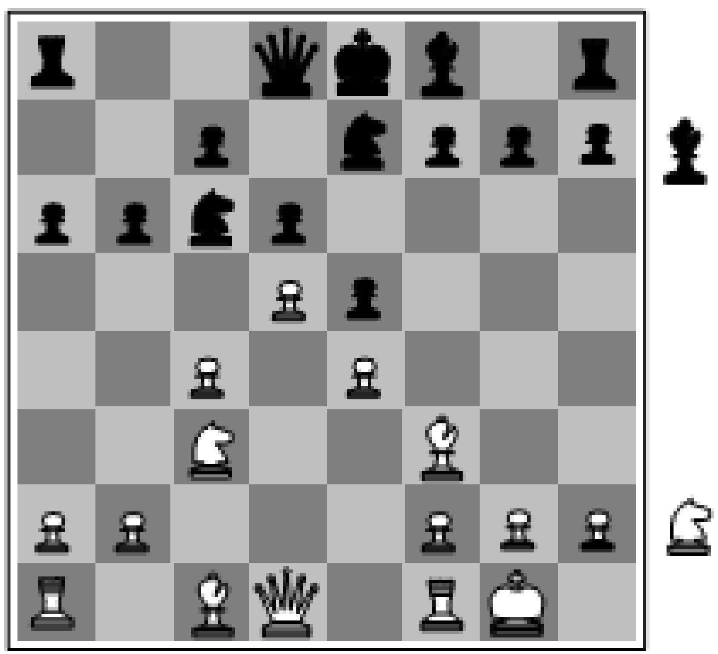
Games and adversarial search

Chapter 6



Games vs. Search problems

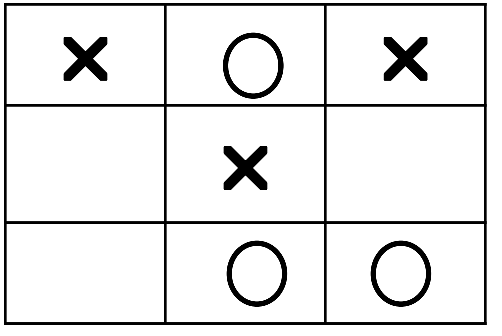
- there are one or some **opponents** in a game vs. search problems.
- An opponent is a person or a computer that increases his/its utility by decreasing ours.
- "Unpredictable" opponent ⇒ solution is a **strategy** specifying a move for every possible opponent reply.
- Time limits ⇒ unlikely to find goal, must approximate plan of attack.

Type of games environment

	deterministic	chance
Perfect information	Chess, othello	backgammon
Imperfect information	battleships	Bridge , poker

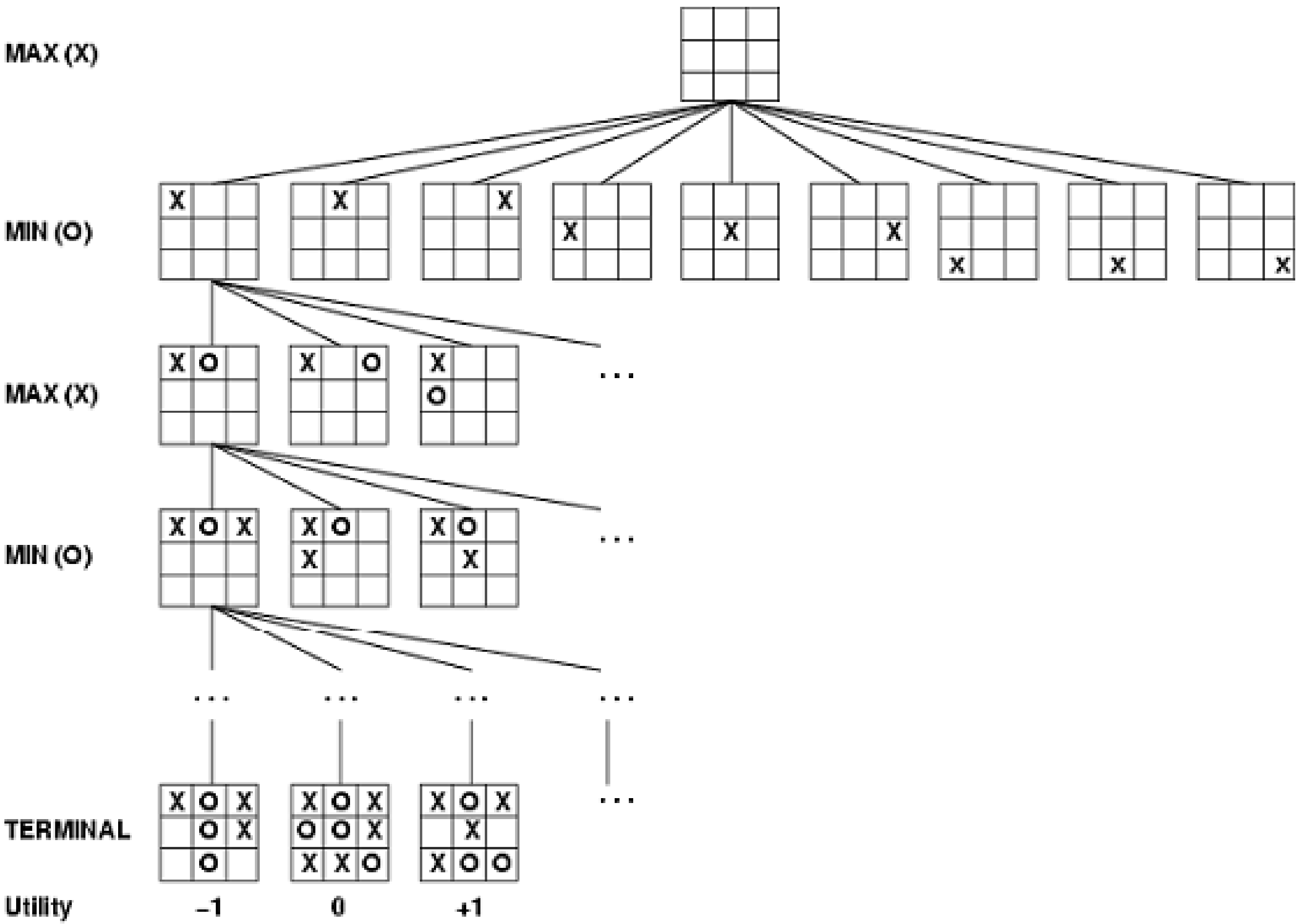
Zero-sum games

- Suppose we have **two** players:
  - Players take turns Players take turns
  - Each game outcome or **terminal state** has a utility for each player (e.g.,**1 for win,0 for loss**)
  - The sum of both players' utilities is a **constant 0**.
  - Like chess, tic-tac-toe, etc.



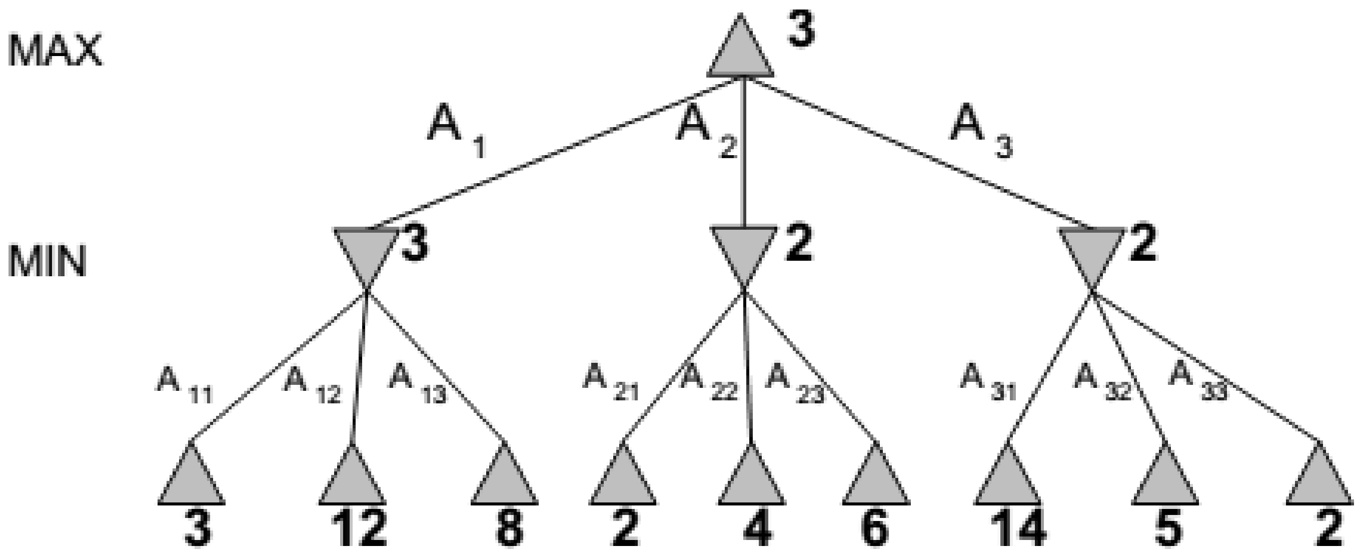
Game tree

A game of tic-tac-toe between two players, “max” , “min”



Minmax

- Perfect play for **deterministic**, **perfect-information** games.
- Idea: choose move to position with **highest minmax value**
- E.g., 2-**ply** games:



Minmax algorithm

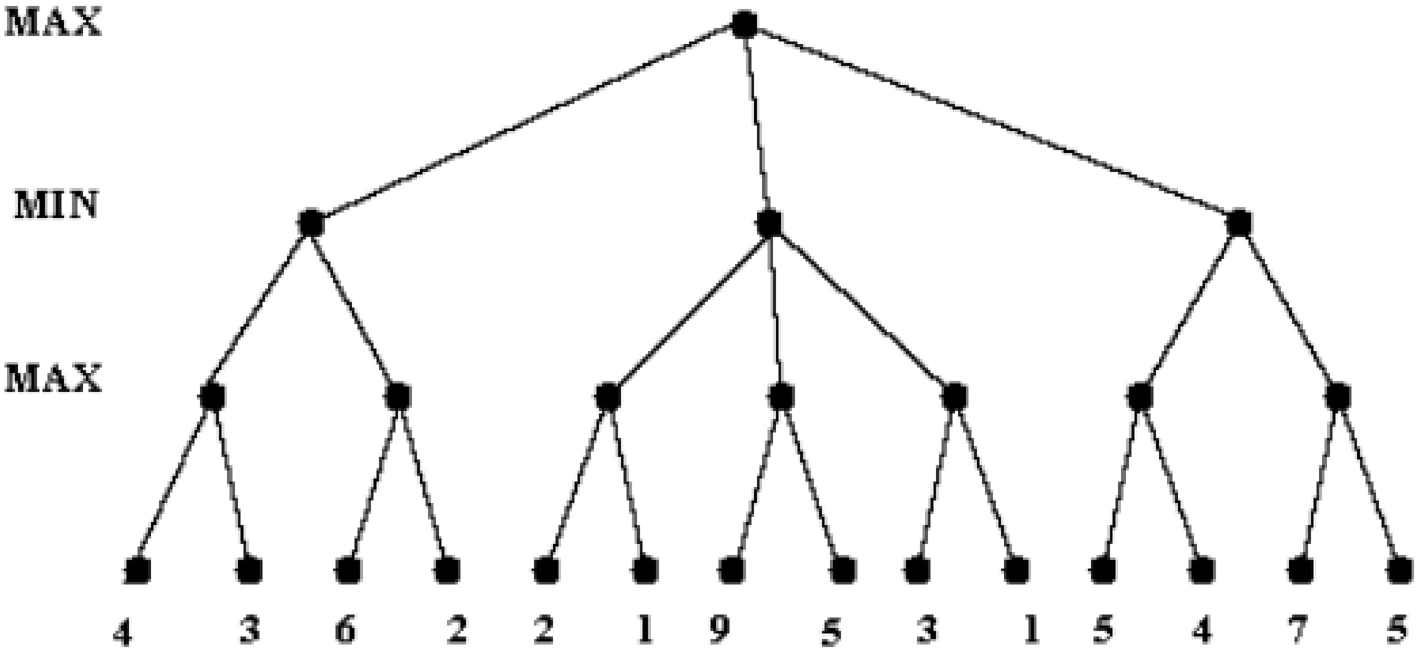
```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game

    return the a in ACTIONS(state) maximizing MIN-VALUE(RESULT(a, state))
```

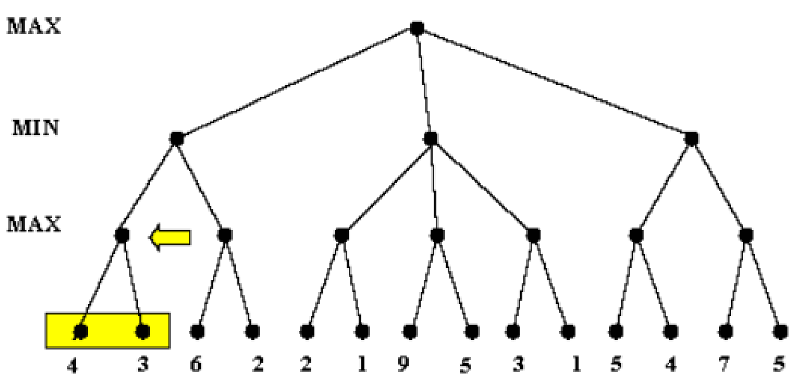
```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v
```

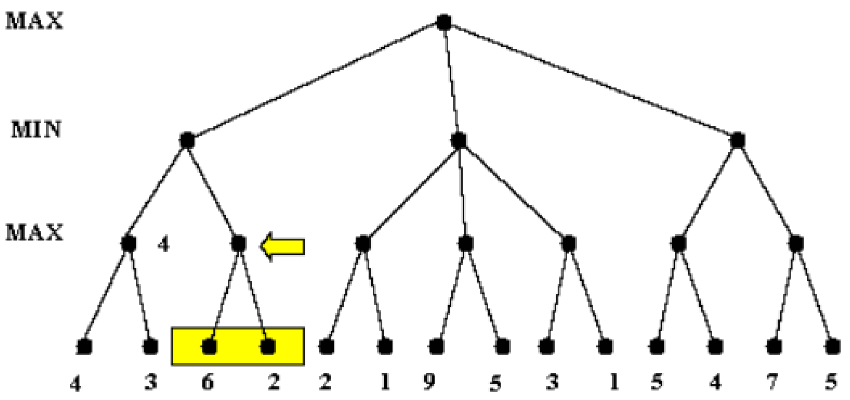
```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
    return v
```

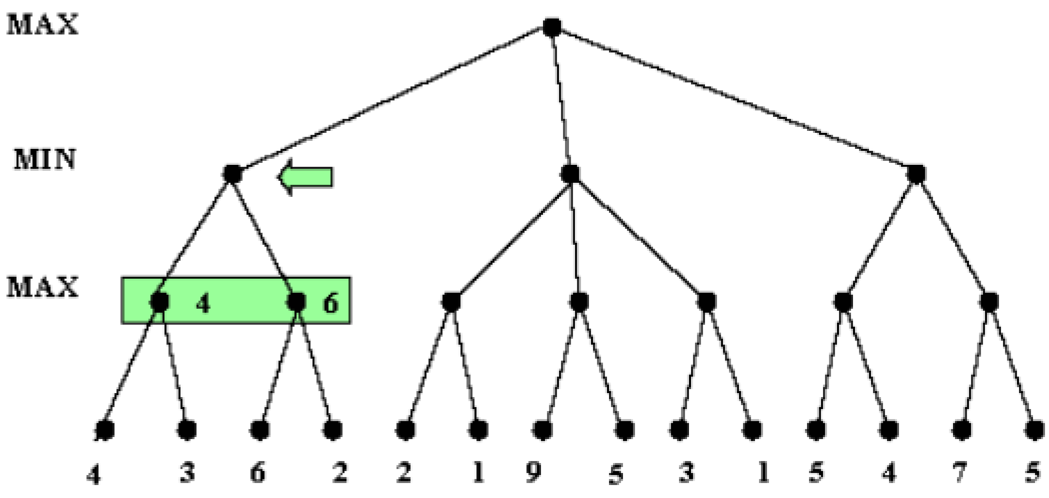
Example: non-zero-sum



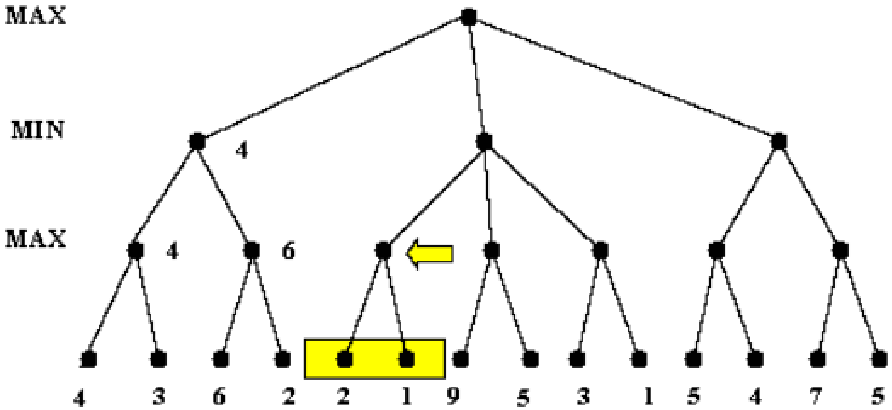
Example

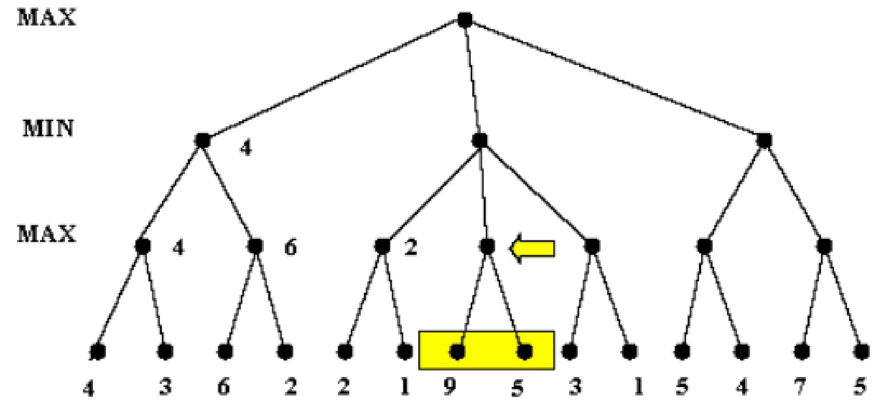

MAX
MIN
MAX
4 3 6 2 2 1 9 5 3 1 5 4 7 5

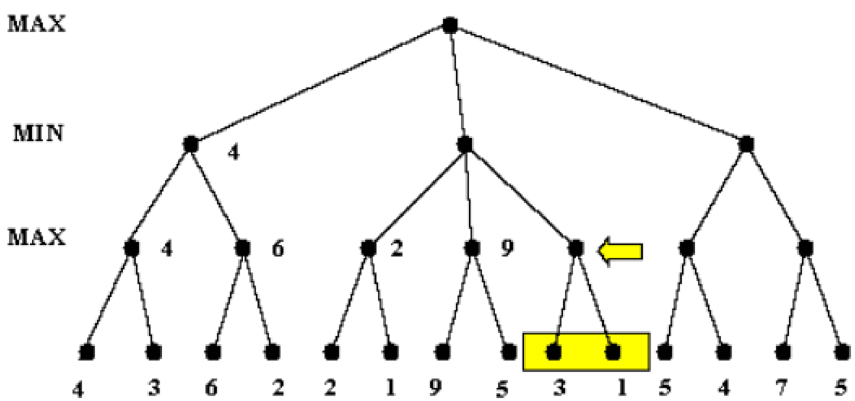

MAX
MIN
MAX
4
4 3 6 2 2 1 9 5 3 1 5 4 7 5

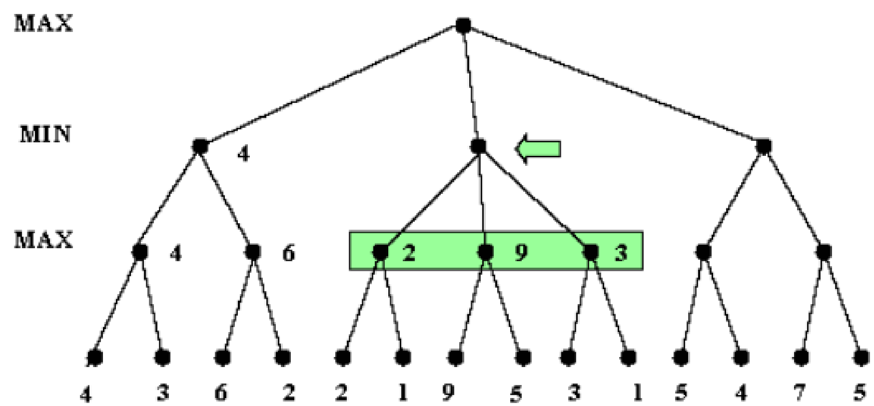

MAX
MIN
MAX
4
6
4 3 6 2 2 1 9 5 3 1 5 4 7 5

Example

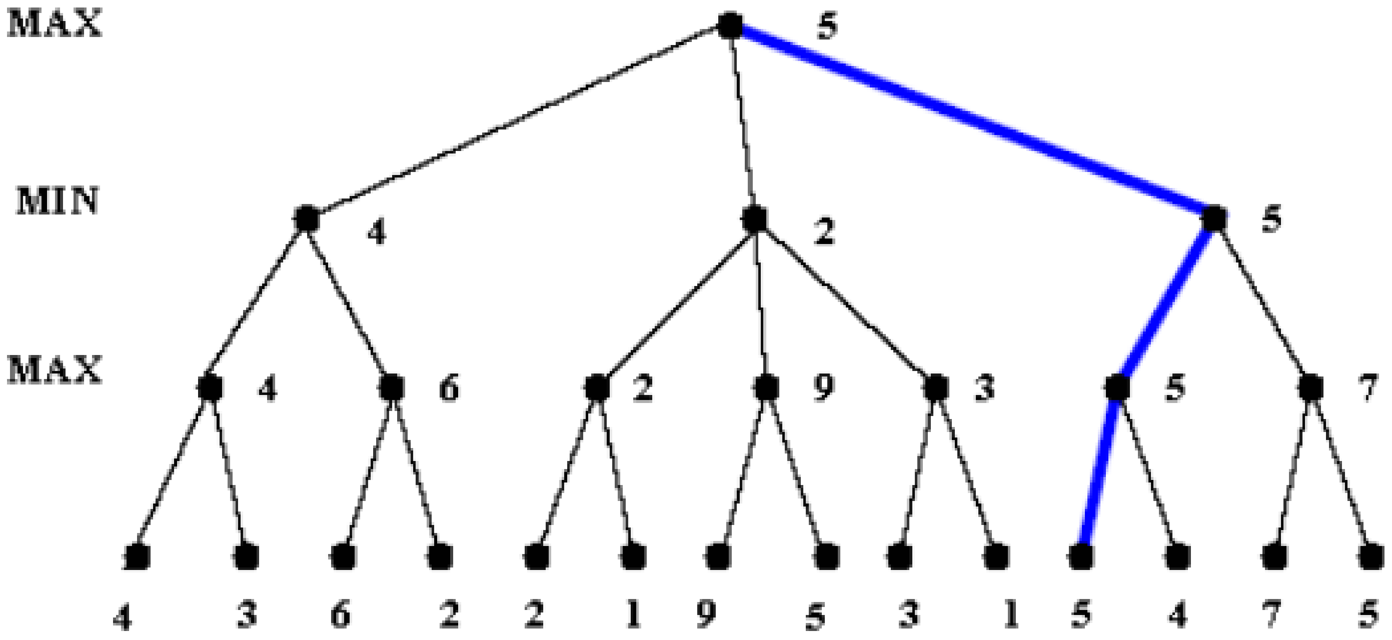

MAX
MIN
MAX
4
4
6
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
4
4
6
2
4 3 6 2 2 1 9 5 3 1 5 4 7 5

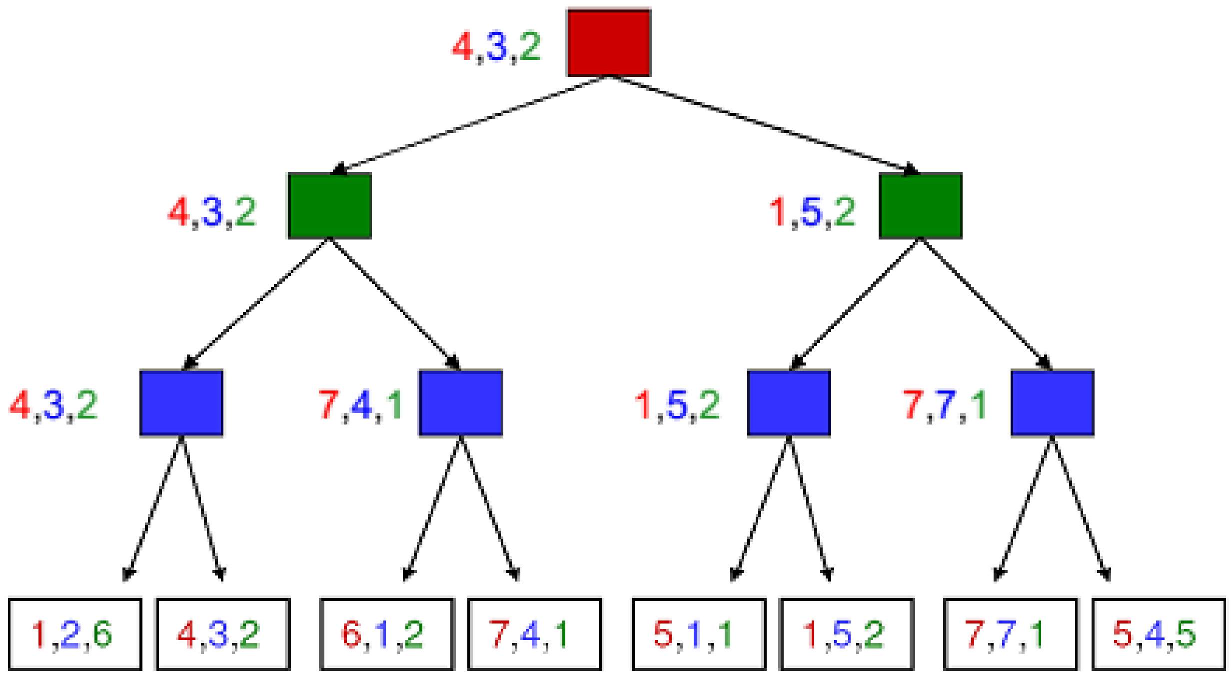

MAX
MIN
MAX
4
4
6
2
9
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
4
4
6
2
9
3
4 3 6 2 2 1 9 5 3 1 5 4 7 5

Example



What about more than two players??



- More than two players, non-zero-sum
- Each player maximizes their own utility at each node

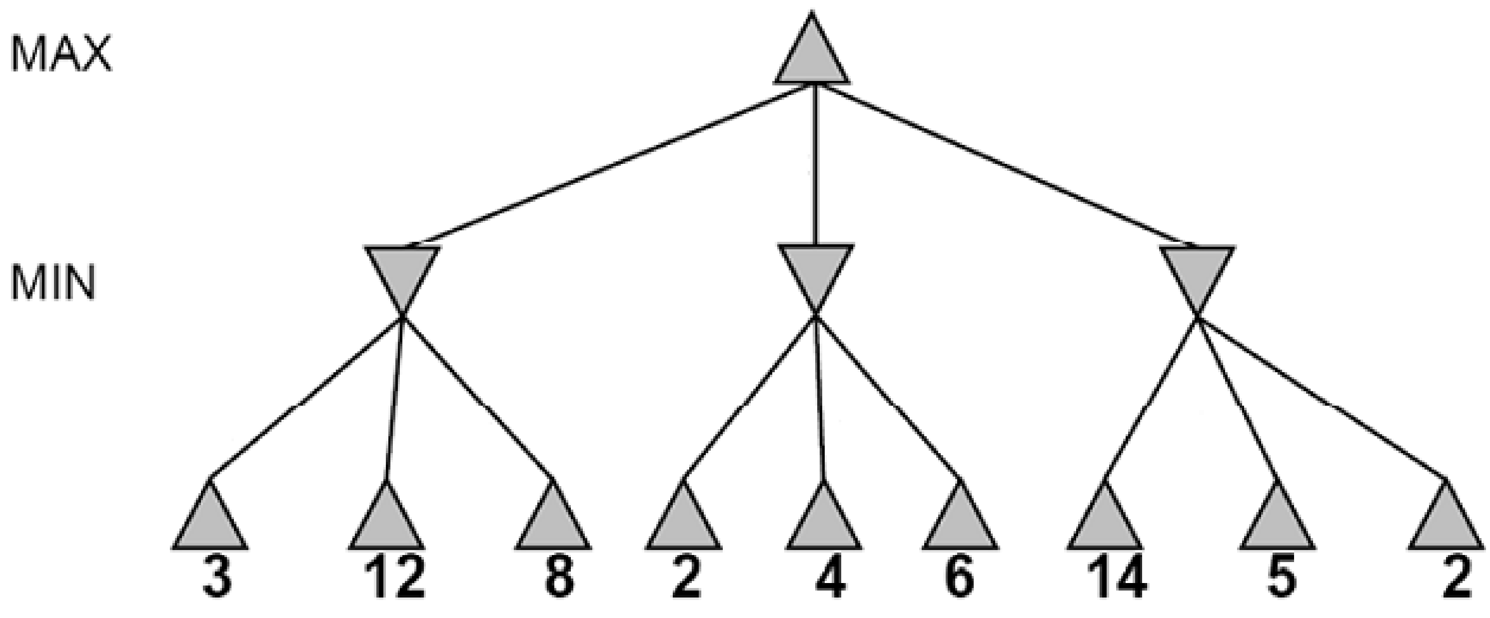
Properties of minmax

- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? O(bm) (depth-first exploration)

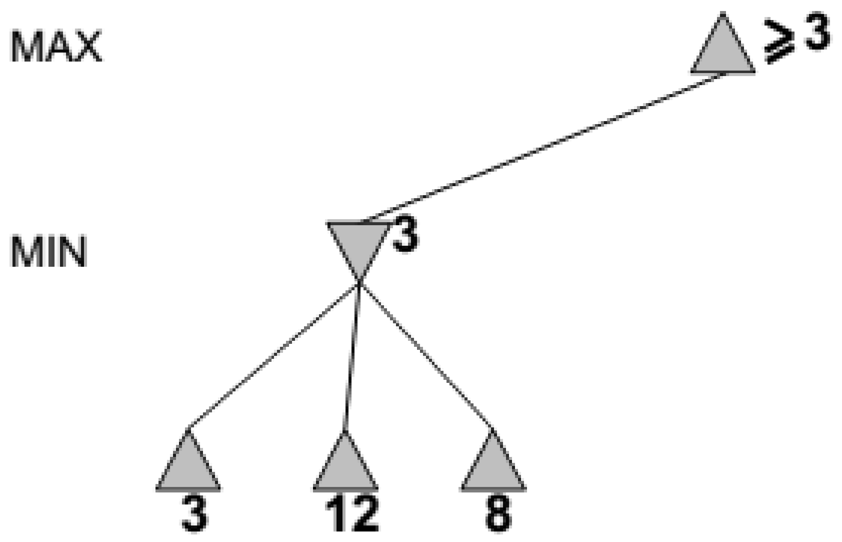
- But do we need to explore every path??
- for chess, $b \approx 35$, $m \approx 100$ for"reasonable"games ⇒exact solution completely **infeasible**

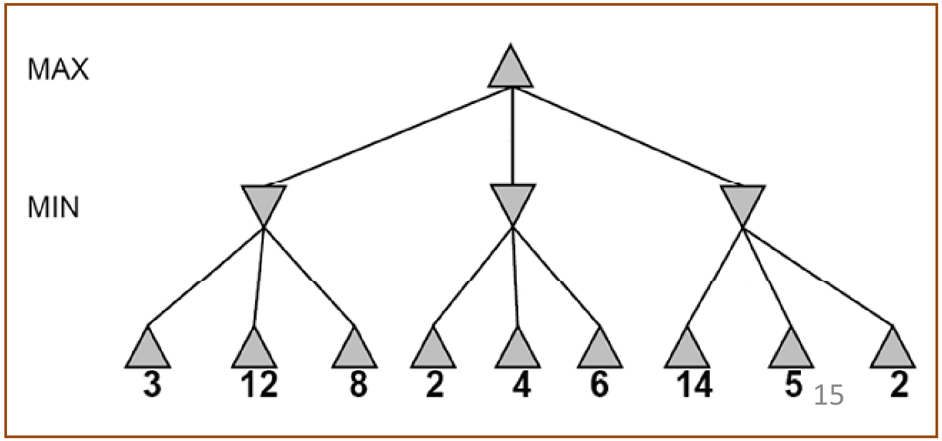
Alpha-beta pruning

- It is possible to compute the **exact minimax decision** <u>without</u> expanding every node in the game tree

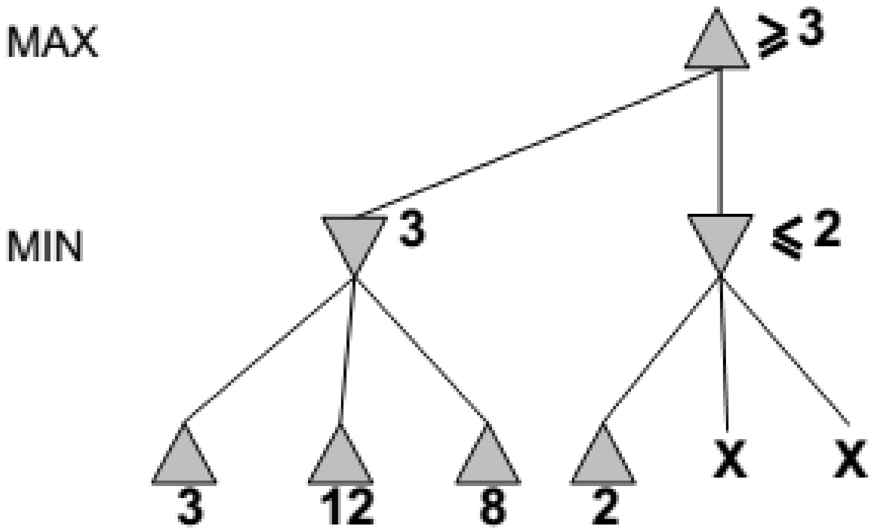


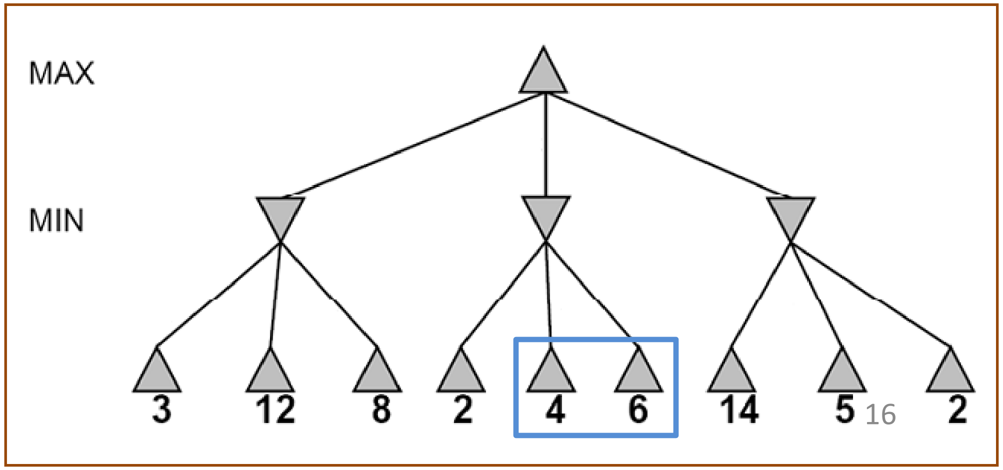
Alpha-beta pruning



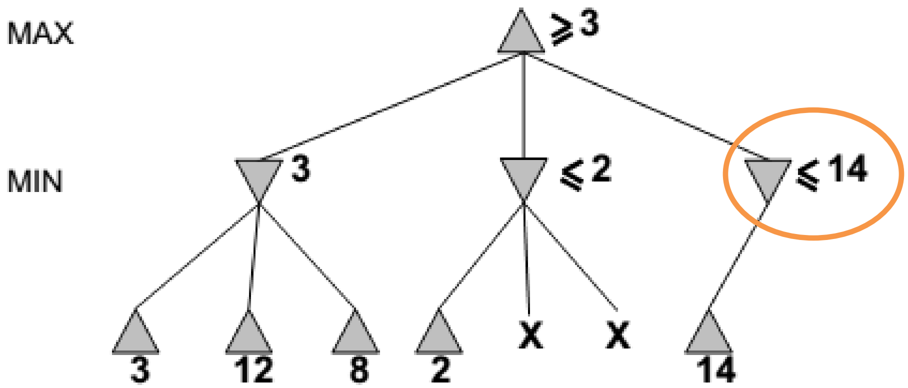


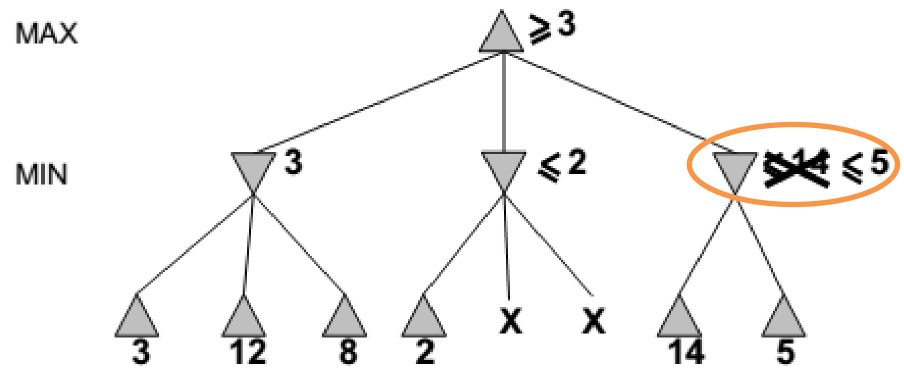
Alpha-beta pruning

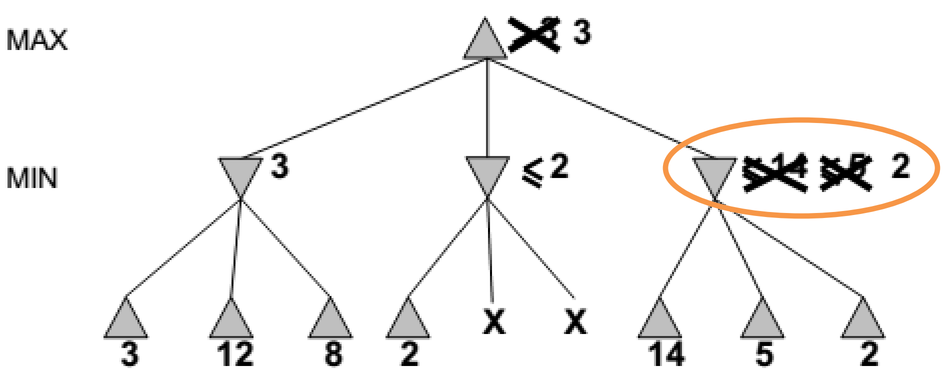


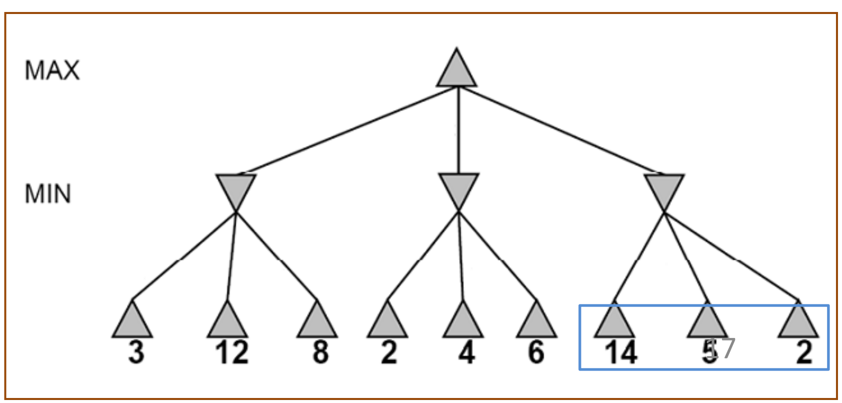


Alpha-beta pruning

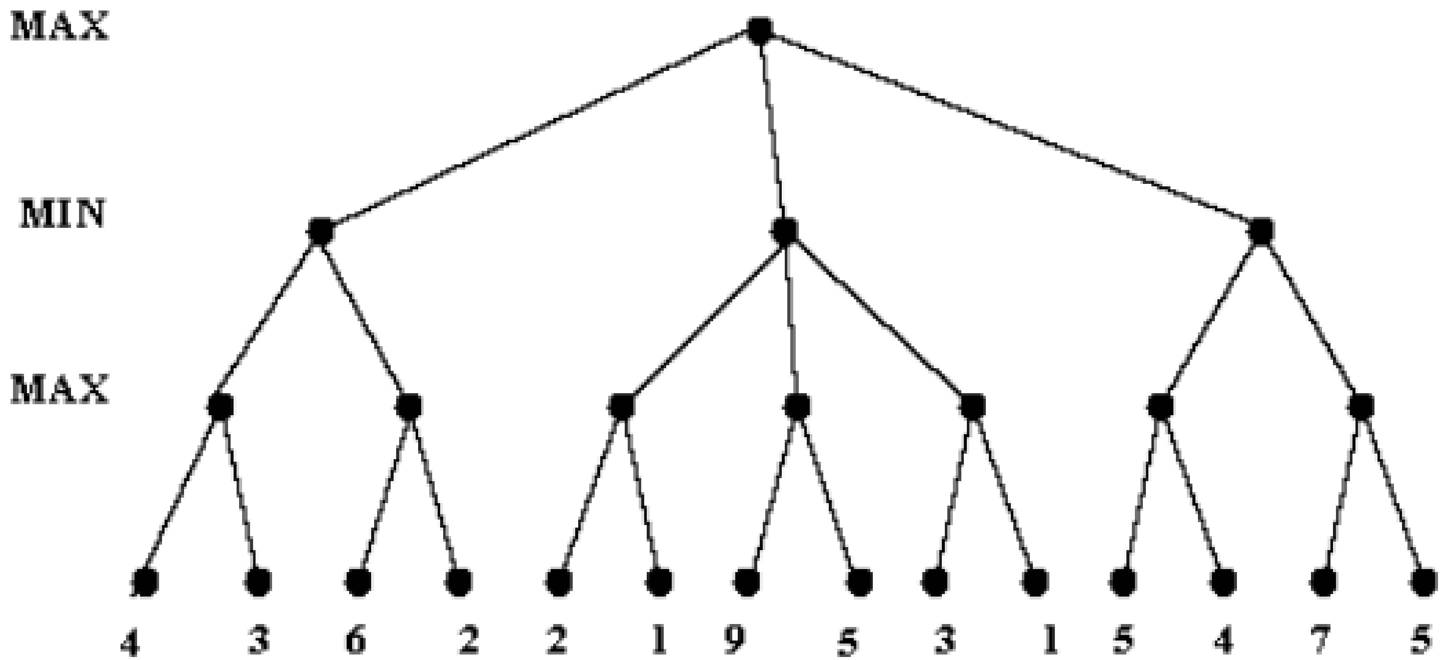




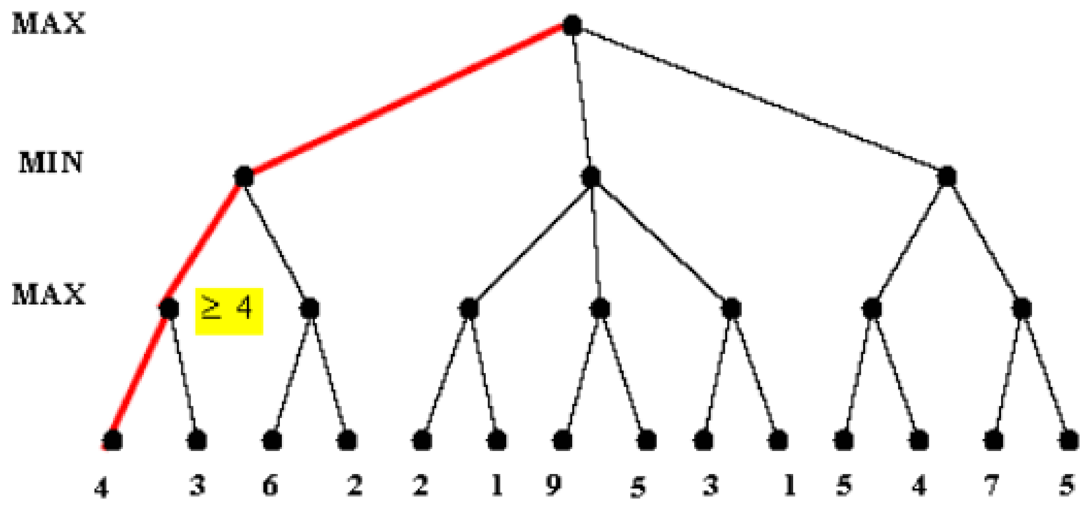


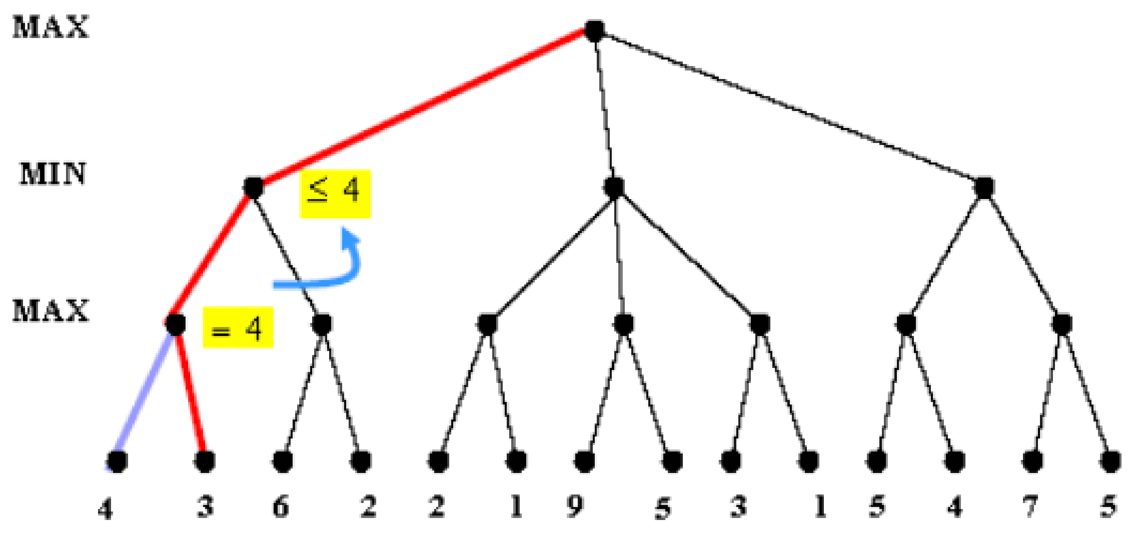


Example: Alpha-beta pruning

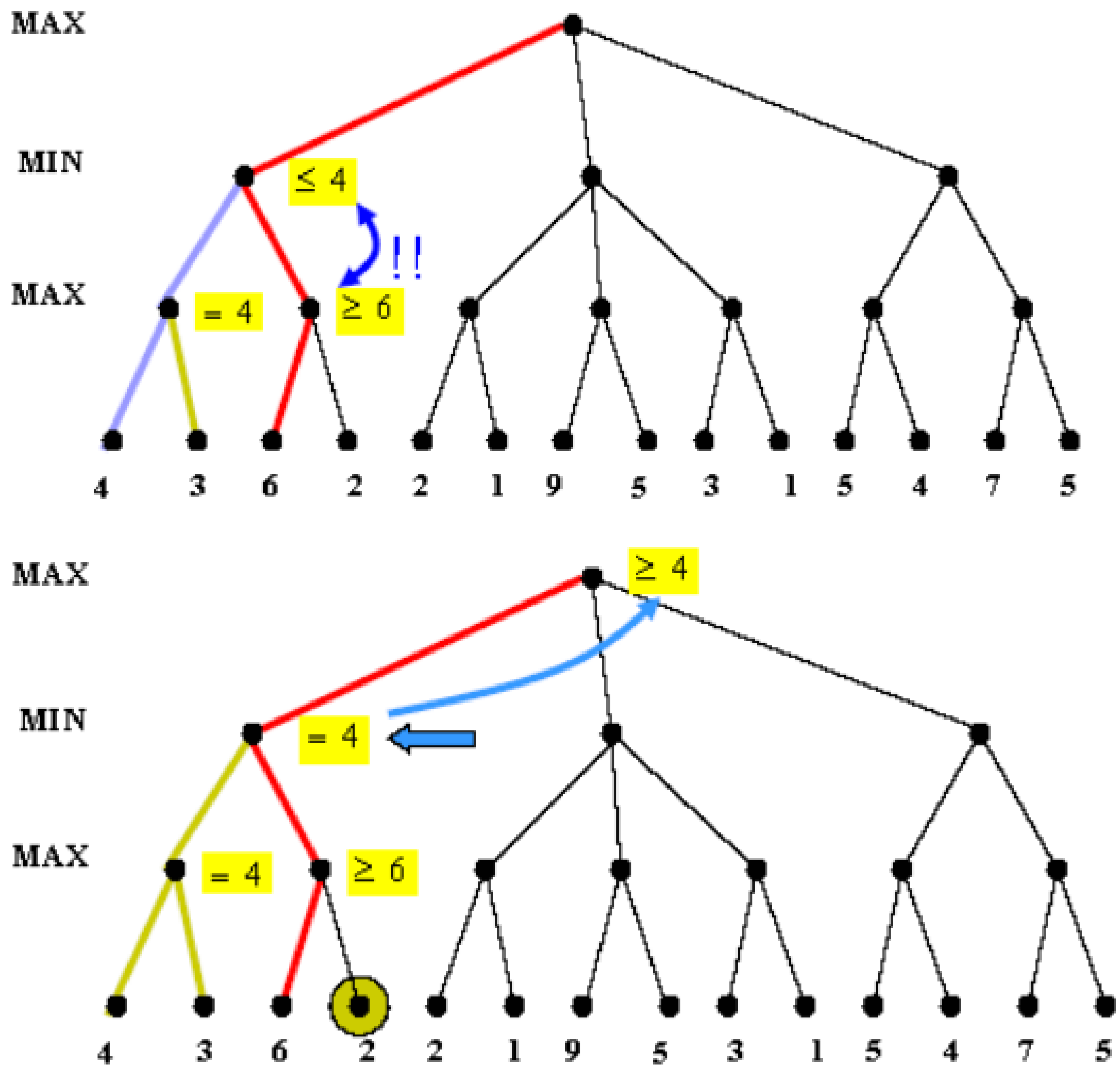


Example: Alpha-beta pruning

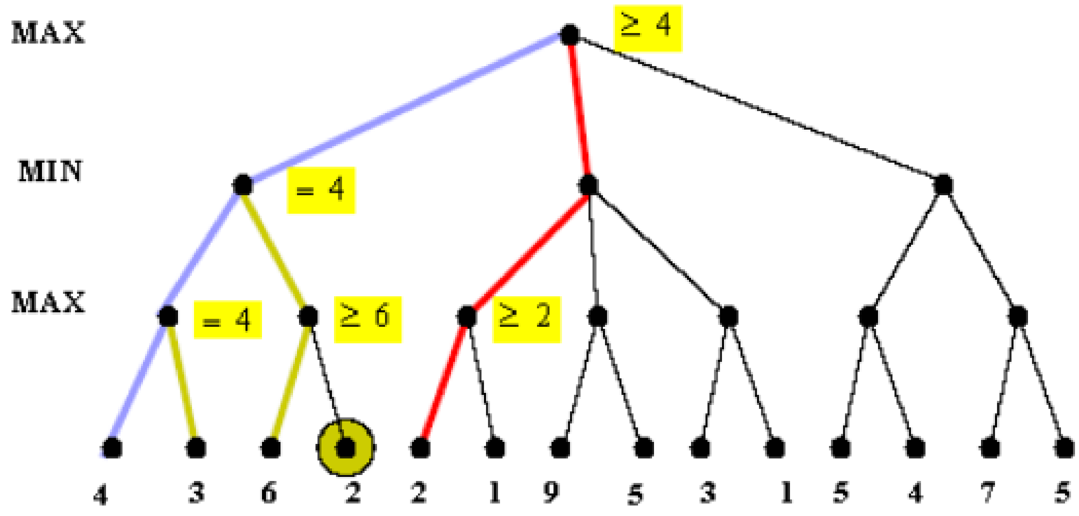


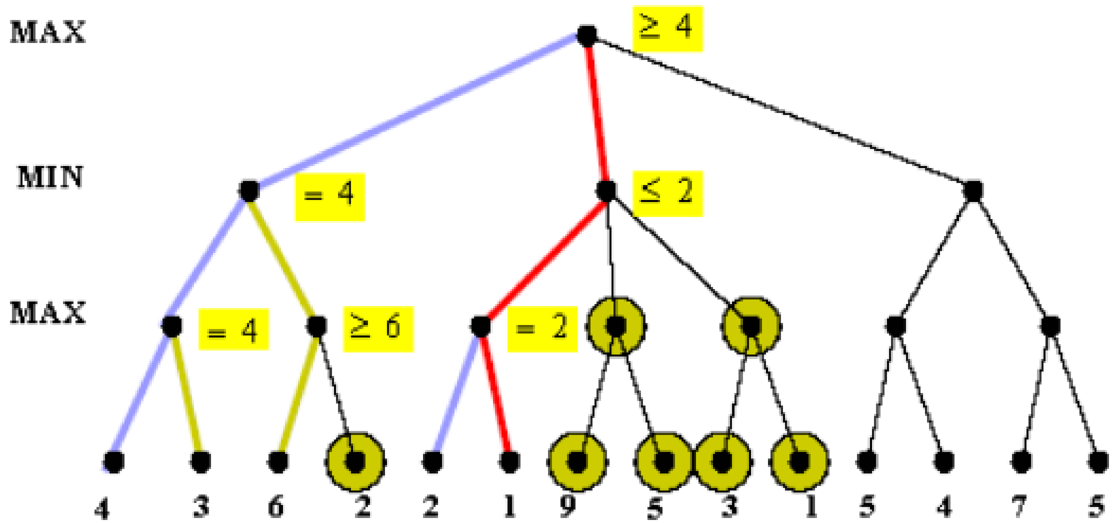


Example: Alpha-beta pruning

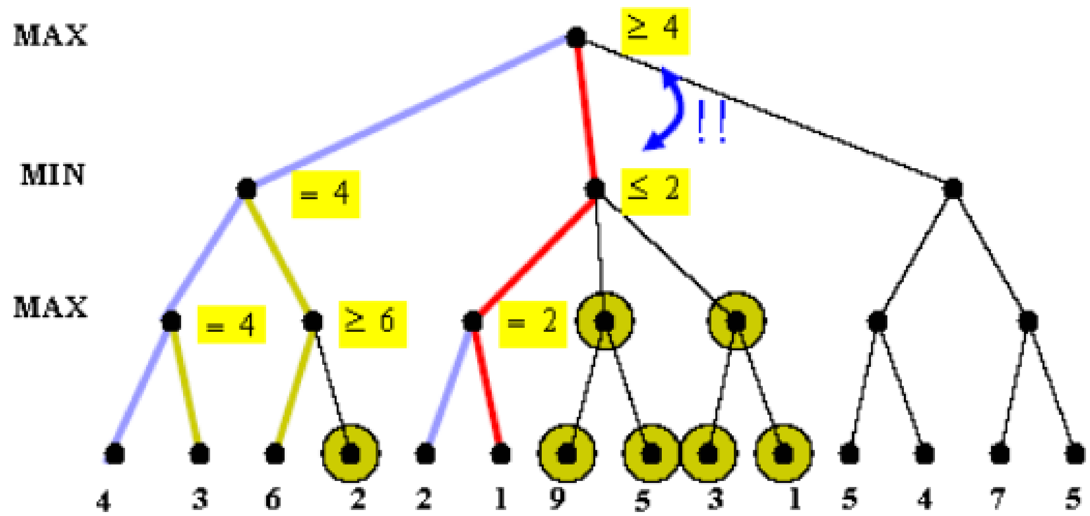


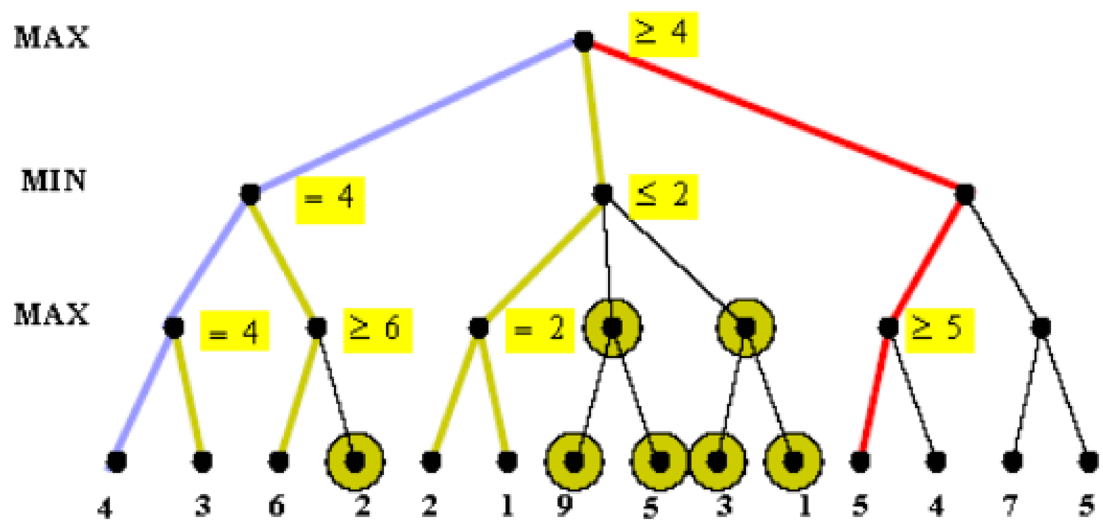
Example: Alpha-beta pruning



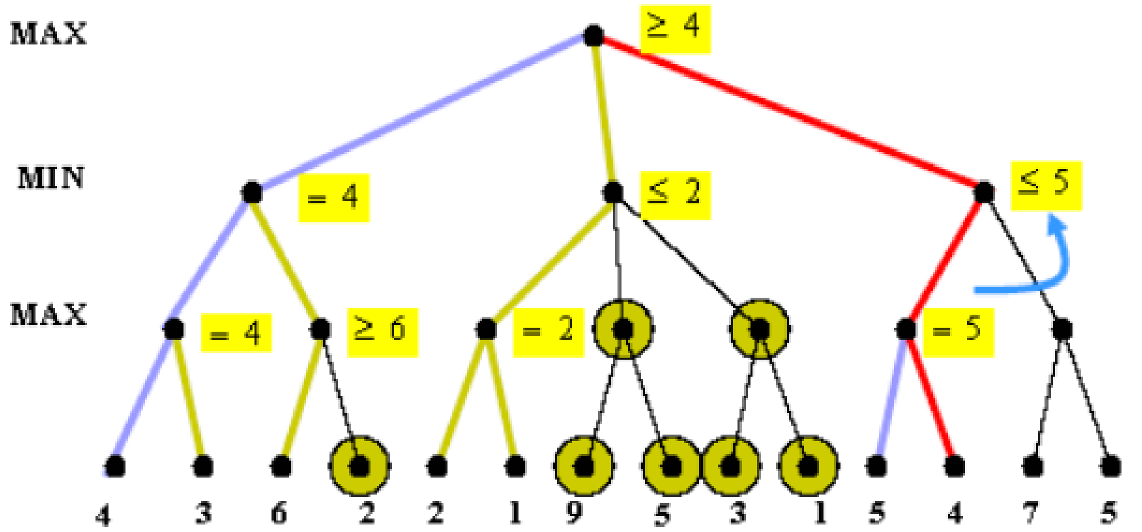


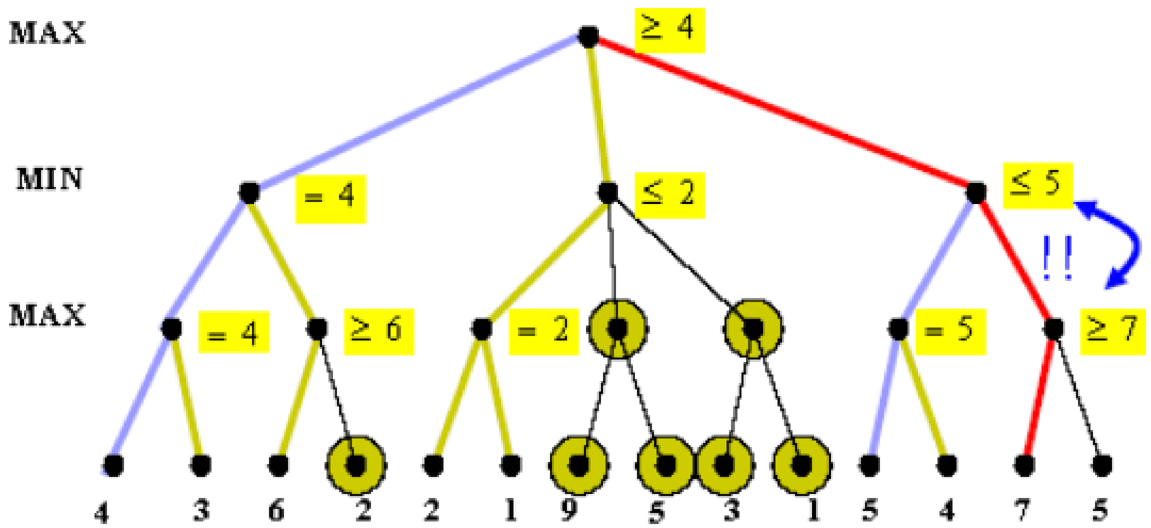
Example: Alpha-beta pruning



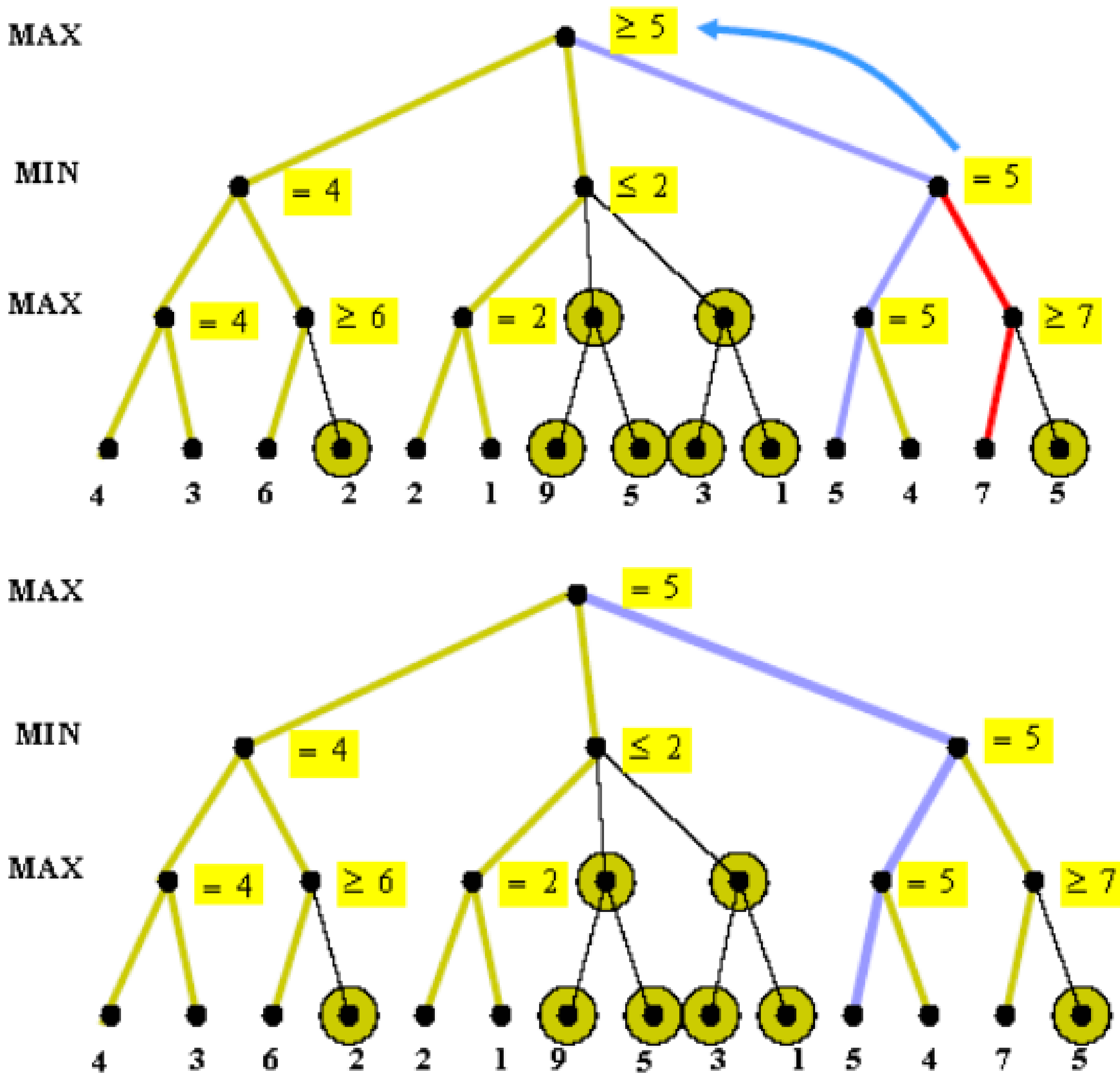


Example: Alpha-beta pruning



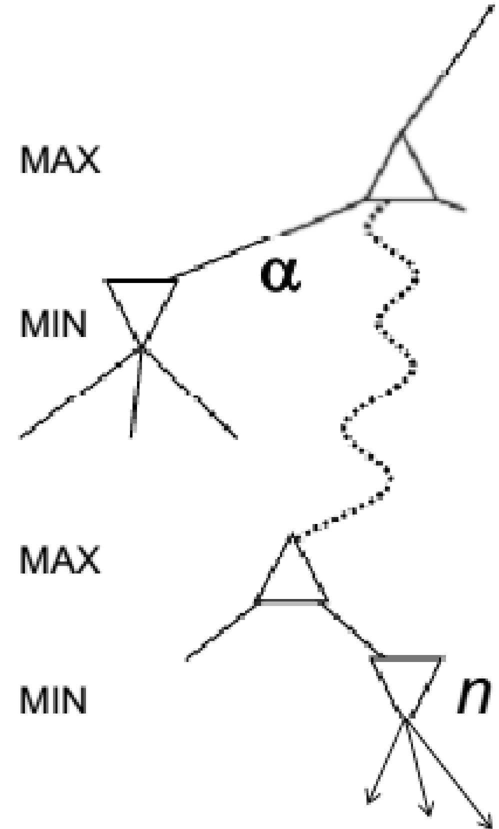


Example: Alpha-beta pruning



Alpha-beta pruning

- **α** is the value of **the best choice** for the MAX player found so far at any choice point above ***n***
- We want to compute the MAXMIN-value at ***n***
- As we loop over n's children, the MIN value decreases
- If it drops below **α**, MAX will never take this branch, so we can ignore *n*'s remaining children



Alpha-beta pruning

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With “perfect ordering”, time complexity = $O(b^{m/2})$

  ⇒**doubles** solvable depth

Evaluation function

- **Cut off** search at a certain depth and compute the value of an **evaluation function** for a state instead of its minimax value.
  - Use Cutoff-Test instead of Terminal-Test
    - e.g.,depth limit
  - Use Eval instead of Utility
    - i.e., evaluation function that estimates desirability of position

Evaluation function

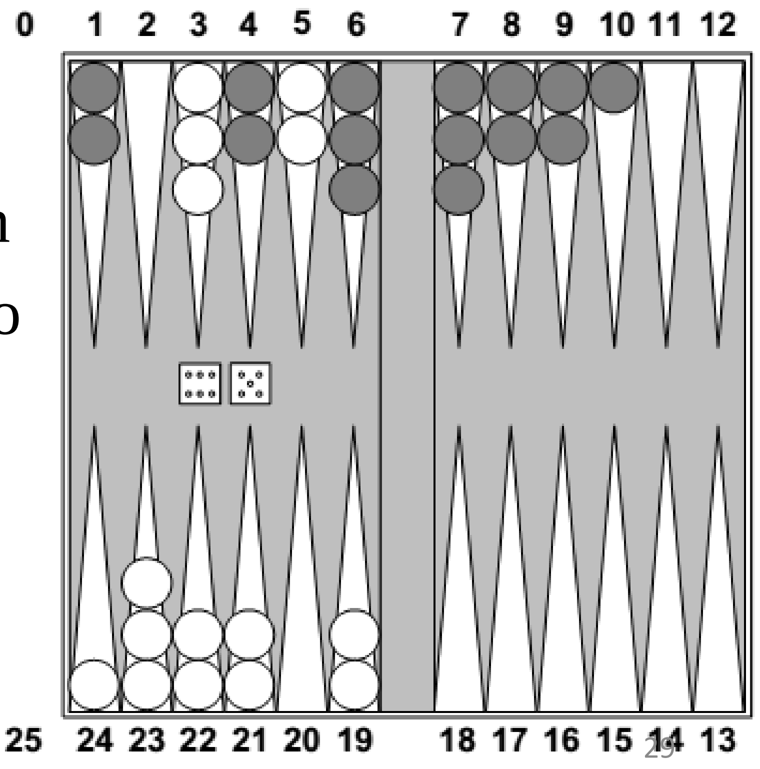
- the evaluation function may be thought of as the probability of winning from a given state or the expected value of that state.
- A common evaluation function is a **linear weighted sum** of features: $\text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
  - For chess,
    - wk may be the value of a piece

      pawn = 1, knight = 3, rook = 5, queen = 9
    - fk(s) may be the number of each kind of piece

      f1(S)= (number of white queens)-(number of black queens)
- Evaluation functions may be designed by a designer or by having the program play many games against itself.

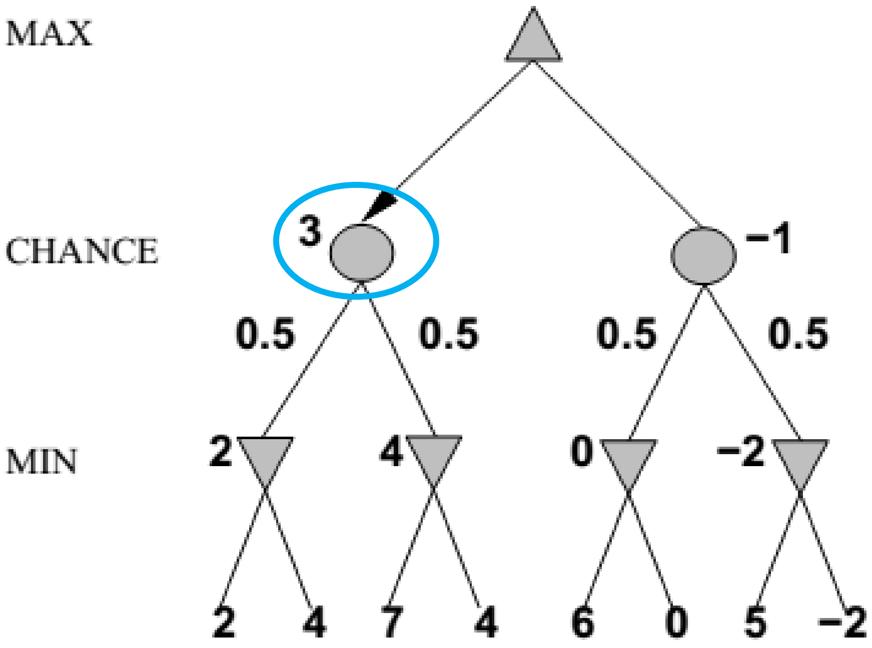
Games of chance

- Deterministic games in practice:
  - Chess: deep blue
  - Othello
  - Checkers: chinook
  - Go
- Nondeterministic game: backgammon
- How to incorporate **dice** throwing into the game tree?



Nondeterministic games

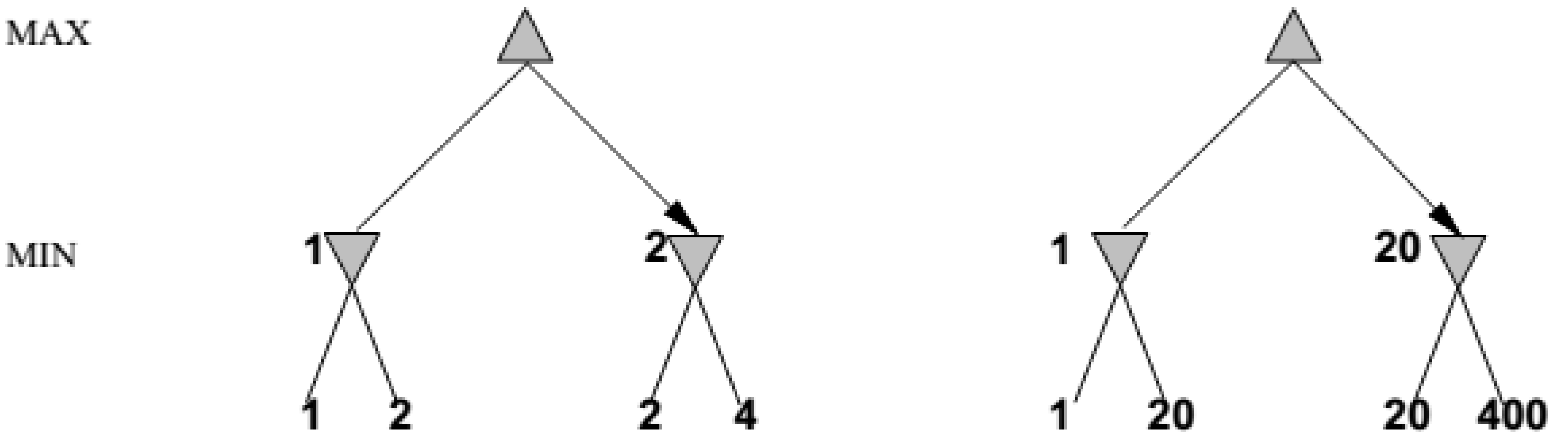
- simplified example with coin-flipping:

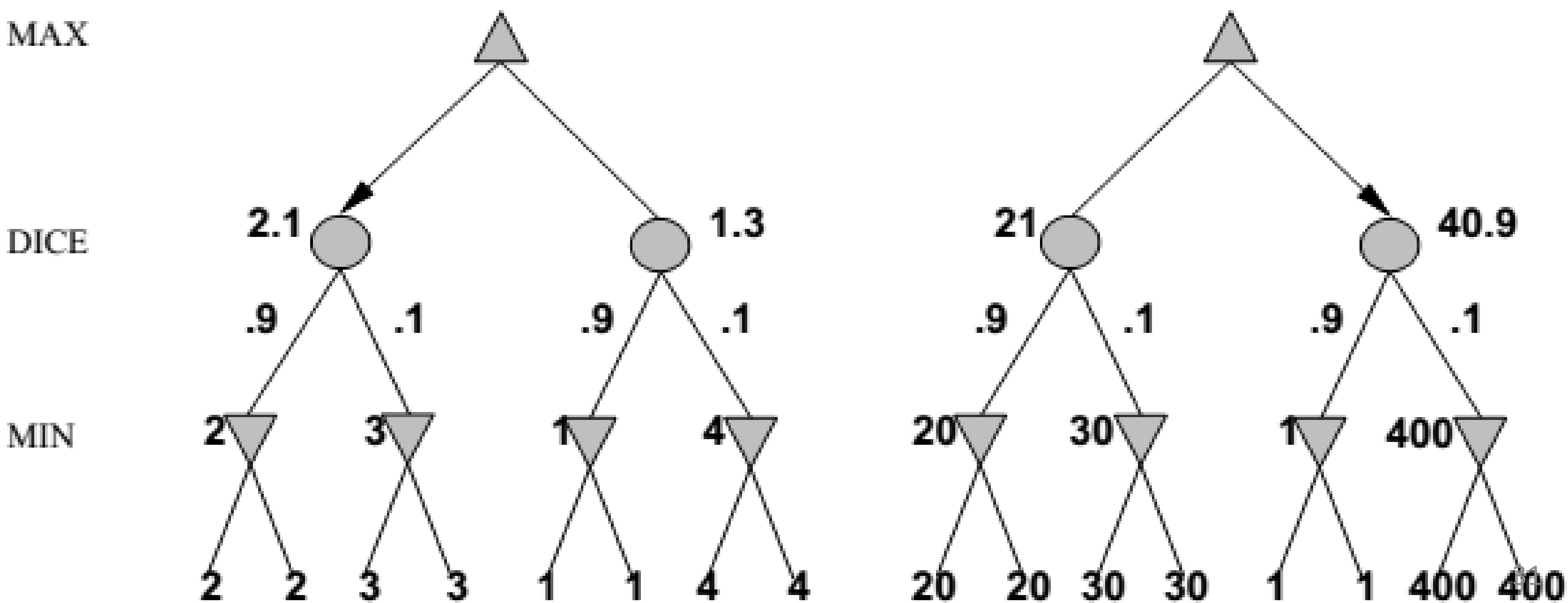


- Chance: 0.5*2+0.5*4=3

  0.5*0+0.5*(-2)= -1

Exact values don't/do matter???





Games of imperfect information

- What about this type of games???
  - Like: card games, where opponent's initial cards are unknown.

End of chapter 6