

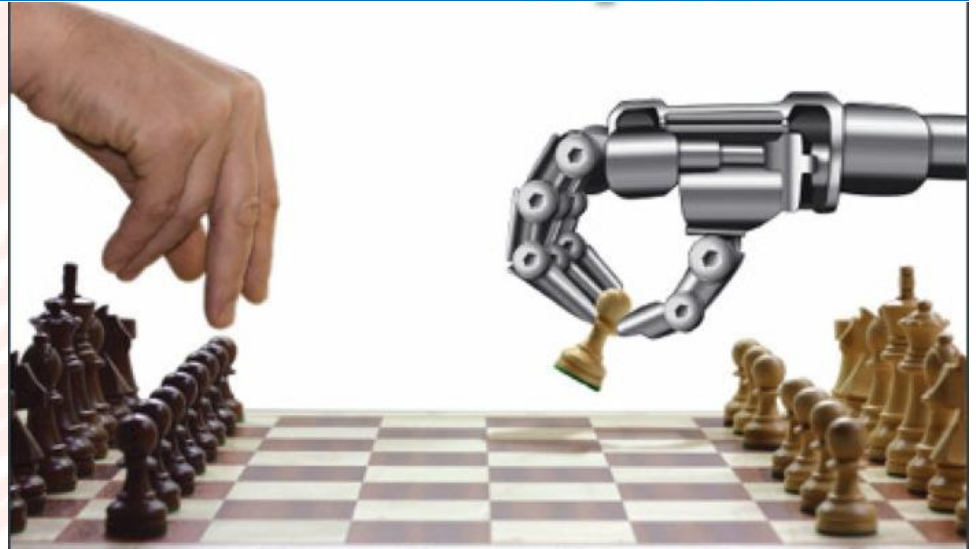


FACULTY OF INFORMATION TECHNOLOGY

# Artificial Intelligence Fundamentals (NM TTNT)

Semester 1, 2023/2024

# Chapter 5. Adversarial Search



# Content

- ▶ What are games?
- ▶ Optimal decisions in games
  - Which strategy leads to success?
- ▶  $\alpha$ - $\beta$  pruning
- ▶ Games of imperfect information
- ▶ Games that include an element of chance

# What are and why study games?

- ▶ Games are a form of multi-agent environment
  - What do other agents do and how do they affect our success?
  - Cooperative vs. competitive multi-agent environments.
  - Competitive multi-agent environments give rise to adversarial problems a.k.a. games
- ▶ Game playing is a good problem for AI research

# What are and why study games?

- ▶ Game playing is non-trivial
  - Players **need "human-like" intelligence**
  - Games can be very **complex** (e.g. chess, go)
  - Requires **decision making within limited time**
- ▶ Games usually are:
  - Well-defined and repeatable
  - Limited and accessible

# Types of Games

## ► Perfect Information (fully observable)

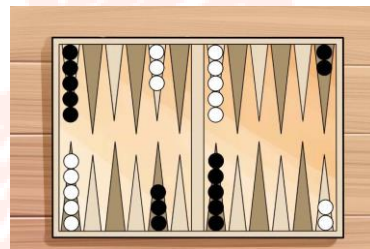
### ◦ Deterministic

- Chess,
- Checkers,
- Go,
- Othello



### ◦ Chance

- Backgammon,
- Monopoly





# Types of Games

## ► Imperfect Information (partially observable)

- Deterministic

- Stratego
- Battleship



- Chance

- Brigde,
- Poker,
- Scrabble,
- Nuclear War



# Types of Games

- ▶ This course focuses on:
  - Perfect Information (fully observable)
  - Deterministic





# Relation of Games to Search

- ▶ Solution is (heuristic) method for finding goal
- ▶ **Heuristics** and **CSP** (Constraint Satisfaction Problems) techniques can find *optimal* solution
- ▶ **Evaluation function**: estimate of **cost from start to goal** through given node
- ▶ Examples: path planning, scheduling activities
- ▶ Solution is strategy (specifies move for every possible opponent reply).
- ▶ Time limits force an approximate solution
- ▶ **Evaluation function**: evaluate "**goodness**" of game position
- ▶ Examples: chess, checkers, Othello, backgammon

Search – no adversary

Games– adversary

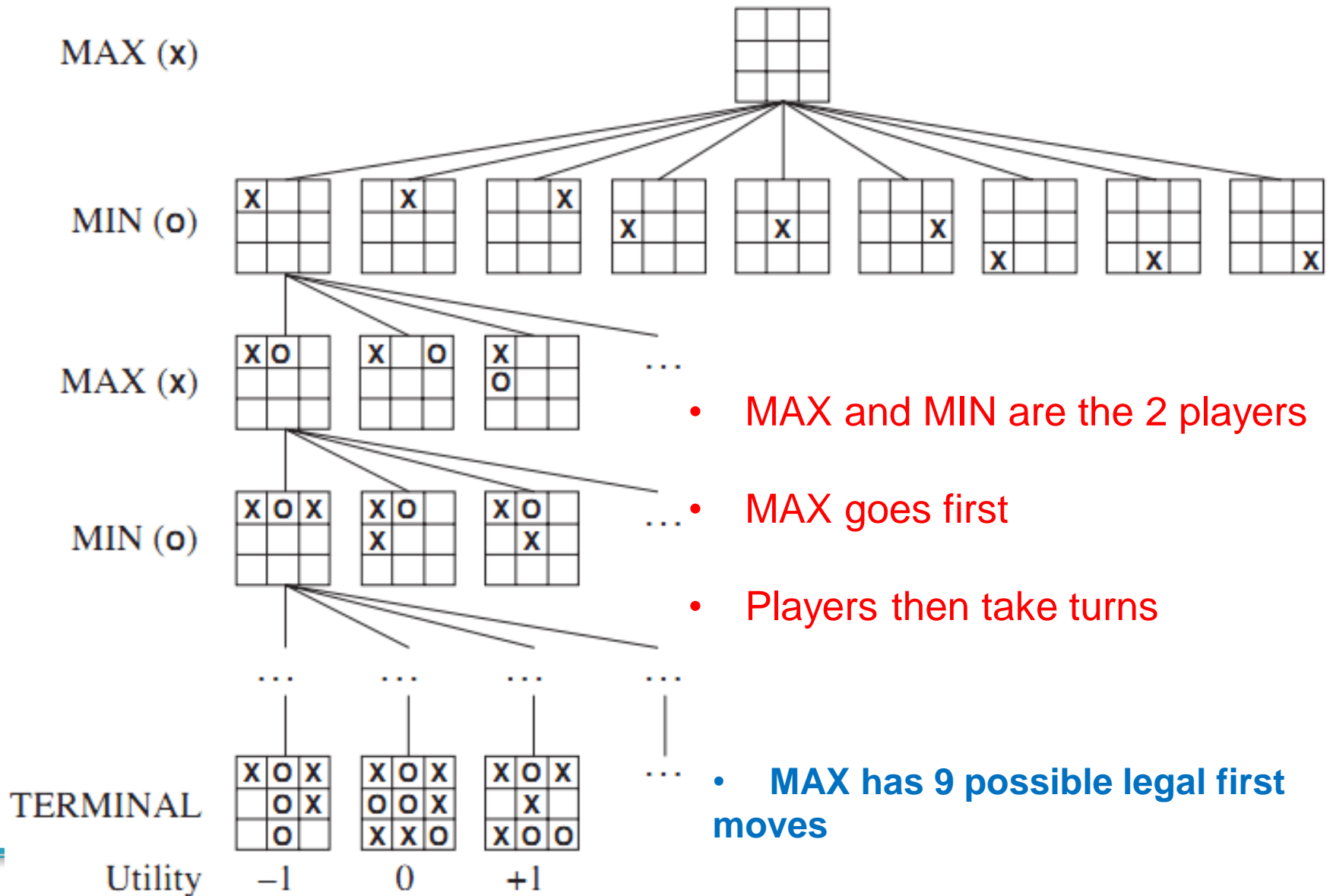
# Game setup

- ▶ Two players: *MAX* and *MIN*
- ▶ *MAX* moves first and they take turns until the game is over.
  - Winner gets award,
  - Loser gets penalty.
- ▶ MAX uses search tree to determine next move.

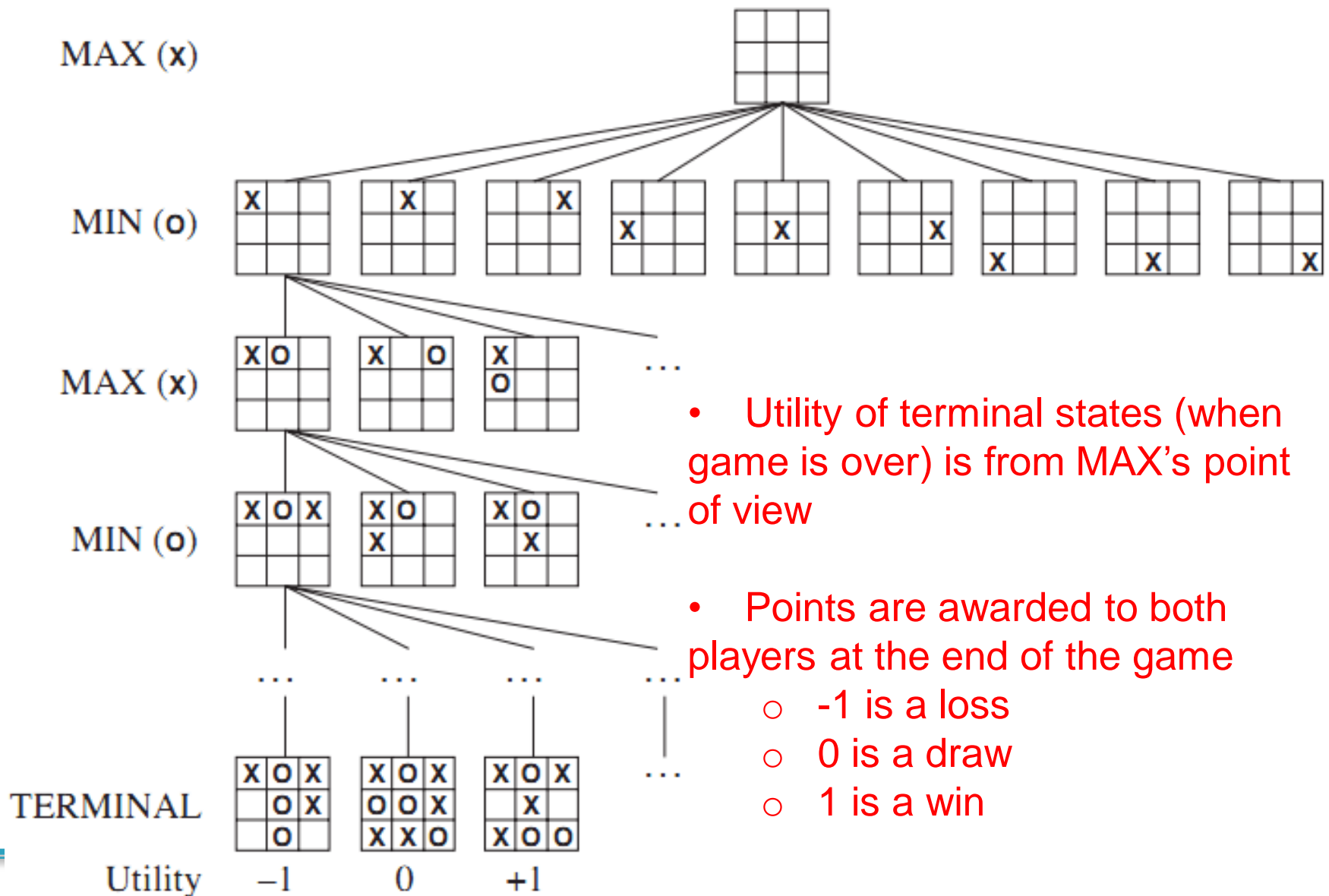
# Game Search

- ▶ Problem Formulation:
  - **States**: board configuration of chess
  - **Successor function**: legal moves a player can make.
  - **Goal test**: determines when the game is over.
  - **initial state**: start board configuration
  - **Utility function**: measures the outcome of the game and its desirability
- ▶ Search objective:
  - Find **the sequence of player's decisions** (moves) maximizing its utility
  - Consider **the opponent's moves** and **their utility**

# Game Tree



# Game Tree





# Game Playing as Search: Complexity

- ▶ Assume the opponent's moves *can* be predicted given the computer's moves.
- ▶ How **complex** would search be in this case?
  - Worst case:  $O(b^d)$ , **b** branching factor, **d** depth
  - **Tic-Tac-Toe**: ~5 legal moves, max of 9 moves
    - $5^9 = 1,953,125$  states
  - **Chess**: ~35 legal moves, ~100 moves per game
    - $35^{100} \sim 10^{154}$  states (but “only”  $\sim 10^{40}$  legal states)

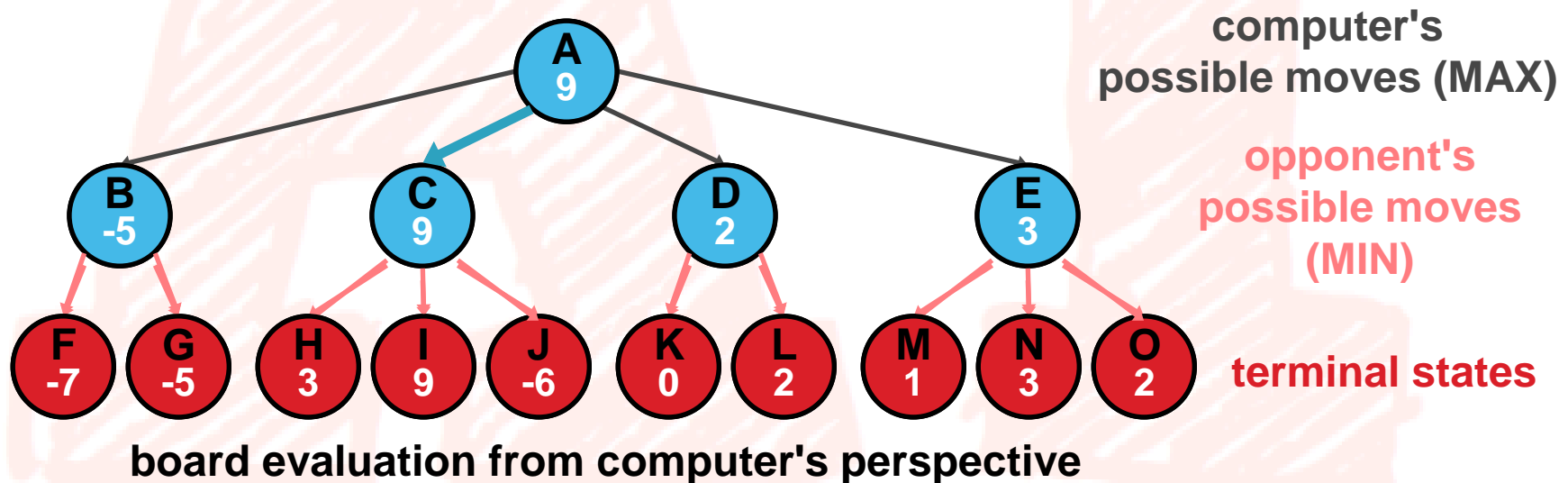
*Common games produce enormous search trees!!*

# Greedy Search Game Playing

- ▶ A *utility* function maps each terminal state of the board to a numeric value corresponding to the value of that state to the computer.
  - **positive for winning**, large positive value: means better for computer (MAX)
  - **negative for losing**, large negative value: means better for opponent (MIN)
  - **zero for a draw**
  - typical values (lost to win):
    - **-infinity to +infinity**
    - **-1.0 to +1.0**

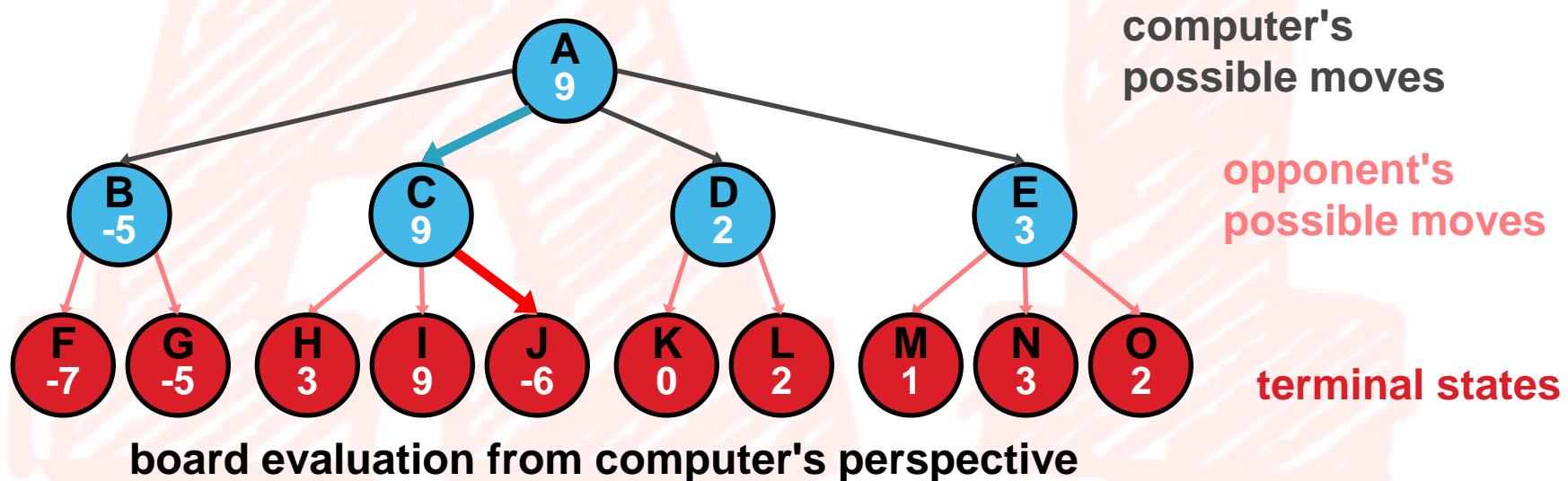
# Greedy Search Game Playing

- ▶ Expand each branch to the **terminal states**
- ▶ Evaluate the utility of each terminal state
- ▶ Choose the move that results in the board configuration with **the maximum value**



# Greedy Search Game Playing

- ▶ Assuming a reasonable search space, what's the problem with greedy search?
  - It ignores what the opponent might do!
  - e.g. MAX (computer) chooses C.  
MIN (opponent) chooses J and defeats computer.



# Minimax principle – Optimal strategies

- ▶ Chooses the best move considering both its move and the opponent's best move
- ▶ Assumption: Both players play optimally!!
  - **MAX** (computer) **maximizing** the utility under the assumption after it moves **MIN** (opponent) will choose the **minimizing** move.
- ▶ Given a game tree, the optimal strategy can be determined by using the minimax value of each node:

**MINIMAX-VALUE( $n$ ) =**

**UTILITY( $n$ )** If  $n$  is a terminal

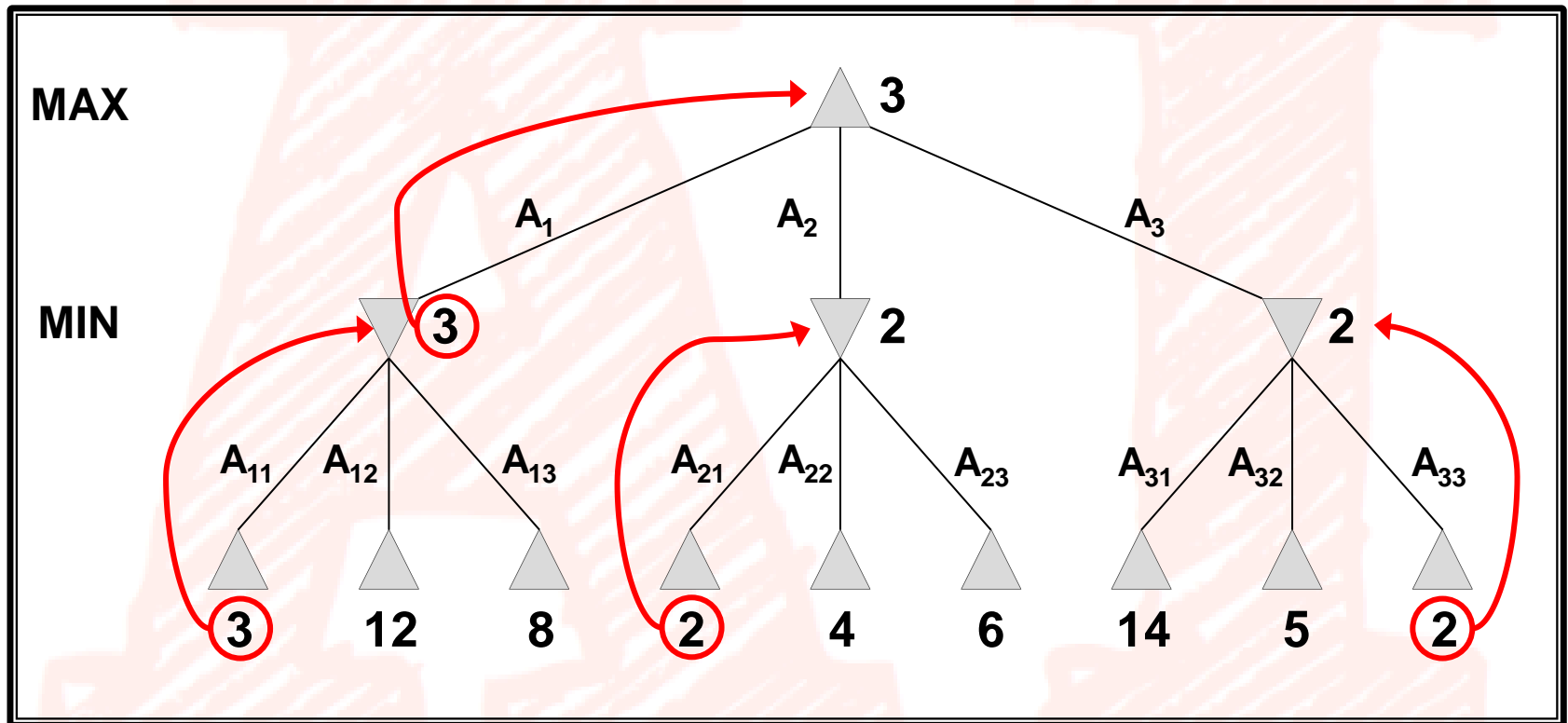
**$\max_{s \in \text{successors}(n)} \text{MINIMAX-VALUE}(s)$ ,** If  $n$  is a max node

**$\min_{s \in \text{successors}(n)} \text{MINIMAX-VALUE}(s)$ ,** If  $n$  is a min node



# Two-Players Game Tree

## The minimax decision



***Minimax maximizes the worst-case outcome for max.***

# What if MIN does not play optimally?

- ▶ Definition of optimal play for **MAX** assumes **MIN** plays optimally: maximizes worst-case outcome for **MAX**.
- ▶ But if **MIN** does not play optimally, **MAX** will do even better. [Can be proved.]

# Minimax: Direct Algorithm

For each move by the MAX (computer):

- ▶ Perform **depth-first search to a terminal state**
- ▶ **Evaluate each terminal state**
- ▶ **Propagate upwards the minimax values**
  - if opponent's move minimum value of children backed up
  - if computer's move maximum value of children backed up
- ▶ **choose move with the maximum of minimax values of children**
- ▶ **Note:**
  - minimax values gradually propagate upwards as DFS proceeds: i.e., minimax values propagate up in “left-to-right” fashion
  - minimax values for sub-tree backed up “as we go”, so only  **$O(bd)$**  nodes need to be kept in memory at any time

# Minimax Algorithm

**function** MINIMAX-DECISION(*state*) *returns an action*

**inputs:** *state*, current state in game

$v \leftarrow \text{MAX-VALUE}(\textit{state})$

**return** the *action* in  $\text{SUCCESSORS}(\textit{state})$  with value  $v$

**function** MAX-VALUE(*state*) *returns a utility value*

**if**  $\text{TERMINAL-TEST}(\textit{state})$  **then return**  $\text{UTILITY}(\textit{state})$

$v \leftarrow -\infty$

**for each**  $s$  **in**  $\text{SUCCESSORS}(\textit{state})$  **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$

**return**  $v$

**function** MIN-VALUE(*state*) *returns a utility value*

**if**  $\text{TERMINAL-TEST}(\textit{state})$  **then return**  $\text{UTILITY}(\textit{state})$

$v \leftarrow +\infty$

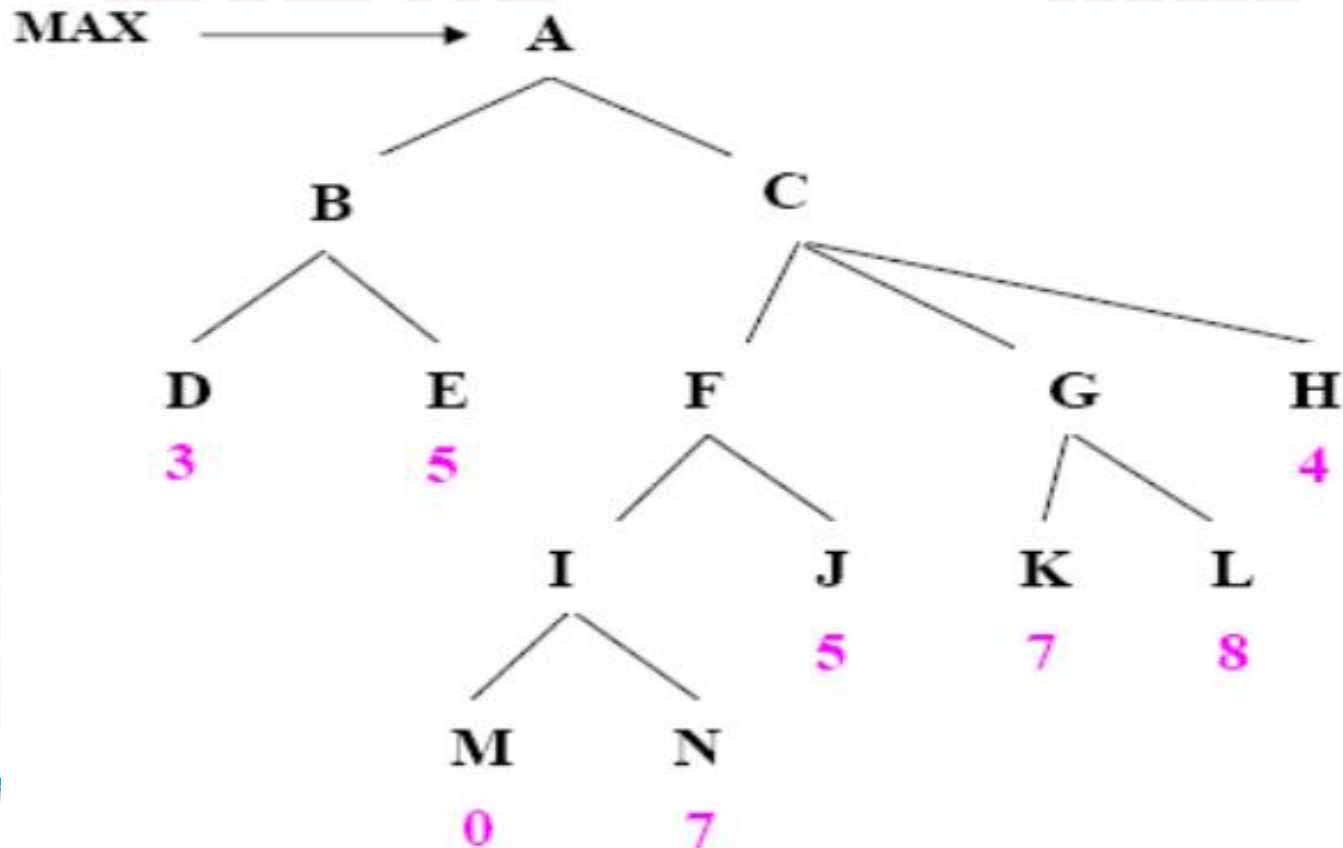
**for each**  $s$  **in**  $\text{SUCCESSORS}(\textit{state})$  **do**

$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$

**return**  $v$

# Exercise 1

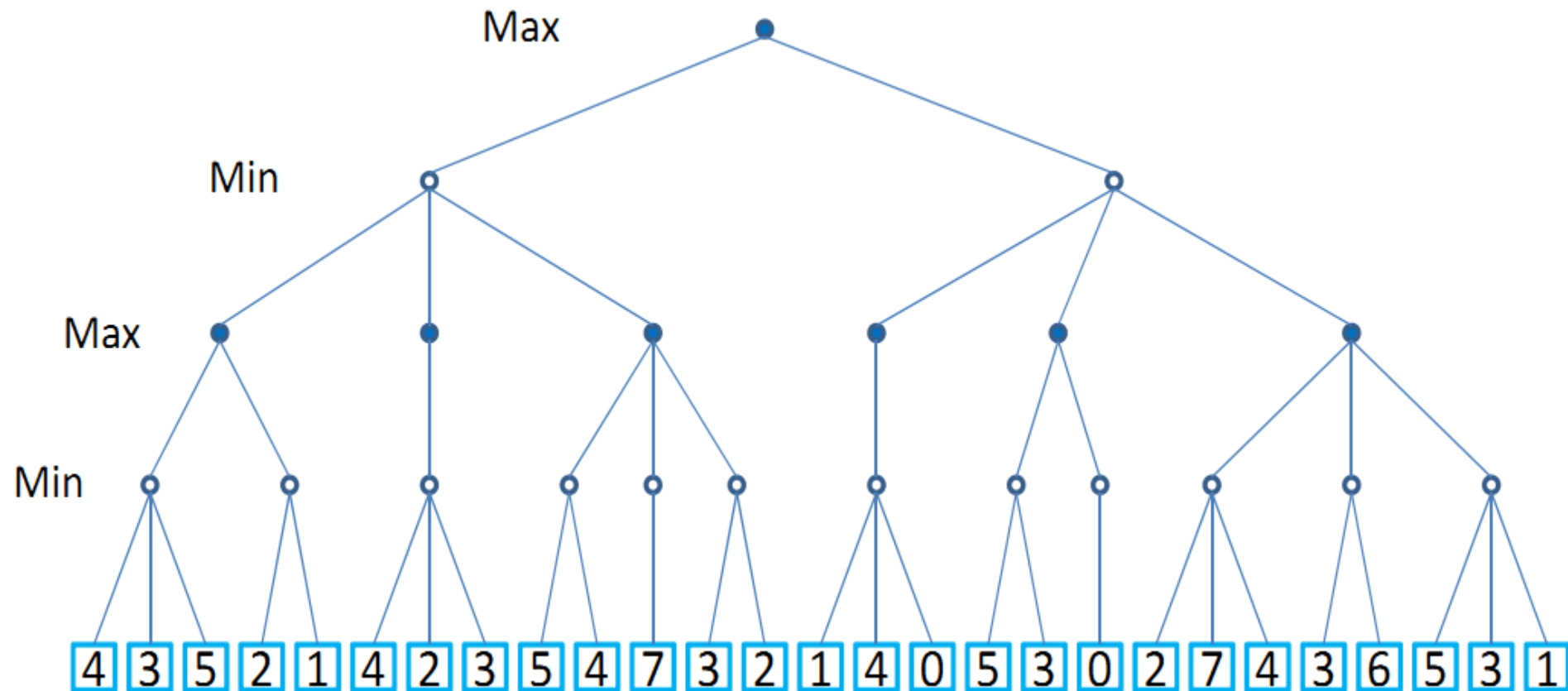
- ▶ Perform the minimax algorithm on the figure below.





# Exercise 2

- Perform the minimax algorithm on the figure below.



# Properties of Minimax

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

- ▶ **Time complexity is a major problem!**  
Player typically only has a finite amount of time to make a move!!
- ▶ For chess,  $b \approx 35$ ,  $m \approx 100$  for "reasonable" games  
→ exact solution completely infeasible

# Problem of minimax search

- ▶ Number of games states is **exponential to the number of moves**.
- ▶ Some of the branches of the game tree won't be taken if playing against an intelligent opponent
- ▶ Solution: can "**prune**" those branches from the tree ==> **Alpha-beta pruning**
- ▶ While doing DFS of game tree, keep track of:
  - **Alpha** = Highest value found so far at any choice point along the MAX path
    - Lower bound on node's utility
  - **Beta** = Lowest value found so far at any choice point along the MIN path
    - Higher bound on node's utility

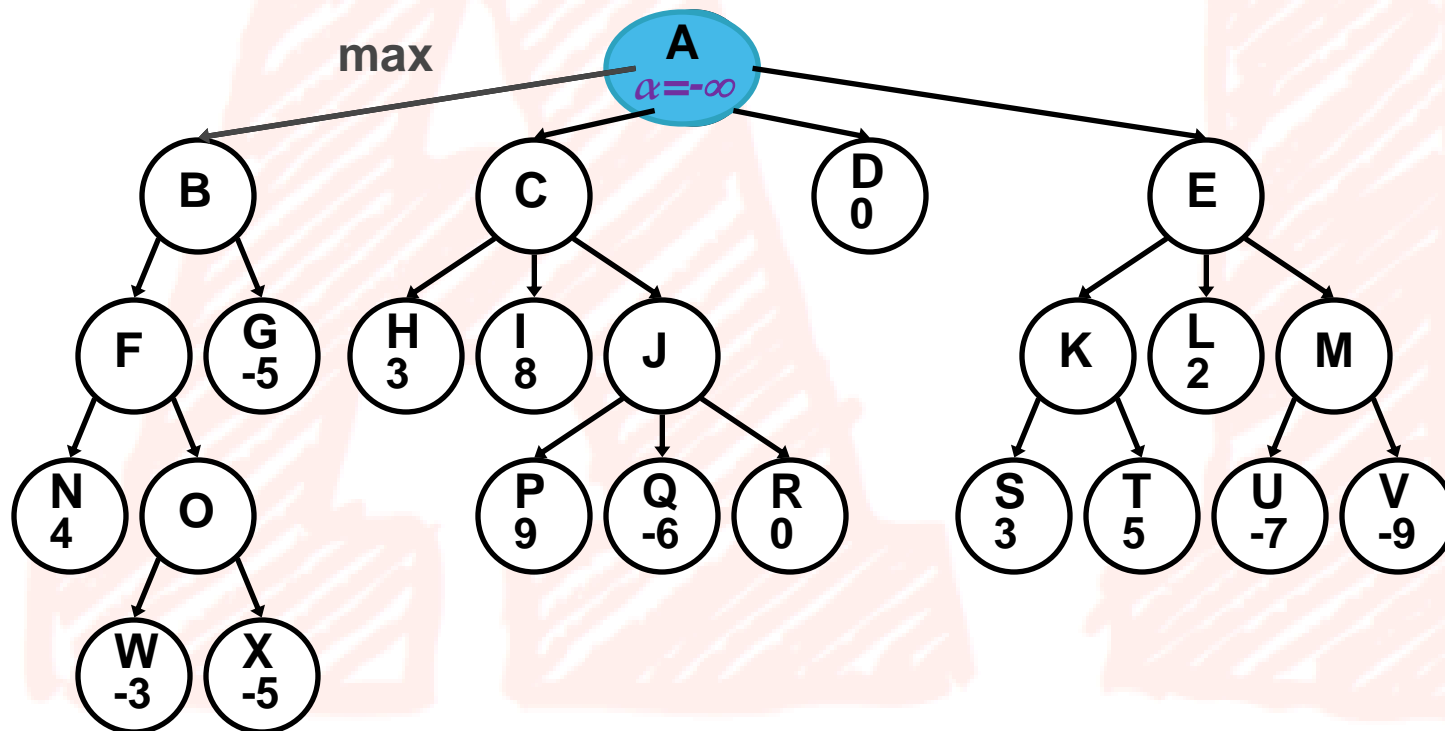
# Alpha-Beta pruning

- ▶ Beta cutoff pruning occurs when maximizing (MAX's turn):
  - If  $\alpha \geq \text{parent's beta}$ , stop expanding
  - Why stop expanding children?
    - Opponent shouldn't allow the MAX to make this move
- ▶ Alpha cutoff pruning occurs when minimizing (MIN's turn):
  - If  $\beta \leq \text{parent's alpha}$ , stop expanding
  - Why stop expanding children?
    - MAX shouldn't take this route

# Alpha-Beta Search Example

$\text{minimax}(A, 0, 4)$       alpha initialized to  $-\infty$

**Expand A?** Yes since there are successors, no cutoff test for root



Call  
Stack

A

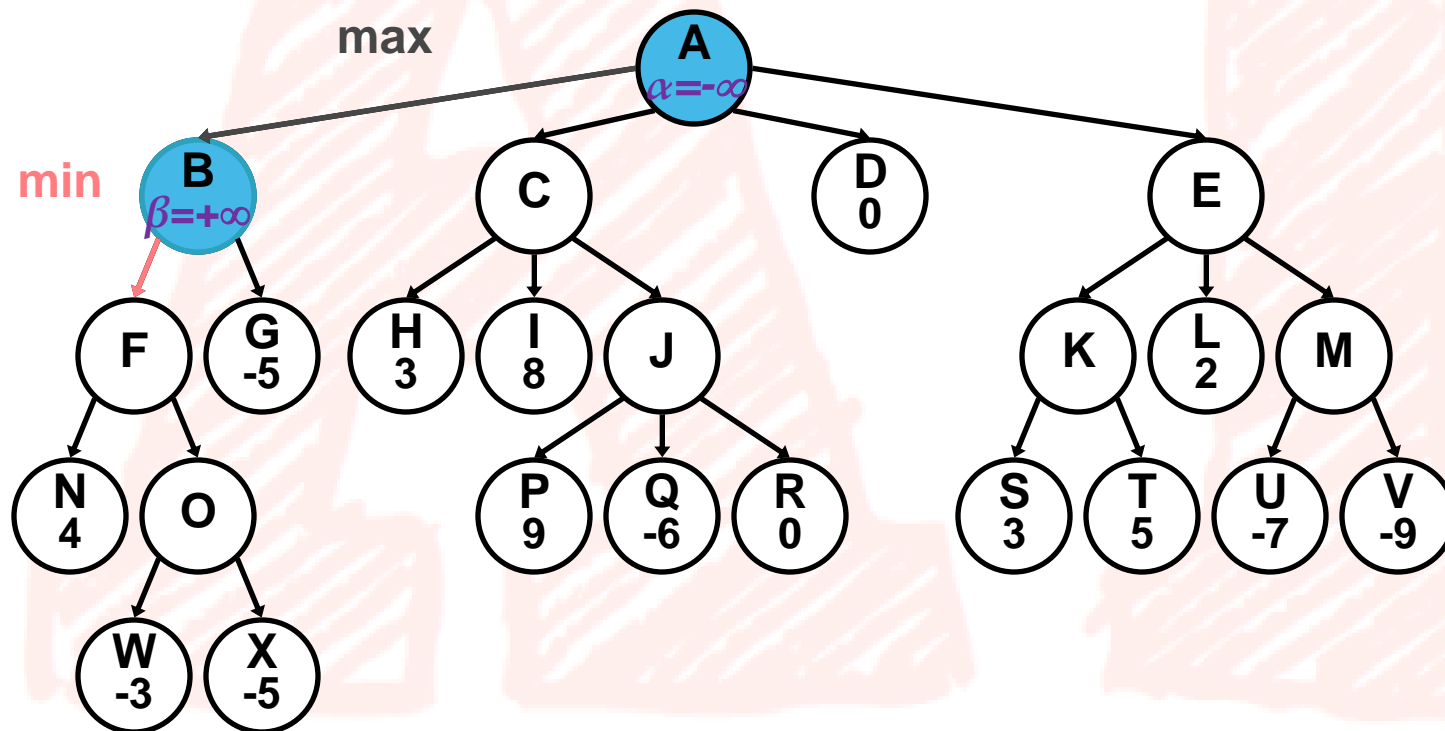


# Alpha-Beta Search Example

**minimax(B,1,4)**

beta initialized to +infinity

**Expand B? Yes** since A's alpha  $\geq$  B's beta is **false**, no alpha cutoff



# Call Stack

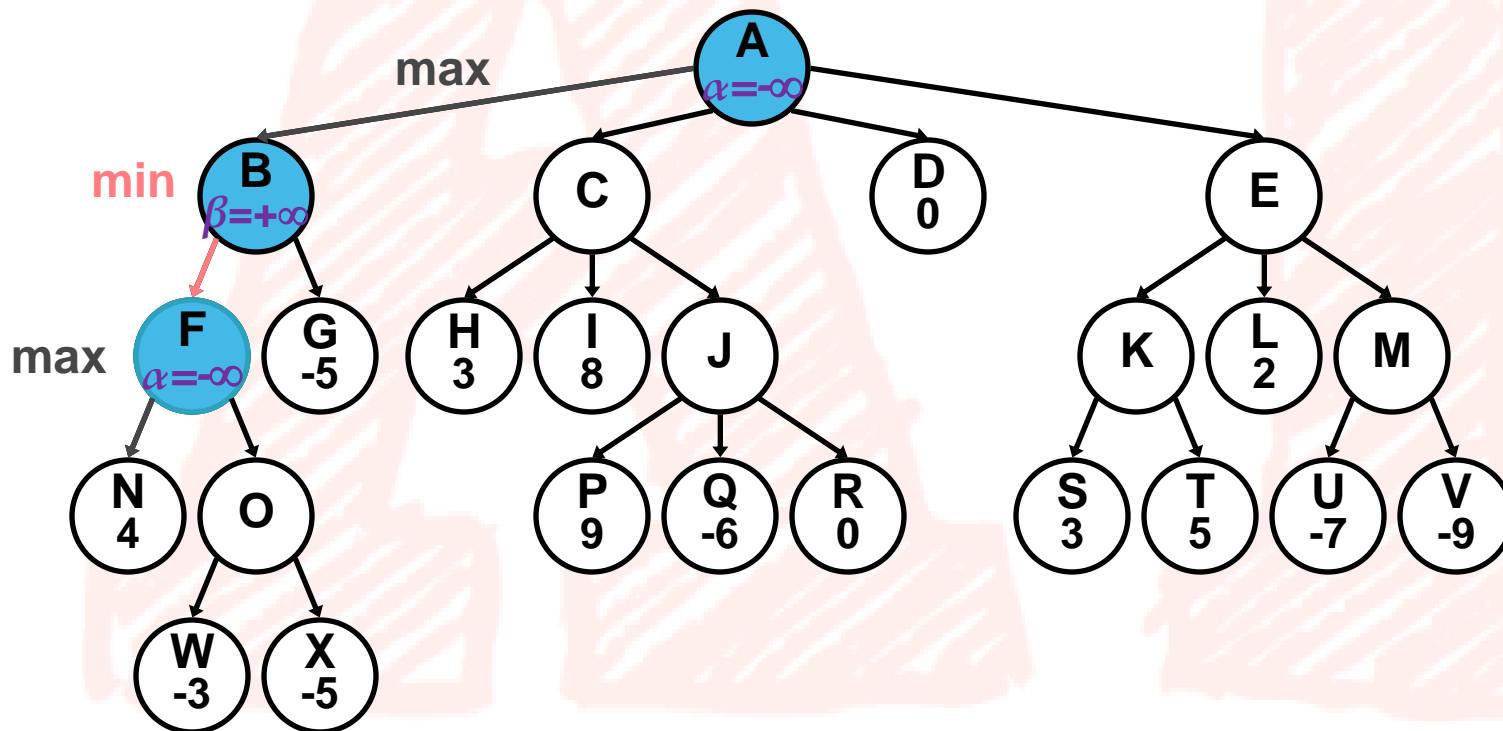
**B**  
**A**

# Alpha-Beta Search Example

$\text{minimax}(F, 2, 4)$

alpha initialized to -infinity

Expand F? Yes since F's alpha  $\geq$  B's beta is **false**, no beta cutoff



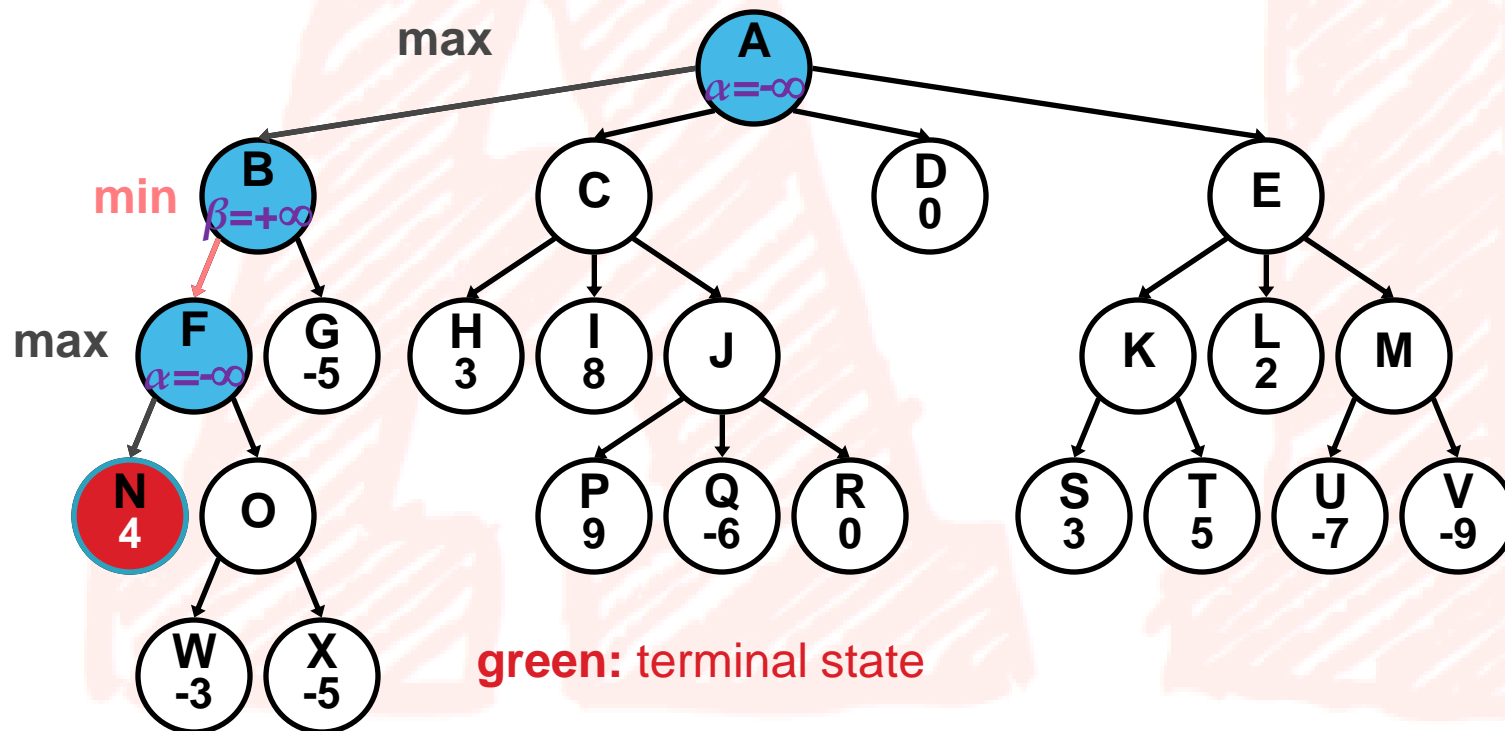
Call  
Stack

F  
B  
A

# Alpha-Beta Search Example

$\text{minimax}(N, 3, 4)$

evaluate and return SBE value



Call  
Stack

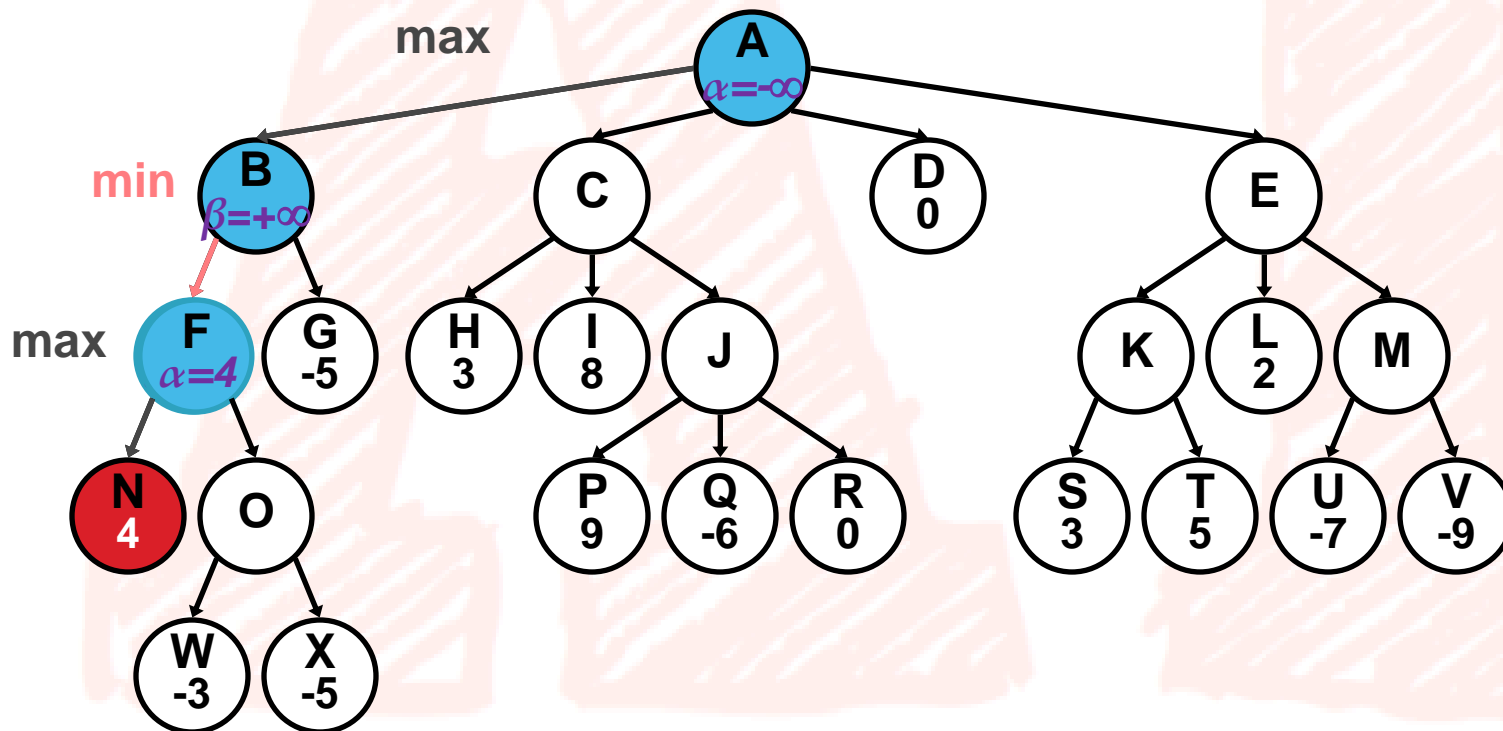
N  
F  
B  
A

# Alpha-Beta Search Example

back to  
 $\text{minimax}(F, 2, 4)$

$\alpha = 4$ , since  $4 \geq -\infty$  (maximizing)

Keep expanding F? Yes since F's  $\alpha \geq$  B's beta is **false**, no beta cutoff



Call  
Stack

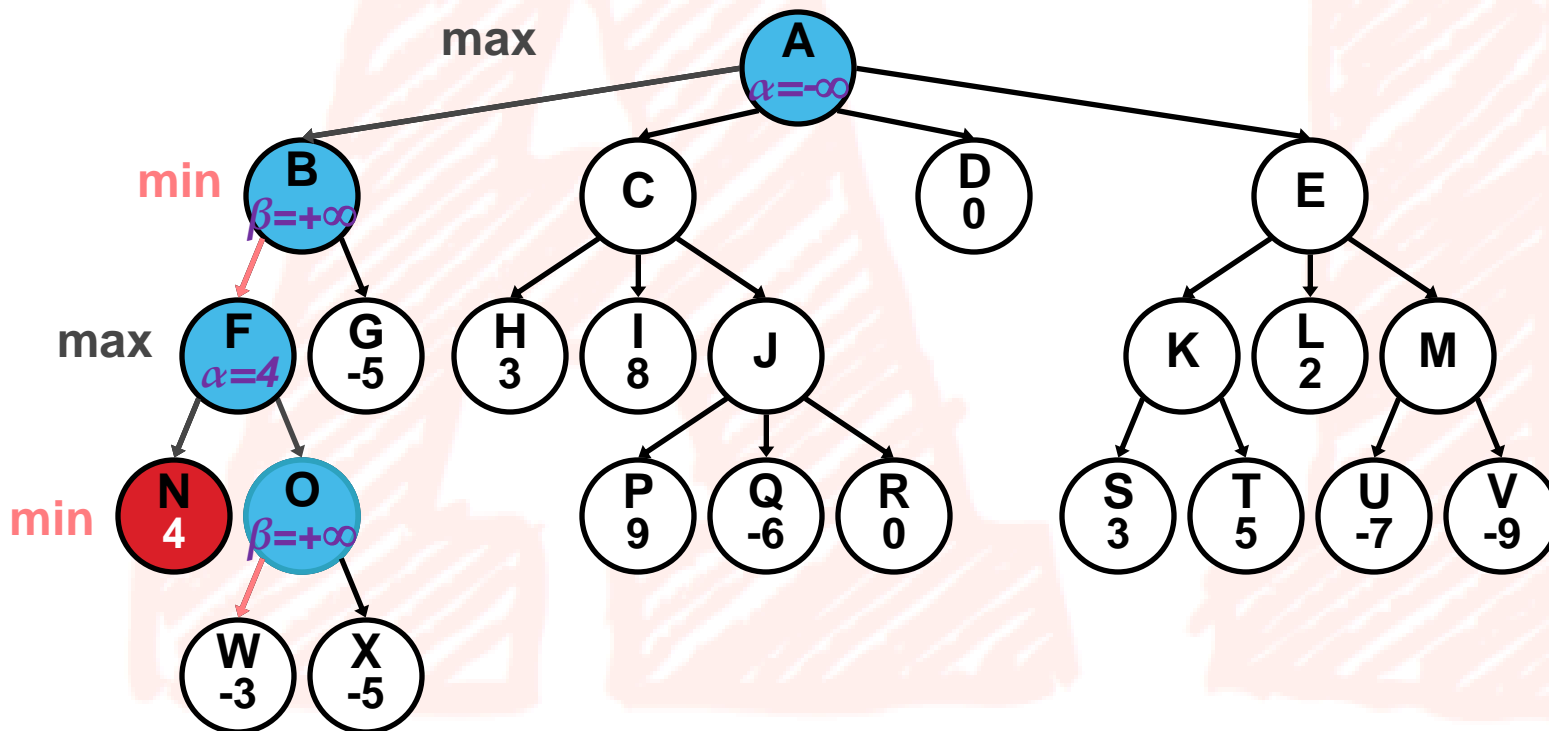
F  
B  
A

# Alpha-Beta Search Example

$\text{minimax}(0, 3, 4)$

beta initialized to +infinity

Expand O? Yes since F's alpha  $\geq$  O's beta is **false**, no alpha cutoff



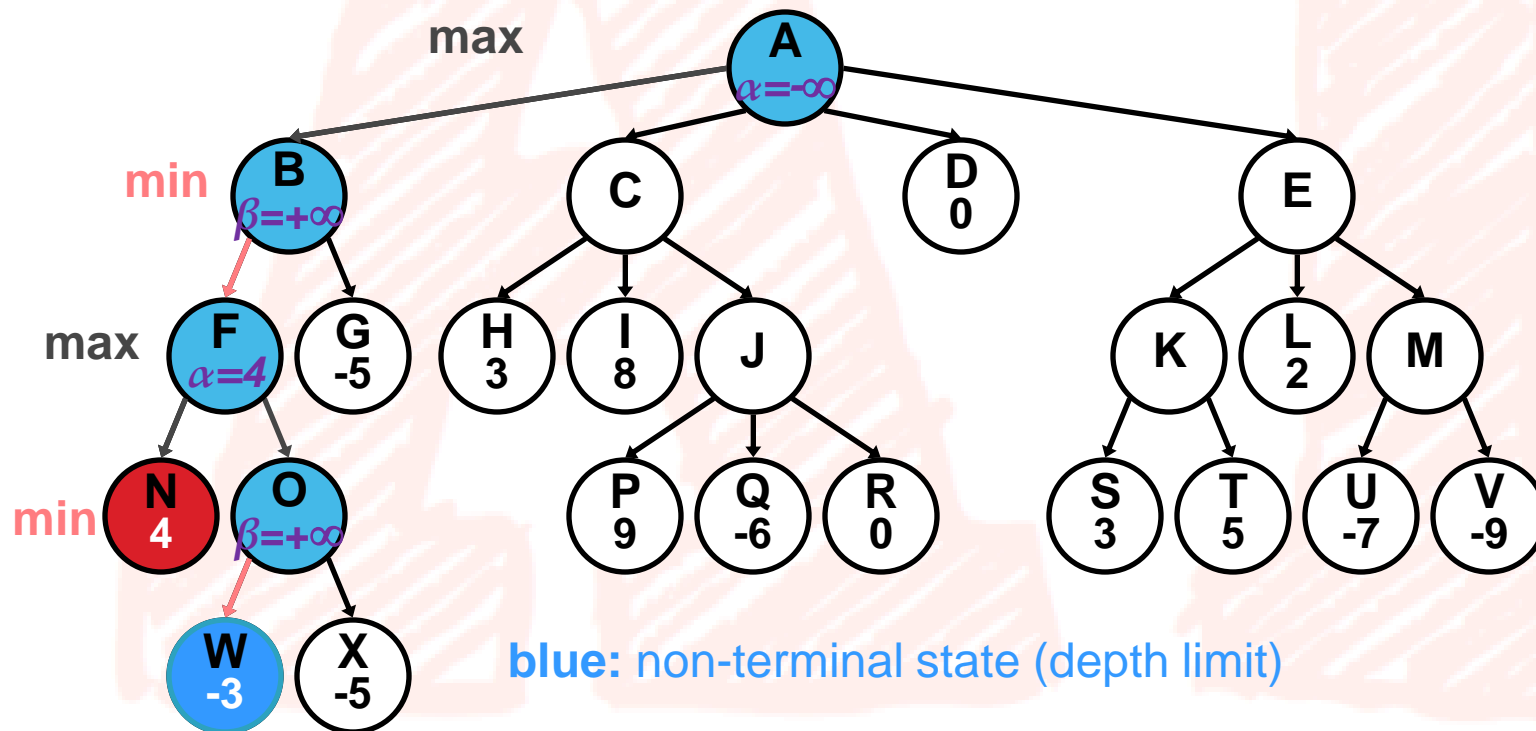
Call  
Stack

O  
F  
B  
A

# Alpha-Beta Search Example

$\text{minimax}(W, 4, 4)$

evaluate and return SBE value



Call  
Stack

W  
O  
F  
B  
A

blue: non-terminal state (depth limit)

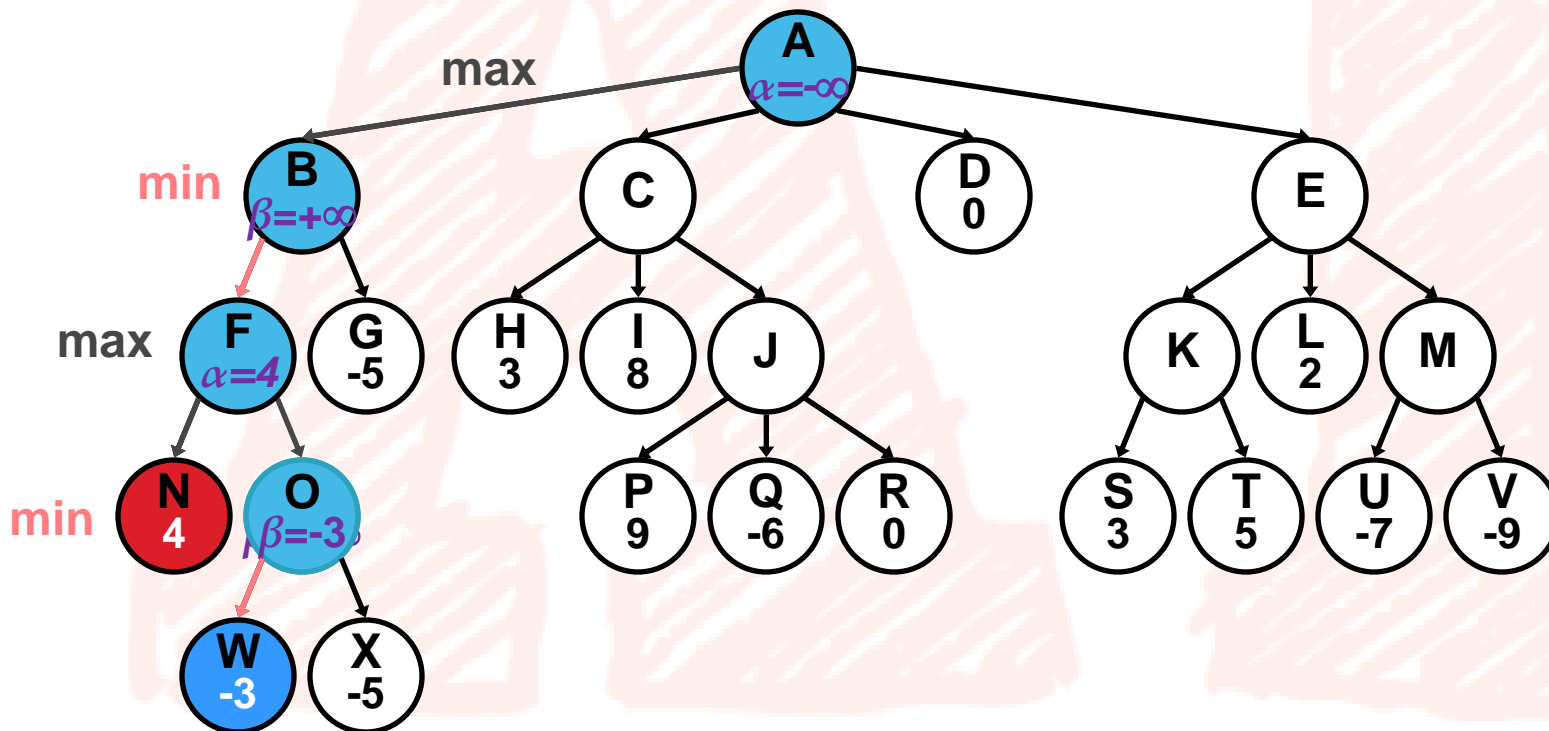
# Alpha-Beta Search Example

back to  
 $\text{minimax}(0, 3, 4)$

$\text{beta} = -3$ , since  $-3 \leq +\text{infinity}$  (minimizing)

Keep expanding O?

**No** since F's  $\alpha \geq$  O's  $\beta$  is **true**: **alpha cutoff**



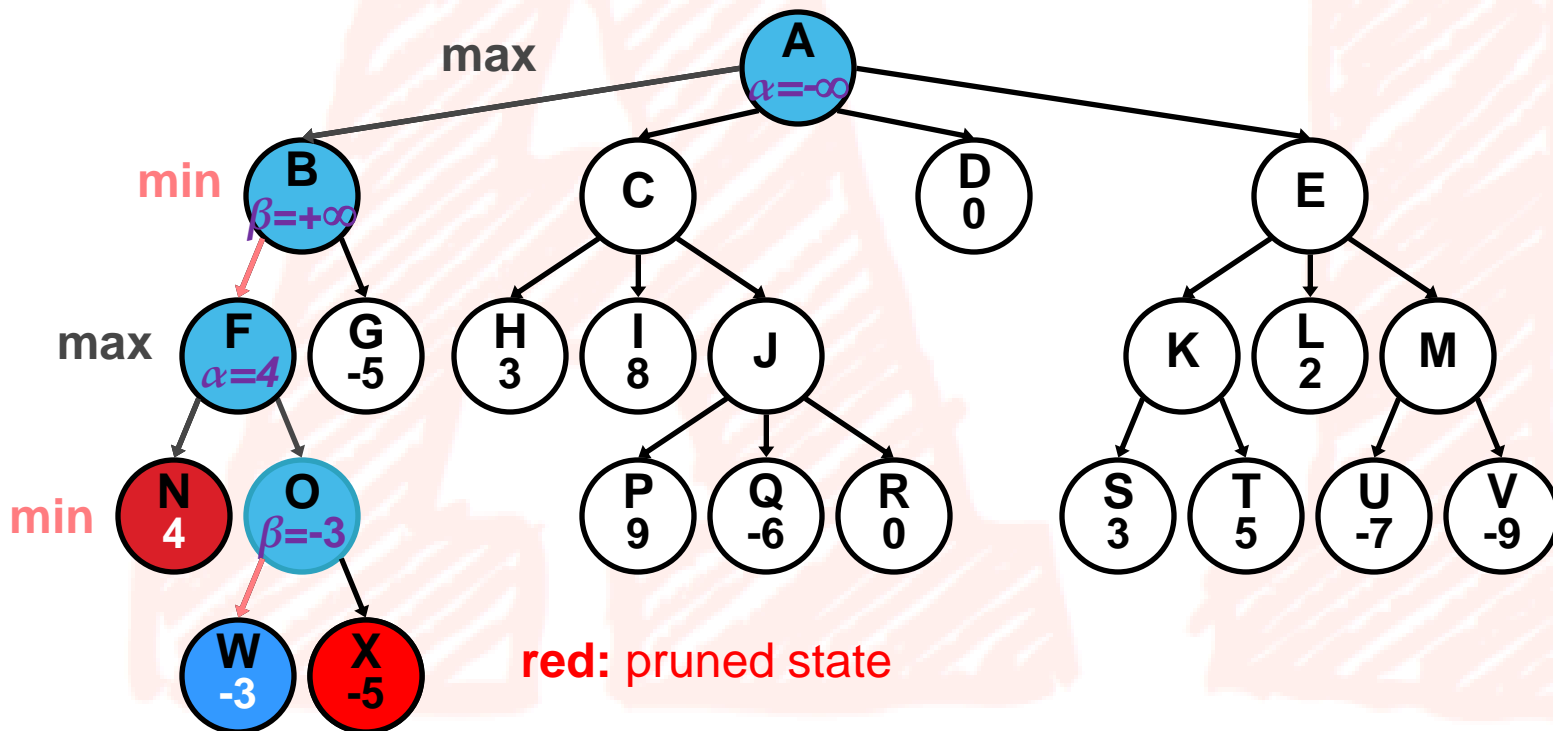
Call  
Stack

O  
F  
B  
A

# Alpha-Beta Search Example

## ► Why?

- Smart opponent will choose W or worse, thus O's upper bound is -3.  
Computer already has better move at N.



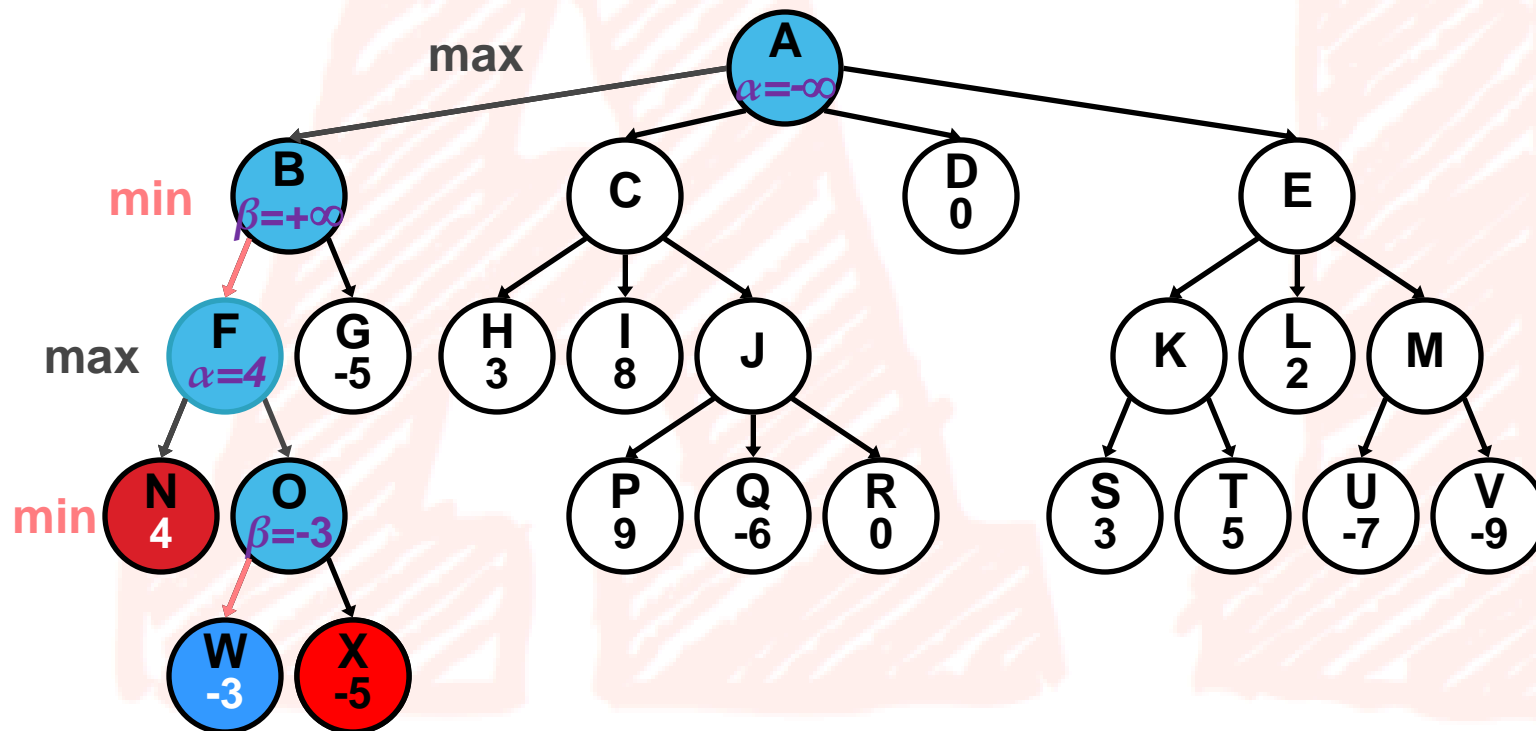


# Alpha-Beta Search Example

back to  
 $\text{minimax}(F, 2, 4)$

alpha doesn't change, since  $-3 < 4$  (maximizing)

Keep expanding F? **No** since no more successors for F



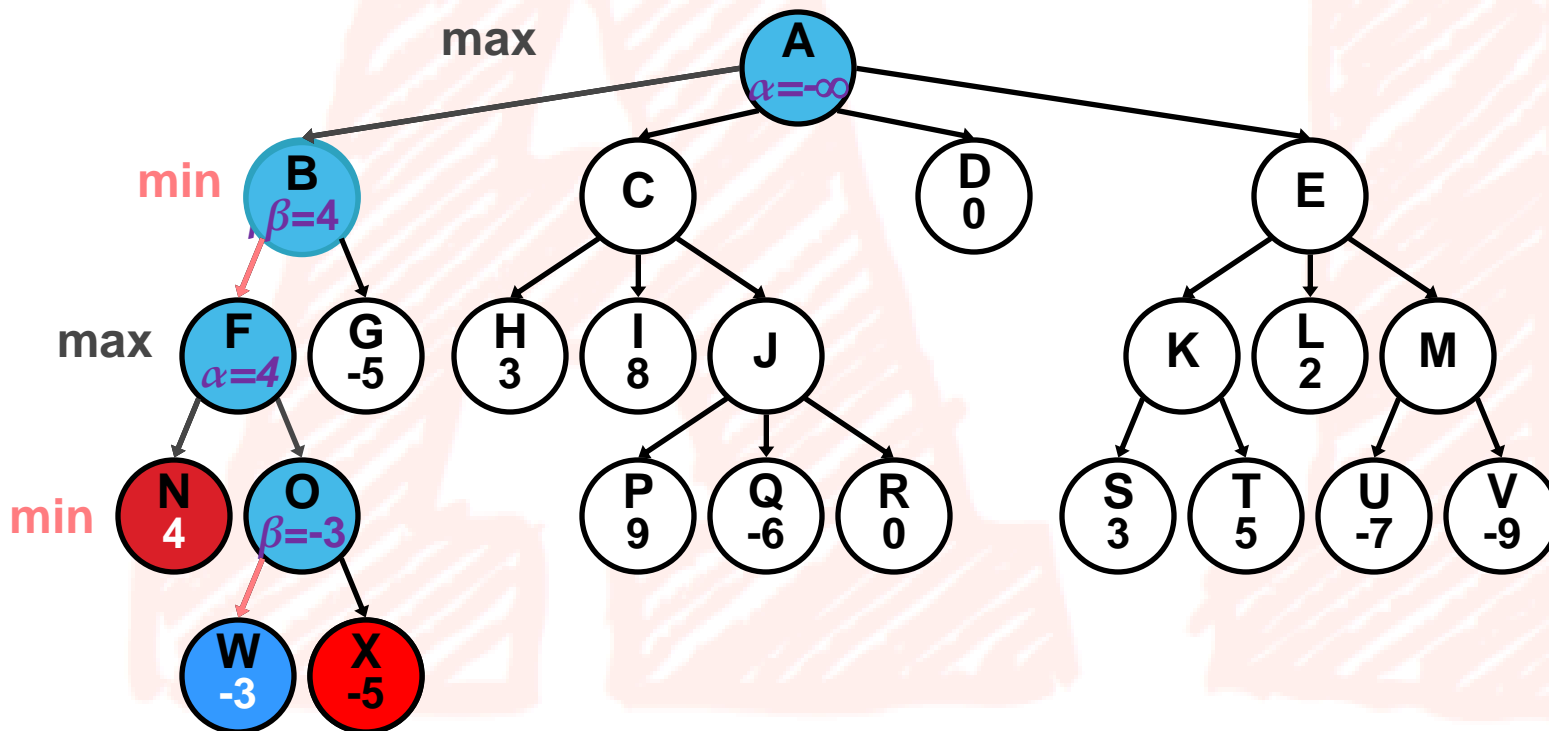
Call  
Stack

F  
B  
A

# Alpha-Beta Search Example

back to  $\text{minimax}(B, 1, 4)$        $\text{beta} = 4$ , since  $4 \leq +\text{infinity}$  (minimizing)

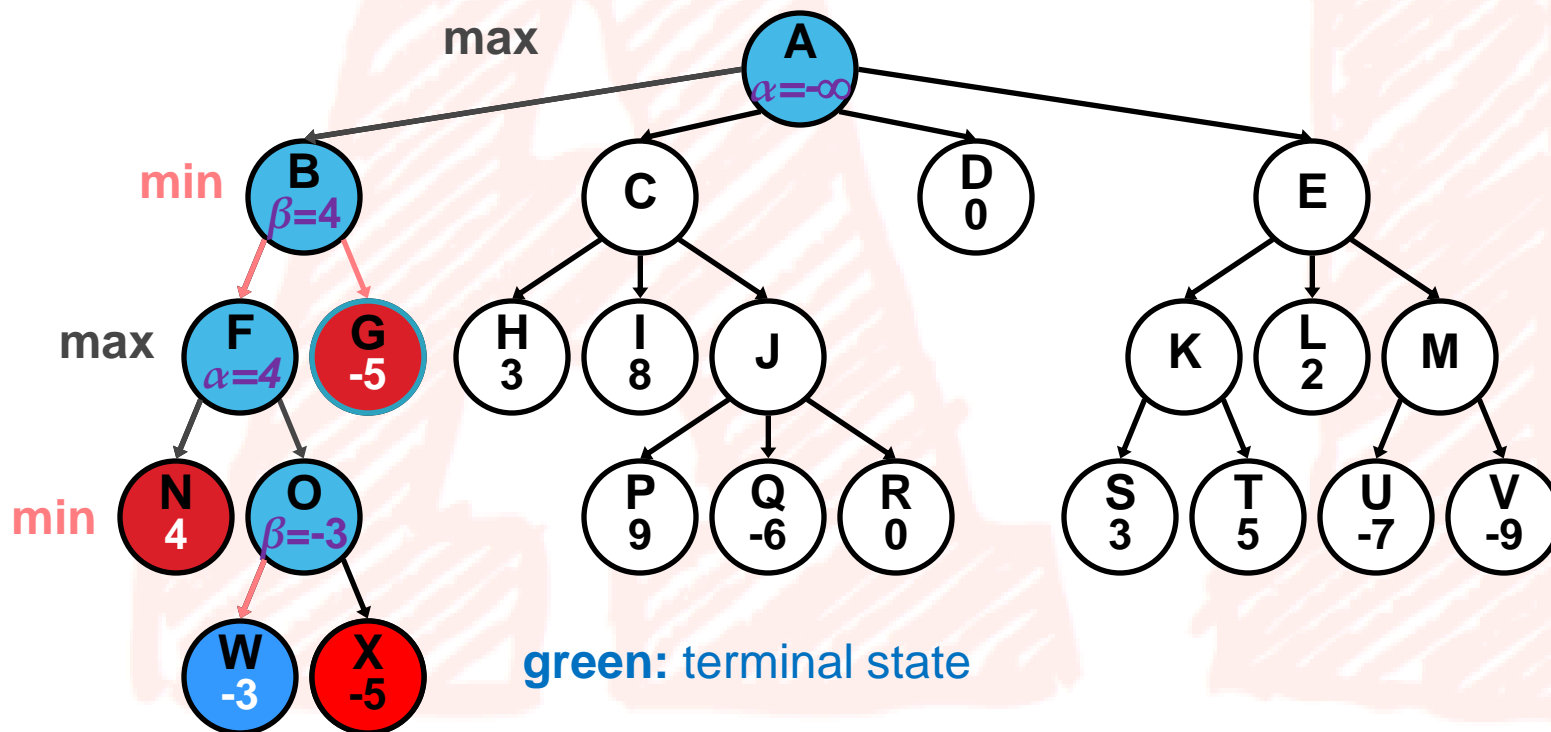
Keep expanding B? Yes since A's  $\alpha \geq B$ 's  $\beta$  is **false**, no alpha cutoff



# Alpha-Beta Search Example

$\text{minimax}(G, 2, 4)$

evaluate and return SBE value



Call  
Stack

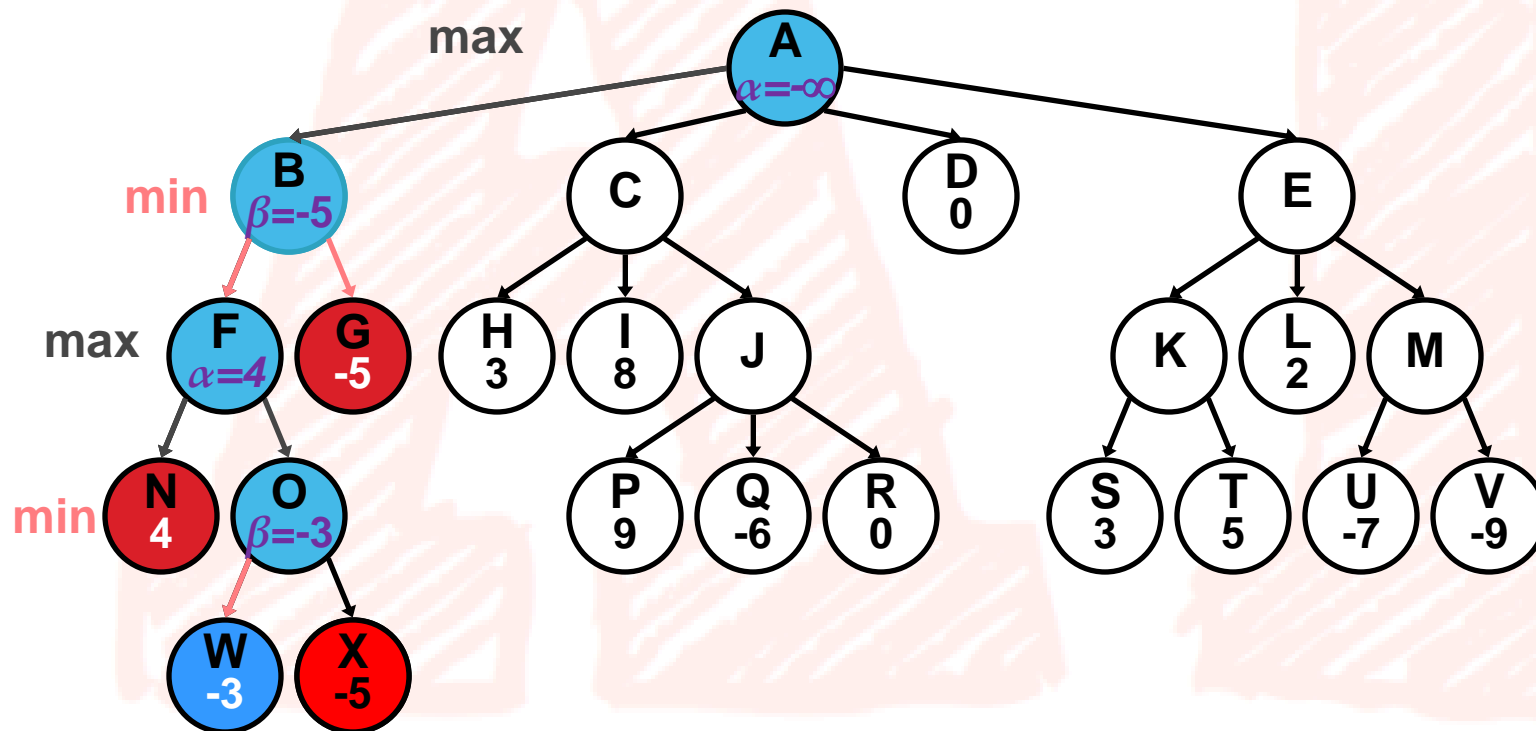
G  
B  
A

# Alpha-Beta Search Example

back to  
 $\text{minimax}(B, 1, 4)$

$\beta = -5$ , since  $-5 \leq 4$  (minimizing)

Keep expanding B? **No** since no more successors for B



Call  
Stack

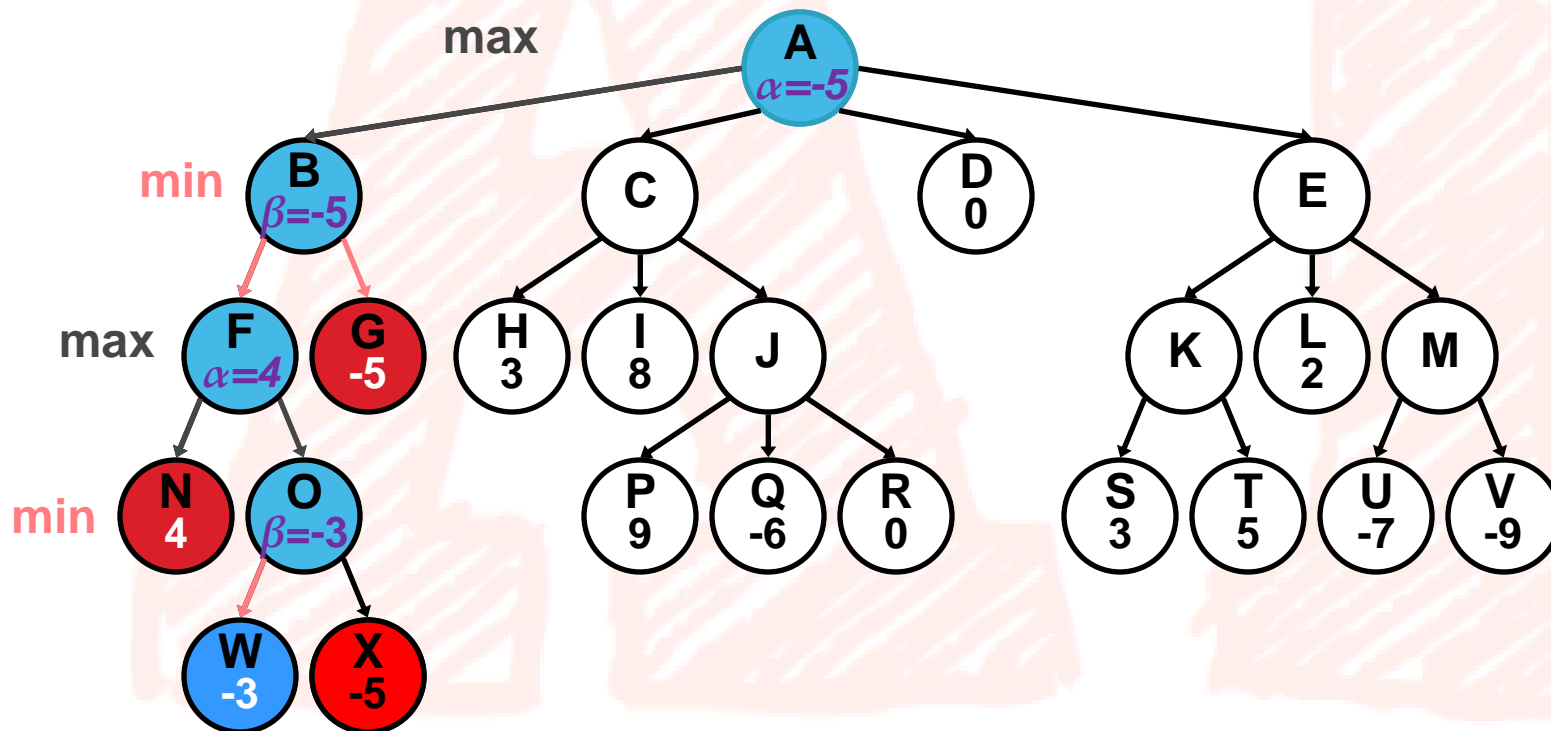
B  
A

# Alpha-Beta Search Example

back to  
 $\text{minimax}(A, 0, 4)$

$\alpha = -5$ , since  $-5 \geq -\text{infinity}$  (maximizing)

Keep expanding A? Yes since there are more successors, no cutoff test



Call  
Stack

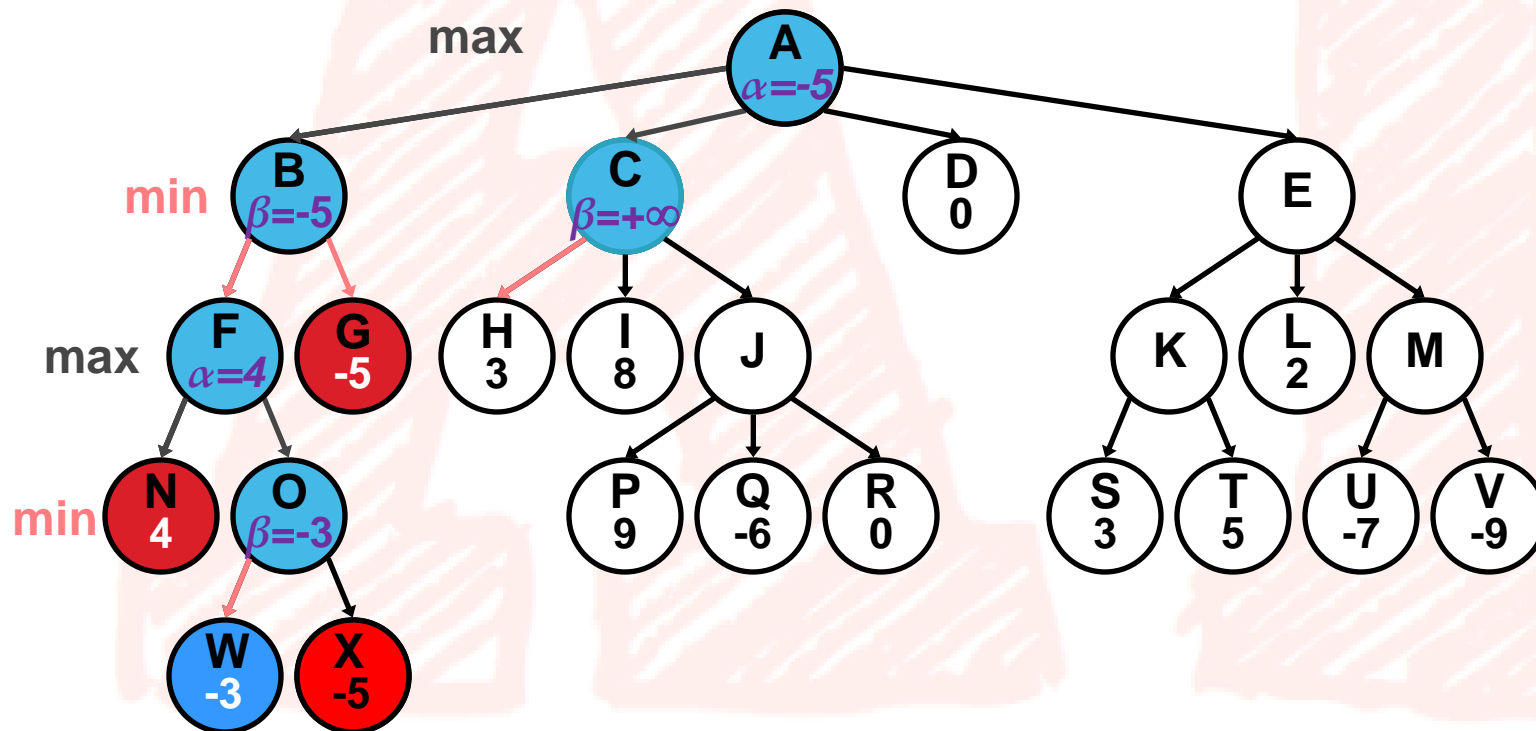
A

# Alpha-Beta Search Example

$\text{minimax}(C, 1, 4)$

beta initialized to +infinity

Expand C? Yes since A's alpha  $\geq$  C's beta is **false**, no alpha cutoff



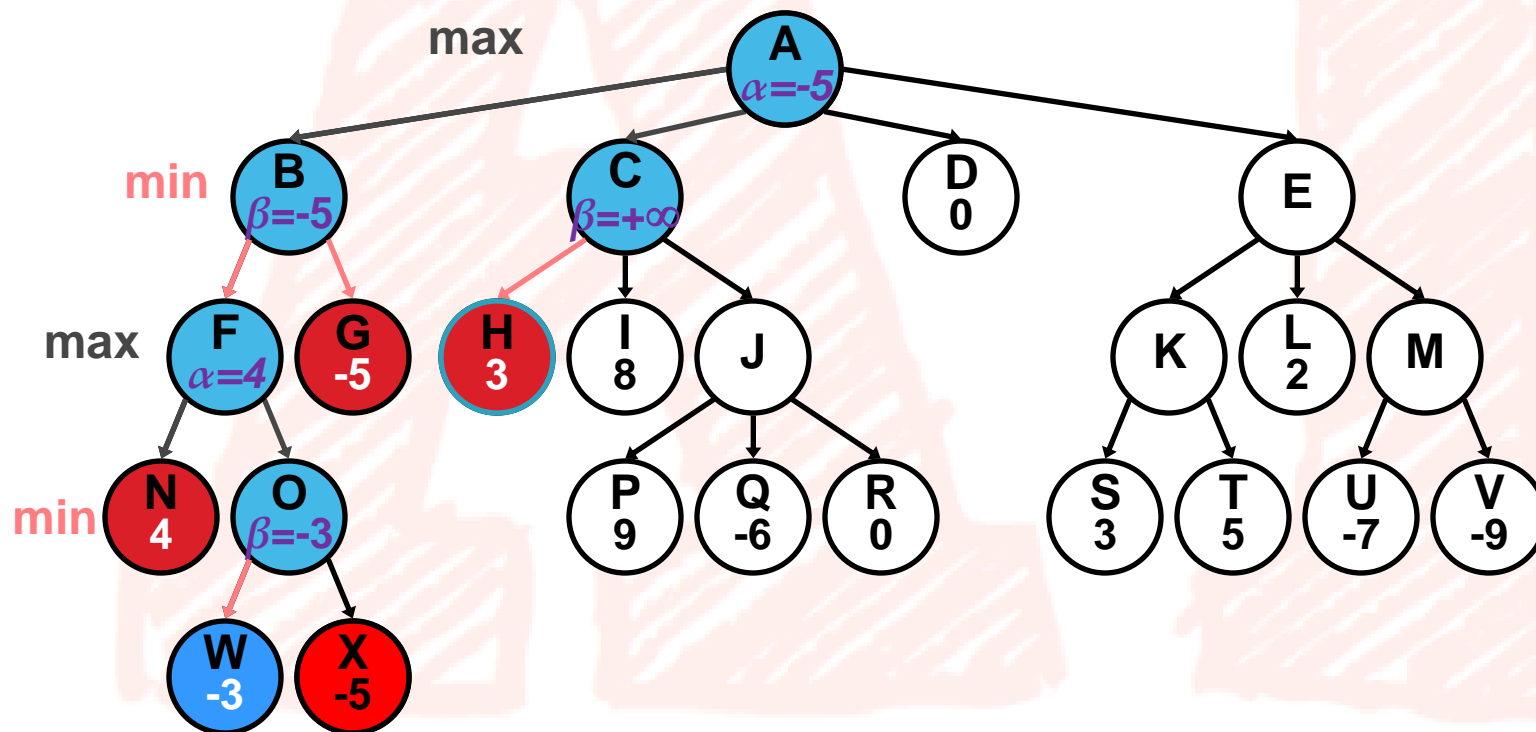
Call  
Stack

C  
A

# Alpha-Beta Search Example

$\text{minimax}(H, 2, 4)$

evaluate and return SBE value



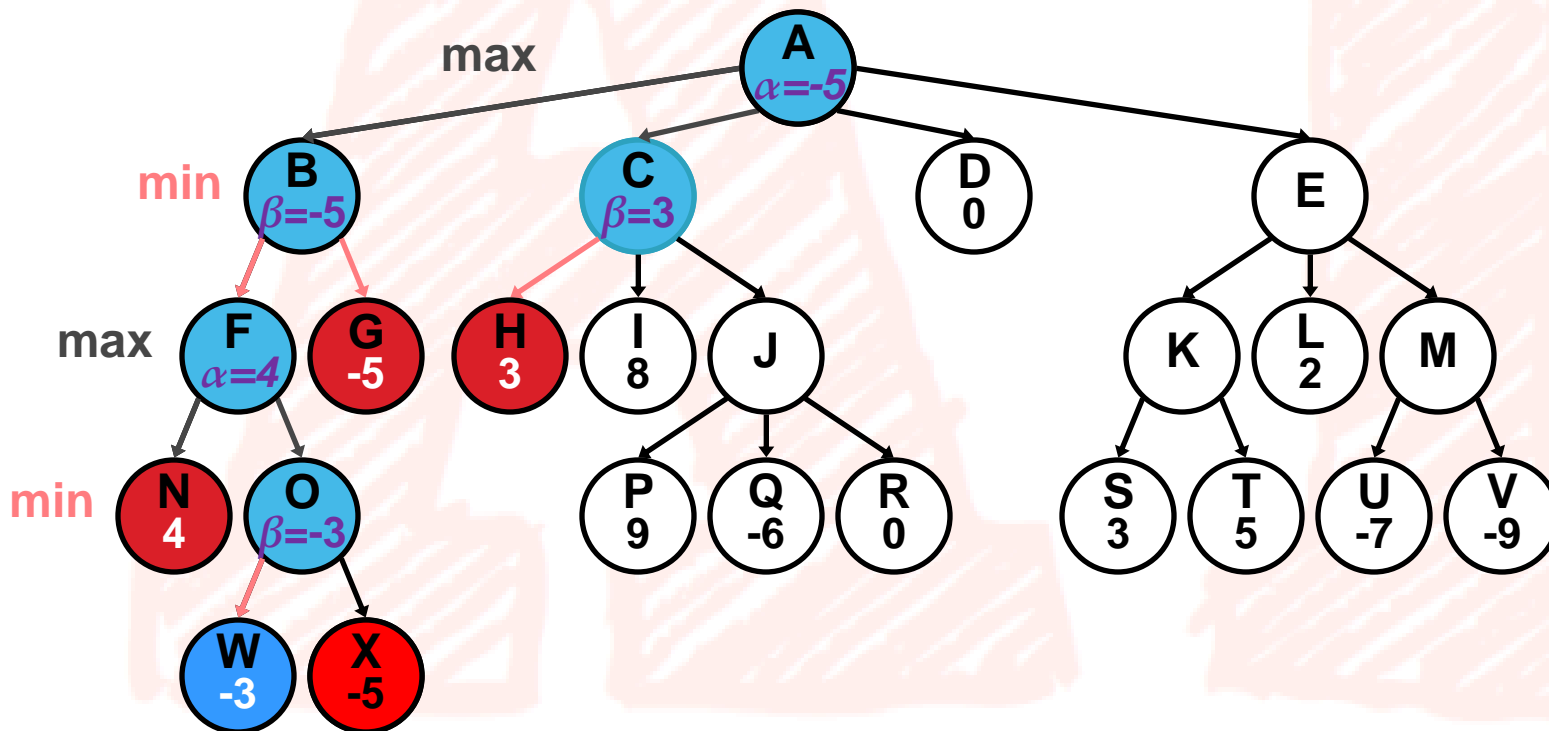
Call  
Stack

H  
C  
A

# Alpha-Beta Search Example

back to  $\text{minimax}(C, 1, 4)$        $\text{beta} = 3$ , since  $3 \leq +\text{infinity}$  (minimizing)

Keep expanding C? Yes since A's  $\alpha \geq C$ 's  $\beta$  is **false**, no alpha cutoff



Call  
Stack

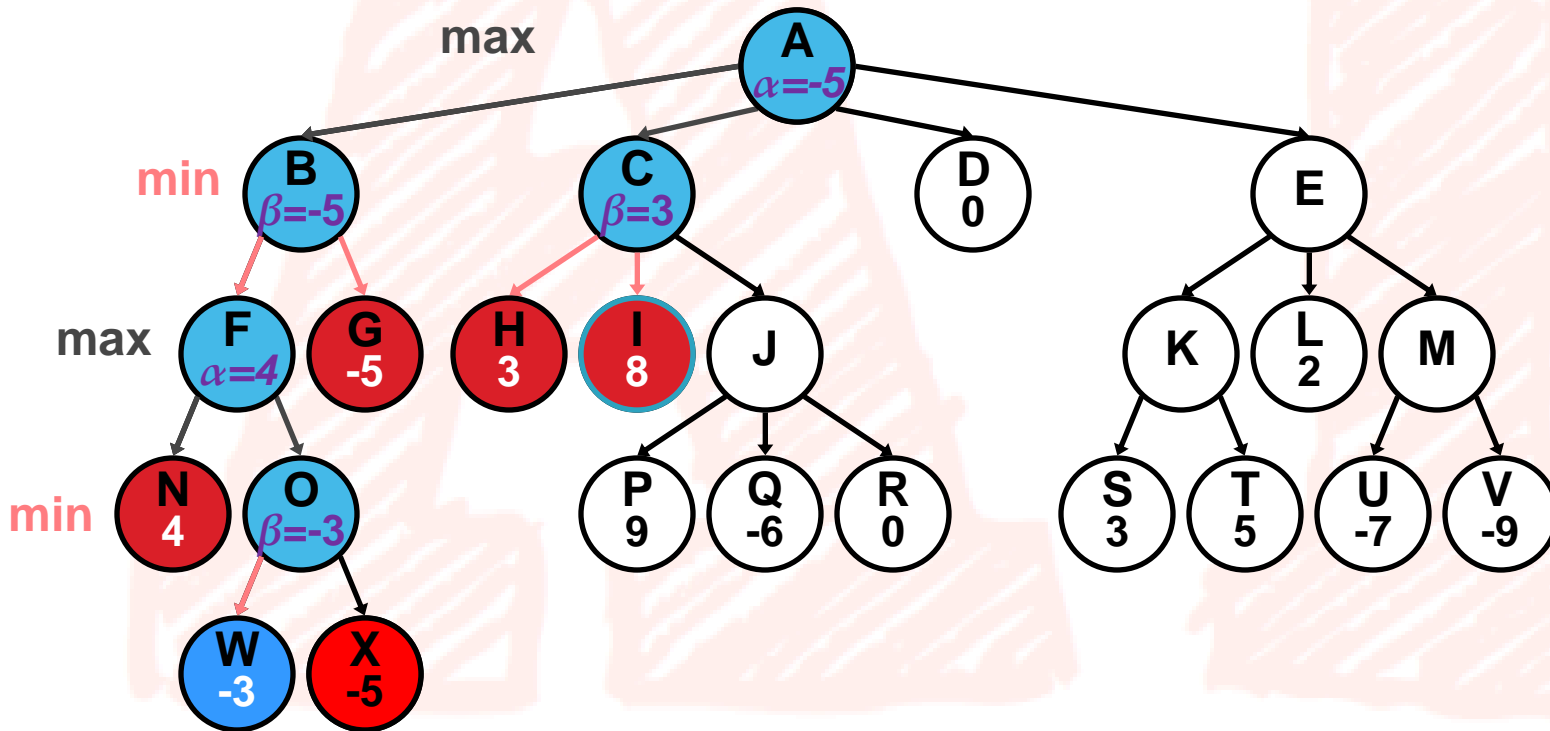
C  
A



# Alpha-Beta Search Example

minimax(I,2,4)

evaluate and return SBE value



# Call Stack

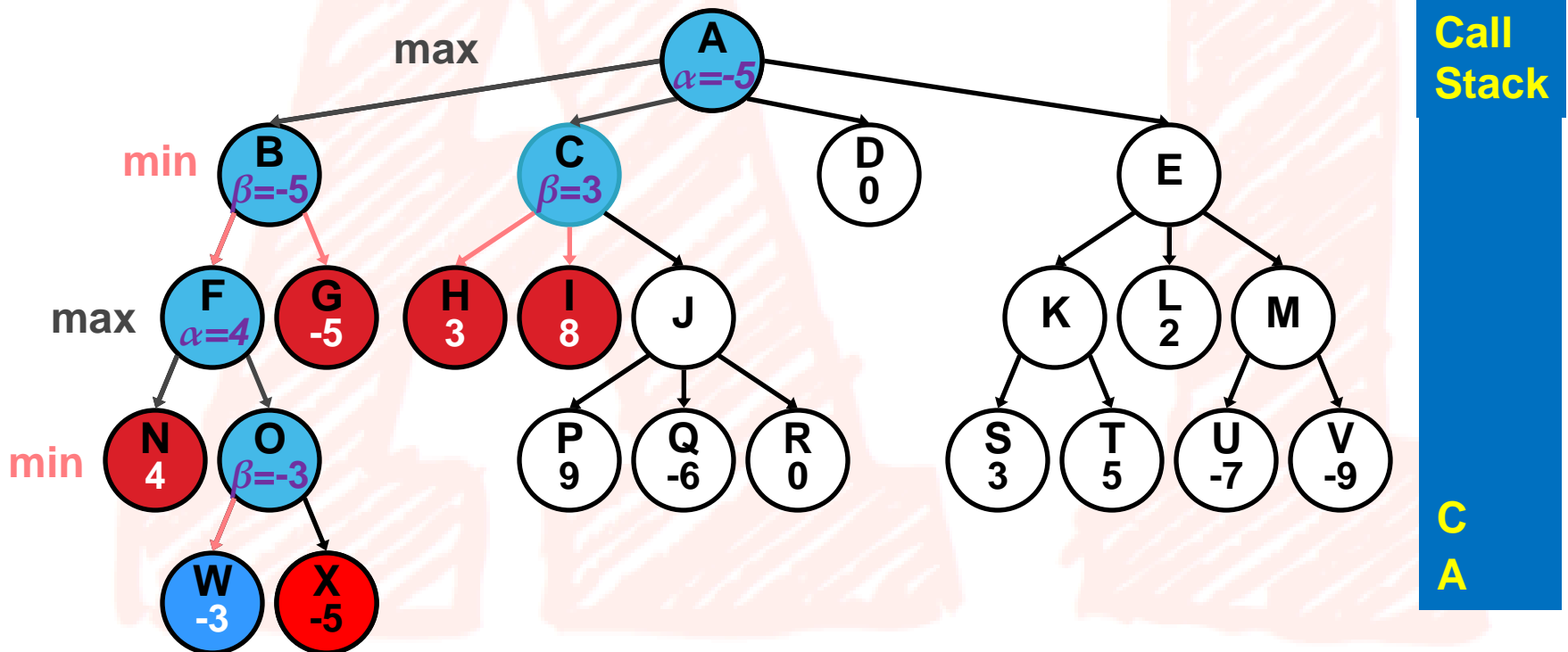
ICA

# Alpha-Beta Search Example

back to  
 $\text{minimax}(C, 1, 4)$

beta doesn't change, since  $8 > 3$  (minimizing)

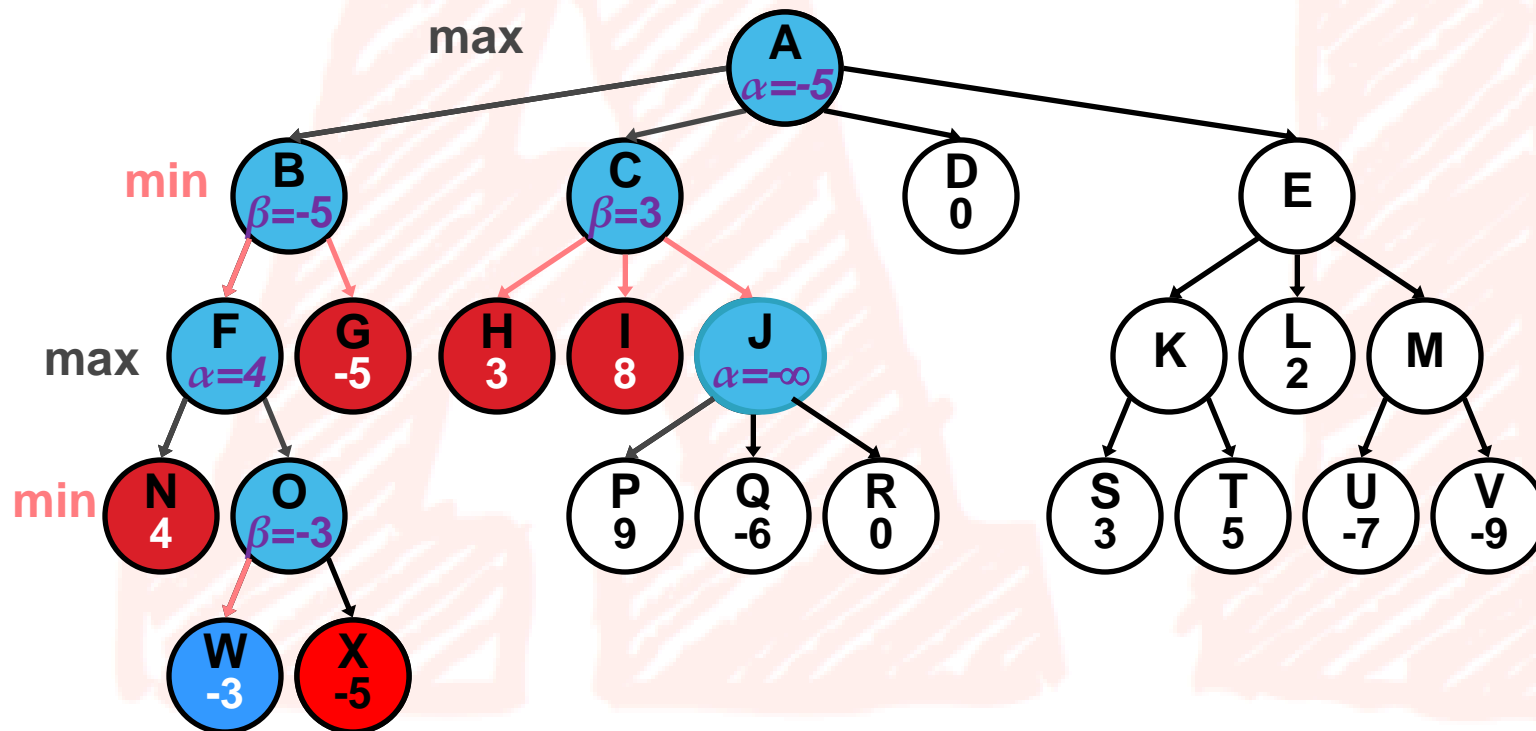
Keep expanding C? Yes since A's alpha  $\geq$  C's beta is **false**, no alpha cutoff



# Alpha-Beta Search Example

$\text{minimax}(J, 2, 4)$       alpha initialized to -infinity

**Expand J?** Yes since J's alpha  $\geq$  C's beta is **false**, no beta cutoff



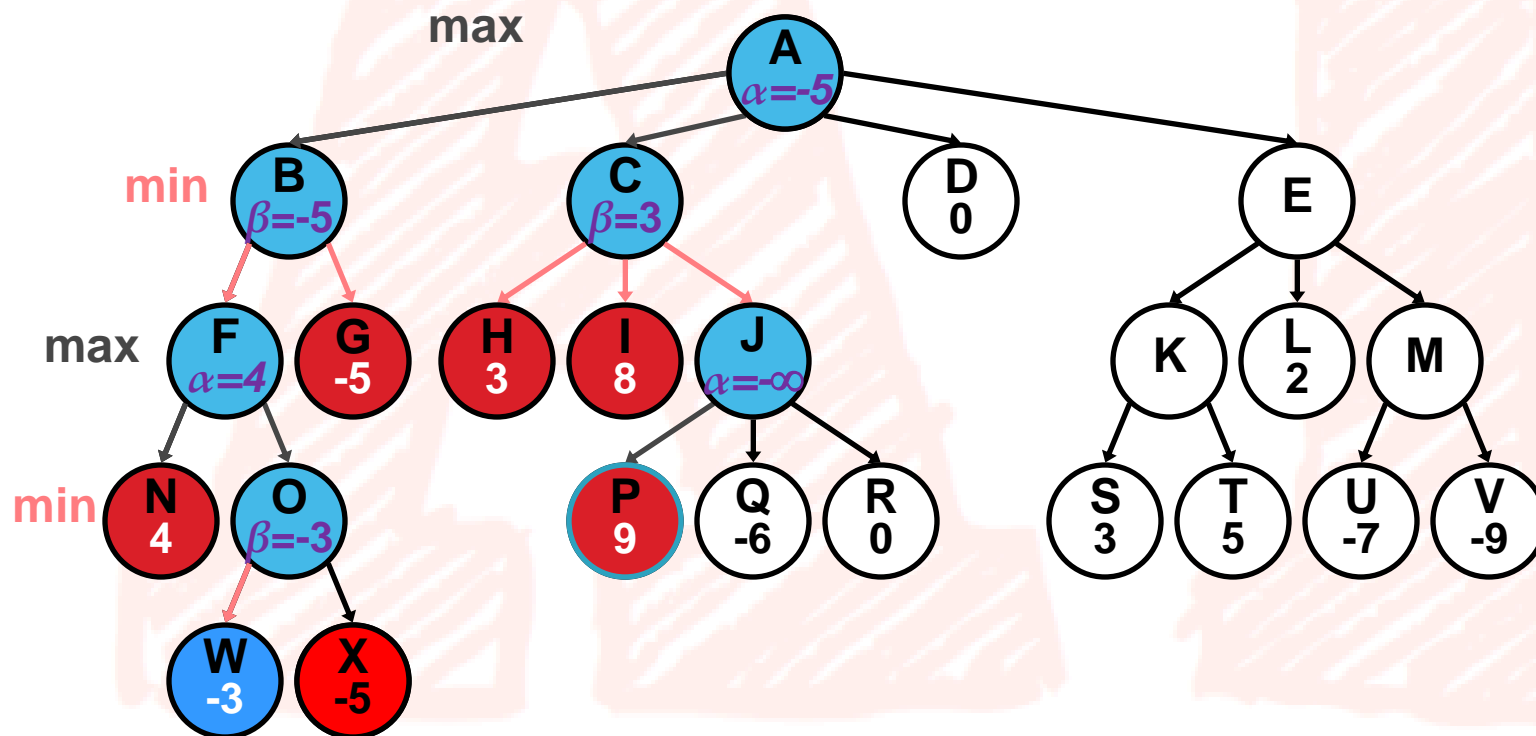
Call  
Stack

J  
C  
A

# Alpha-Beta Search Example

$\text{minimax}(P, 3, 4)$

evaluate and return SBE value



Call  
Stack

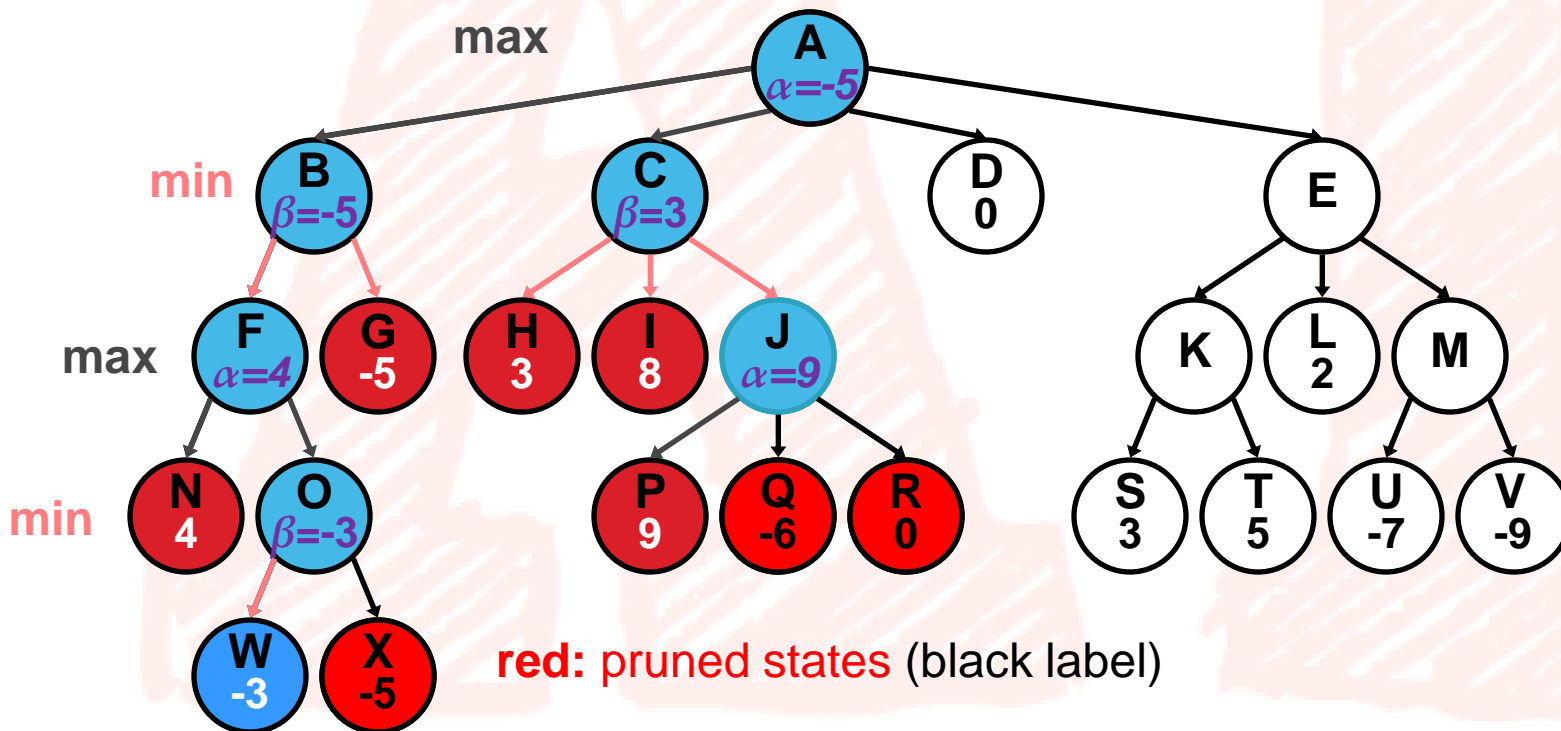
P  
J  
C  
A

# Alpha-Beta Search Example

back to  
 $\text{minimax}(J, 2, 4)$

$\alpha = 9$ , since  $9 \geq -\text{infinity}$  (maximizing)

Keep expanding J? **No** since J's  $\alpha \geq$  C's  $\beta$  is **true**: **beta cutoff**



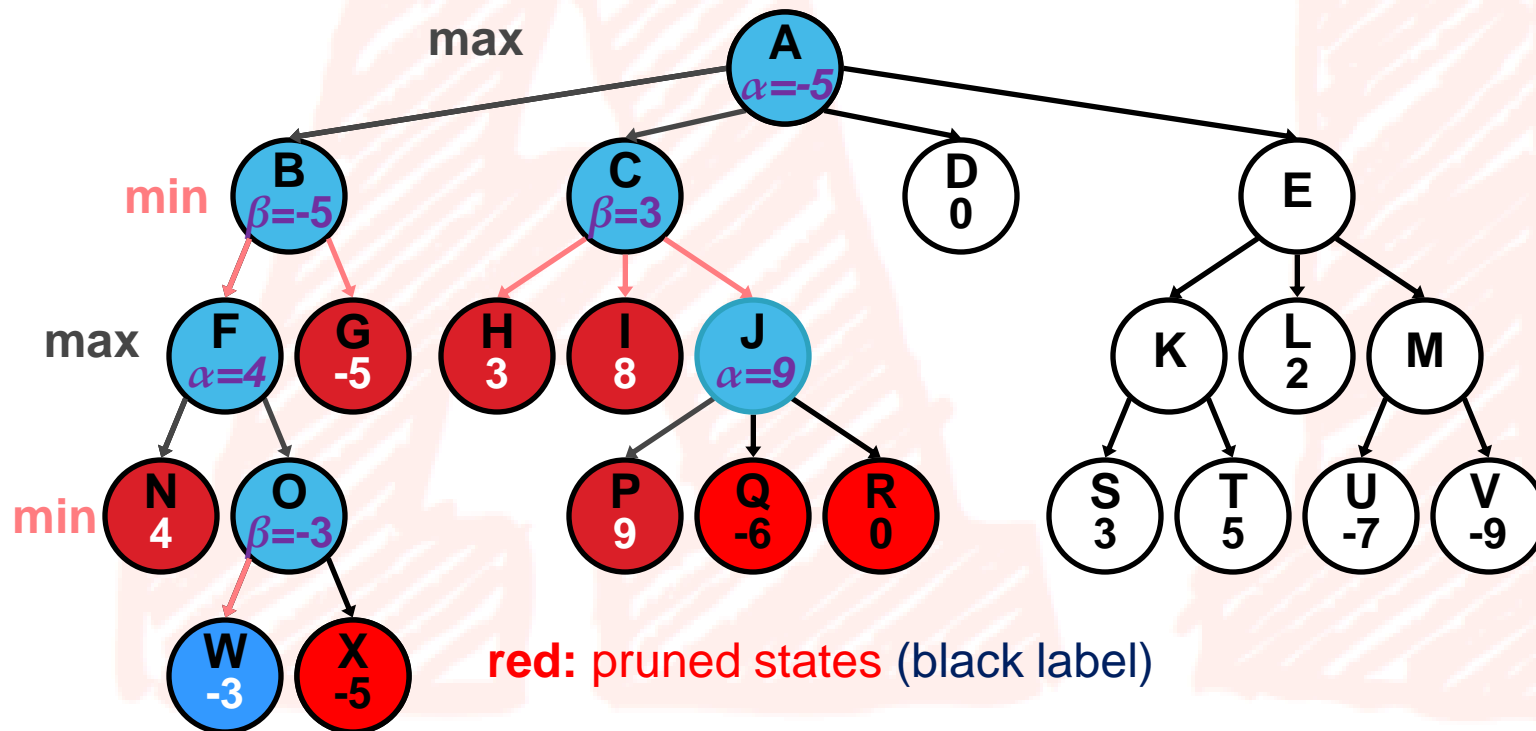
Call  
Stack

J  
C  
A

# Alpha-Beta Search Example

## ► Why?

- Computer will choose P or better, thus J's lower bound is 9.  
Smart opponent won't let computer take move to J  
(since opponent already has better move at H).



Call  
Stack

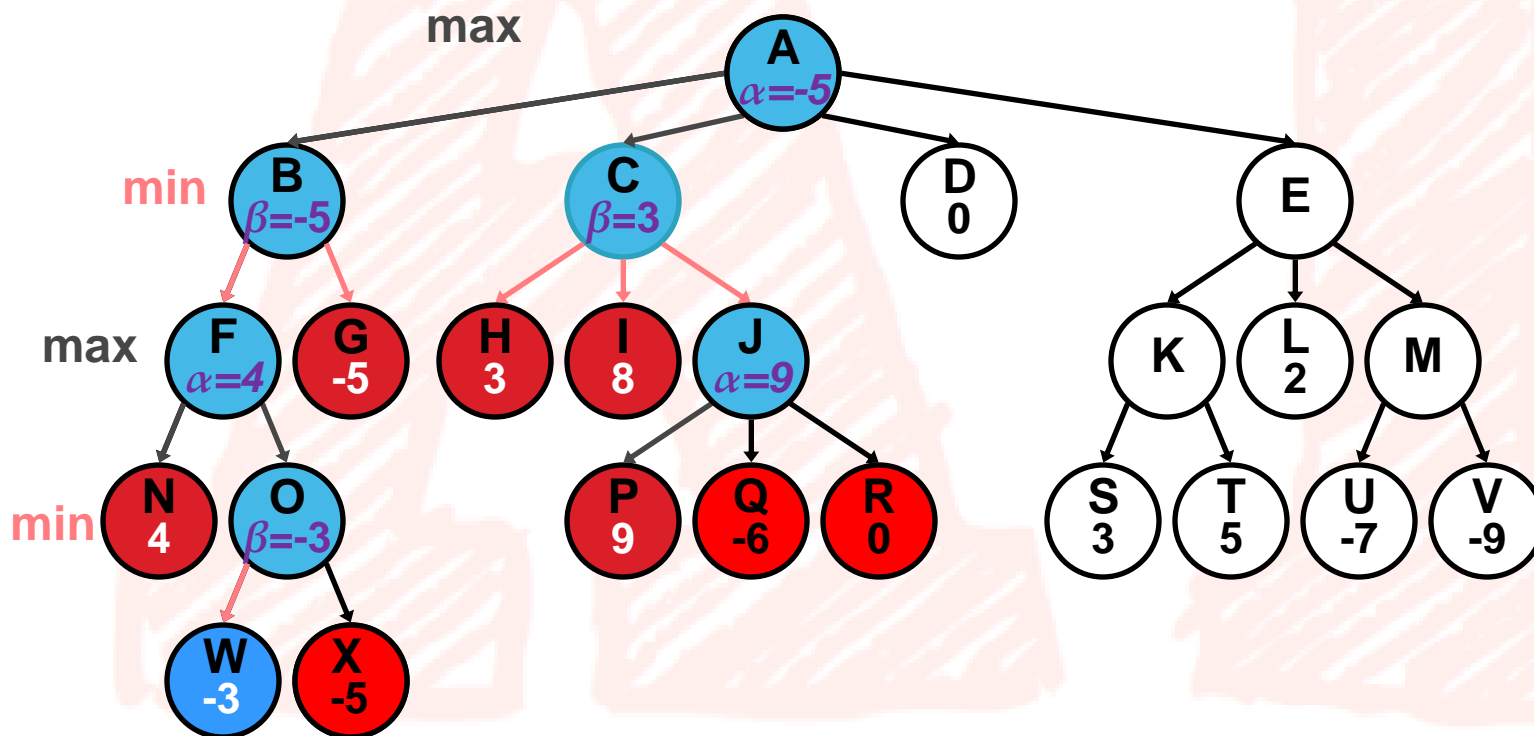
J  
C  
A

# Alpha-Beta Search Example

back to  
 $\text{minimax}(C, 1, 4)$

beta doesn't change, since  $9 > 3$  (minimizing)

Keep expanding C? **No** since no more successors for C

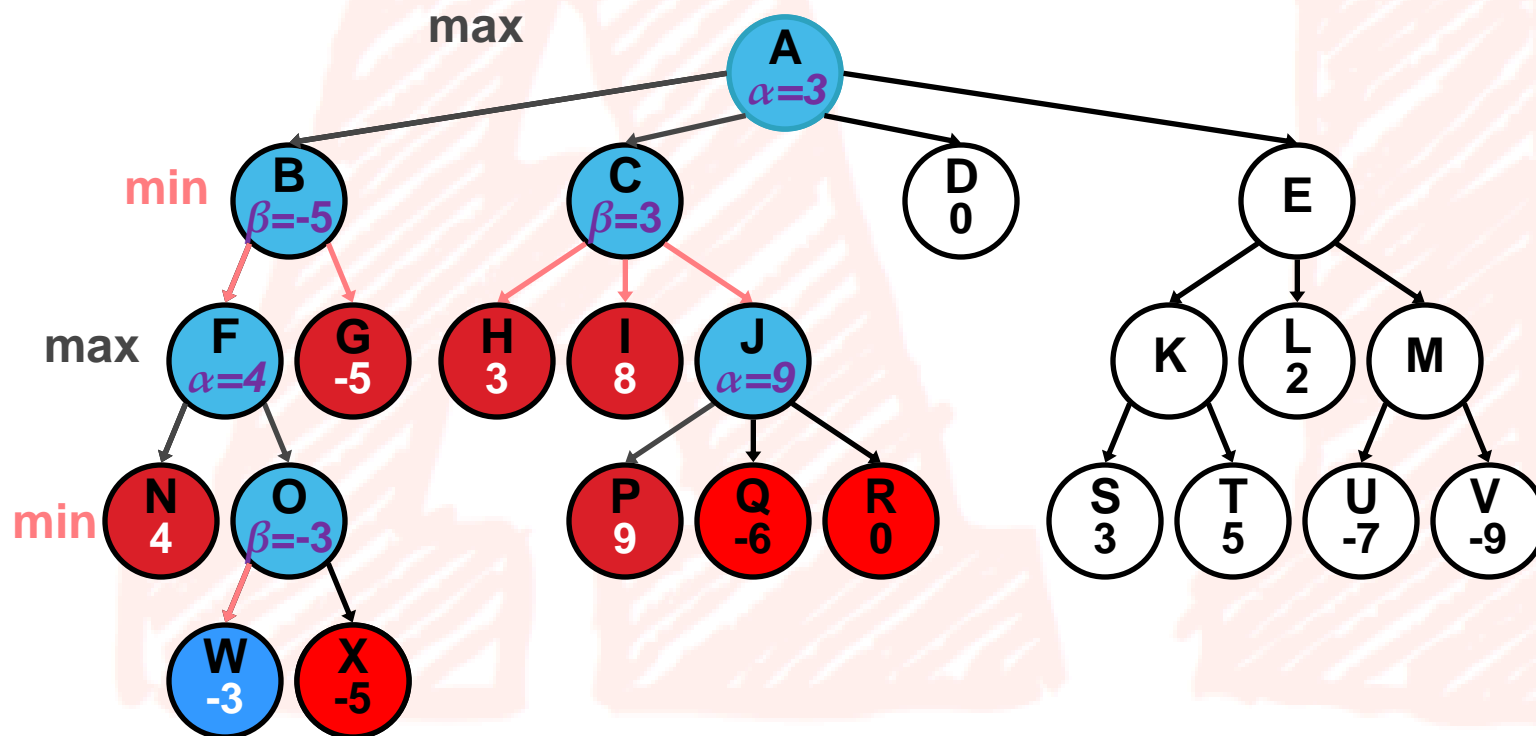


# Alpha-Beta Search Example

back to  
 $\text{minimax}(A, 0, 4)$

$\alpha = 3$ , since  $3 \geq -5$  (maximizing)

Keep expanding A? Yes since there are more successors, no cutoff test



Call  
Stack

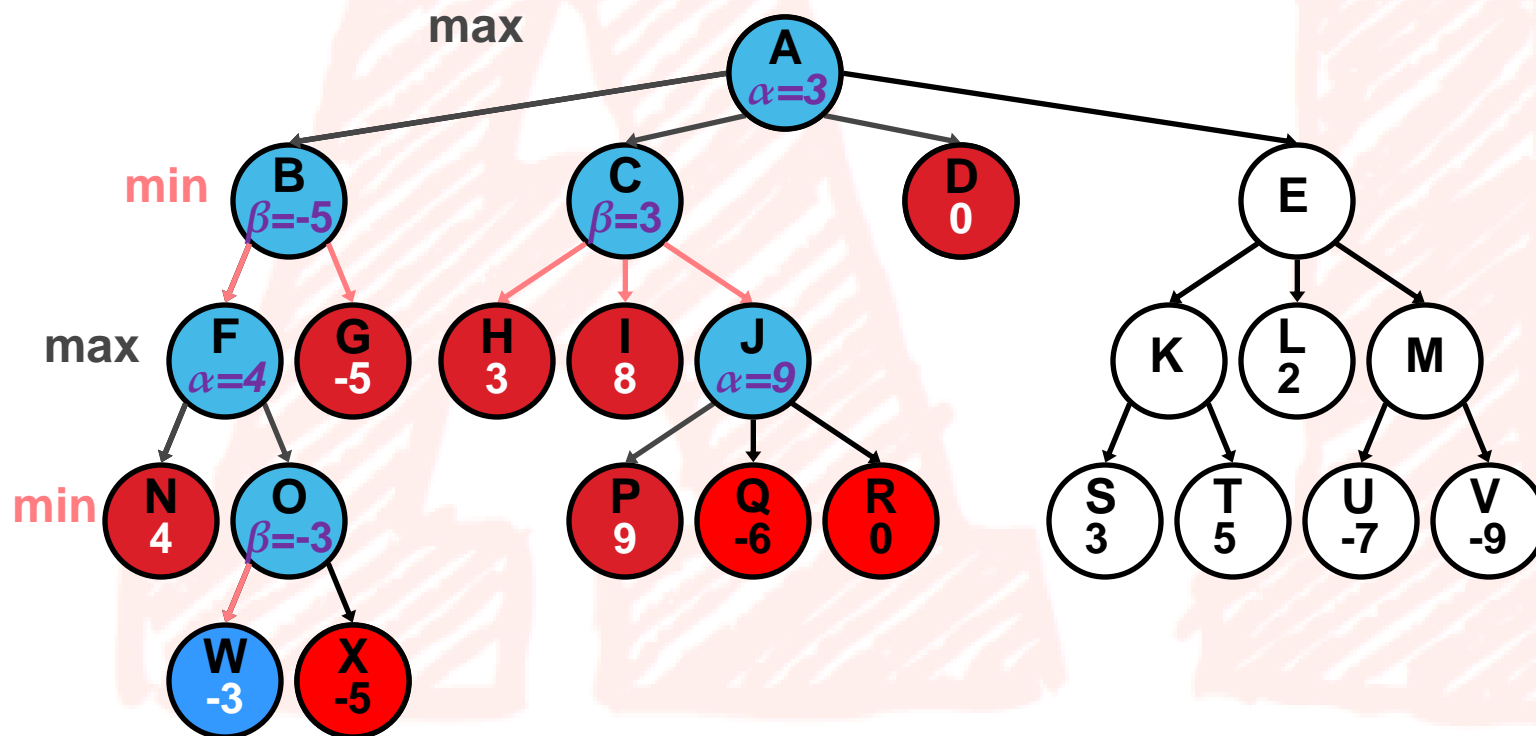
A



# Alpha-Beta Search Example

$\text{minimax}(D, 1, 4)$

evaluate and return SBE value



Call  
Stack

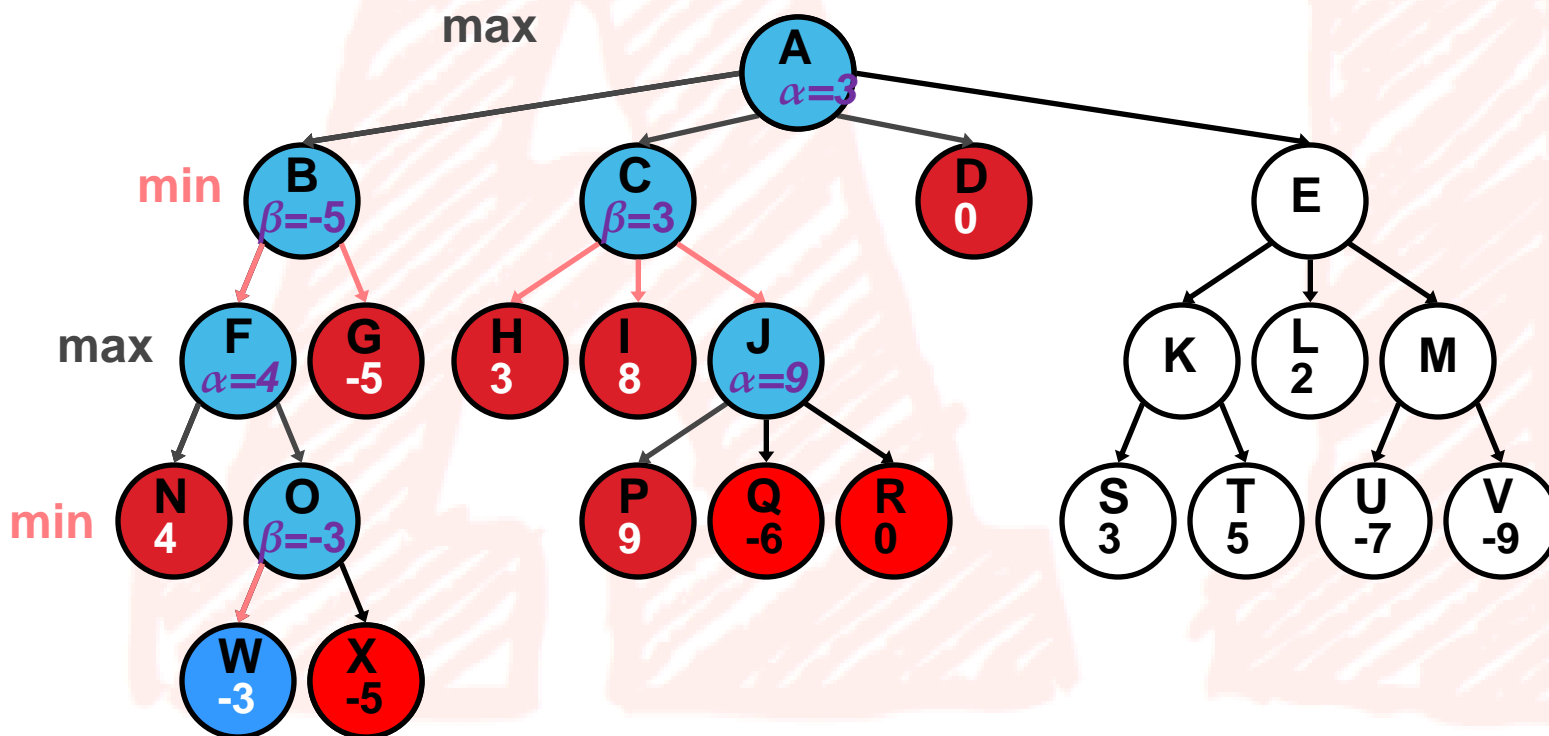
D  
A

# Alpha-Beta Search Example

back to  
 $\text{minimax}(A, 0, 4)$

alpha doesn't change, since  $0 < 3$  (maximizing)

Keep expanding A? Yes since there are more successors, no cutoff test

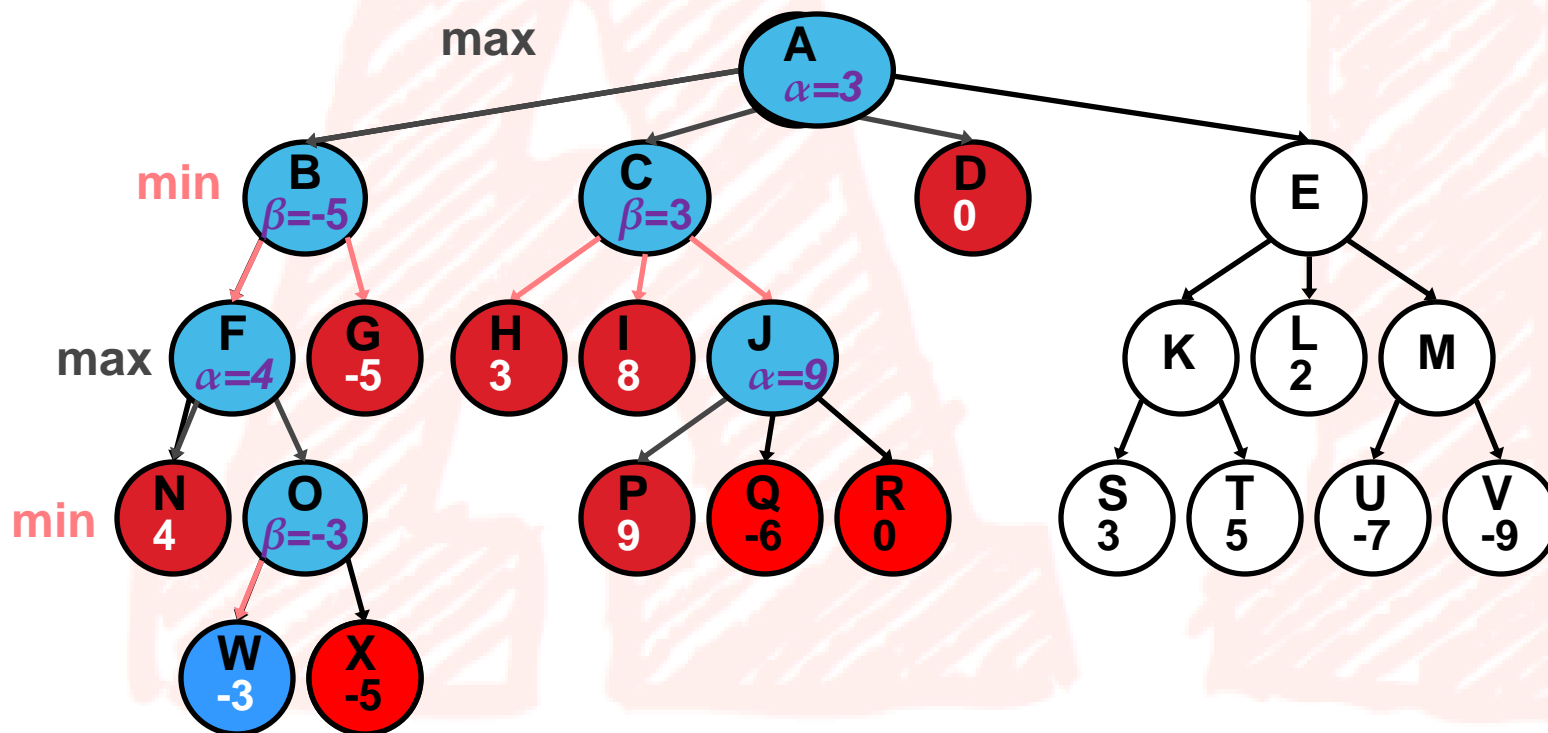


Call  
Stack

A

# Alpha-Beta Search Example

- ▶ How does the algorithm finish searching the tree?

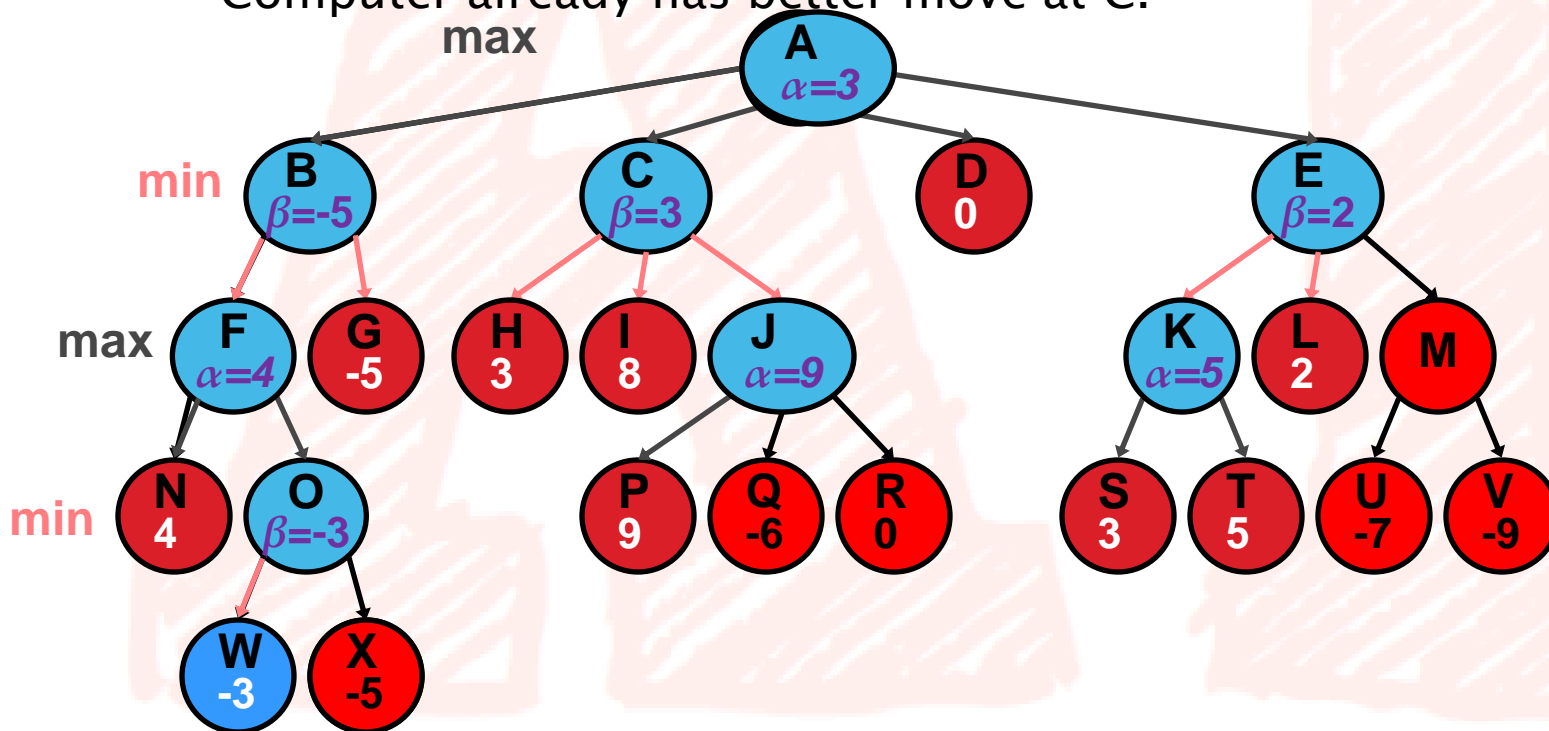


Call  
Stack

A

# Alpha-Beta Search Example

- ▶ Stop Expanding E since A's alpha  $\geq$  E's beta is true: alpha cutoff
- ▶ Why?
  - Smart opponent will choose L or worse, thus E's upper bound is 2.
  - Computer already has better move at C.

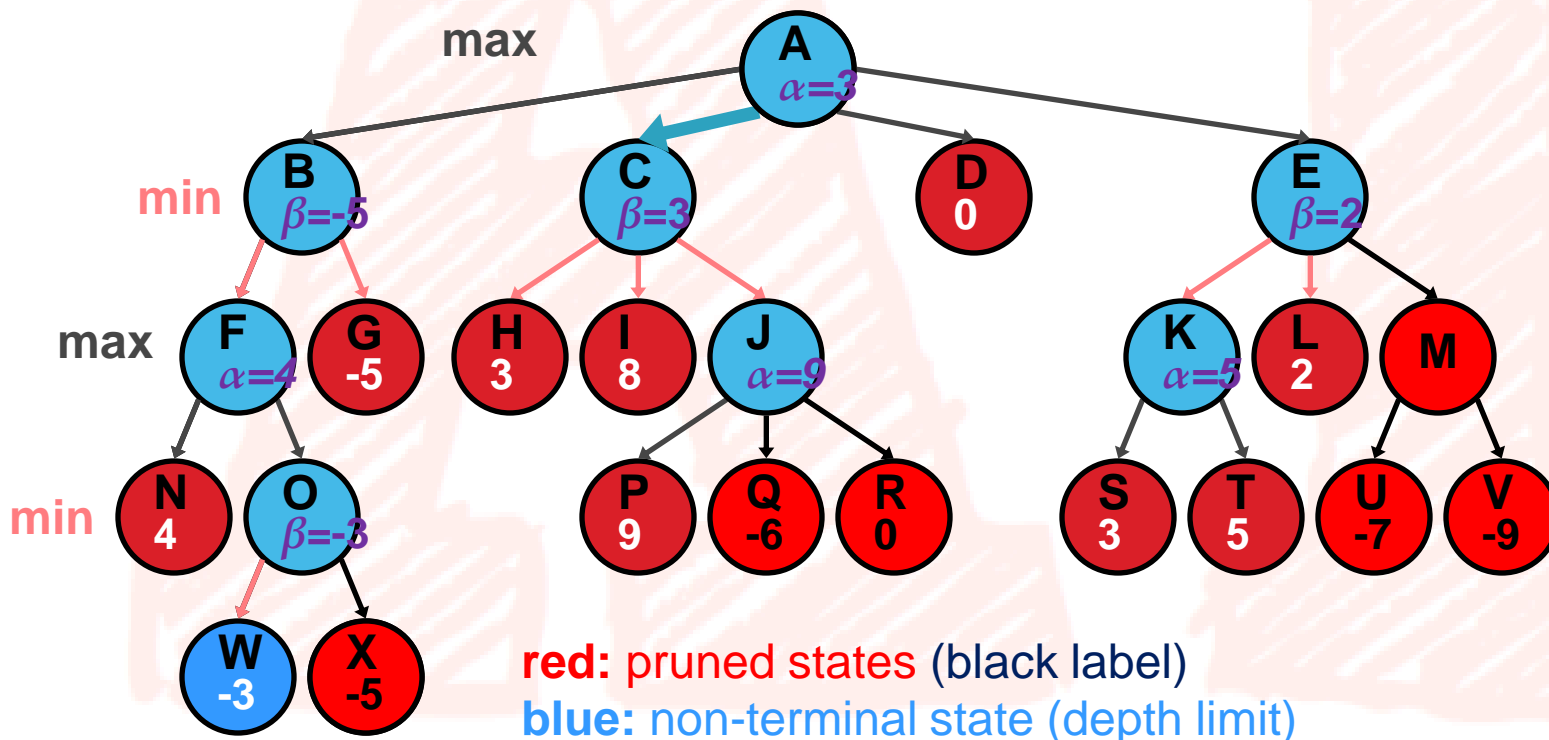


Call  
Stack

A

# Alpha-Beta Search Example

- Result: Computer chooses move to C.

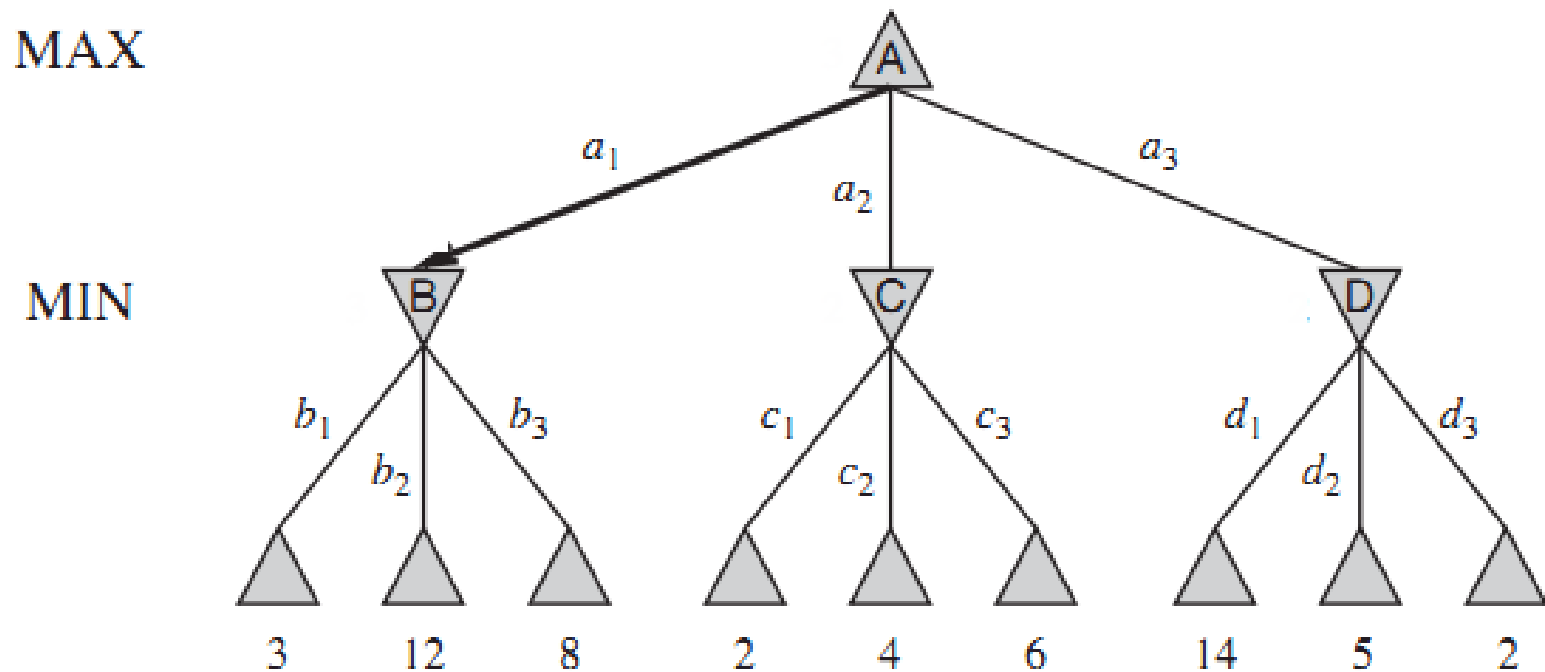


Call  
Stack

A

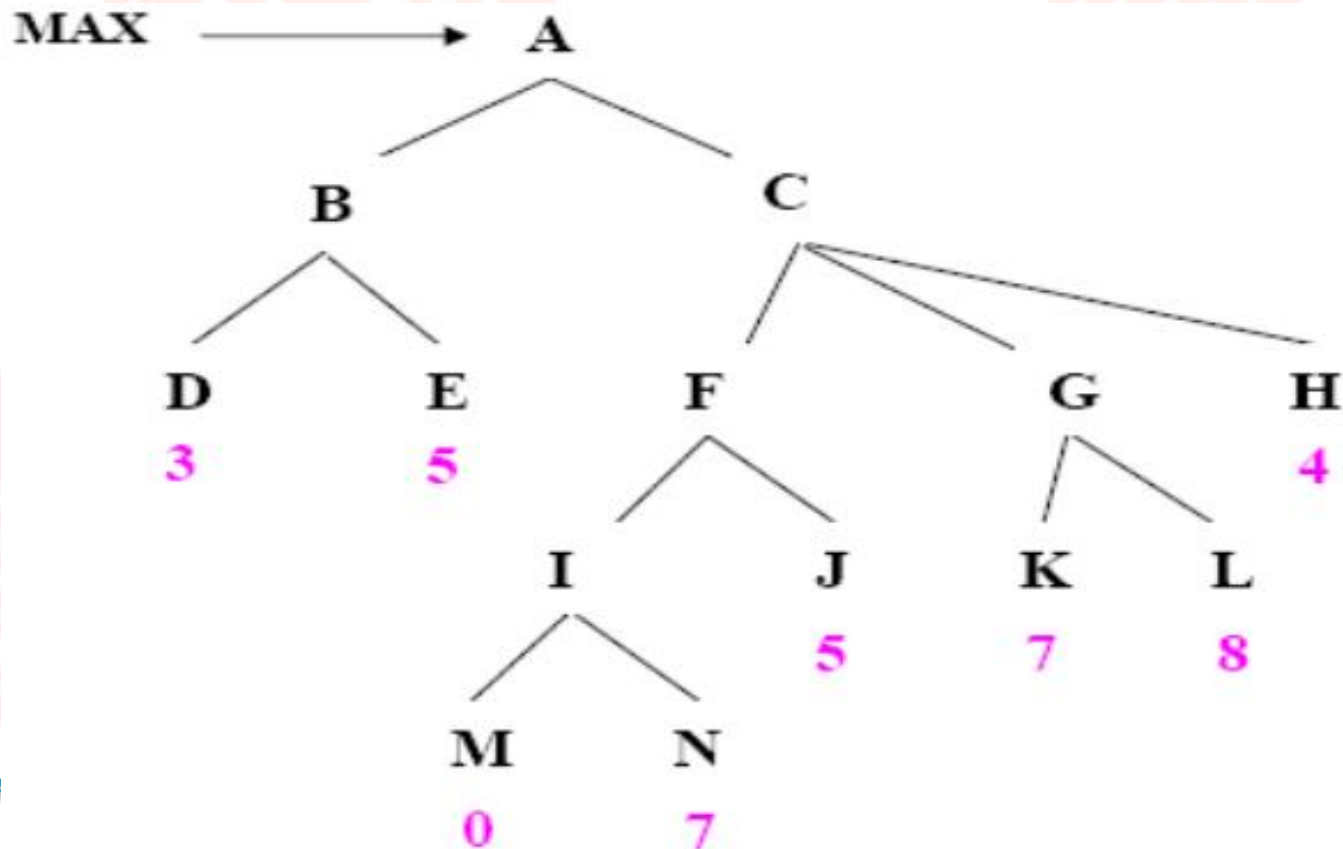
# Exercise 3

- Perform the minimax algorithm on the figure below with **Alpha-Beta-pruning**



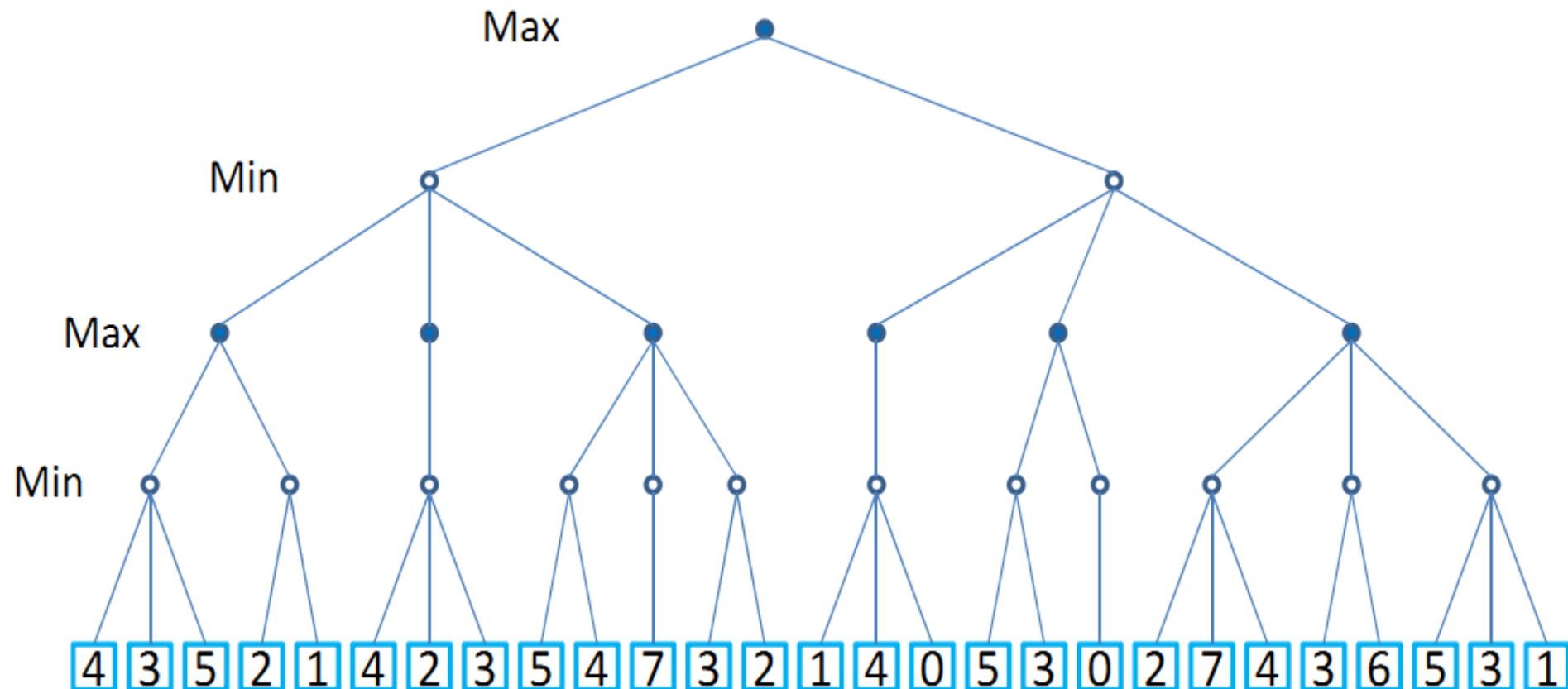
# Exercise 4

- ▶ Perform the minimax algorithm on the figure below with Alpha-Beta-pruning



# Exercise 5

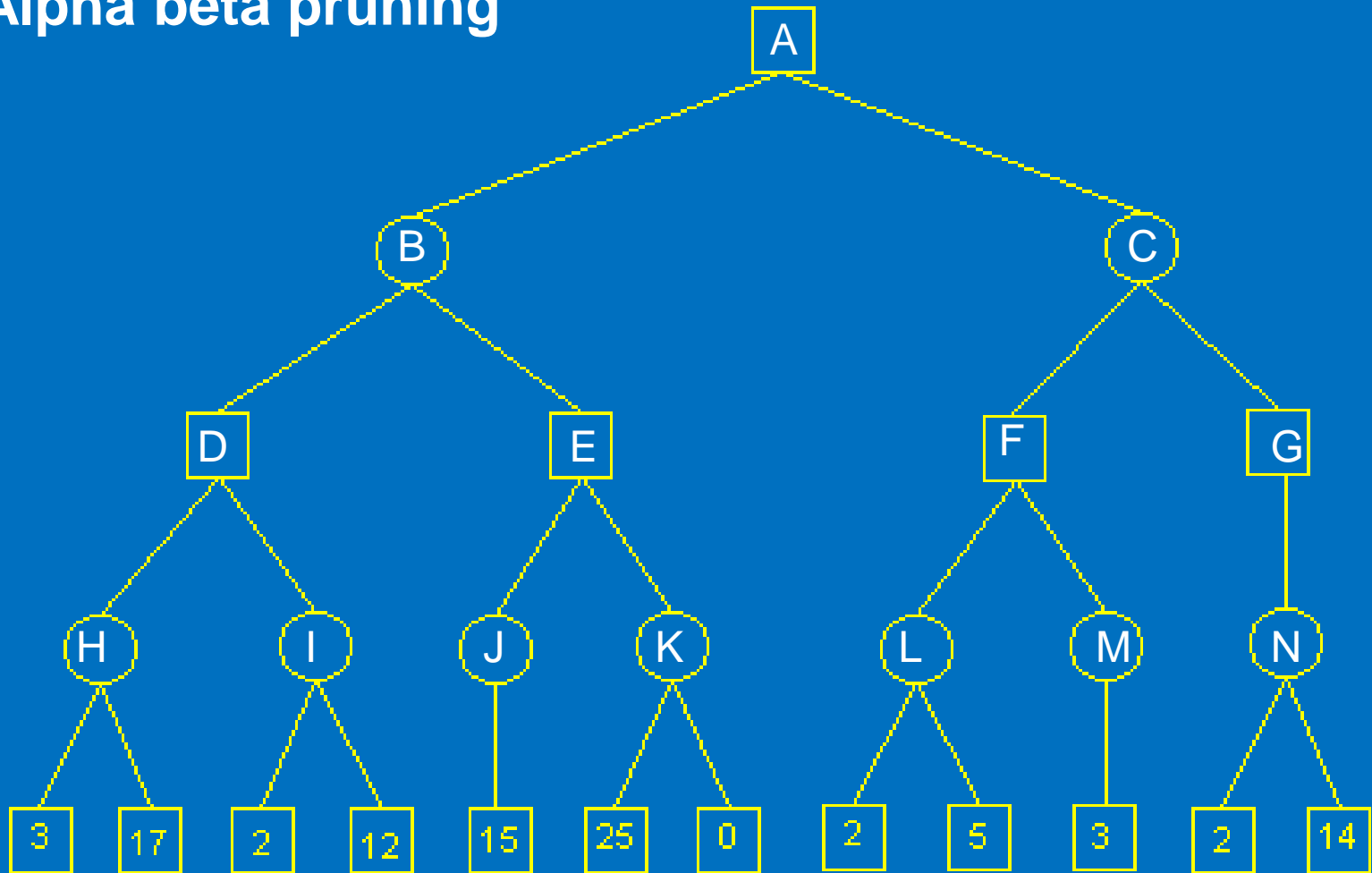
- Perform the minimax algorithm on the figure below with Alpha-Beta-pruning





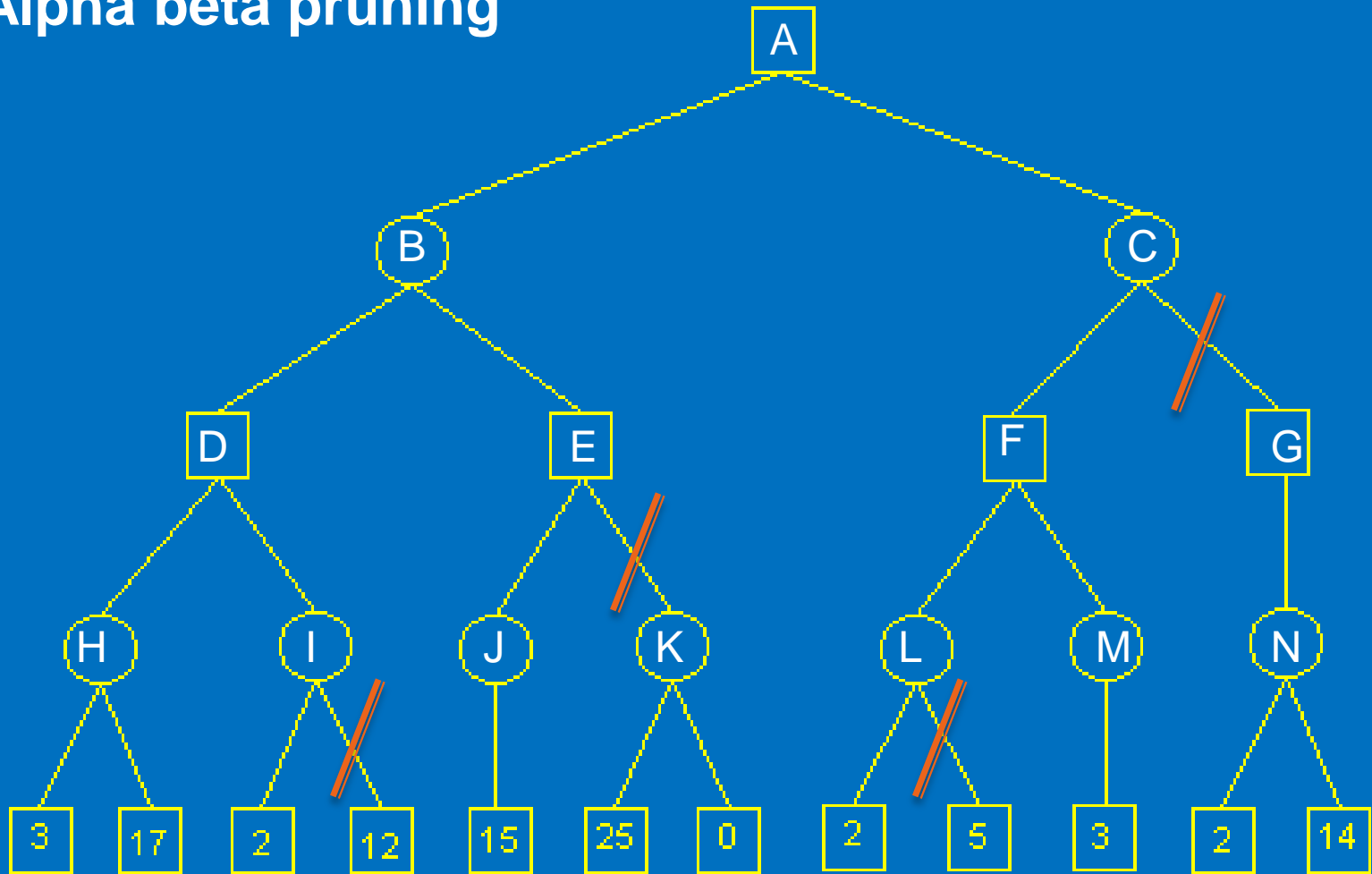
# Exercise 6

## Alpha beta pruning

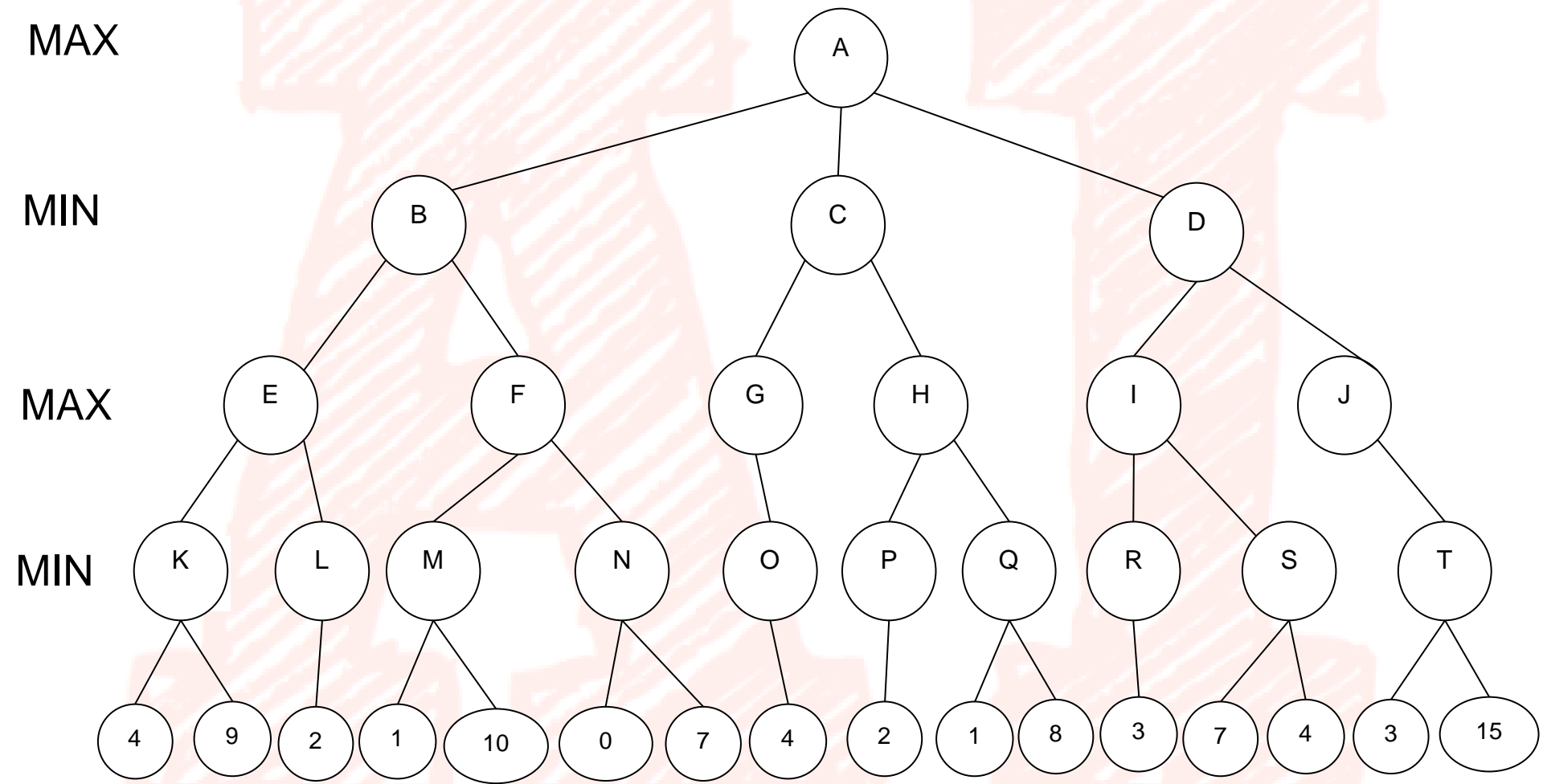


# Exercise 6: Result

## Alpha beta pruning



# Exercise 7



# Alpha-Beta Algorithm

**function** ALPHA-BETA-SEARCH(*state*) *returns an action*

**inputs:** *state*, current state in game

$v \leftarrow \text{MAX-VALUE}(\text{state}, -\infty, +\infty)$

**return** the *action* in SUCCESSORS(*state*) with value  $v$

**function** MAX-VALUE(*state*,  $\alpha$ ,  $\beta$ ) *returns a utility value*

**if** TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow -\infty$

**for each**  $s$  **in** SUCCESSORS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s, \alpha, \beta))$

**if**  $v \geq \beta$  **then return**  $v$

$\alpha \leftarrow \text{MAX}(\alpha, v)$

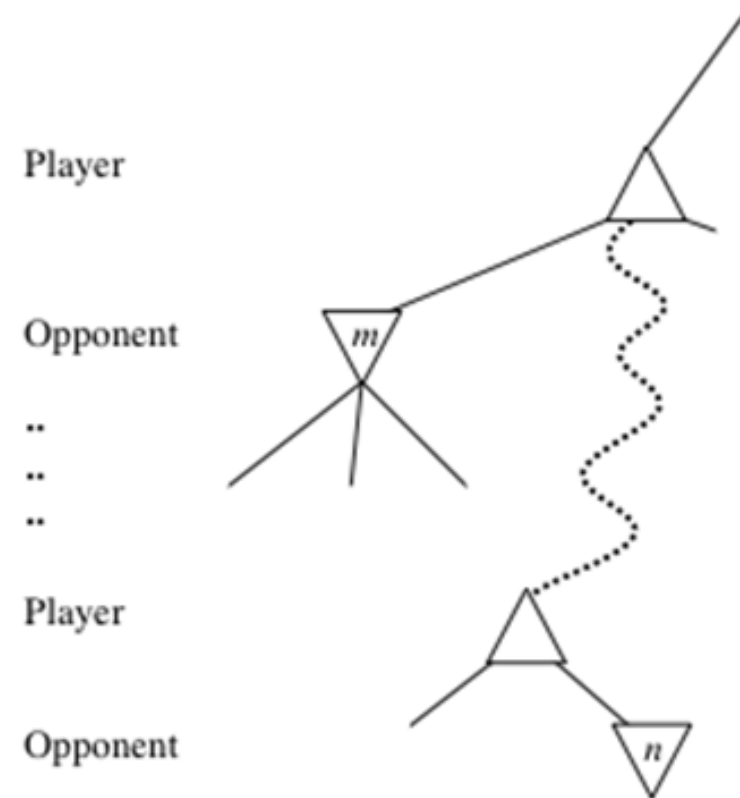
**return**  $v$

# Alpha-Beta Algorithm

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

# General alpha-beta pruning

- ▶ Consider a **node  $n$**  somewhere in the tree
- ▶ If player has a **better choice at**
  - Parent node of  **$n$**
  - Or any choice point further up
- ▶  **$n$**  will never be reached in actual play.
- ▶ Hence when enough is known about  **$n$** , it can be pruned.



# Final Comments: Alpha–Beta Pruning

- ▶ Pruning does not affect final results
- ▶ Entire subtrees can be pruned.
- ▶ Good *move ordering* improves effectiveness of pruning
- ▶ With "perfect ordering" time complexity is  $O(b^{m/2})$ 
  - Branching factor of  $\sqrt{b}$  !!
  - Alpha–beta pruning can look twice as far as Minimax in the same amount of time
- ▶ Repeated states are again possible.
  - Store them in memory = transposition table

# Imperfect Real-Time Decisions

- ▶ Minimax require **too much leaf-node evaluations**.
- ▶ May be impractical within a reasonable amount of time.
- ▶ SHANNON (1950):
  - Cut off search earlier (**replace TERMINAL-TEST by CUTOFF-TEST**)
  - Apply heuristic evaluation function EVAL (replacing utility function of alpha-beta)



# Cutting off search

- ▶ Change:

```
if TERMINAL-TEST(state) then  
  return UTILITY(state)
```

into



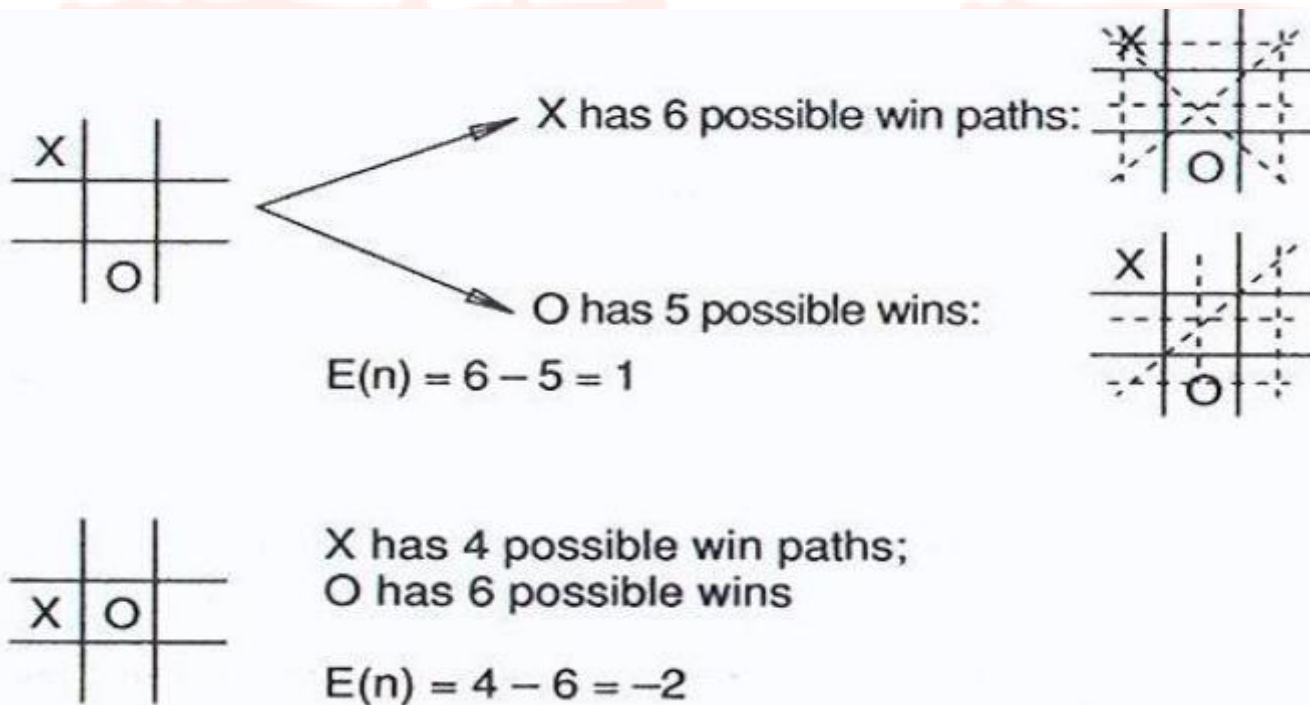
```
if CUTOFF-TEST(state, depth) then  
  return EVAL(state)
```

- ▶ Introduces a fixed-depth limit depth
  - Is selected so that the amount of time will not exceed what the rules of the game allow.
- ▶ When cutoff occurs, the evaluation is performed.

# Heuristic EVAL

- ▶ EVAL function returns an estimate of the expected utility of the game from a given position.
- ▶ Performance of game playing depends on quality of EVAL.
- ▶ Requirements:
  - EVAL must agree with terminal-nodes in the same way as UTILITY.
  - Computation may not take too long.
  - For non-terminal states the EVAL should be strongly correlated with the actual chance of winning.
- ▶ Only useful for quiescent (no wild swings in value in near future) states

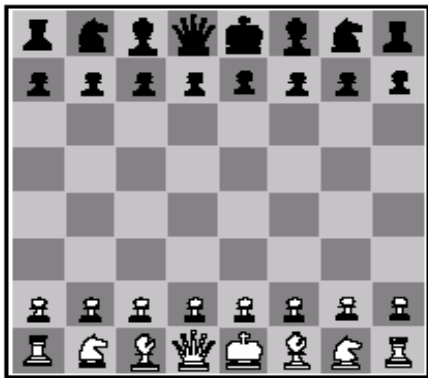
# Heuristic **EVAL** example



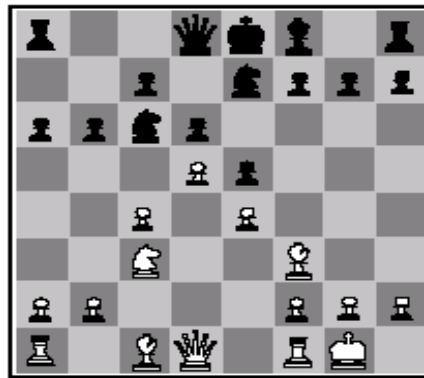
- Heuristic:  $E(n) = M(n) - O(n)$ 
  - $M(n)$ : total win paths of X,
  - $O(n)$ : total win paths of O

# Heuristic EVAL example

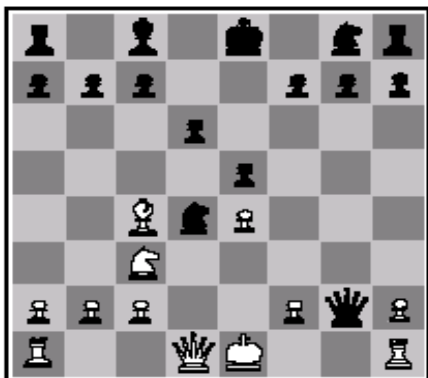
$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s)$$



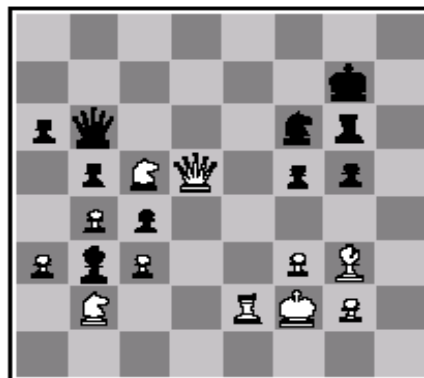
(a) White to move  
Fairly even



(b) Black to move  
White slightly better



(c) White to move  
Black winning



(d) Black to move  
White about to lose

- $w_i$ : a weight,
- $f_i$ : a feature of the position

- Pawn: 1,
- Knight: 3,
- Rook: 5,
- Queen: 9

# Multiplayer games

- ▶ Games allow **more than two players**
- ▶ Single **minimax values become vectors**

to move

A

(1, 2, 6)

B

(1, 2, 6)

(-1, 5, 2)

C

(1, 2, 6)

X

(6, 1, 2)

(-1, 5, 2)

(5, 4, 5)

A

(1, 2, 6)

(4, 2, 3)

(6, 1, 2)

(7, 4, -1)

(5, -1, -1)

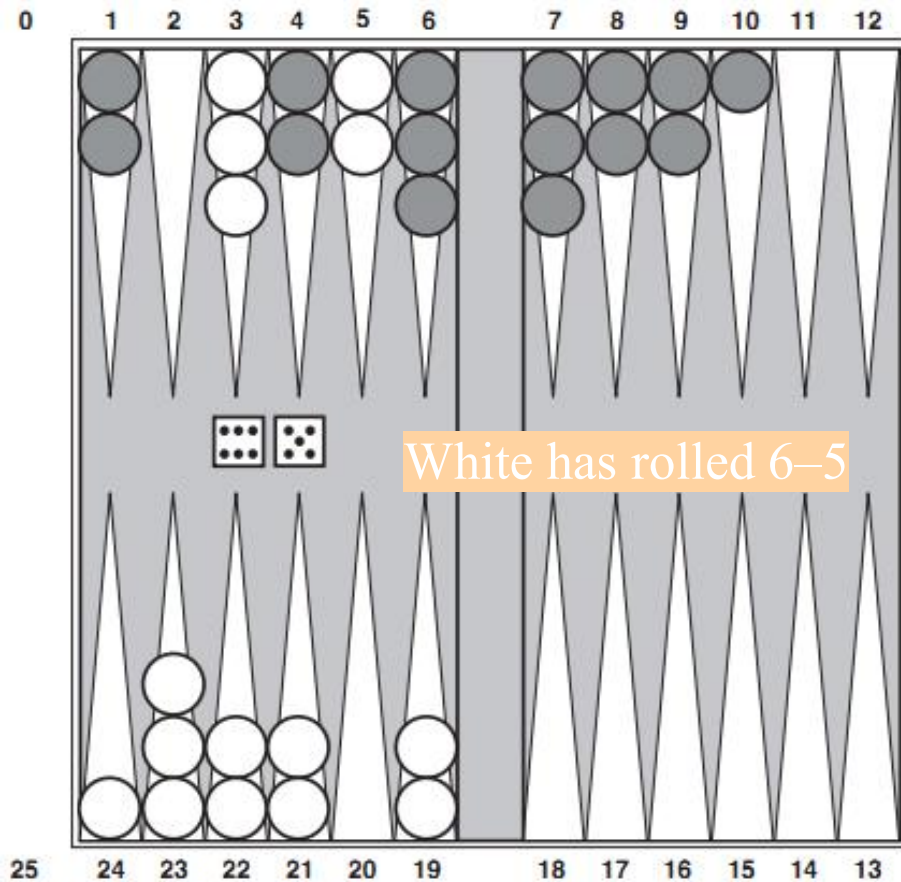
(-1, 5, 2)

(7, 7, -1)

(5, 4, 5)

# Games that include chance

Backgammon



MAX

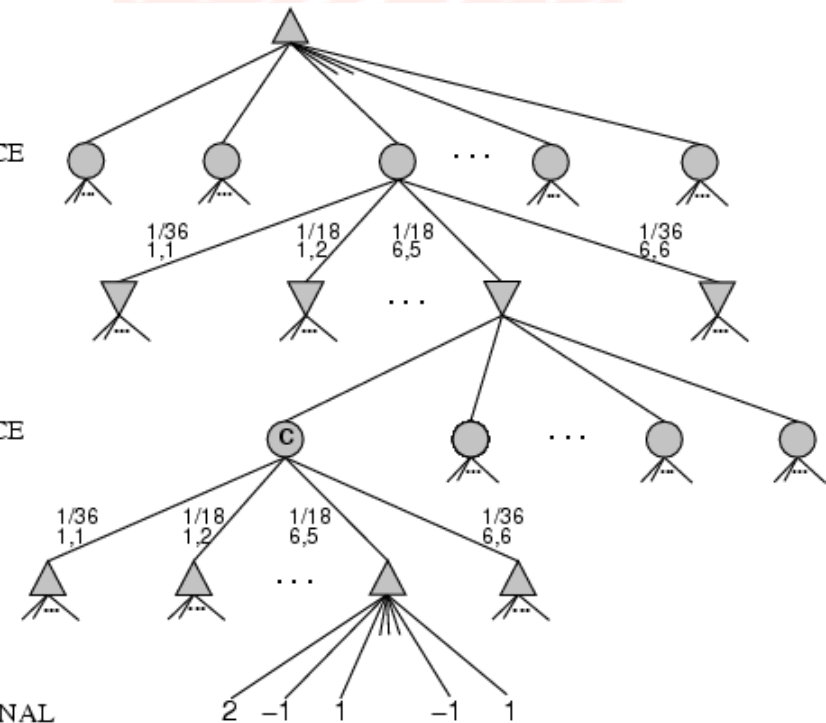
CHANCE

MIN

CHANCE

MAX

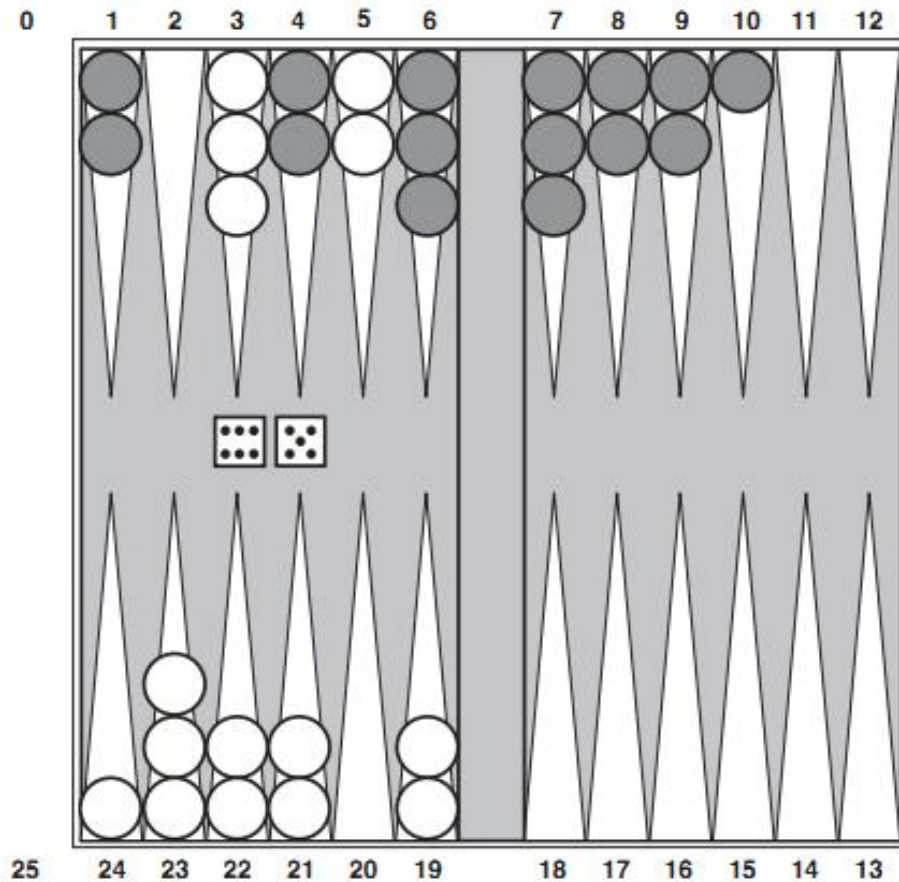
TERMINAL



- Possible moves (5-10, 5-11), (5-11, 19-24), (5-10, 10-16) and (5-11, 11-16)

# Games that include chance

Backgammon



MAX

CHANCE

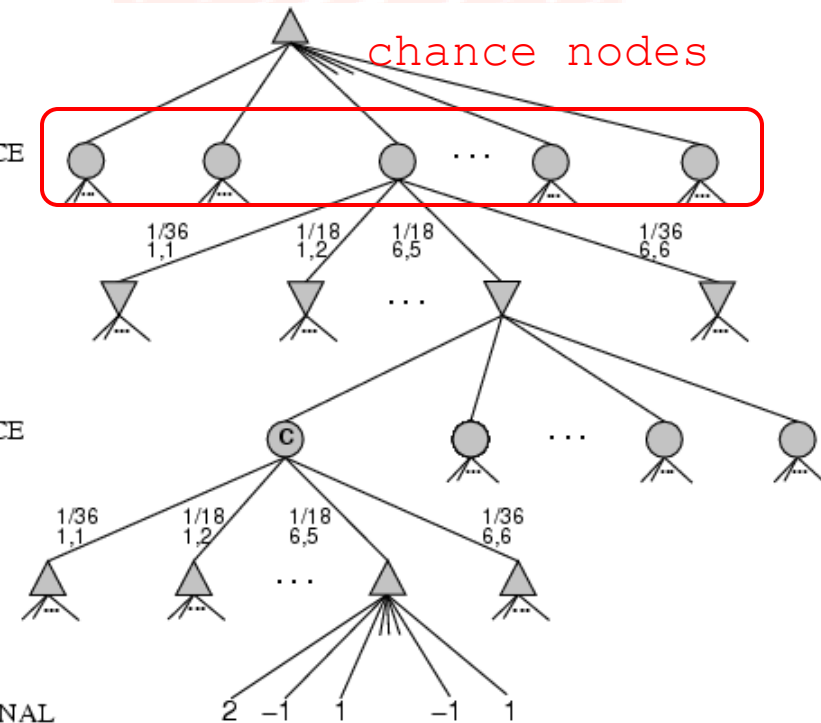
MIN

CHANCE

MAX

TERMINAL

chance nodes

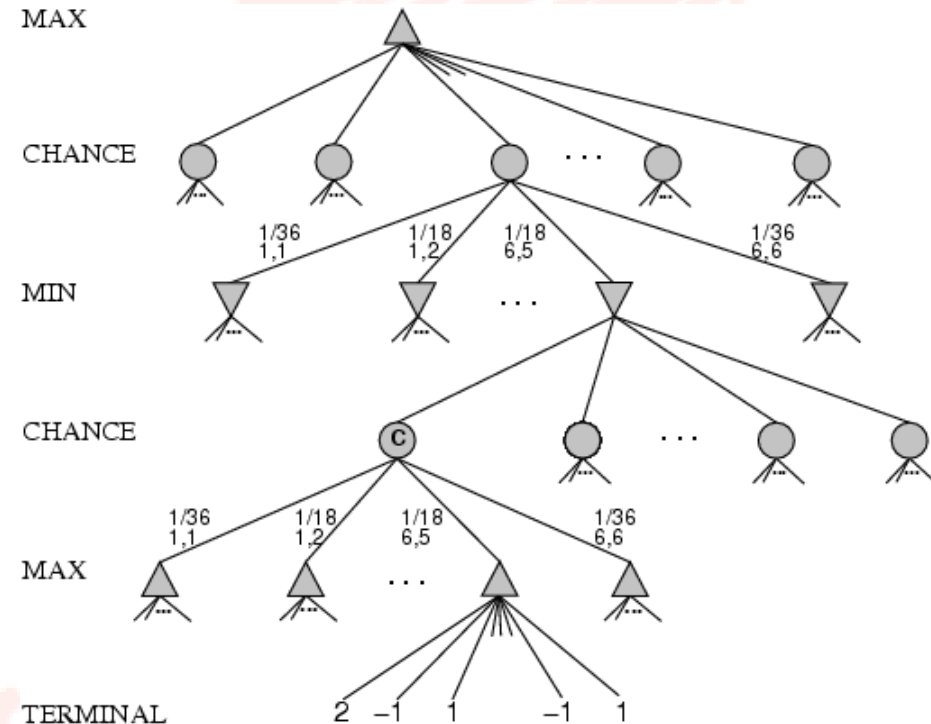
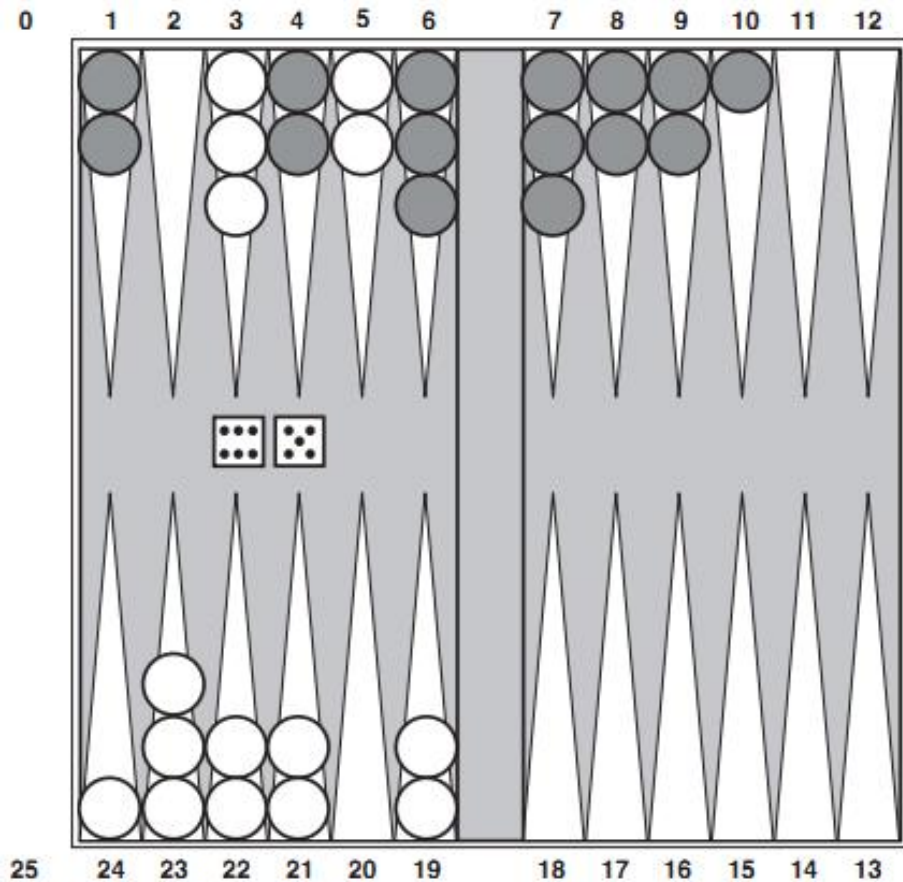


- ▶ Possible moves (5-10, 5-11), (5-11, 19-24), (5-10, 10-16) and (5-11, 11-16)
- ▶ [1,1], ..., [6,6] chance  $1/36$ , all other chance  $1/18$



# Games that include chance

Backgammon



- ▶  $[1,1], \dots, [6,6]$  chance  $1/36$ , all other chance  $1/18$
- ▶ Can not calculate definite minimax value, only **expected value**



# Expected minimax value

EXPECTED-MINIMAX-VALUE(n) =

UTILITY(n)

If n is a terminal

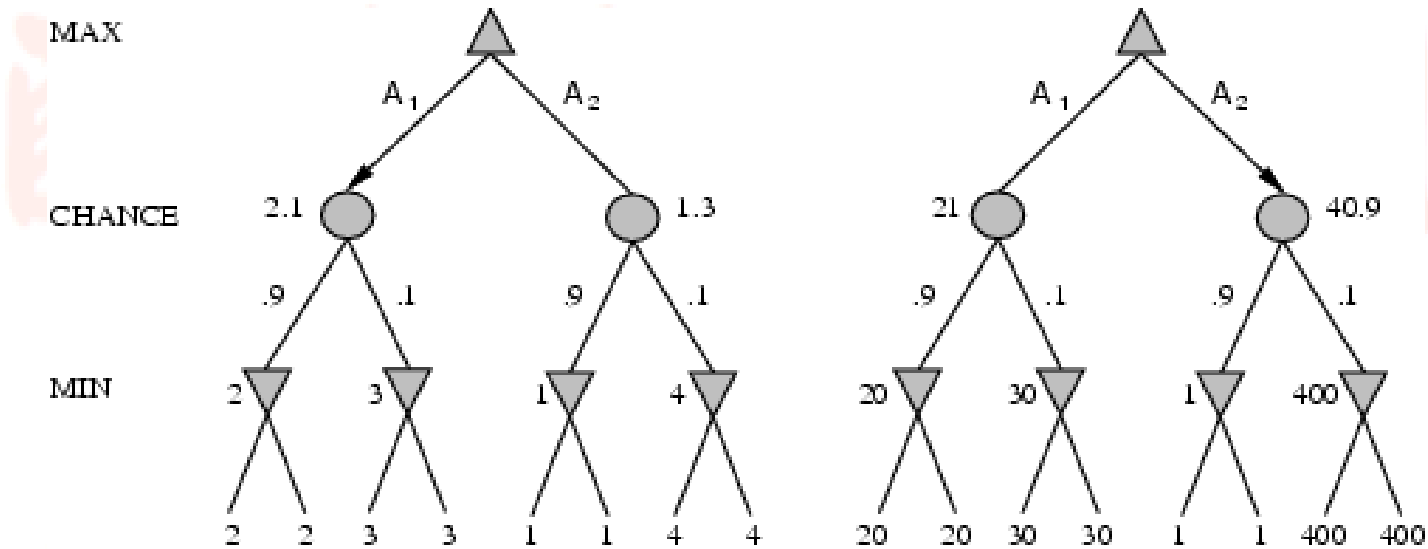
$\max_{s \in \text{successors}(n)} \text{MINIMAX-VALUE}(s)$  If n is a max node

$\min_{s \in \text{successors}(n)} \text{MINIMAX-VALUE}(s)$  If n is a min node

$\sum_{s \in \text{successors}(n)} P(s) \cdot \text{EXPECTEDMINIMAX}(s)$  If n is a  
chance node

- ▶ These equations can be backed-up recursively all the way to the root of the game tree.

# Position evaluation with chance nodes



- ▶ Left,  $A_1$  wins
- ▶ Right,  $A_2$  wins
- ▶ **Outcome of evaluation function may not change** when values are scaled differently.
- ▶ Behavior is preserved only by a positive linear transformation of EVAL.

# Summary

- ▶ Games are fun (and dangerous)
- ▶ They illustrate several important points about AI
  - Perfection is unattainable  $\rightarrow$  approximation
  - Good idea what to think about
  - Uncertainty constrains the assignment of values to states
- ▶ Games are to AI as grand prix racing is to automobile design.



FACULTY OF INFORMATION TECHNOLOGY

**Thank you for your attention!**