Game Playing

Alpha–Beta Pruning

Minimax: Properties

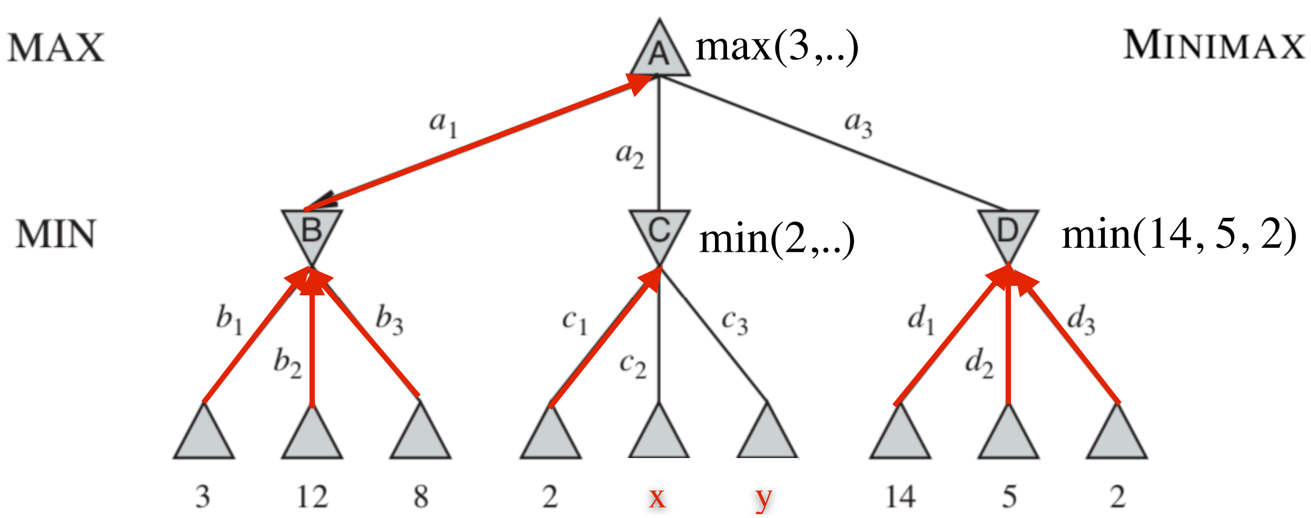
- Depth-first traversal (branching factor b, depth m)
- Complete? Yes if tree is finite
- Optimal? Yes against an optimal opponent
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$
- Time cost is not practical for real games (Chess: $m \approx 100$, $b \approx 35$)

Minimax Problems

- The number of game states it has to examine is exponential in the depth of the tree
- Can we ignore some nodes?
- Can eliminate large parts of the tree using pruning
  - eliminating possibilities from consideration without having to examine them
  - allows us to ignore portions of the search tree that make no difference to the final choice

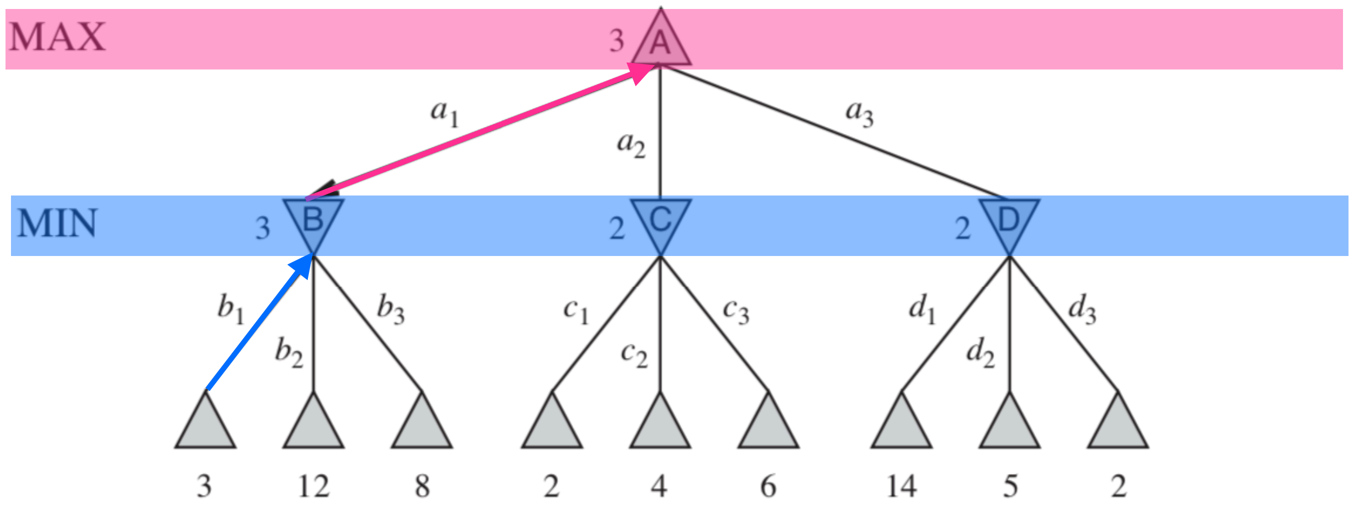
Minimax: Are There Shortcuts?

- Minimax problem: Number of game states it has to examine is exponential in the depth of the tree
- We can cut it in half by pruning, i.e. ignore portions of the search tree that makes ***no difference*** to the final choice (reduce the number of evaluations and branching)
- Calculation of optimal decision by considering what we already know at each point in the process could lead to minimax decision without evaluating some nodes
- Remove the nodes that don't have to be evaluated – removing redundancy



$$
\begin{aligned}
\textsc{Minimax}(root) &= \max(\min(3,12,8), \min(2,x,y), \min(14,5,2)) \\
&= \max(3, \min(2,x,y), 2) \\
&= \max(3, z, 2) \qquad \boxed{\text{where } z = \min(2,x,y) \leq 2} \\
&= 3.
\end{aligned}
$$

Node Evaluation



- $B \leq 3$ MIN – the biggest value that I can have is 3
- $A \geq 3$ MAX – the smallest value that I can have is 3

8. Alpha-Beta Pruning

- Main idea: If Player has a choice to move to node *n* for consideration, and if there is already a better choice of value for Player from previously processed nodes, then *n* can be ignored

- Alpha (α) is concerning what is the minimum, >= that I can take (worst-case scenario), i.e. the first player who is trying to maximise the score

- Beta (β) is concerning what is the maximum, <= that I have to give (worst-case scenario), i.e. the second player who is trying to minimise the score

- Pruning (termination of the recursive call) happens when the value of the current node is **worse than** the current alpha (Max) or beta (Min)

Alpha-Beta: Pseudocode

```
function ALPHA-BETA-SEARCH(state) returns an action
  v ← MAX-VALUE(state, −∞, +∞)
  return the action in ACTIONS(state) with value v
```

```
function MAX-VALUE(state, α, β) 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), α, β))
    if v ≥ β then return v
    α ← MAX(α, v)
  return v
```

```
function MIN-VALUE(state, α, β) 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) , α, β))
    if v ≤ α then return v
    β ← MIN(β, v)
  return v
```

- 2 main functions with 3 parameters – s (current state), α (best explored option for Max from root to s) and β (best explored option for Min from root to s) and an output value, v
- At the start node:
  - Smallest value that I can have, $\alpha = -\infty$
  - Biggest value that I can have, $\beta = +\infty$
- Keep track of alpha & beta globally and locally (lower and upper bounds)
- v is the value used to manipulate α and β and it is passed back from the function

$\alpha - \beta$ Pruning Walkthrough

function **Alpha-Beta-Search**(s) returns an action
$s \leftarrow$ root   *# start from root*
$v \leftarrow$ **Minimax-ab**($s, -\infty, +\infty$)   *# initialise alpha and beta*
return the action in Actions(s) with value v

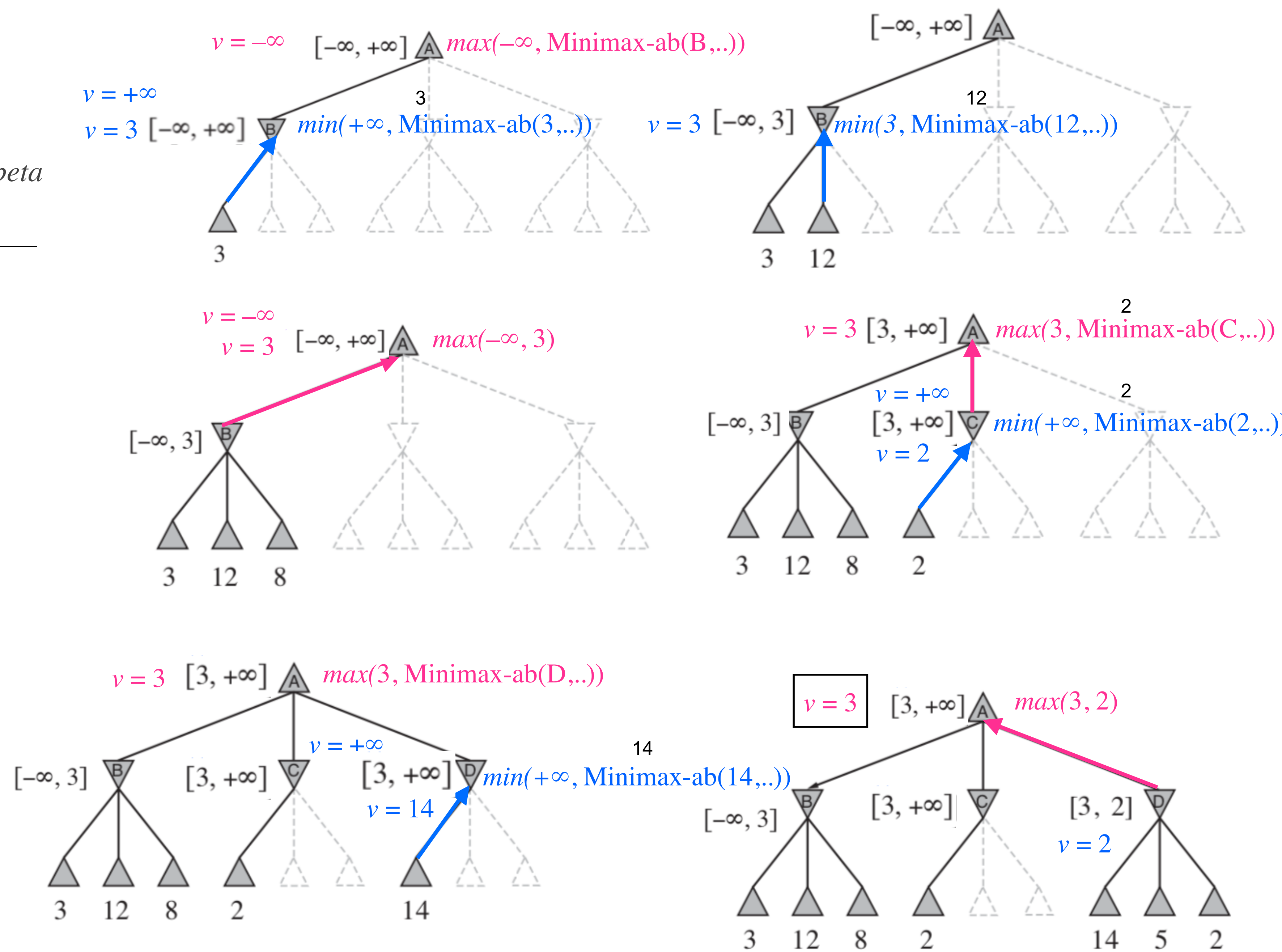
---

function **Minimax-ab**(s, α, β) returns a value v
if Terminal-test(s) then **return** Utility(s) *# base case*

else if (Player == Max)
$v \leftarrow -\infty$ *# reset v to $-\infty$ at every Max Player node*
for each child, c of s
    $v \leftarrow$ max(v, **Minimax-ab**(c, α, β))
    if $v \geq \beta$ then **return** v *# Pruning*
    else $\alpha \leftarrow$ max(α, v)

else if (Player == Min)
$v \leftarrow +\infty$ *# reset v to $+\infty$ at every Min Player* node
for each child, c of s
    $v \leftarrow$ min(v, **Minimax-ab**(c, α, β))
    if $v \leq \alpha$ then **return** v *# Pruning*
    else $\beta \leftarrow$ min(β,v)

return v



Alpha-beta Pruning Points

- α and β are inherited from the parent and they are manipulated locally at the current node for its own use
  - but they are not passed back up. Only v is passed back up (via return)
- At every node, v is reset to $-\infty$ if it is the node is a Max player or $+\infty$ if it is a Min Player
- At Max, **pruning** happens when v is bigger than β – Min Player above will ignore this path because it is worse (bigger) than the best that they have so far, β
- At Min, **pruning** happens when v is smaller than α – Max Player above will ignore this path because it is worse (smaller) than the best they have so far, α

Alpha–Beta: Properties

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With perfect ordering, time complexity = $O(b^{\frac{m}{2}})$
  - ➡ doubles solvable depth
  - ➡ can easily reach depth 8 and play good chess

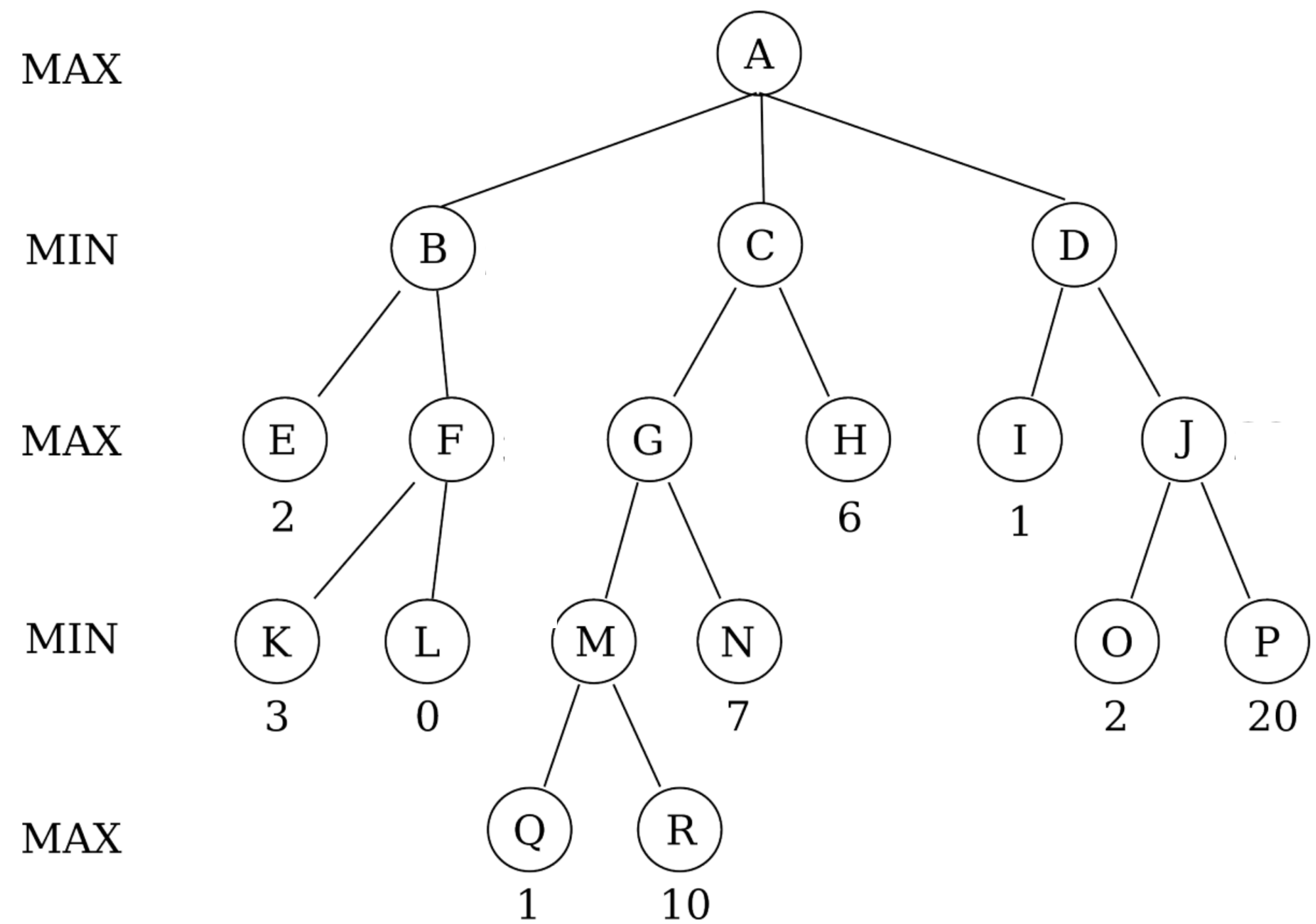
Summary

- Game playing as adversarial search
  - zero-sum games
  - utility values
- Search in games with perfect information:
  - Minimax
  - Alpha-beta pruning
    - ➡ Alpha-Beta has been used by popular programs like Deep Blue to efficiently play against Chess Grandmasters

References

- Russel and Norvig, Chapter 5, until 5.3
- J. Schrum, Alpha-beta pruning intuition [Video]
- S. Kambhapati, Alpha-beta intuition [Video]
- Historical reading:
  - Computer considers possible lines of play (Babbage, 1846)
  - Algorithm for perfect play (Zermelo, 1912; Von Neumann 1944)
  - Finite horizon, approximate evaluation (Zuse, 1945; Wiener, 1948; Shannon 1950)
  - First chess program (Turing, 1950)
  - Machine learning to improve evaluation accuracy (Samuel, 1952–57)
  - Pruning to allow deeper search (McCarthy, 1956)

Exercise: $\alpha - \beta$ Pruning



- Order the evaluations by nodes and α, β and v values
- First step:

1. $A : \alpha = -\infty, \beta = +\infty, v = -\infty$

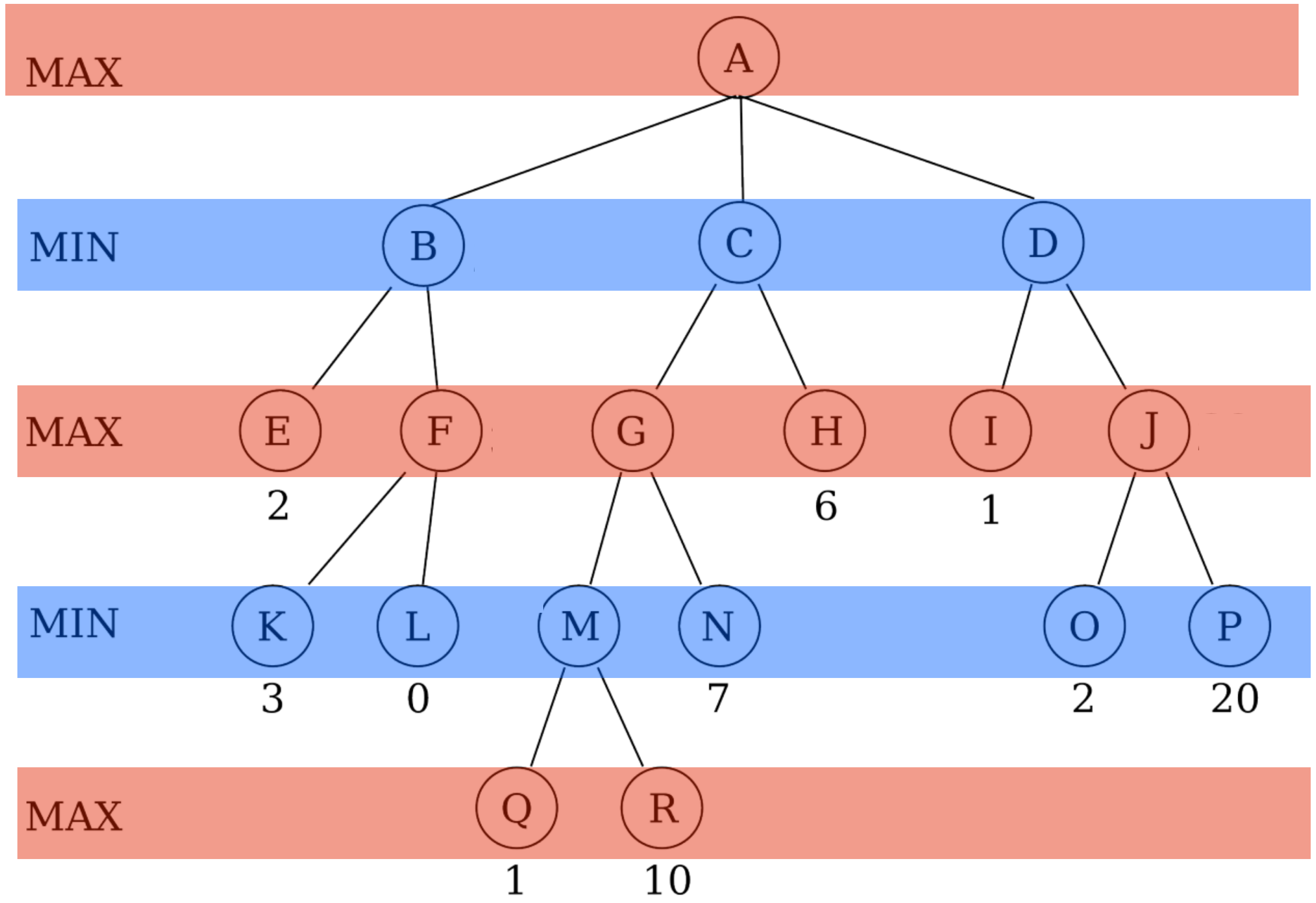
```
function Alpha-Beta-Search(s) returns an action
s ⟵ root   # start from root
v ⟵ Minimax-ab(s, −∞, +∞)   # initialise alpha and beta
return the action in Actions(s) with value v
```

```
function Minimax-ab(s, α, β) returns a value v
if Terminal-test(s) then return Utility(s)  # base case

else if (Player == Max)
v ⟵ −∞  # reset v to −∞ at every Max Player node
for each child, c of s
        v ⟵ max(v, Minimax-ab(c, α, β))
        if v ≥ β then return v  # Pruning
        else α ⟵ max(α, v)

else if (Player == Min)
v ⟵ +∞  # reset v to +∞ at every Min Player node
for each child, c of s
        v ⟵ min(v, Minimax-ab(c, α, β))
        if v ≤ α then return v   # Pruning
        else β ⟵ min(β,v)

return v
```



function **Alpha-Beta-Search**(s) returns an action
$s \leftarrow$ root *# start from root*
$v \leftarrow$ **Minimax-ab**(s, $-\infty$, $+\infty$) *# initialise alpha and beta*
return the action in Actions(s) with value v

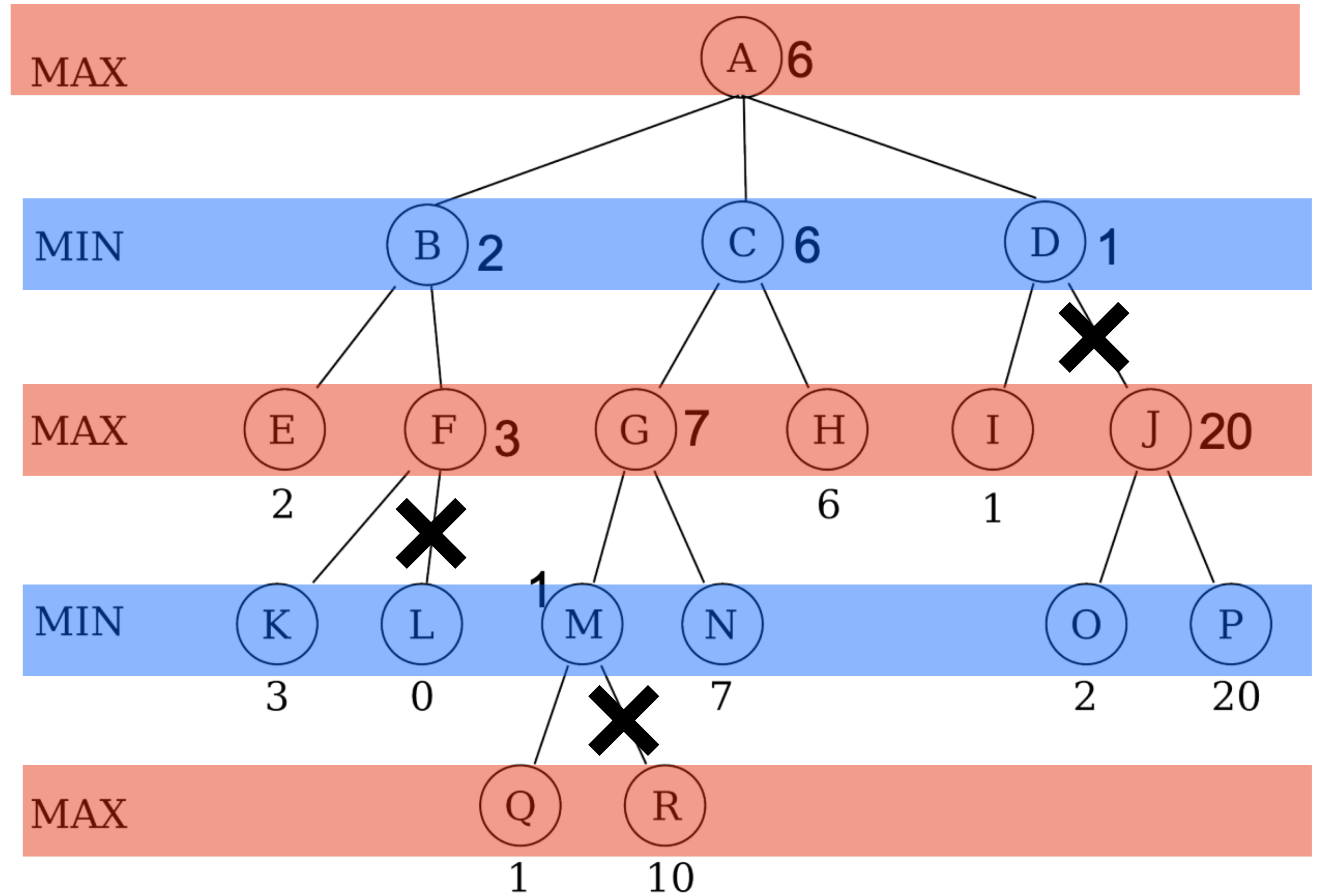
---

function **Minimax-ab**(s, α, β) returns a value v
if Terminal-test(s) then **return** Utility(s) *# base case*

else if (Player == Max)
$v \leftarrow -\infty$ *# reset v to $-\infty$ at every Max Player node*
for each child, c of s
  $v \leftarrow$ max(v, **Minimax-ab**(c, α, β))
  if $v \geq \beta$ then **return** v *# Pruning*
  else $\alpha \leftarrow$ max(α, v)

else if (Player == Min)
$v \leftarrow +\infty$ *# reset v to $+\infty$ at every Min Player node*
for each child, c of s
  $v \leftarrow$ min(v, **Minimax-ab**(c, α, β))
  if $v \leq \alpha$ then **return** v *# Pruning*
  else $\beta \leftarrow$ min(β,v)

return v



1. A: α=-∞, β=+∞, v=-∞
2. B: α=-∞, β=+∞, v=+∞
3. E: α=-∞, β=+∞, v=2
4. B: α=-∞, β=$\cancel{+\infty}$ 2, v=$\cancel{+\infty}$ 2
5. F: α=-∞, β=2, v=-∞
6. K: α=-∞, β=2, v=3
7. F: α=-∞, β=2, v=$\cancel{-\infty}$ 3 (L ignored)
8. B: α=-∞, β=2, v=2
9. A: α=$\cancel{-\infty}$ 2, β=+∞, v=$\cancel{-\infty}$ 2
10. C: α=2, β=+∞, v=+∞
11. G: α=2, β=+∞, v=-∞
12. M: α=2, β=+∞, v=+∞
13. Q: α=2, β=+∞, v=1
14. M: α=2, β=+∞, v=$\cancel{+\infty}$ 1 (R ignored)
15. G: α=2, β=+∞, v=$\cancel{-\infty}$ 1
16. N: α=2, β=+∞, v=7
17. G: α=$\cancel{2}$ 7, β=+∞, v=$\cancel{1}$ 7
18. C: α=2, β=$\cancel{+\infty}$ 7, v=$\cancel{+\infty}$ 7
19. H: α=2, β=7, v=6
20. C: α=2, β=$\cancel{7}$ 6, v=$\cancel{7}$ 6
21. A: α=$\cancel{2}$ 6, β=+∞, v=$\cancel{2}$ 6
22. D: α=6, β=+∞, v=+∞
23. I: α=6, β=+∞, v=1
24. D: α=6, β=+∞, v=$\cancel{+\infty}$ 1 (J,O,P ignored)
25. A: α=6, β=+∞, v=6

Q: Which move should Max take? A→C

Q. Which move should Min take after that? C→H

Poll Results