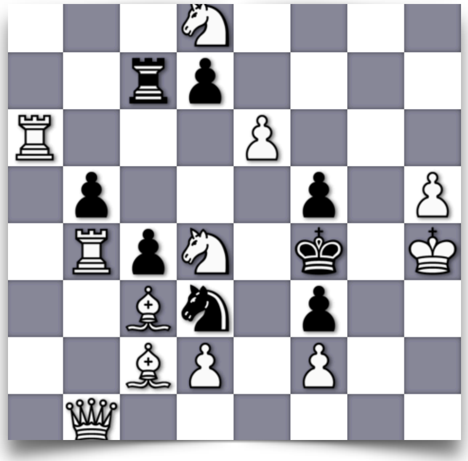
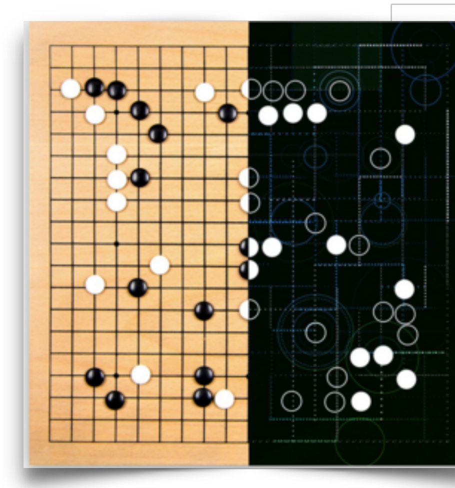
5.1. Game Playing
with Perfect Information

Outline

- Game playing as adversarial search
- Minimax
- Alpha-beta pruning



Game Playing



- In a competitive environment, agents' goals are in conflict, giving rise to adversarial search problems
- Most common games are deterministic, turn-taking, two-player, zero-sum games of perfect information (meaning deterministic, fully observable environments where two agents act alternately in which the utility values at the end of the game are always equal and opposite)
- Games are *too hard* to solve, e.g. chess has a branching factor of about 35 and each player plays up to 50 moves (infeasible to predict all the possible steps ahead because there are more chess moves than the number of atoms in the universe!). Go's average branching factor is 250!
- Like the real world, games require the ability to make *some* decision even when calculating the *optimal* decision is infeasible

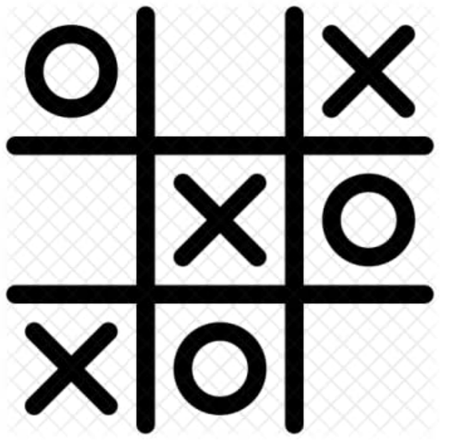
Games vs. Search Problems

- In normal search problem, the optimal solution would be a sequence of actions leading to a goal state
- "Unpredictable" opponent – solution is a strategy specifying a move for every possible opponent reply
- Time limits – unlikely to find goal, must approximate

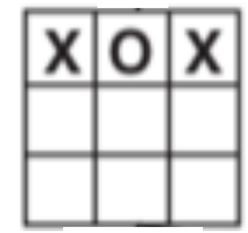
	deterministic	chance
perfect information there's one best way to play for each player	**chess, checkers, go, othello**	**backgammon monopoly**
imperfect information e.g. not all cards are visible to each player	**battleships, blind tictactoe**	**bridge, poker, scrabble nuclear war**

2-player Games

- Call players **Max** and **Min**; Max moves first, they take turns until the game is over (e.g. Tic-Tac-Toe, Chess, <u>Connect4</u>), at each turn you tend to make a move based what you think the opponent will do next
- At the end of the game, points are awarded to the winner and penalties given to the loser

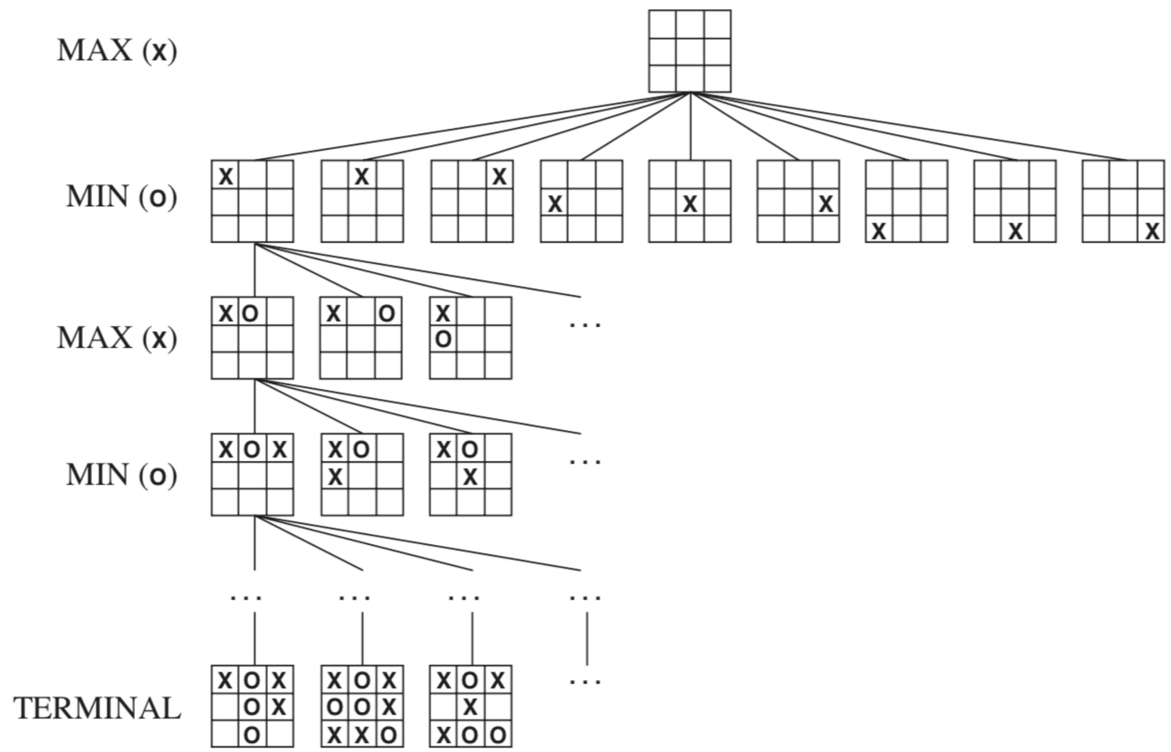


2-player Games cont.



- Game formal definition:
  - s_0 initial state
  - `Player(s)` which player has to move in state `s`
  - `Actions(s,a)` returns the set of legal moves in state `s`
  - `Result(s,a)` the transition model which defines the result of a move
  - `Terminal-test(s)` returns True when the game is over and False otherwise. States after the game has ended are called terminal states
  - `Utility(s, p)` also objective function or payoff function defines the final numeric value for a game that ends in terminal state `s` for a player p
    - ➡ In chess, the outcome is a win (1), loss (0) or draw (½)
    - ➡ A zero-sum game: total payoff to all players is the same for every instance of the game, e.g. chess payoff is either 0 + 1, 1 + 0, or ½ + ½ which is 1 all the time

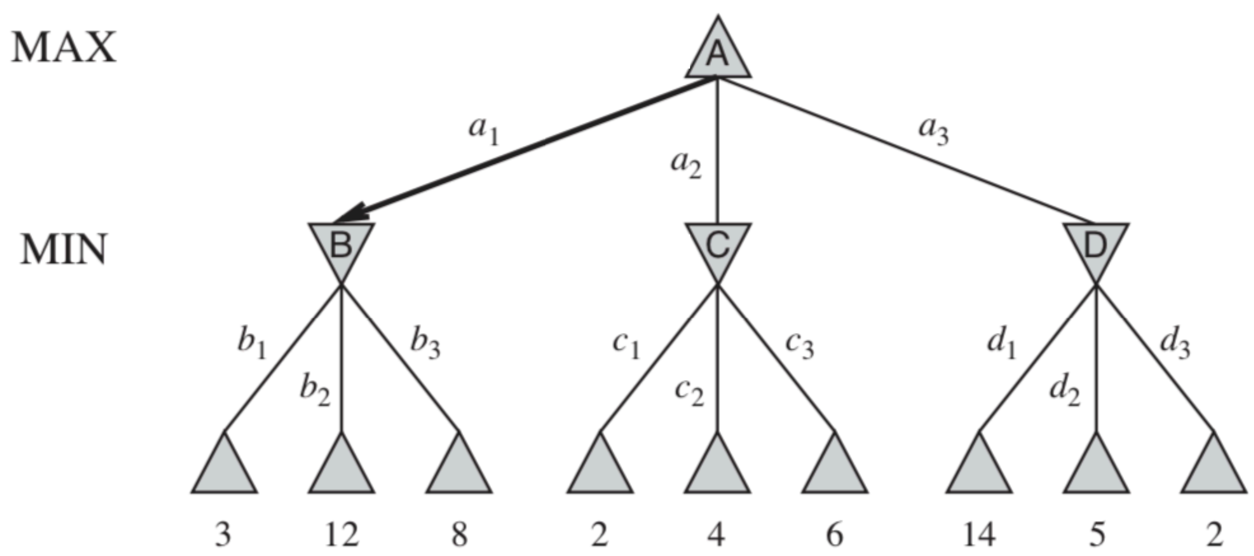
Game Tree for Tic-Tac-Toe



- For simple games like Tic-Tac-Toe with less than 9! (approx. 300,000) possible games, the moves can be represented as a game tree, nodes are the states and edges are the moves
- Alternate layers correspond to the different players
- Both players know all about the current state of the game
- Each leaf in the tree represents win for one player (or draw)

Toy Problem: Two-ply Game

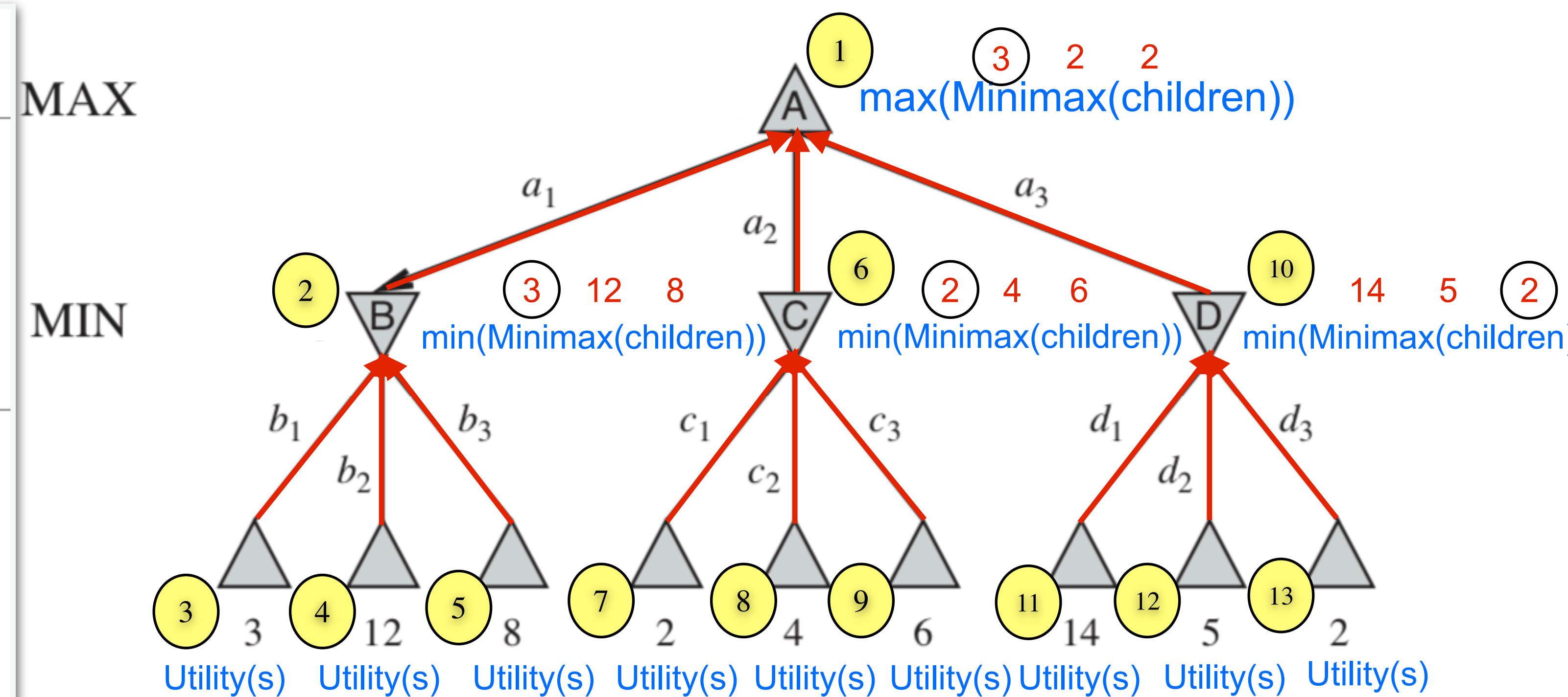
- Just two moves in the game, one move by Max followed by one move by Min
- Possible moves for Max at the root are a_1, a_2 and a_3 and possible moves for Min are $b_1,\ldots,d_3$
- Given a game tree, the optimal strategy can be determined from the minimax value of each node, `Minimax(n)` which is the utility (for Max) of being in the corresponding state, assuming that *both players play optimally* from there to the end of the game
- Minimax value of a terminal state is just its utility
- Max prefers prefers to move to a state of maximum value, whereas Min prefers a state of minimum value



* For this ply-game, what is the best move for:
- Max? ($a1/a2/a3$)
- Min after Max's move? ($b1/b2\ldots/d3$)

Minimax – Pseudocode

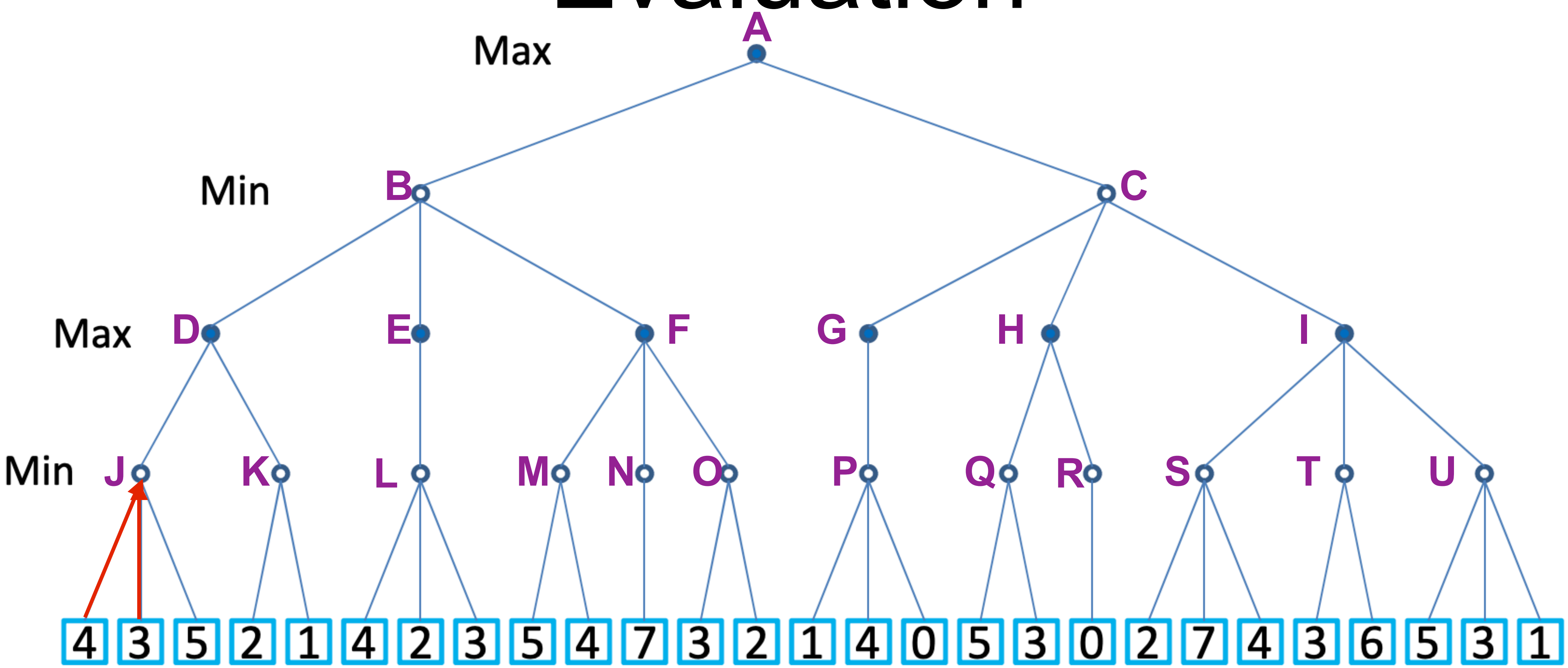
```
function MINIMAX-DECISION(state) returns an action
  return arg max_{a ∈ ACTIONS(s)} MIN-VALUE(RESULT(state, a))

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

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



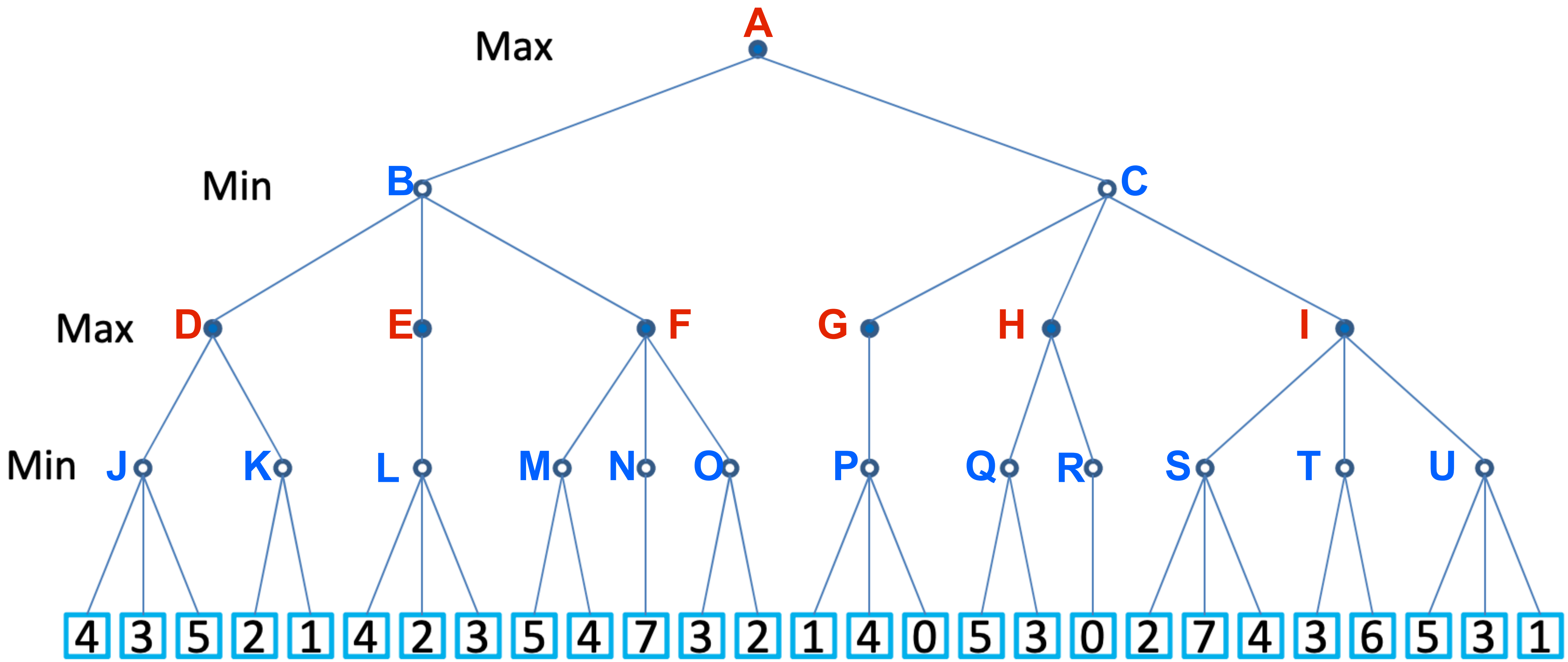
- Recursive functions have a recursive (self-calling) case and a base (trivial) case
- Here the base case applies to all leaf nodes and recursive calls are for all other nodes
- Leaf nodes return their utility values while non-leaf nodes calculate the minimum or maximum values of their children

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

Demo – Minimax Node Evaluation



- Order the evaluations by nodes and minimax values
- First two steps:
  1. J <= 4
  2. J <= 3



1. J <= 4
2. J <= 3
3. **J = 3**
4. D >= 3
5. K <= 2
6. **K = 1**
7. **D = 3**
8. B <= 3
9. L <= 4
10. L <= 2
11. **L = 2**
12. **E = 2**
13. B <= 2
14. M <= 5
15. **M = 4**
16. F >= 4
17. **N = 7**
18. F >= 7
19. O <= 3
20. **O = 2**
21. **F = 7**
22. **B = 2**
23. A >= 2
24. P <= 1
25. P <= 1
26. **P = 0**
27. **G = 0**
28. C <= 0
29. Q <= 5
30. **Q = 3**
31. H >= 3
32. **R = 0**
33. **H = 3**
34. C <= 0
35. S <= 2
36. S <= 2
37. **S = 2**
38. I >= 2
39. T <= 3
40. **T = 3**
41. I >= 3
42. U <= 5
43. U <= 3
44. **U = 1**
45. **I = 3**
46. **C = 0**
47. **A = 2**

References

- Russel and Norvig, Chapter 5, until 5.2
- S. Markovitch, Minimax algorithm animation [Video]

Quiz Week 6 – 15%

- Written quiz next week Wednesday, April 9th during class (3.00pm – 4.15pm)
- Closed-book exam – NO alternate online quiz will be given to those who have a valid excuse for absence (score for quiz will be based on your final exam or other assignments)
- Cover all topics until minimax
- NO notes, calculators, mobile phones, scrap paper – you will be given a quiz booklet
- Use pen or pencil+eraser (refrain from using red ink)
- You will be given a sheet containing all the main algorithms
- Remember to review agents and rationality too