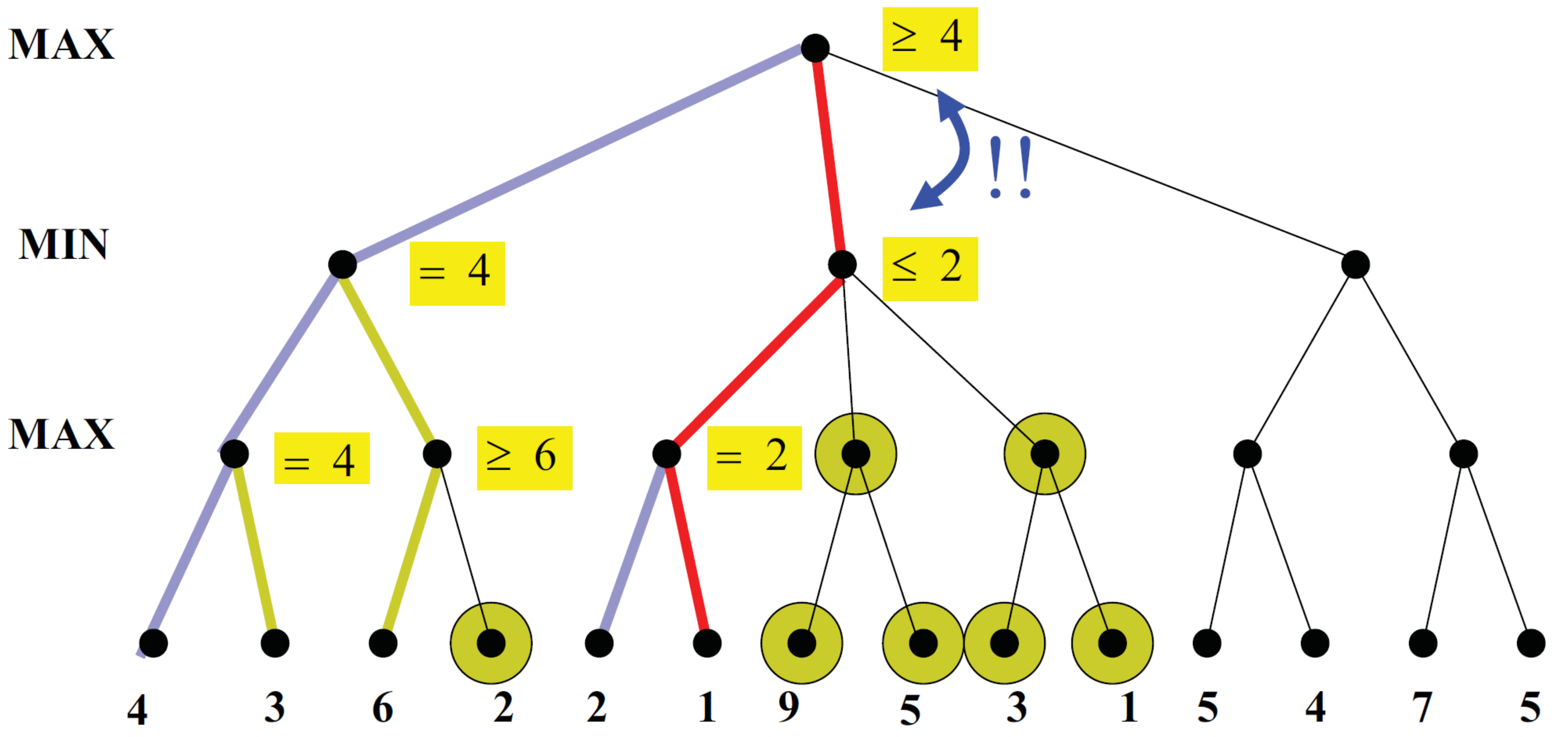
α-β Pruning Example



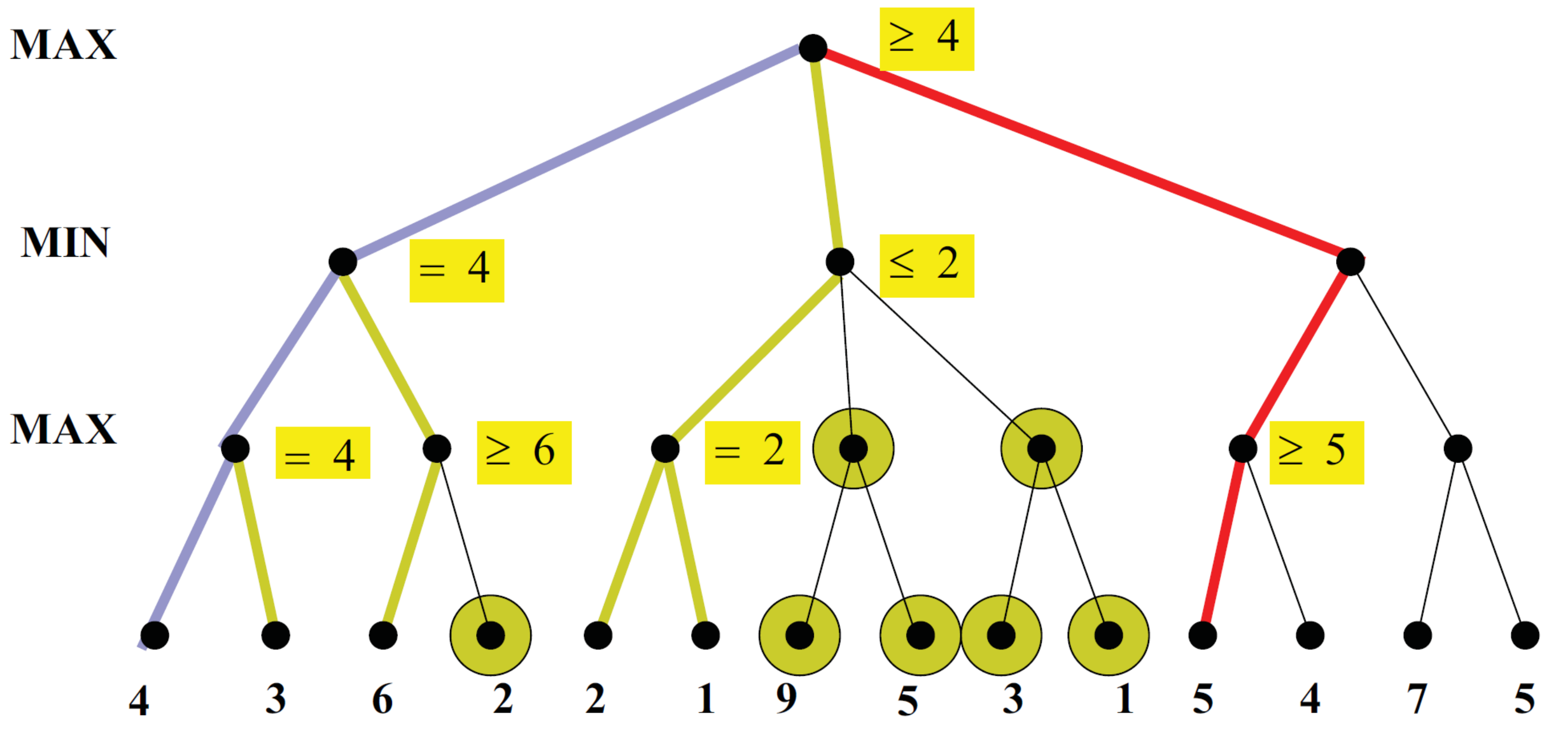
α-β Pruning Example



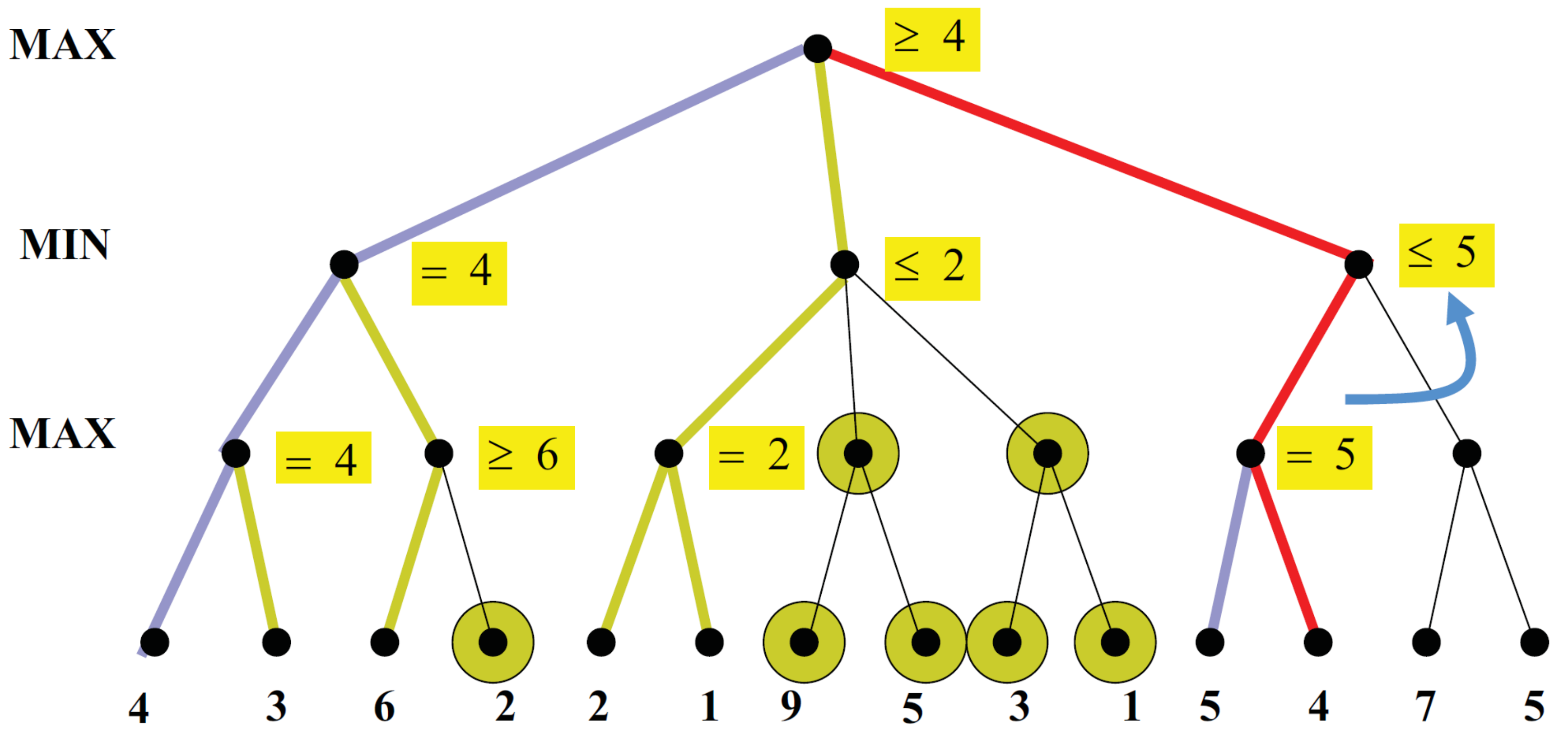
α-β Pruning Example



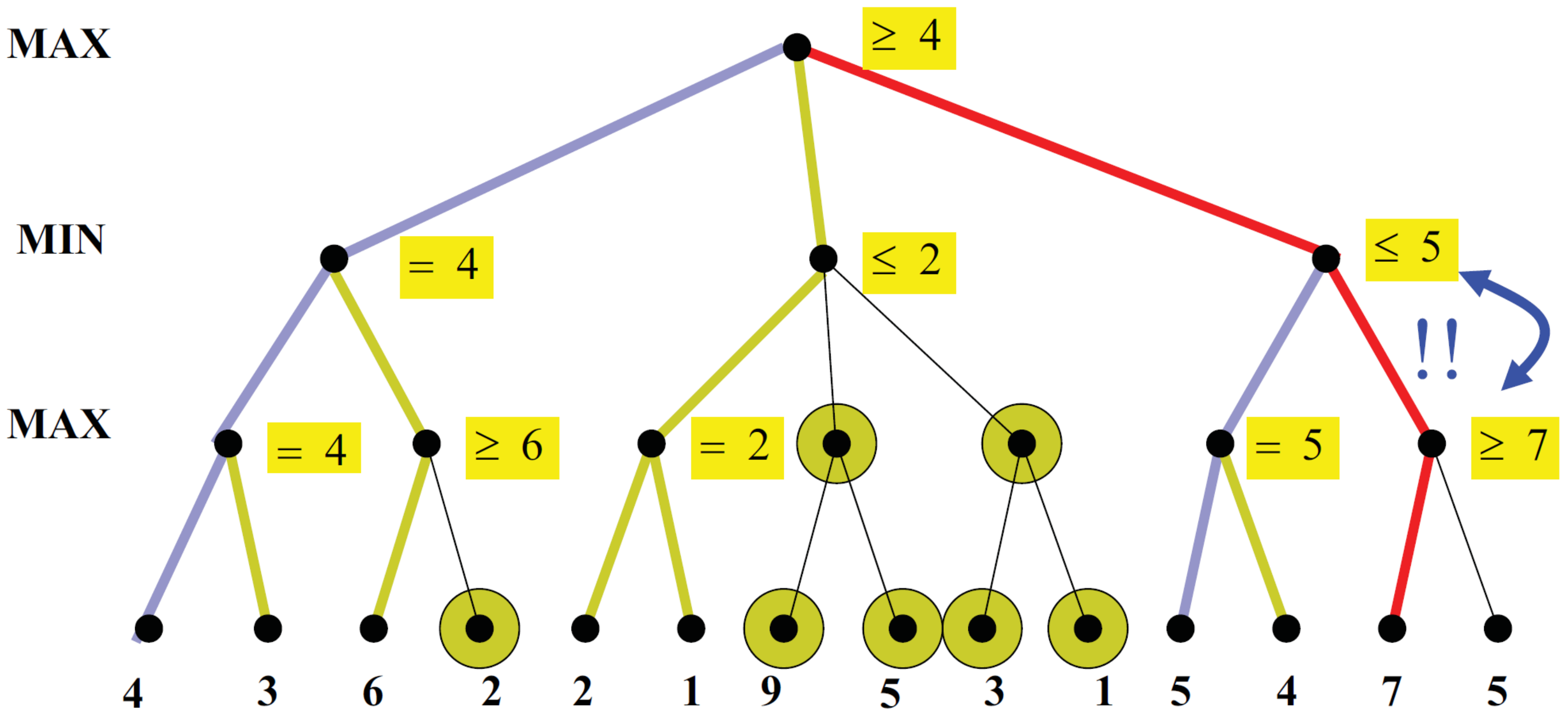
α-β Pruning Example



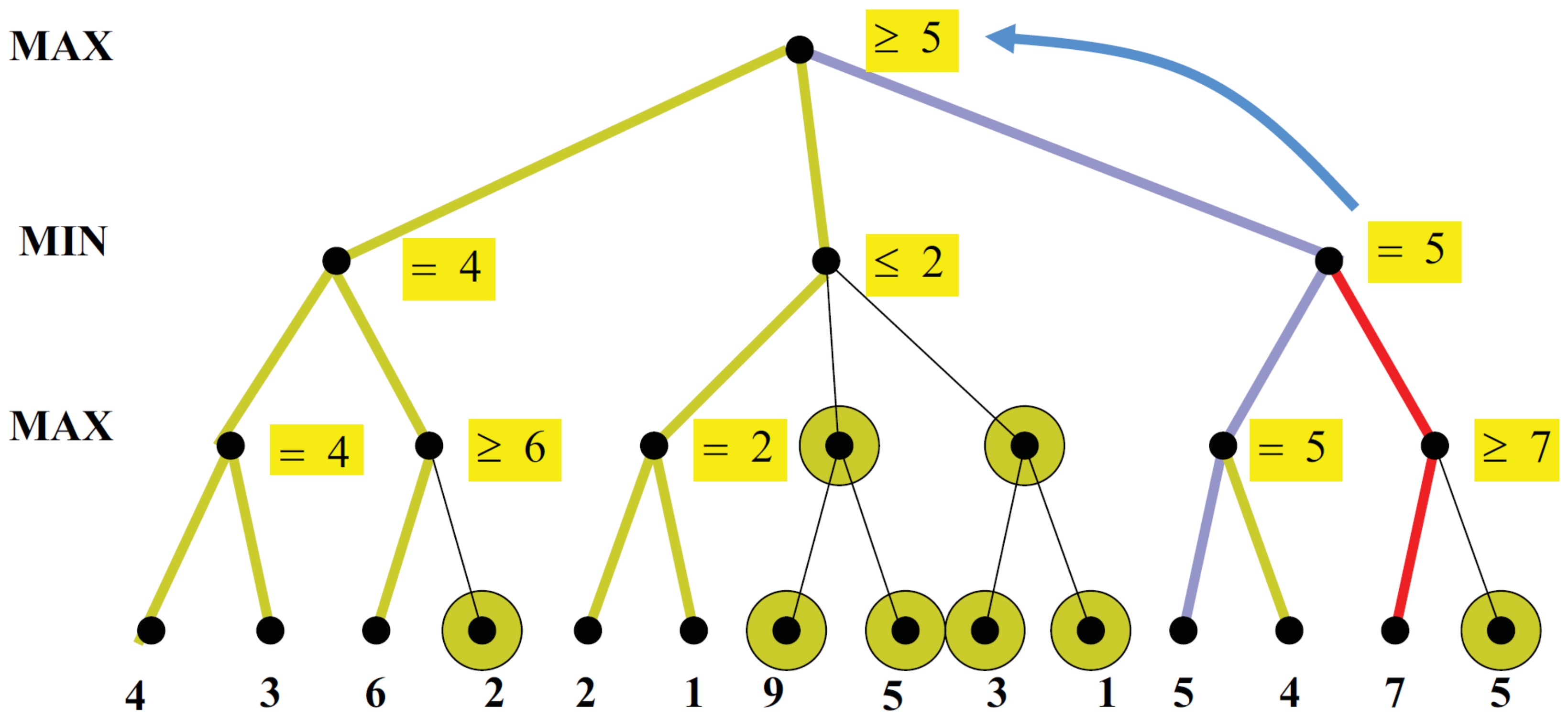
α-β Pruning Example



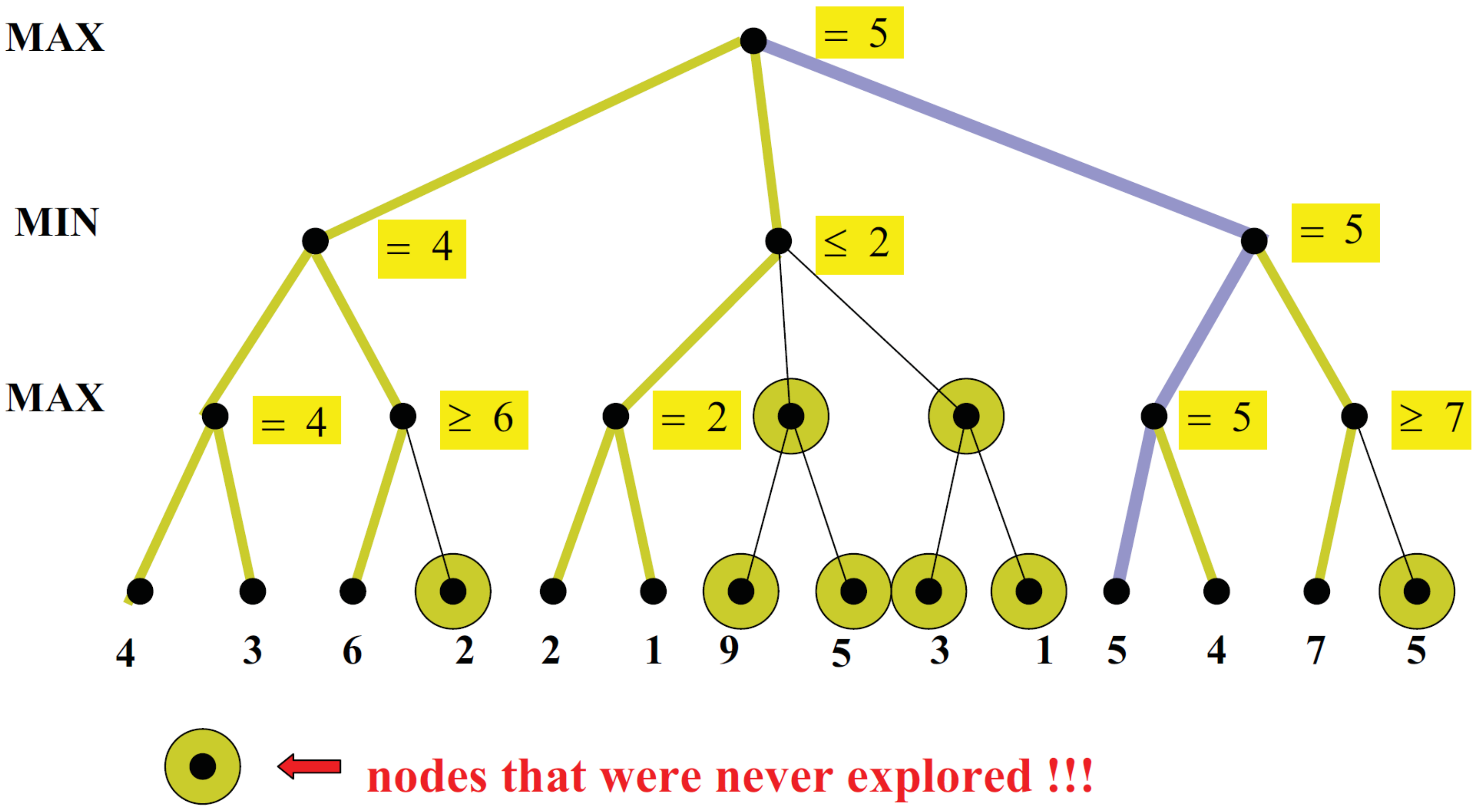
α-β Pruning Example



α-β Pruning Example



α-β Pruning Example

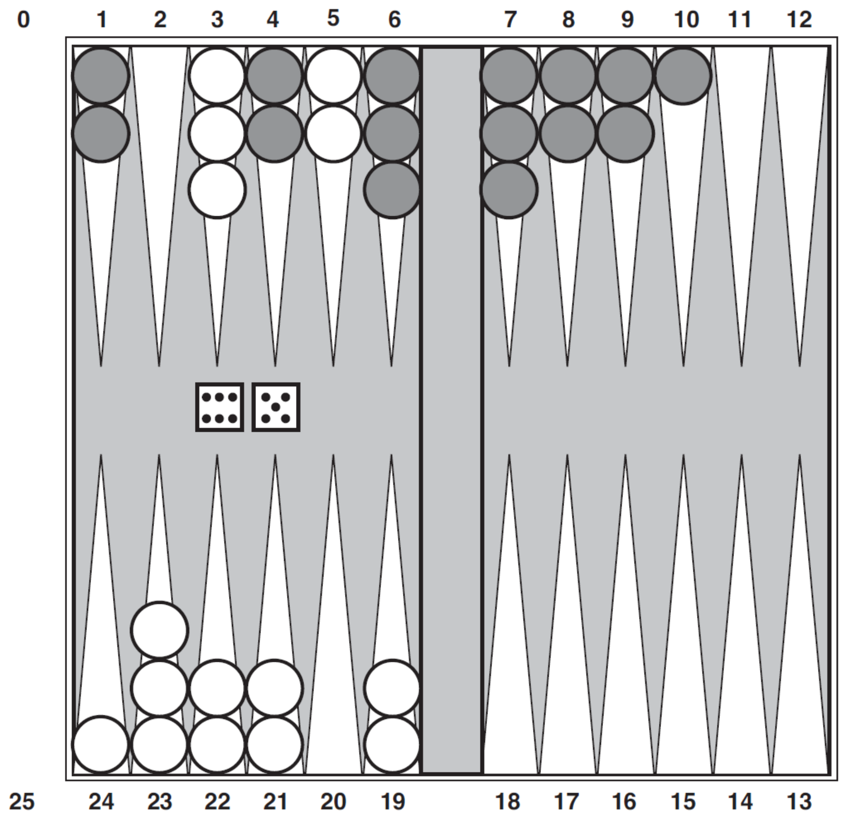


Properties of α-β Pruning

- Pruning does not affect the final result (the sequence of actions is the same as that found without pruning).

Stochastic Games

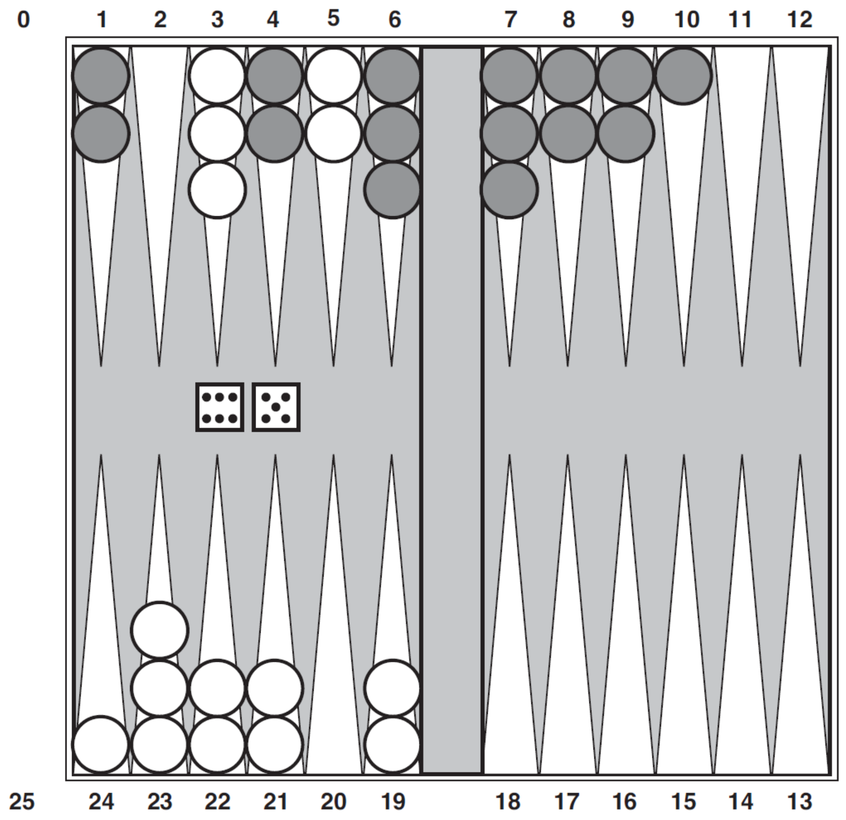
- Backgammon
  - Each player rolls dice to determine the legal moves



- 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
- Goal: move all one's checkers off the board

Stochastic Games

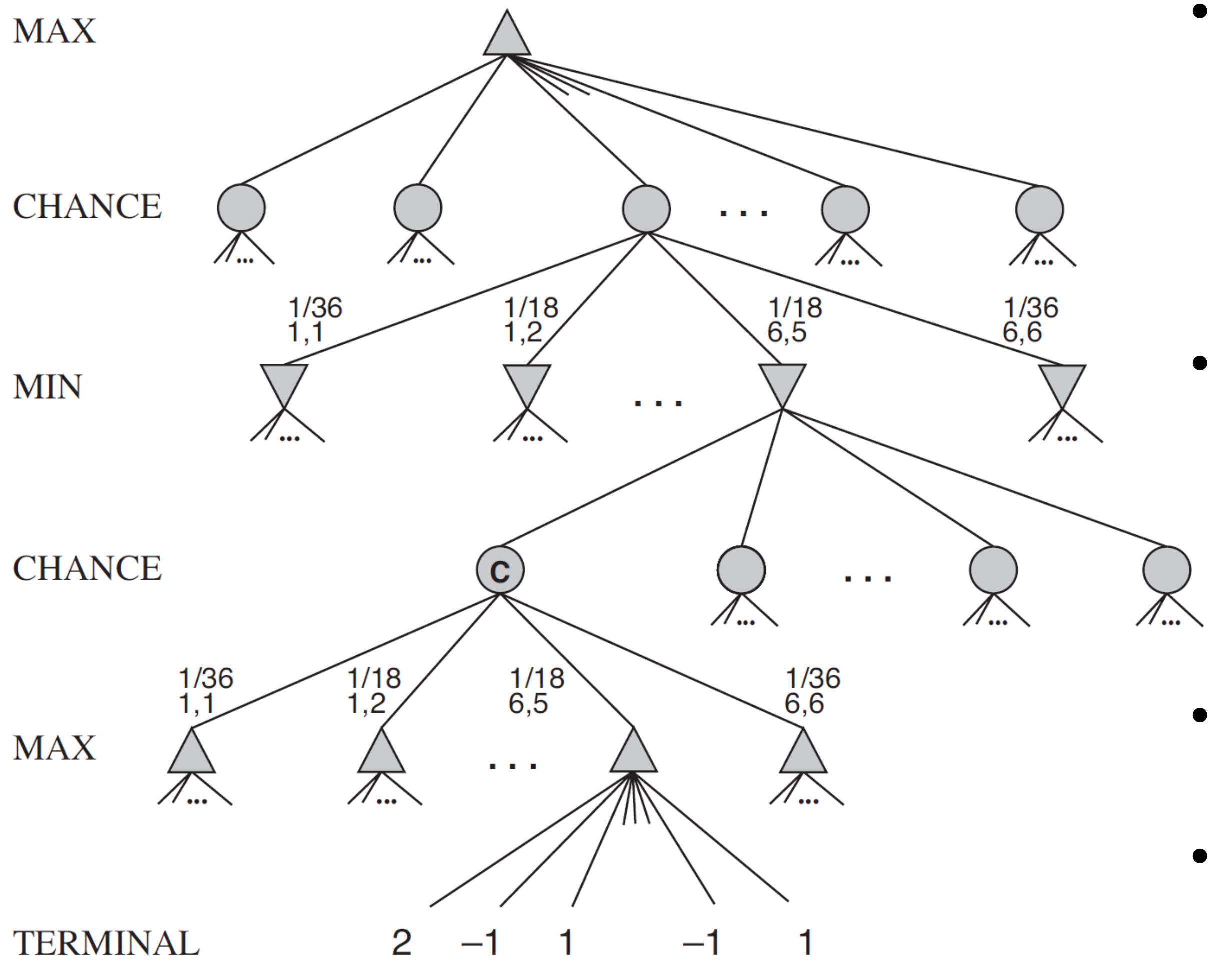
- Backgammon
  - Each player rolls dice to determine the legal moves



- 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
- We cannot construct a standard game tree

Stochastic Games

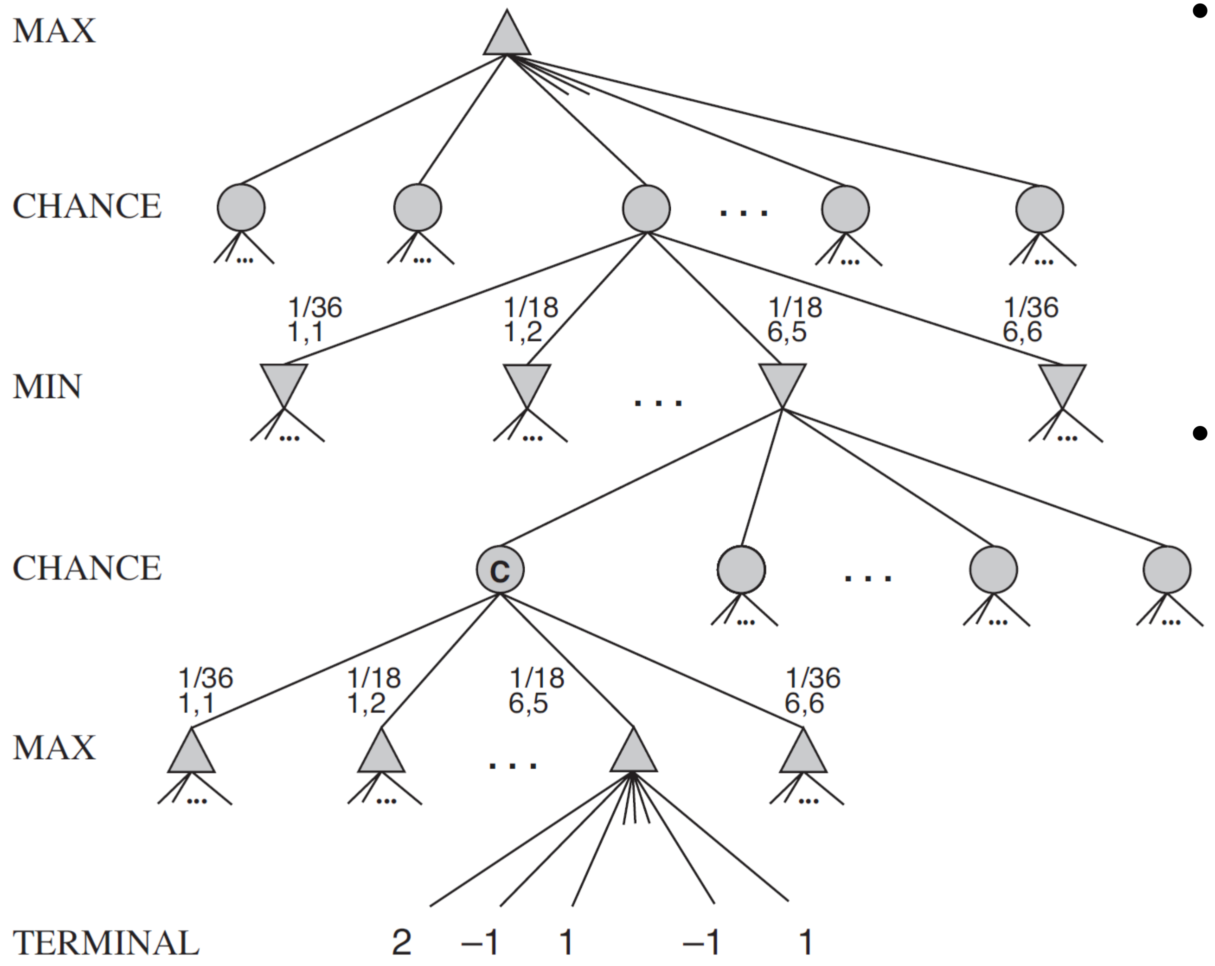
- **Chance Nodes:** Circles in this figure



- 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



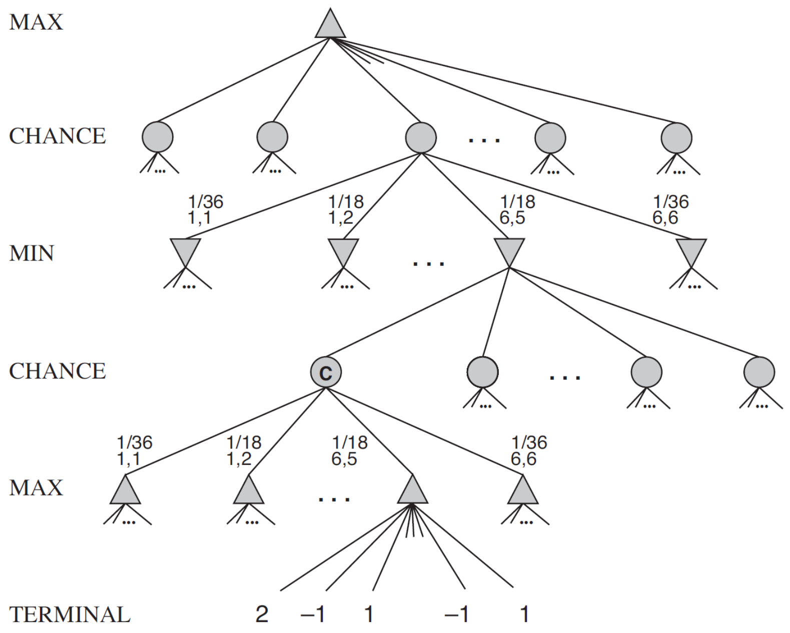
- The six doubles (1-1 through 6-6): each has a probability of 1/36
- The other 15 distinct rolls each has a 1/18 probability

Stochastic Games

- Expecti-minimax values
  - The minimax values for nodes in a game tree with chance nodes
- For terminal nodes, MAX and MIN nodes: this value is calculated in 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

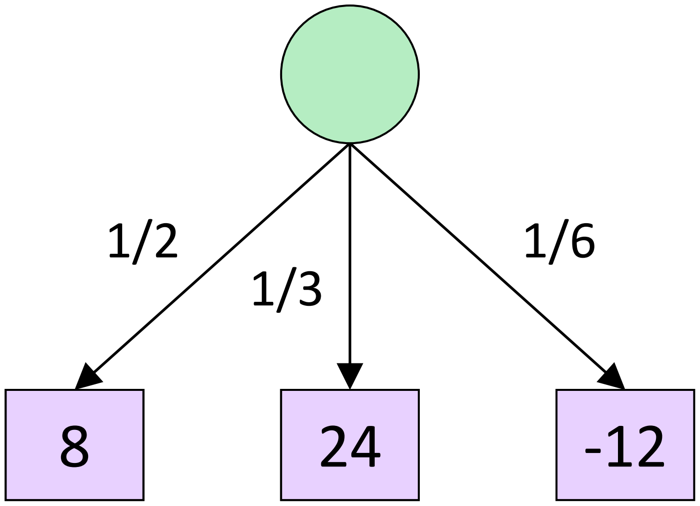
- Calculate the expecti-minimax value



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

Example 1

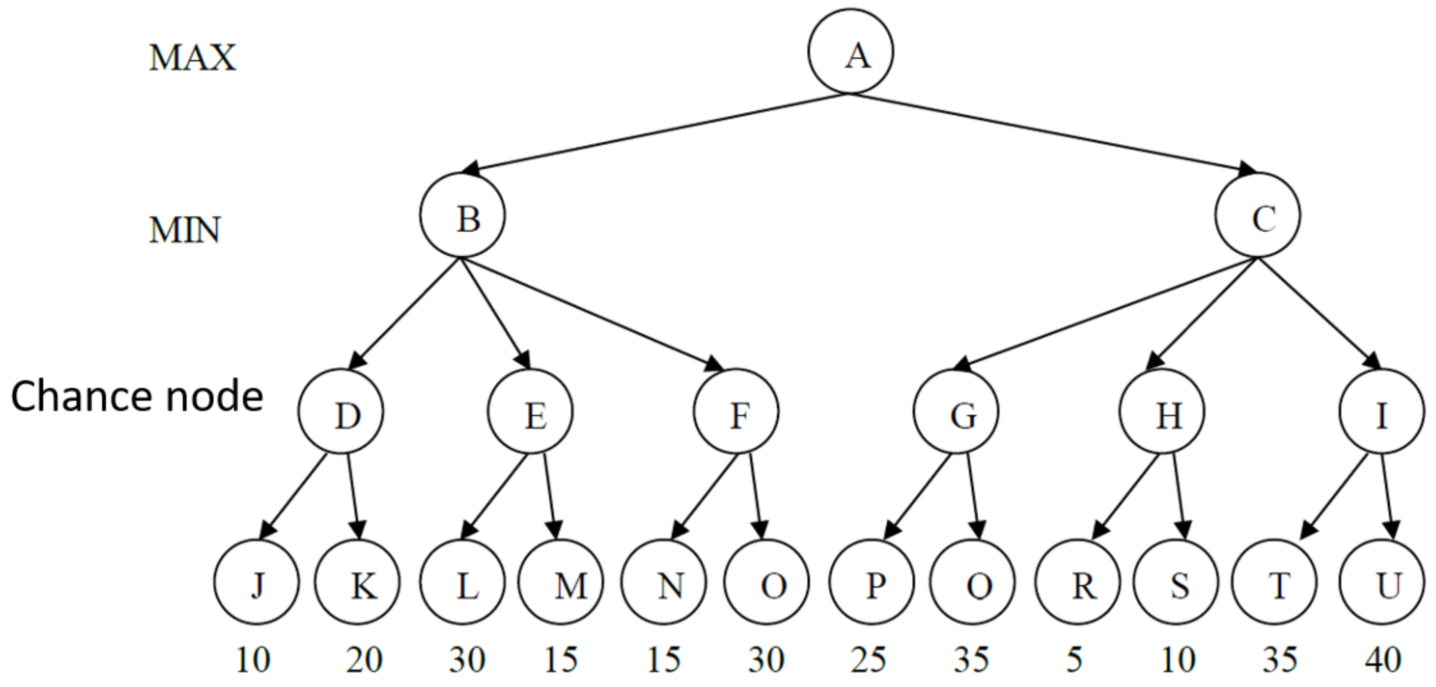
- Calculate the expectiminimax value of the chance node



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

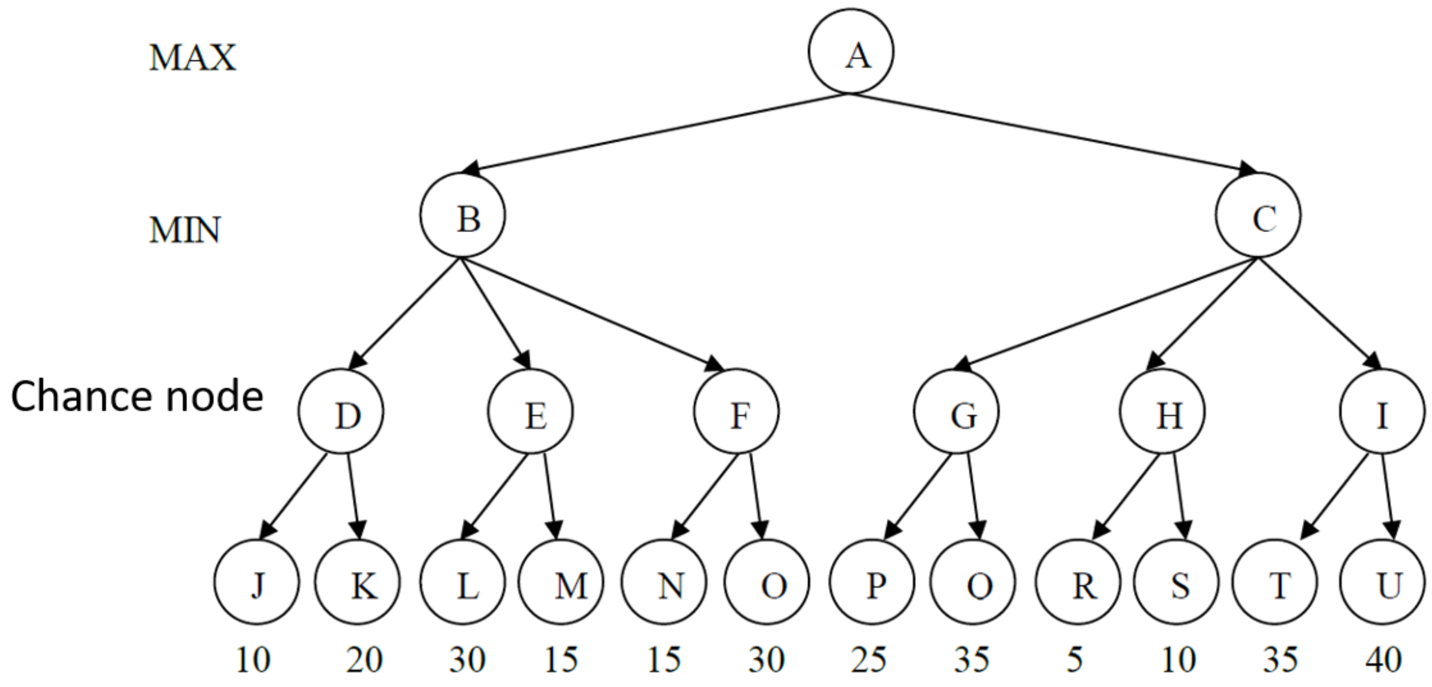
Example 2

- In the tree below assume that the nodes D, …, I 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?



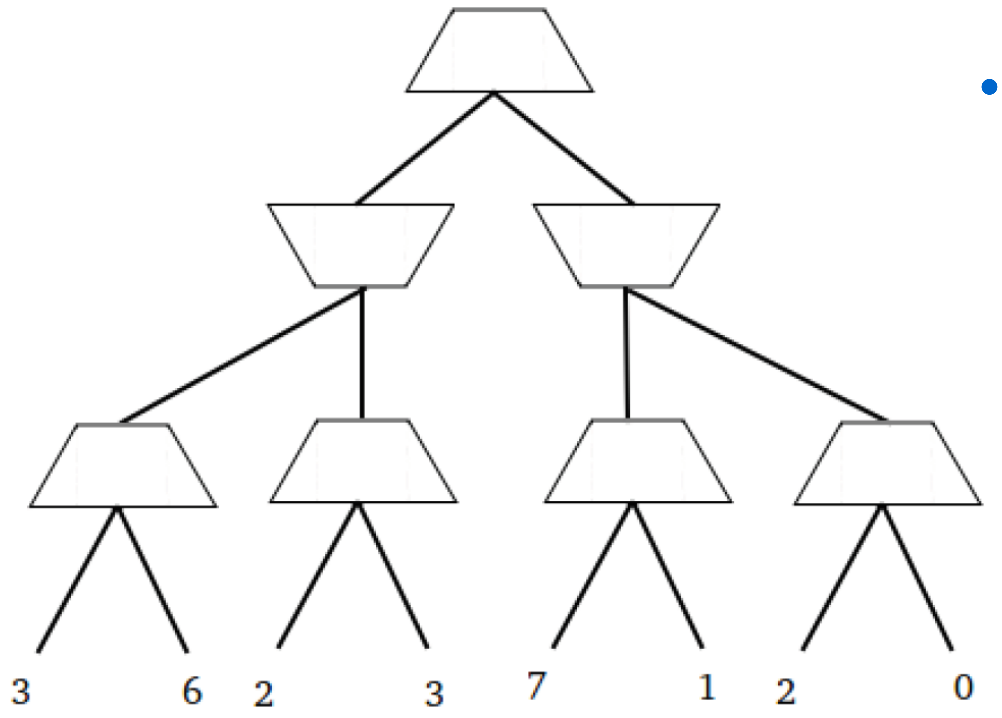
Example 2

- Answer: 15



Example 3: Standard Minimax

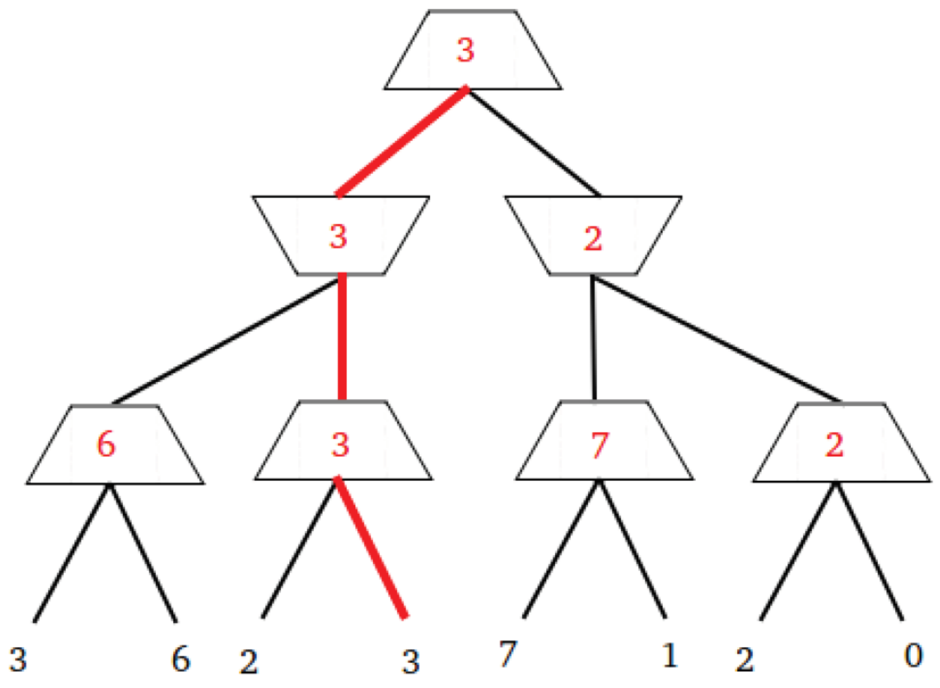
- Fill in the values of each of the nodes in the following Minimax tree. The upward pointing trapezoids correspond to maximizer nodes (layer 1 and 3), and the downward pointing trapezoids correspond to minimizer nodes (layer 2). Each node has two actions available, Left and Right.



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

Example 3: Standard Minimax

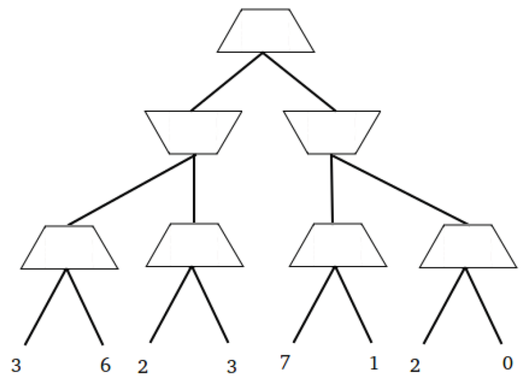
- Fill in the values of each of the nodes in the following Minimax tree. The upward pointing trapezoids correspond to maximizer nodes (layer 1 and 3), and the downward pointing trapezoids correspond to minimizer nodes (layer 2). Each node has two actions available, Left and Right.



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

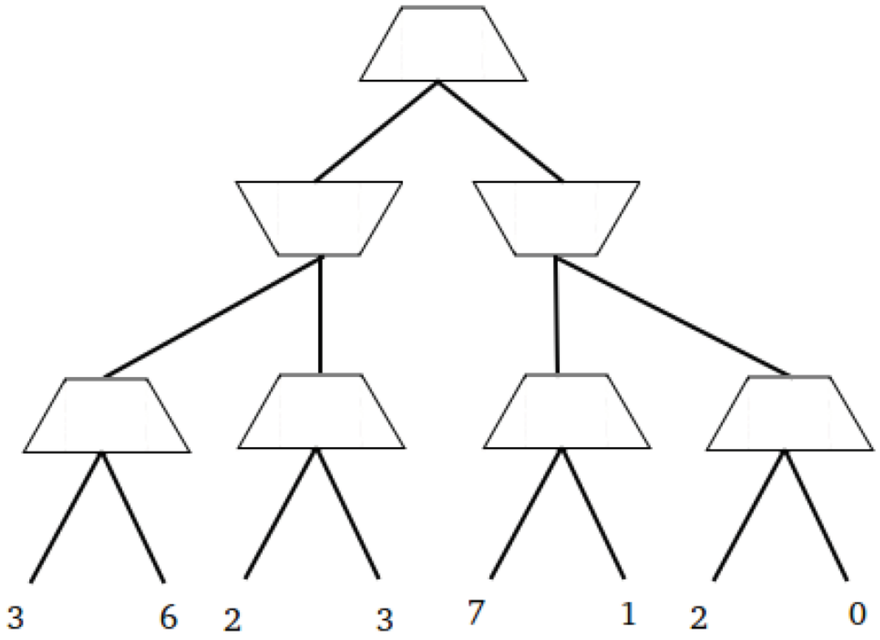
Example 4: Dark Magic

- 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.



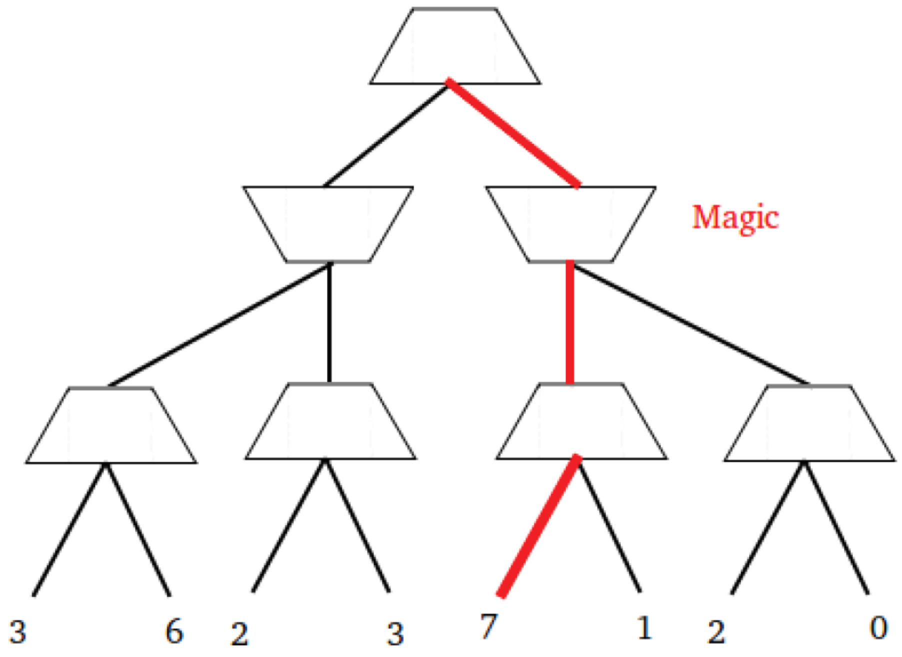
Example 4: Dark Magic

- (i) Dark Magic at Cost $c = 2$
- Let Pacman have 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.



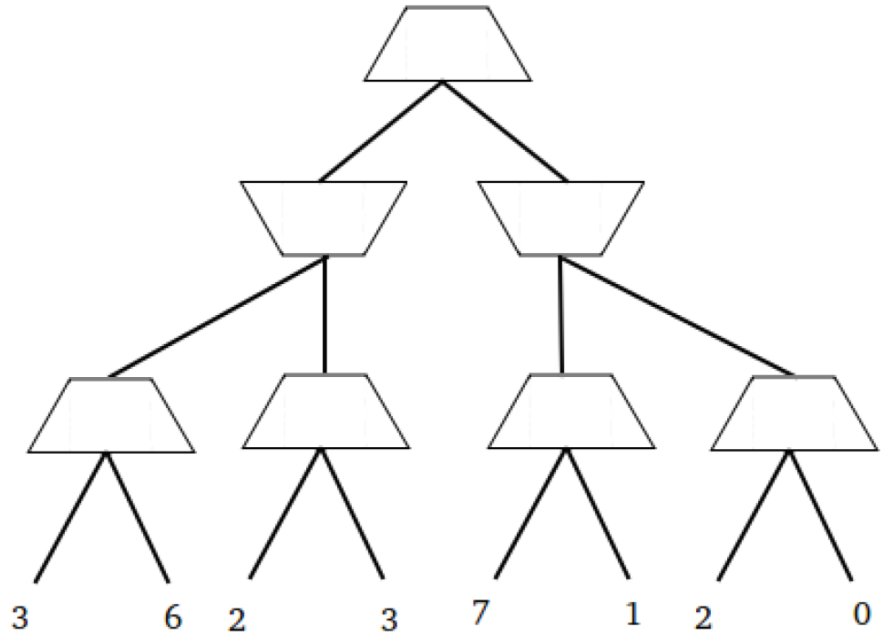
Example 4: Dark Magic

- (i) Dark Magic at Cost $c = 2$
- Let Pacman have 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.



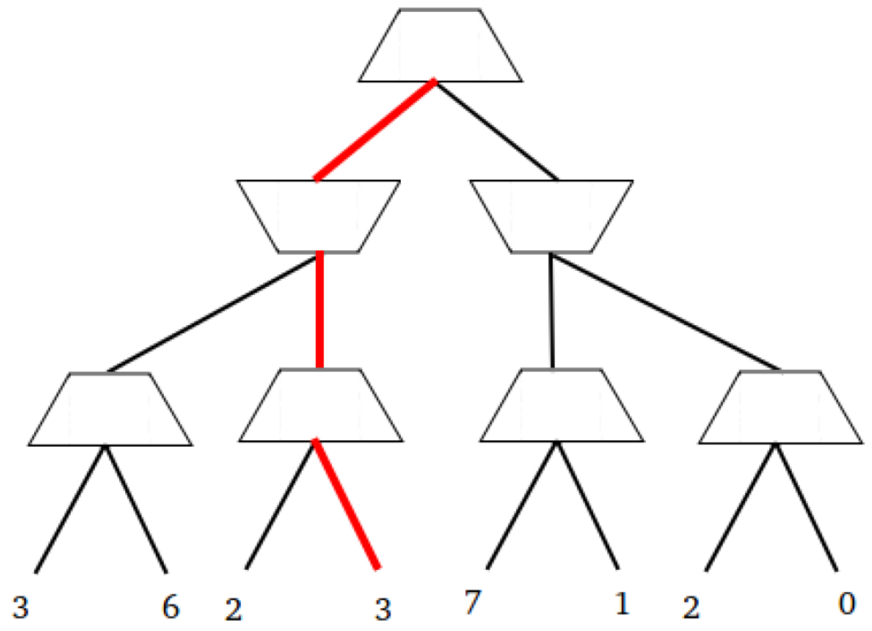
Example 4: Dark Magic

- (ii) Dark Magic at Cost $c = 5$
- Let Pacman have 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.



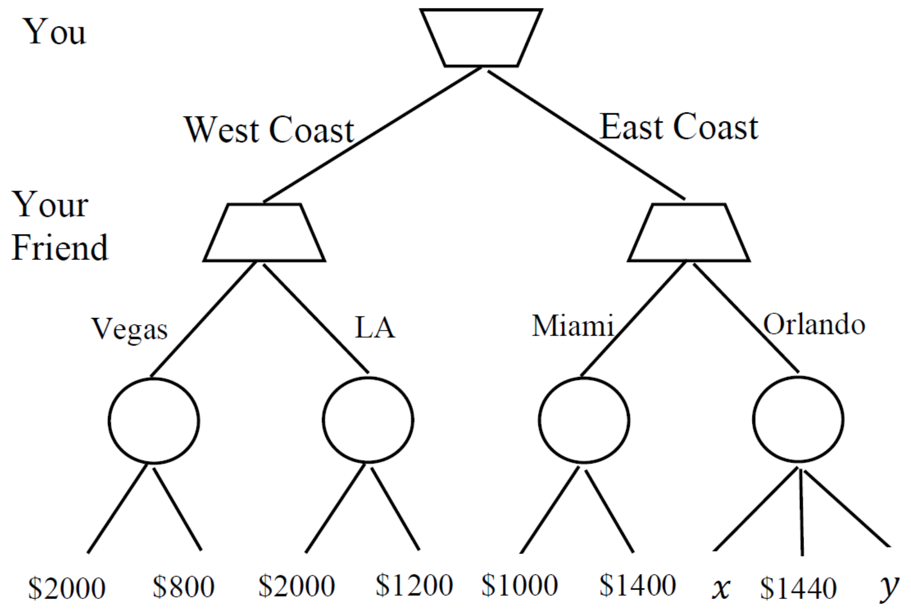
Example 4: Dark Magic

- (ii) Dark Magic at Cost $c = 5$
- Let Pacman have 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.



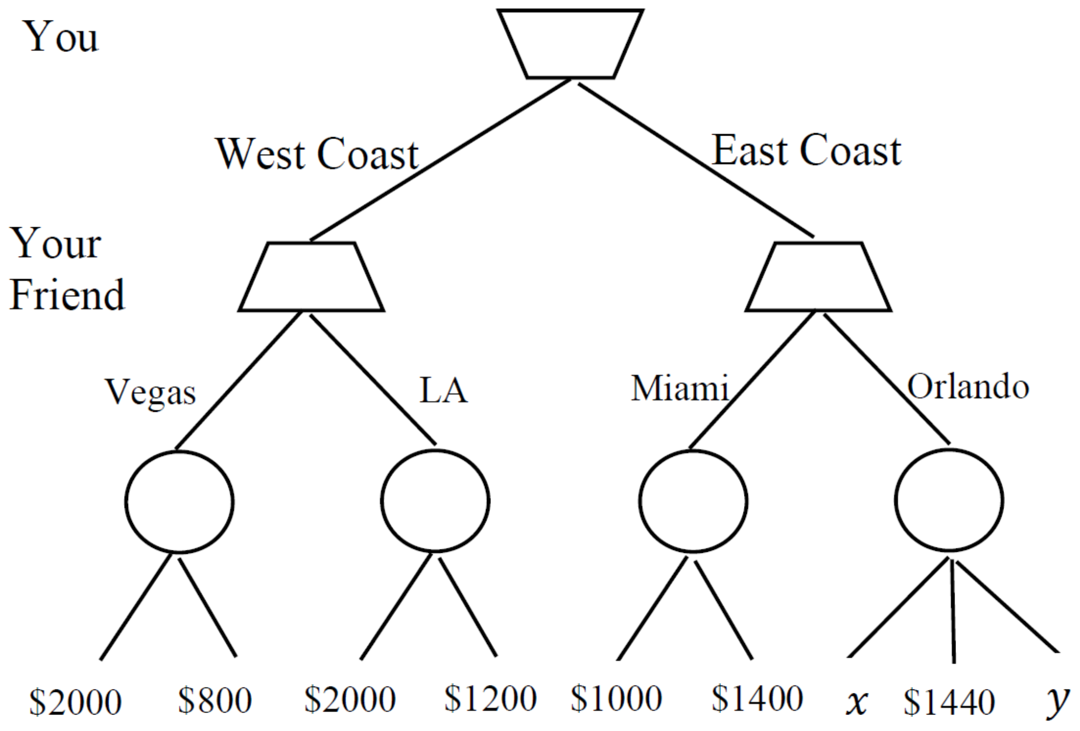
Example 5:

- 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.



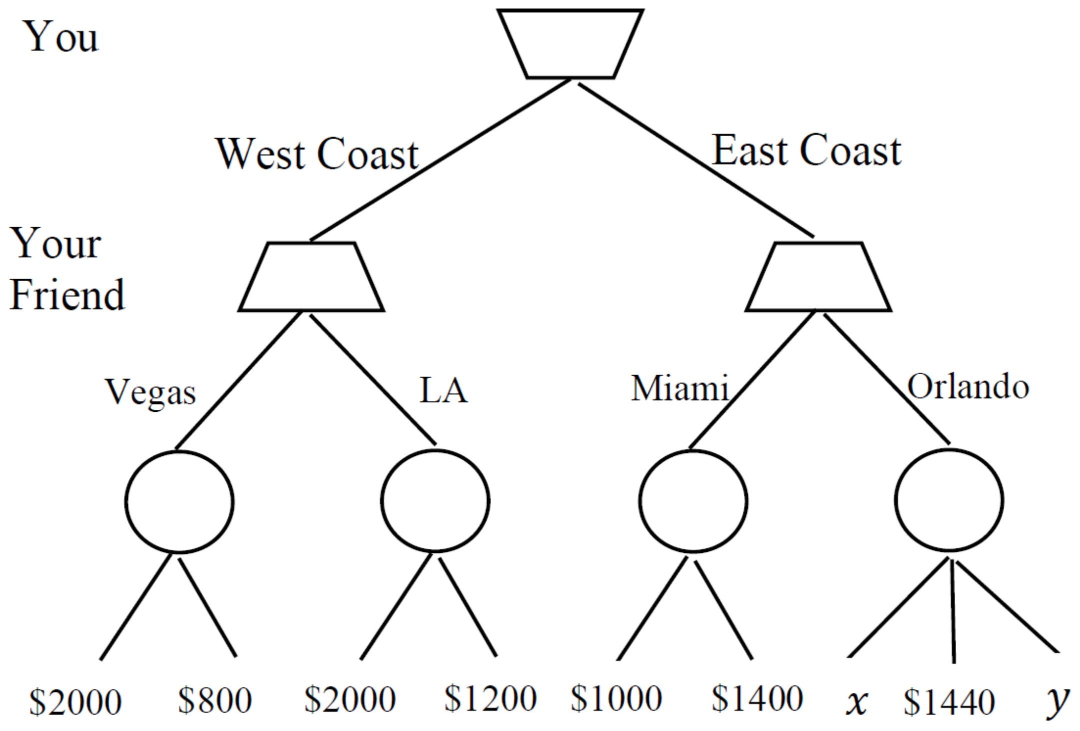
Example 5:

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



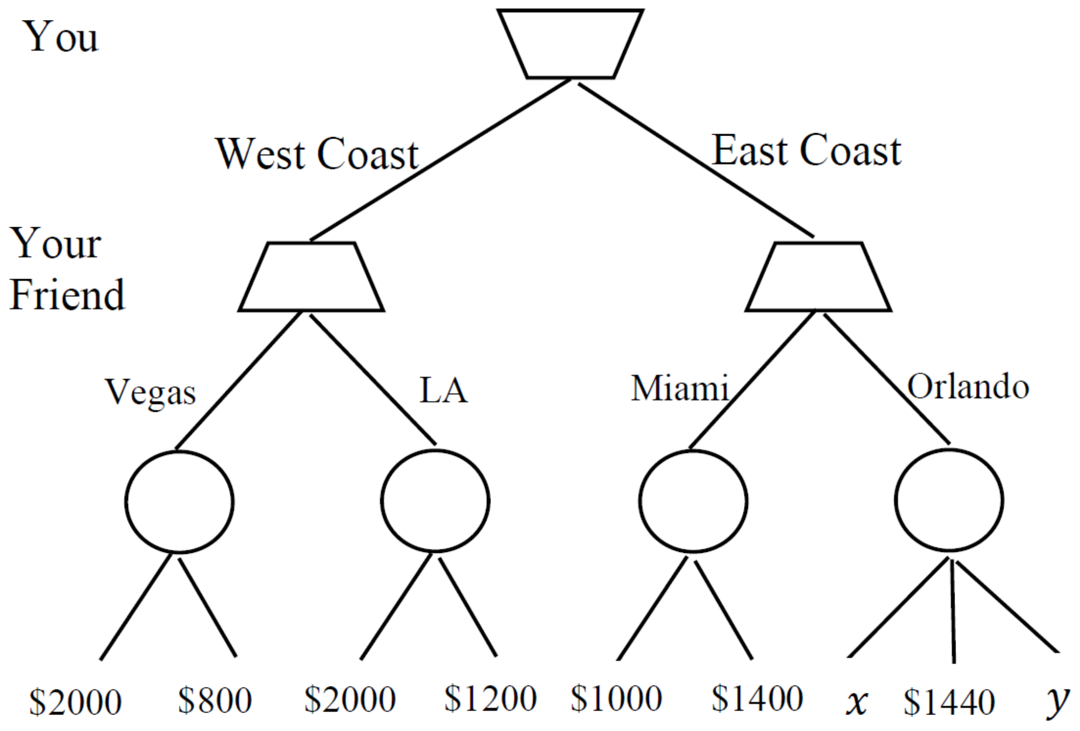
Example 5:

- 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?


You
West Coast
East Coast
Your Friend
Vegas
LA
Miami
Orlando
$2000
$800
$2000
$1200
$1000
$1400
x
$1440
y