



Adversarial Search

Game Search, Min- Max Search,
Alpha-beta Pruning

Games



- Games have engaged the intellectual faculties of humans.
- Game playing is an hot AI research area.
- In 1950, first chess program written by Alan Turing, as soon as computer become programmable.
- It proofs machine having intelligence.
- Average branching factor of chess is 35, each player need 50 moves. Search tree has 35^{100} nodes.
- This complexity introduced new kind of uncertainty.





Games

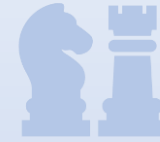
- Mathematical **game theory**, a branch of economics, views any multiagent environment as a game, provided that the impact of each agent on the others is “significant,” regardless of whether the agents are cooperative or competitive.
- the most common games we will focus are special kind—what game theorists call deterministic, turn-taking, two-player, zero-sum games of perfect information (such as chess).
- Games, like the real world, therefore require the ability to make some decision even when calculating the optimal decision is infeasible.
- We look at techniques for choosing a good move when time is limited. Pruning allows us to ignore portions of the search tree that make no difference to the final choice, and heuristic evaluation functions allow us to approximate the true utility of a state without doing a complete search.



Games

- Consider games with two players, whom we call MAX and MIN
- MAX moves first, and then they take turns moving until the game is over. At the end of the game, points are awarded to the winning player and penalties are given to the loser.
- A game can be formally defined as a kind of search problem with the following elements:
 - S_0 : 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.

Games

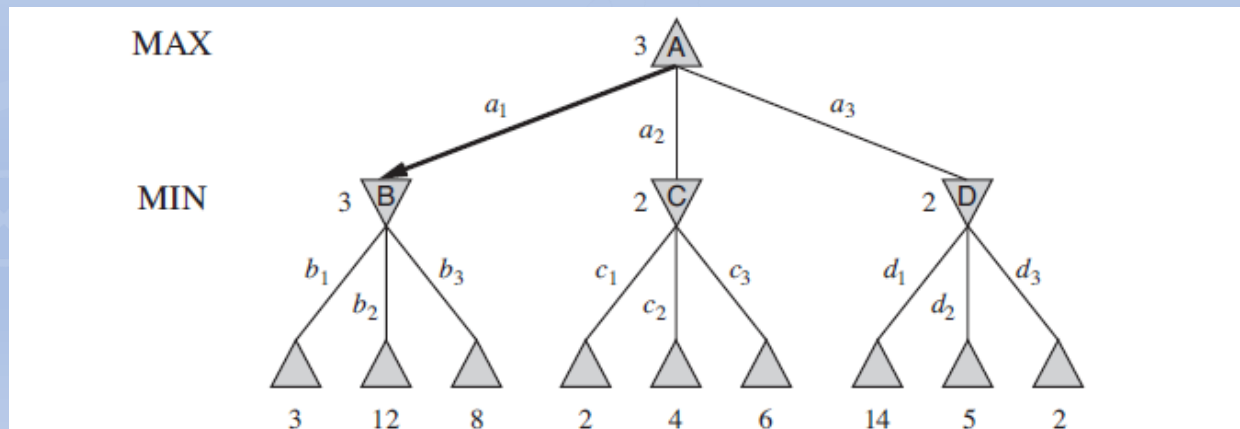


- $\text{RESULT}(s, a)$: The transition model, which defines the result of a move.
- $\text{TERMINAL-TEST}(s)$: A terminal test, which is true when the game is over and false otherwise. States where the game has ended are called terminal states.
- $\text{UTILITY}(s, p)$: A utility function (also called an objective function or payoff function), defines the final numeric value for a game that ends in terminal state s for a player p .

Game Tree: 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.

OPTIMAL DECISIONS IN GAMES

- Even a simple game like tic-tac-toe is too complex for us to draw the entire game tree on one page, so we will switch to the trivial game
- In game parlance, we say that this tree is one move deep, consisting of two half-moves, each of which is called a ply.
- The utilities of the terminal states in this game range from 2 to 14.
- the optimal strategy can be determined from the minimax value of each node, which we write as MINIMAX(n).



- The minimax value of a node is the utility (for MAX) of being in the corresponding state, assuming that both players play optimally from there to the end of the game.
- the minimax value of a terminal state is just its utility.
- Given a choice, MAX prefers to move to a state of maximum value, whereas MIN prefers a state of minimum value.
- So we have the following:

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



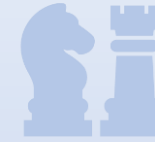
Minimax algorithm

- The minimax algorithm computes the minimax decision from the current state.
- The recursion proceeds all the way down to the leaves of the tree, and then the minimax values are backed up through the tree as the recursion unwinds.

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

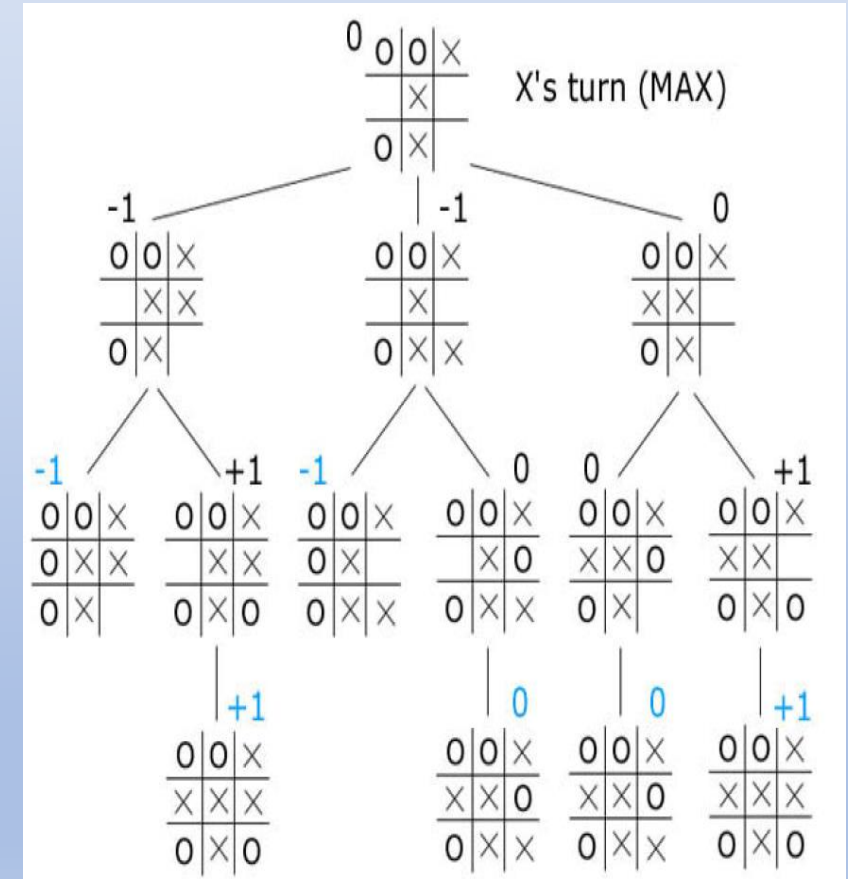
```
function MAX-VALUE(state) returns a utility value  
  if TERMINAL-TEST(state) then return UTILITY(state)  
   $v \leftarrow -\infty$   
  for each a in ACTIONS(state) do  
     $v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(s, a)))$   
  return v
```

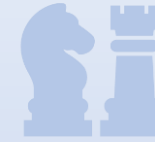
```
function MIN-VALUE(state) returns a utility value  
  if TERMINAL-TEST(state) then return UTILITY(state)  
   $v \leftarrow \infty$   
  for each a in ACTIONS(state) do  
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$   
  return v
```

Tic Tac Toe- Game Tree

- Nodes are the current state of the game and the edges represent the moves.
- Each individual move by one player is called a 'ply'.
- The leaves of the game tree represent terminal positions when outcome is clear.
- Each terminal has a score.
- May associate 1 with win, 0 with draw, -1 with a loss for MAX.





Game of NIM

- Given a number of piles in which each pile contains some numbers of stones/coins. In each turn, a player can choose only one pile and remove any number of stones (at least one) from that pile. The player who cannot move is considered to lose the game (i.e., one who take the last stone is the winner)



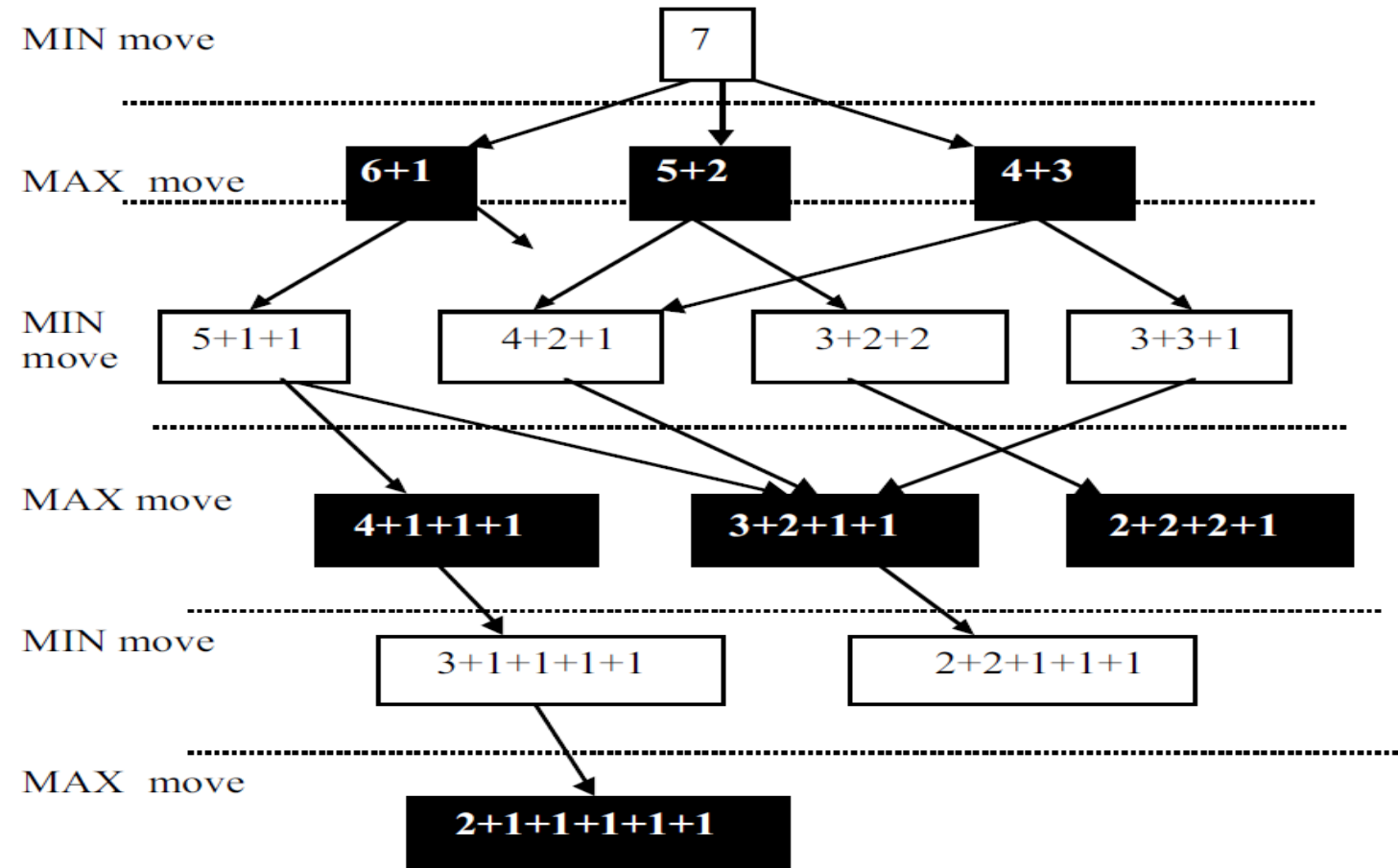


Game of NIM

The game starts with an odd number of match sticks, normally 7 or 9, placed on a single row, called a pile. Each player in his turn has to break a single pile into two piles of unequal sticks, greater than zero. The game will come to an end when either of the two players cannot give a successful move. The player who cannot give a successful move the first time will lose the game.

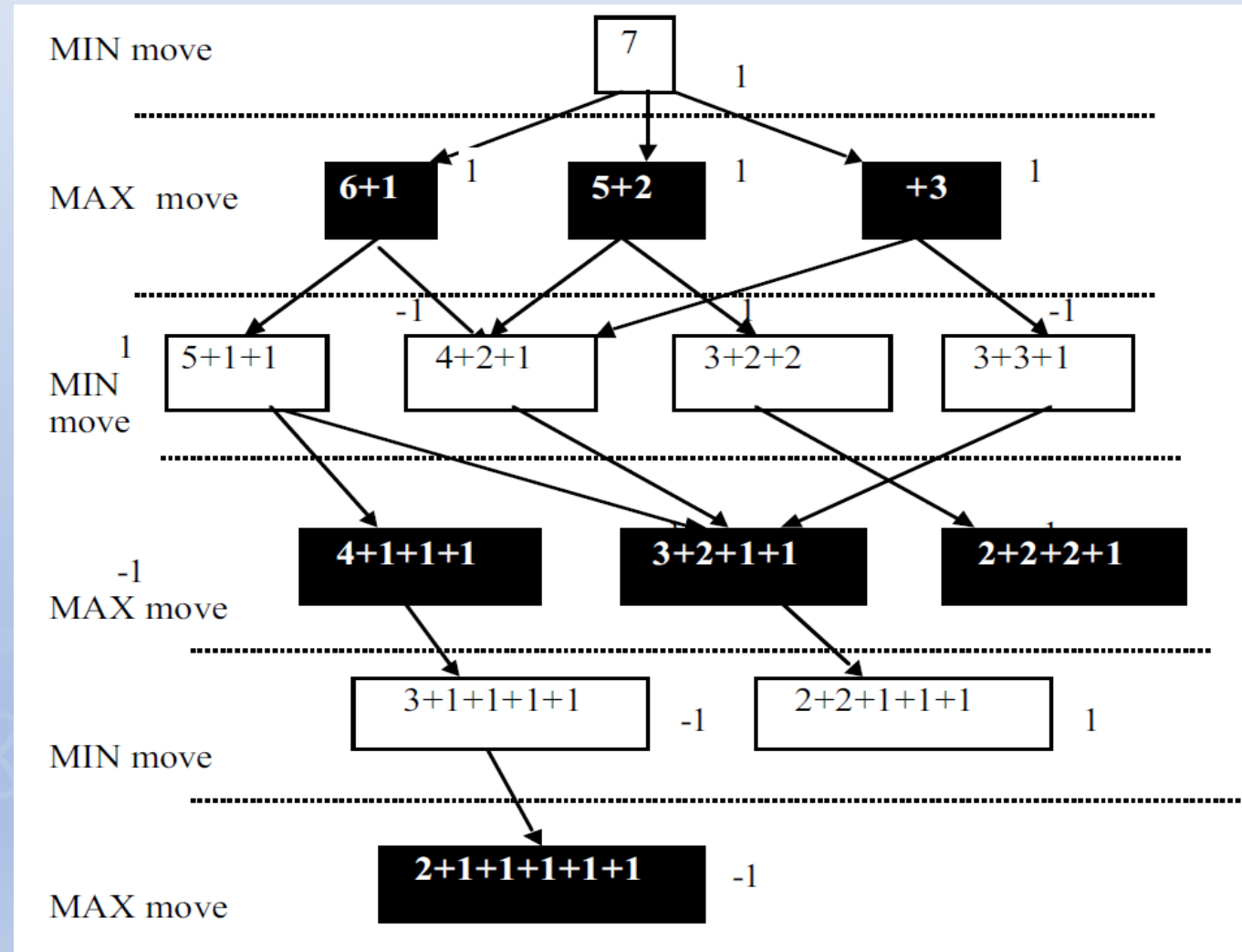
This is a defensive game and consequently the opening player, here, is called the MINIMIZER

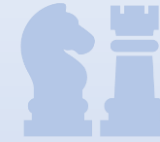
NIM Game Tree



Symbol: Minimizer's move , Maximizer's move

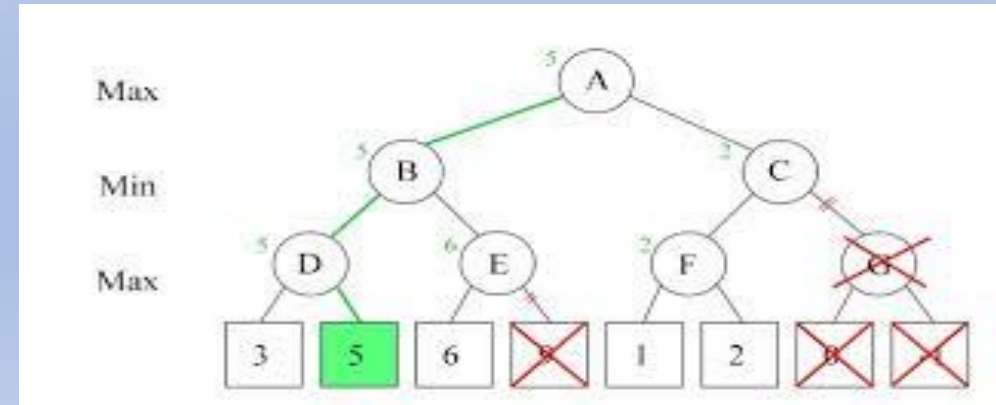
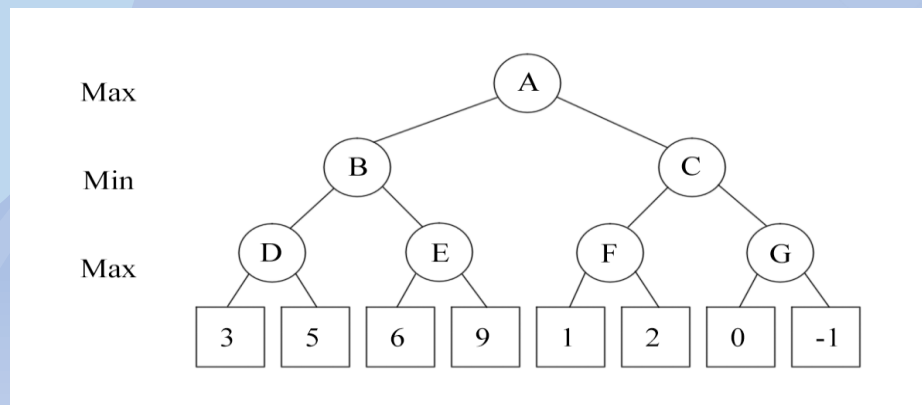
The MAXIMIZER's success is denoted by +1, while the MINIMIZER's success by -1 and a draw by a 0.



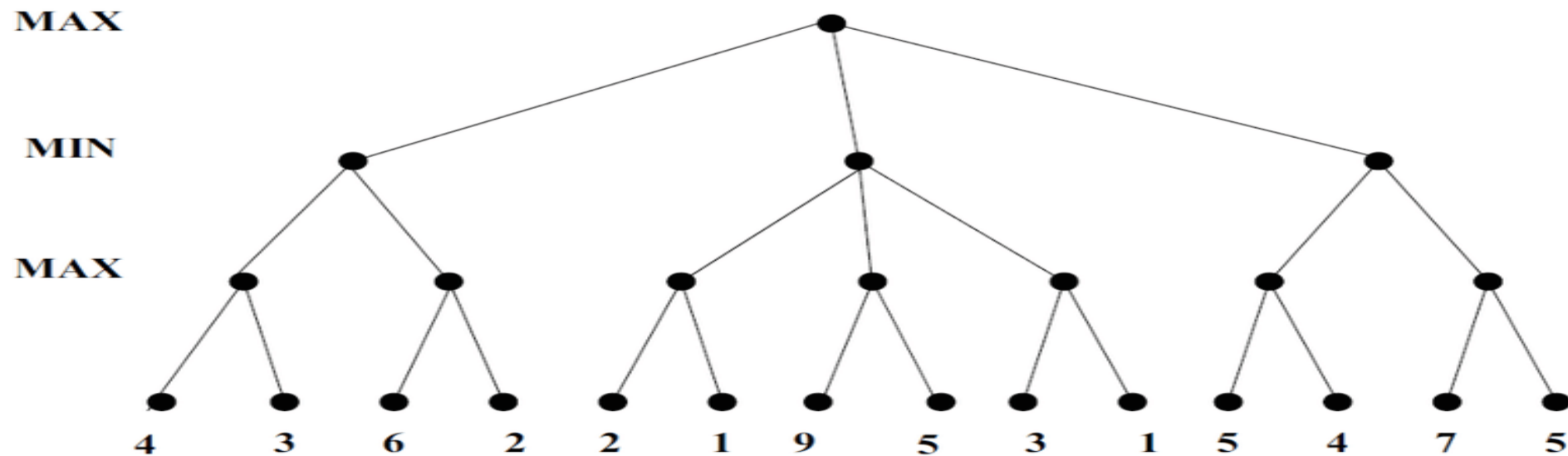


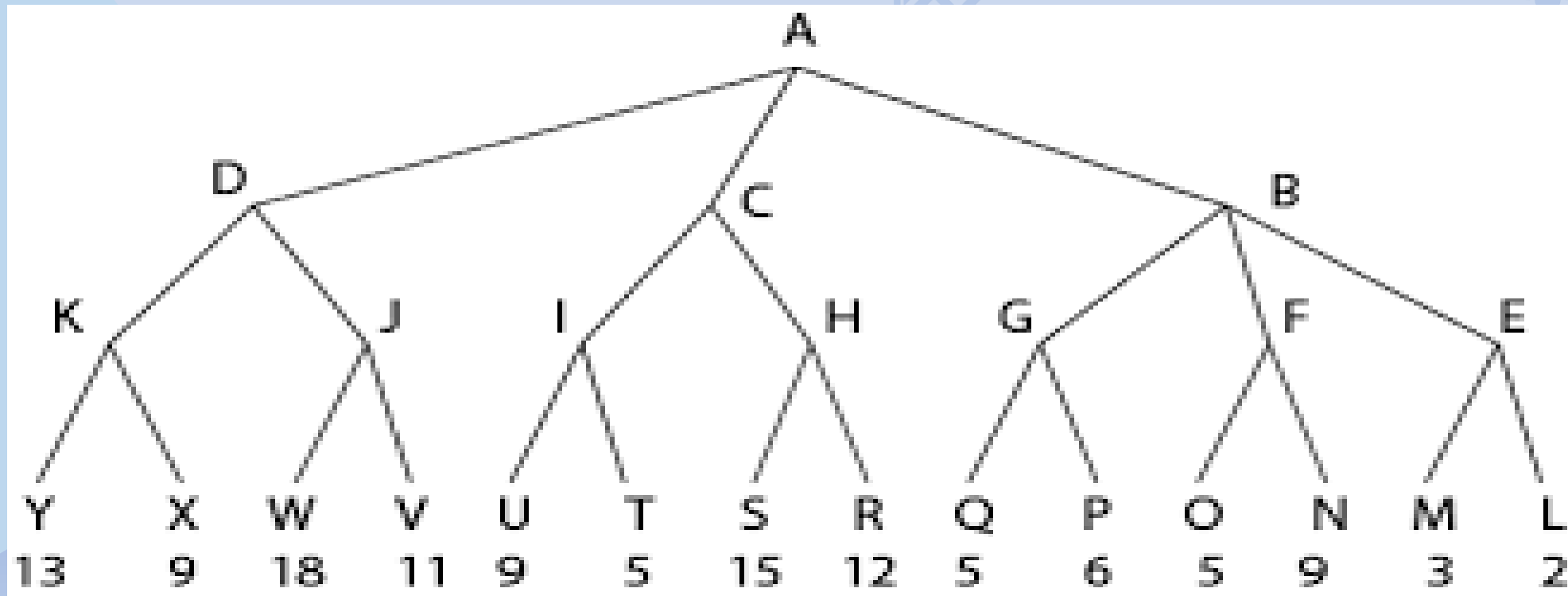
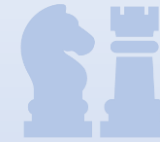
Alpha Beta Pruning

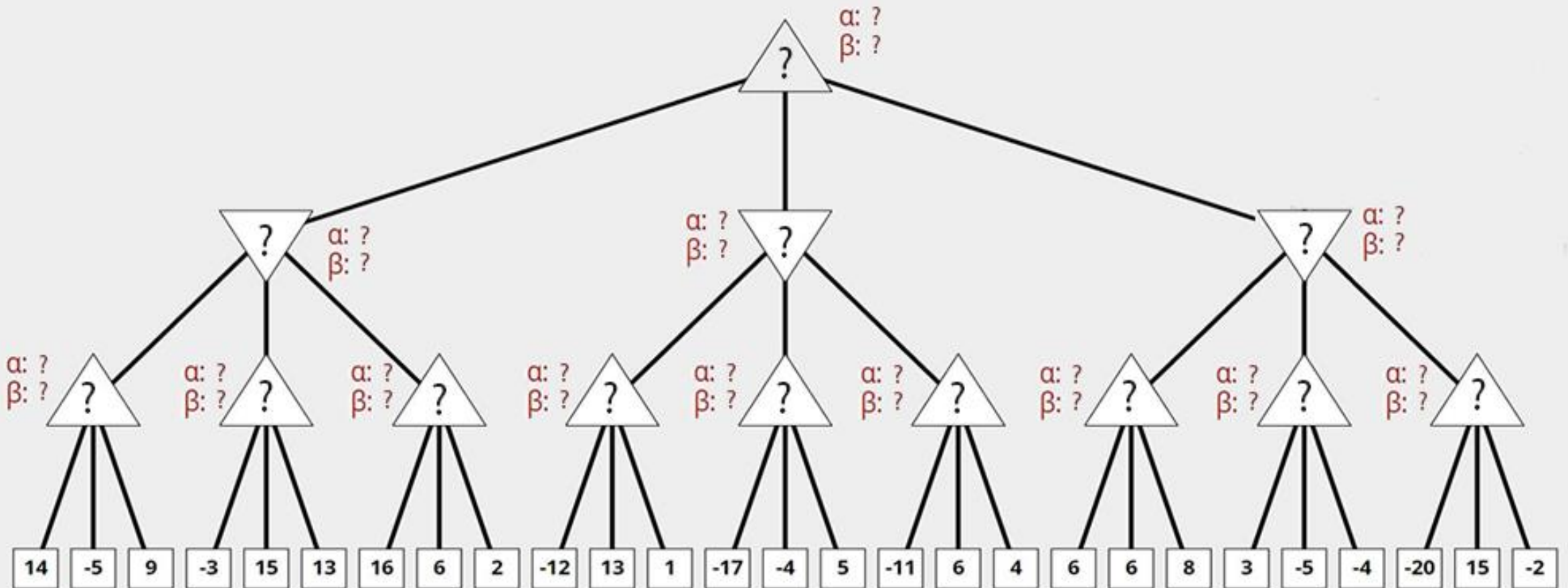
- The alpha beta procedure can speed up a depth first minimax search
- Alpha: a lower bound on the value that a max node may ultimately assigned $v \geq \beta$
- Beta: an upper bound on the value that a minimizing node may ultimately assigned, $v \leq \beta$
- When $\alpha \geq \beta$, prune the branch.





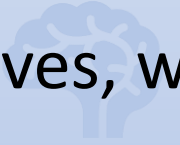




- Initially $\alpha = -\infty$ and $\beta = \infty$
- Alpha(α) is the best value for max along the path.
- Beta(β) is the best value for min along the path.







Analysis of alpha beta pruning

- Bad: worst moves encountered first. 
 - Good: Good moves ordered first. 
 - If we can order moves, we can get more benefit from alpha beta pruning. 
- 
- 
- 
- 
- 