

# Artificial Intelligence

CS 165A

Apr 26, 2022

Instructor: Prof. Yu-Xiang Wang

Today

- Examples of heuristics in A\*-search
- Games and Adversarial Search

# Project 1 submissions so far

- Most students have submitted
  - A few reports are still missing
- Most submissions got full credits for the coding part
  - We are still grading the reports
- Feel free to talk to us if you find it challenging.
- Notes:
  - You could still submit (late days will be automatically applied)
  - Bonus questions have no deadline.
  - Project 2 is out. Please start early!

# Recap: Search algorithms

- State-space diagram vs Search Tree
- Uninformed Search algorithms
  - BFS / DFS
  - Depth Limited Search
  - Iterative Deepening Search.
  - Uniform cost search.
- Informed Search (with an heuristic function  $h$ ):
  - Greedy Best-First-Search. (not complete / optimal)
  - A\* Search (complete / optimal if  $h$  is admissible)

# Recap: Summary table of uninformed search

| Criteria  | BFS               | Uniform-cost          | DFS      | Depth-limited | IDS               | Bidirectional      |
|-----------|-------------------|-----------------------|----------|---------------|-------------------|--------------------|
| Complete? | Yes <sup>#</sup>  | Yes <sup>#&amp;</sup> | No       | No            | Yes <sup>#</sup>  | Yes <sup>#+</sup>  |
| Time      | $O(b^d)$          | $O(b^{1+[C^*/e]})$    | $O(b^m)$ | $O(b^l)$      | $O(b^d)$          | $O(b^{d/2})$       |
| Space     | $O(b^d)$          | $O(b^{1+[C^*/e]})$    | $O(bm)$  | $O(bl)$       | $O(bd)$           | $O(b^{d/2})$       |
| Optimal?  | Yes <sup>\$</sup> | Yes                   | No       | No            | Yes <sup>\$</sup> | Yes <sup>\$+</sup> |

$b$ : Branching factor

$d$ : Depth of the shallowest goal

$l$ : Depth limit

$m$ : Maximum depth of search tree

$e$ : The lower bound of the step cost

<sup>#</sup>: Complete if  $b$  is finite

<sup>&</sup>: Complete if step cost  $\geq e$

<sup>\$</sup>: Optimal if all step costs are identical

<sup>+</sup>: If both direction use BFS

(Section 3.4.7 in the AIMA book.)

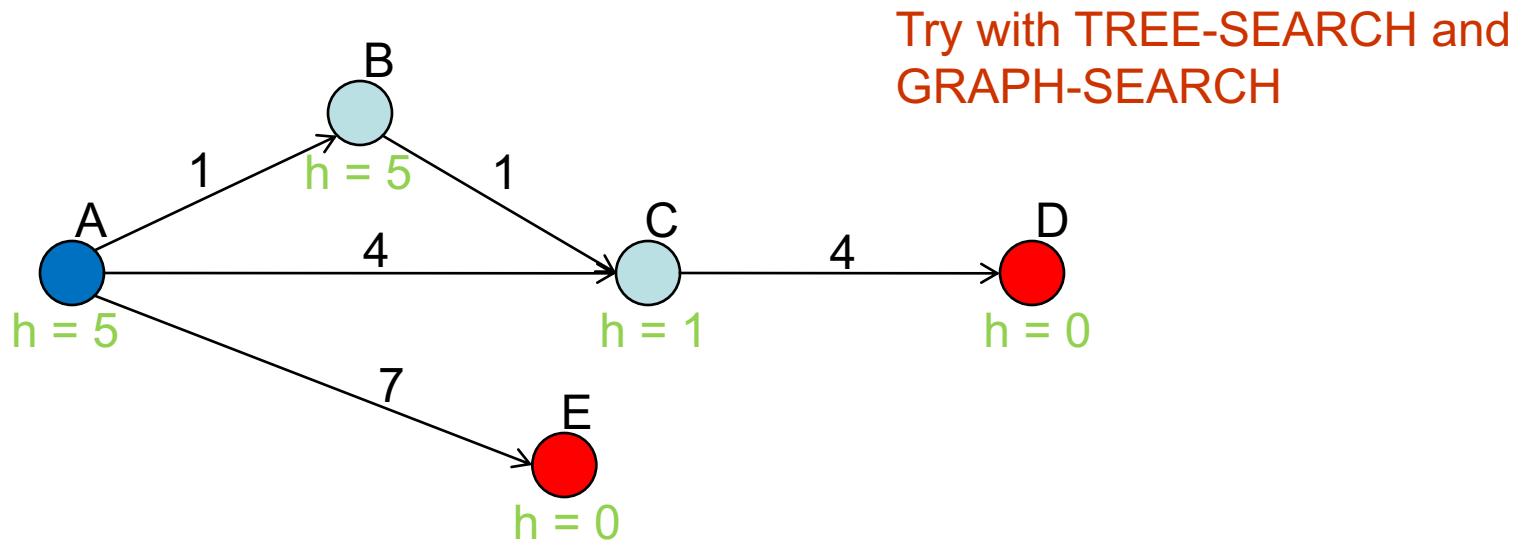
# Recap: A\* Search (Pronounced “A-Star”)

- Uniform-cost search minimizes  $g(n)$  (“past” cost)
- Greedy search minimizes  $h(n)$  (“expected” or “future” cost)
- “A\* Search” combines the two:
  - Minimize  $f(n) = g(n) + h(n)$
  - Accounts for the “past” and the “future”
  - Estimates the cheapest solution (complete path) through node  $n$

```
function A*-SEARCH(problem, h) returns a solution or failure
return BEST-FIRST-SEARCH(problem, f)
```

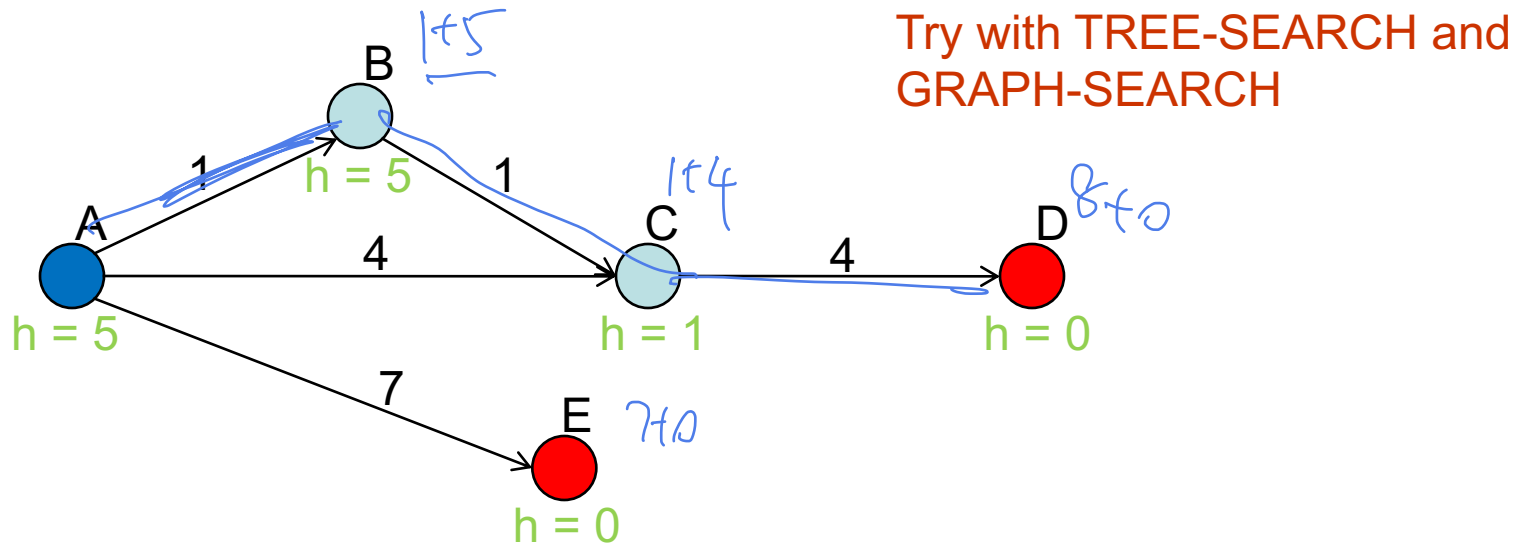
# Recap: Avoiding Repeated States using A\*

- Is GRAPH-SEARCH optimal with A\*?



# Recap: Avoiding Repeated States using A\*

- Is GRAPH-SEARCH optimal with A\*?



Graph Search

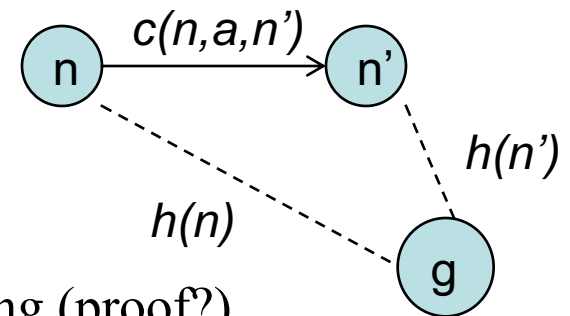
Step 1: Among B, C, E, Choose C

Step 2: Among B, E, D, Choose B

Step 3: Among D, E, Choose E. (you are not going to select C again)

# Recap: Consistency (Monotonicity) of heuristic $h$

- A heuristic is consistent (or monotonic) provided
  - for any node  $n$ , for any successor  $n'$  generated by action  $a$  with cost  $c(n,a,n')$ 
    - $h(n) \leq c(n,a,n') + h(n')$
  - akin to triangle inequality.
  - guarantees admissibility (proof?).
  - values of  $f(n)$  along any path are non-decreasing (proof?).



- GRAPH-SEARCH using consistent  $f(n)$  is optimal.
- Note that  $h(n) = 0$  is consistent and admissible.



# This lecture

- Example of heuristics / A\* search
  - Effective branching factor
- Games
- Adversarial Search

# Heuristics

- What's a heuristic for
  - Driving distance (or time) from city A to city B ?
  - 8-puzzle problem ?
  - M&C ?
  - Robot navigation ?

# Heuristics

- What's a heuristic for

- Driving distance (or time) from city A to city B ? *Straight line distance*
- 8-puzzle problem ? *# of tiles misplaced*
- M&C ? *# of steps to bring all over to RITS* / *# of moves in "swap"s*
- Robot navigation ? *# of ppl on the UTS*

- **Admissible** heuristic

- Does not overestimate the cost to reach the goal
- "Optimistic"

- **Consistent** heuristic:

- Satisfy a triangular inequality:  $h(n) \leq c(n, a, n') + h(n')$

$h(n) - h(n')$

$\leq |h(n) - h(n')|$

$\leq \underline{I = c(n, a, n')}$

$h(n) = h(n') + c$

then  $h(n) \leq h(n') + c$  if  $h(n) \leq h(n') + c$

# Heuristics

- What's a heuristic for
  - Driving distance (or time) from city A to city B ?
  - 8-puzzle problem ?
  - M&C ?
  - Robot navigation ?
- **Admissible** heuristic
  - Does not overestimate the cost to reach the goal
  - “Optimistic”
- **Consistent** heuristic:
  - Satisfy a triangular inequality:  $h(n) \leq c(n,a,n') + h(n')$
- Are the above heuristics admissible? Consistent?

# Example: 8-Puzzle

|   |   |   |
|---|---|---|
| 5 | 4 |   |
| 6 | 1 | 8 |
| 7 | 3 | 2 |

Start State

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 8 |   | 4 |
| 7 | 6 | 5 |

Goal State

# Comparing and combining heuristics

- Heuristics generated by considering relaxed versions of a problem.
- Heuristic  $h_1$  for 8-puzzle
  - Number of out-of-order tiles
- Heuristic  $h_2$  for 8-puzzle
  - Sum of Manhattan distances of each tile *(excluding space)*
- $h_2$  dominates  $h_1$  provided  $h_2(n) \geq h_1(n)$ .
  - $h_2$  will likely prune more than  $h_1$ .
- $\max(h_1, h_2, \dots, h_n)$  is
  - admissible if each  $h_i$  is
  - consistent if each  $h_i$  is
- Cost of sub-problems and pattern databases
  - Cost for 4 specific tiles
  - Can these be added for disjoint sets of tiles?

|   |   |   |
|---|---|---|
| 7 | 8 |   |
| 0 | 1 | 3 |
| 4 | 2 | 5 |

|   |   |   |
|---|---|---|
| 1 | 2 | 3 |
| 8 | 0 | 4 |
| 7 | 6 | 5 |

# Effective Branching Factor

# Effective Branching Factor

- Though informed search methods may have poor *worst-case* performance, they often do quite well if the heuristic is good
  - Even if there is a huge branching factor



# Effective Branching Factor

- Though informed search methods may have poor *worst-case* performance, they often do quite well if the heuristic is good
  - Even if there is a huge branching factor
- One way to quantify the effectiveness of the heuristic: the **effective branching factor,  $b^*$** 
  - N: total number of nodes expanded
  - d: solution depth
  - $N = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$



# Effective Branching Factor

- Though informed search methods may have poor *worst-case* performance, they often do quite well if the heuristic is good
  - Even if there is a huge branching factor
- One way to quantify the effectiveness of the heuristic: the **effective branching factor,  $b^*$** 
  - N: total number of nodes expanded
  - d: solution depth
  - $N = 1 + b^* + (b^*)^2 + \dots + (b^*)^d$
- For a good heuristic,  $b^*$  is close to 1

# Example: 8-puzzle problem

Averaged over 100 trials each at different solution lengths

| $d$ | Search Cost |            |            | Effective Branching Factor |            |            |
|-----|-------------|------------|------------|----------------------------|------------|------------|
|     | IDS         | $A^*(h_1)$ | $A^*(h_2)$ | IDS                        | $A^*(h_1)$ | $A^*(h_2)$ |
| 2   | 10          | 6          | 6          | 2.45                       | 1.79       | 1.79       |
| 4   | 112         | 13         | 12         | 2.87                       | 1.48       | 1.45       |
| 6   | 680         | 20         | 18         | 2.73                       | 1.34       | 1.30       |
| 8   | 6384        | 39         | 25         | 2.80                       | 1.33       | 1.24       |
| 10  | 47127       | 93         | 39         | 2.79                       | 1.38       | 1.22       |
| 12  | 364404      | 227        | 73         | 2.78                       | 1.42       | 1.24       |
| 14  | 3473941     | 539        | 113        | 2.83                       | 1.44       | 1.23       |
| 16  | –           | 1301       | 211        | –                          | 1.45       | 1.25       |
| 18  | –           | 3056       | 363        | –                          | 1.46       | 1.26       |
| 20  | –           | 7276       | 676        | –                          | 1.47       | 1.27       |
| 22  | –           | 18094      | 1219       | –                          | 1.48       | 1.28       |
| 24  | –           | 39135      | 1641       | –                          | 1.48       | 1.26       |

Ave. # of nodes expanded

Solution length

# Memory Bounded Search

# Memory Bounded Search

- Memory, not computation, is usually the limiting factor in search problems
  - Certainly true for A\* search

# Memory Bounded Search

- Memory, not computation, is usually the limiting factor in search problems
  - Certainly true for A\* search
- Why? What takes up memory in A\* search?

# Memory Bounded Search

- Memory, not computation, is usually the limiting factor in search problems
  - Certainly true for A\* search
- Why? What takes up memory in A\* search?
- Solution: Memory-bounded A\* search
  - Iterative Deepening A\* (IDA\*)
  - Simplified Memory-bounded A\* (SMA\*)
  - (Read the textbook for more details.)

*Beam search*

# Summary of informed search

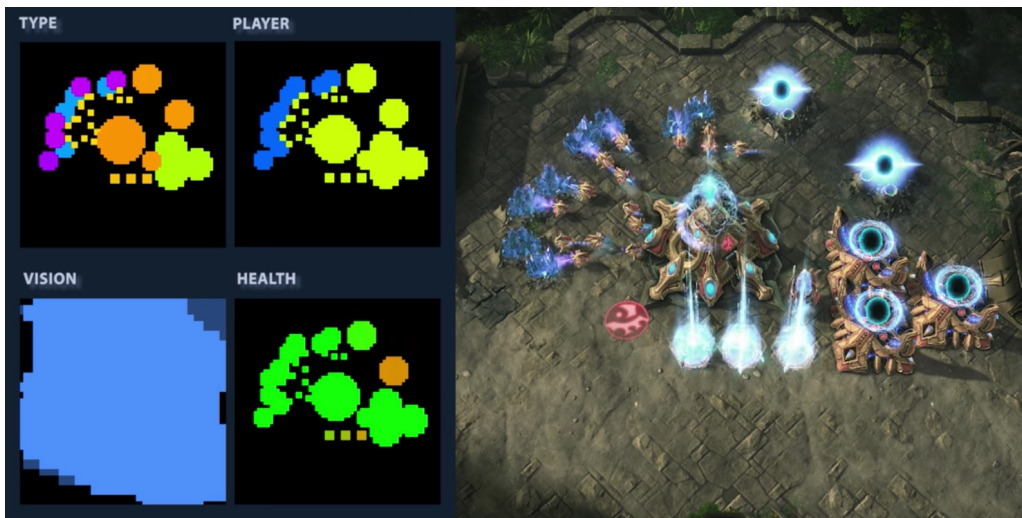
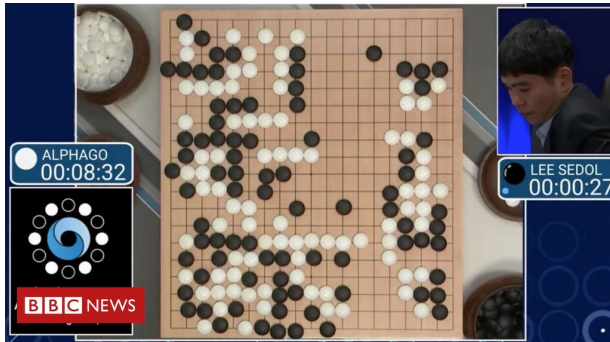
- How to use a heuristic function to improve search
  - Greedy Best-first search + Uniform-cost search = A\* Search
- When is A\* search optimal?
  - h is Admissible (optimistic) for Tree Search
  - h is Consistent for Graph Search
- Choosing heuristic functions
  - A good heuristic function can reduce time/space cost of search by orders of magnitude.
  - Good heuristic function may take longer to evaluate.



# Games and Adversarial Search



- Games: problem setup
- Minimax search
- Alpha-beta pruning

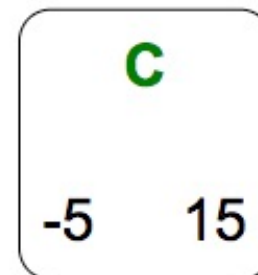
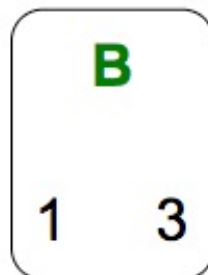
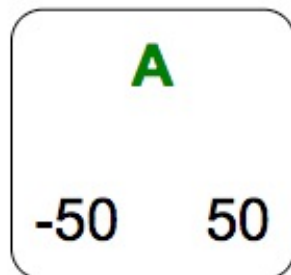


# Illustrative example of a simple game (1 min discussion)



## Example: game 1

You choose one of the three bins.  
I choose a number from that bin.  
Your goal is to maximize the chosen number.



$$E[V]$$
$$=$$
$$(-5) \cdot 0.5$$
$$+ 15 \cdot 0.5$$

(Example taken from Liang and Sadigh)

# Game as a search problem

# Game as a search problem

- $S_0$  The initial state

# Game as a search problem

- $S_0$  The initial state
- $\text{PLAYER}(s)$ : Returns which player has the move

# Game as a search problem

- $S_0$  The initial state
- $PLAYER(s)$ : Returns which player has the move
- $ACTIONS(s)$ : Returns the legal moves.

# Game as a search problem

- $S_0$  The initial state
- $PLAYER(s)$ : Returns which player has the move
- $ACTIONS(s)$ : Returns the legal moves.
- $RESULT(s, a)$ : Output the state we transition to.

# Game as a search problem

- $S_0$  The initial state
- $PLAYER(s)$ : Returns which player has the move
- $ACTIONS(s)$ : Returns the legal moves.
- $RESULT(s, a)$ : Output the state we transition to.
- $TERMINAL-TEST(s)$ : Returns True if the game is over.



# Game as a search problem

- $S_0$  The initial state
- $\text{PLAYER}(s)$ : Returns which player has the move
- $\text{ACTIONS}(s)$ : Returns the legal moves.
- $\text{RESULT}(s, a)$ : Output the state we transition to.
- $\text{TERMINAL-TEST}(s)$ : Returns True if the game is over.
- $\text{UTILITY}(s, p)$ : The payoff of player  $p$  at terminal state  $s$ .



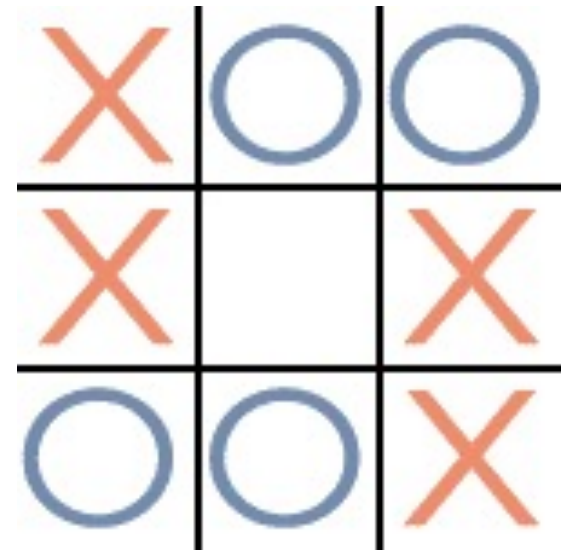
|       |     |       |
|-------|-----|-------|
|       | $s$ | $s$   |
| $p_1$ | 1   | 0     |
| $p_2$ | 0   | 1     |
| <hr/> |     | 1 1 ← |

# Two-player, Turn-based, Perfect information, Deterministic, Zero-Sum Game

- Two-player: Tic-Tac-Toe, Chess, Go!
- Turn-based: The players take turns in round-robin fashion.
- Perfect information: The State is known to everyone
- Deterministic: Nothing is random
- Zero-sum: The total payoff for all players is a **constant**.
  - *The 8-puzzle is a one-player, perfect info, deterministic, zero-sum game.*
  - *How about Rock-Paper-Scissors?*
  - *How about Monopoly?*
  - *How about Starcraft?*

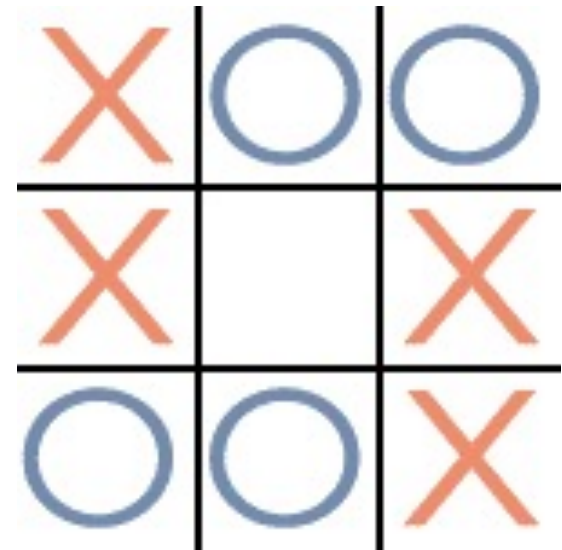
# Tic-Tac-Toe

- The first player is **X** and the second is **O**
- Object of game: get three of your symbol in a horizontal, vertical or diagonal row on a 3x3 game board
- **X** always goes first
- Players alternate placing **Xs** and **O**s on the game board
- Game ends when a player has three in a row (a wins) or all nine squares are filled (a draw)



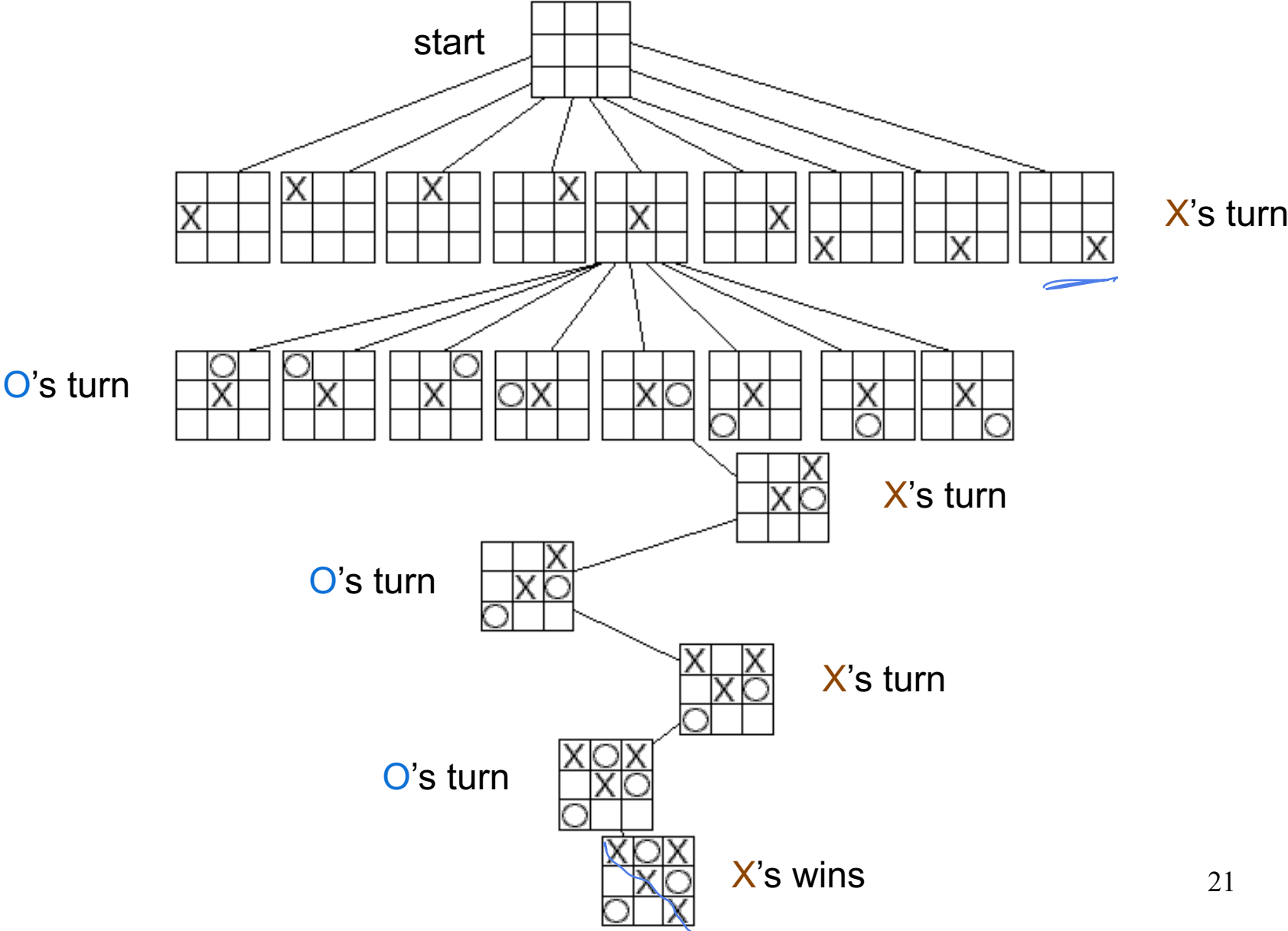
# Tic-Tac-Toe

- The first player is **X** and the second is **O**
- Object of game: get three of your symbol in a horizontal, vertical or diagonal row on a 3x3 game board
- **X** always goes first
- Players alternate placing **Xs** and **O**s on the game board
- Game ends when a player has three in a row (a wins) or all nine squares are filled (a draw)



What's the state, action, transition, payoff for Tic-Tac-Toe?

# Partial game tree for Tic-Tac-Toe



# Game trees

- A game tree is like a search tree in many ways ...
  - nodes are search states, with full details about a position
    - characterize the arrangement of game pieces on the game board
  - edges between nodes correspond to moves
  - leaf nodes correspond to a set of goals
    - { win, lose, draw }
    - usually determined by a score for or against player
  - at each node it is one or other player's turn to move
- A game tree is not like a search tree because you have an opponent!

# Two players: MIN and MAX

- In a zero-sum game:
  - payoff to Player 1 = - payoff to Player 2
- The goal of Player 1 is to maximizing her payoff.
- The goal of Player 2 is to maximizing her payoff as well
  - Equivalent to minimizing Player 1's payoff.

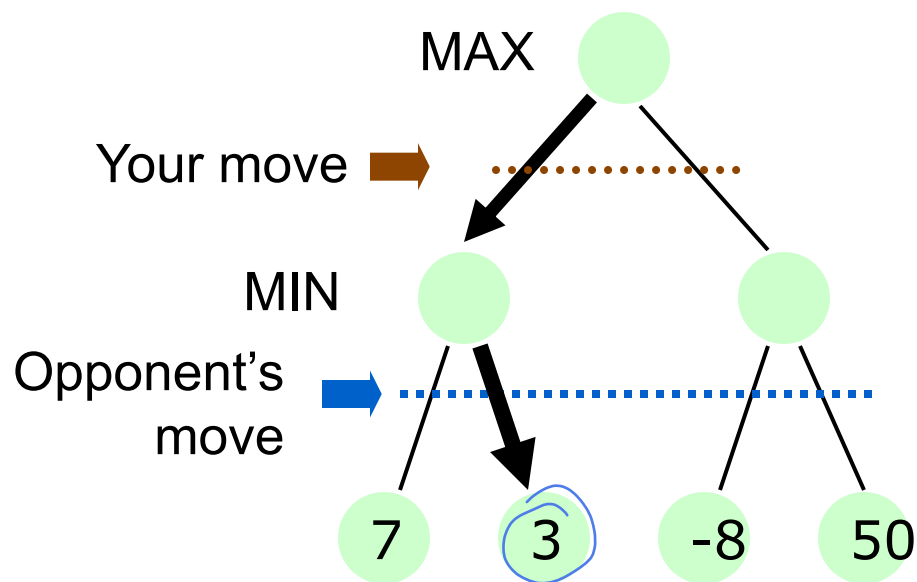
$$\text{max reward} = \text{min}(-\text{reward}) = \text{min}(\text{reward of your opponent})$$

# Minimax search

- Assume that both players play perfectly
  - do not assume player will miss good moves or make mistakes
- Score(s): The score that MAX will get towards the end if both player play perfectly from s onwards.
- Consider MIN's strategy
  - MIN's best strategy:
    - choose the move that **minimizes** the score that will result when MAX chooses the **maximizing** move
  - MAX does the opposite

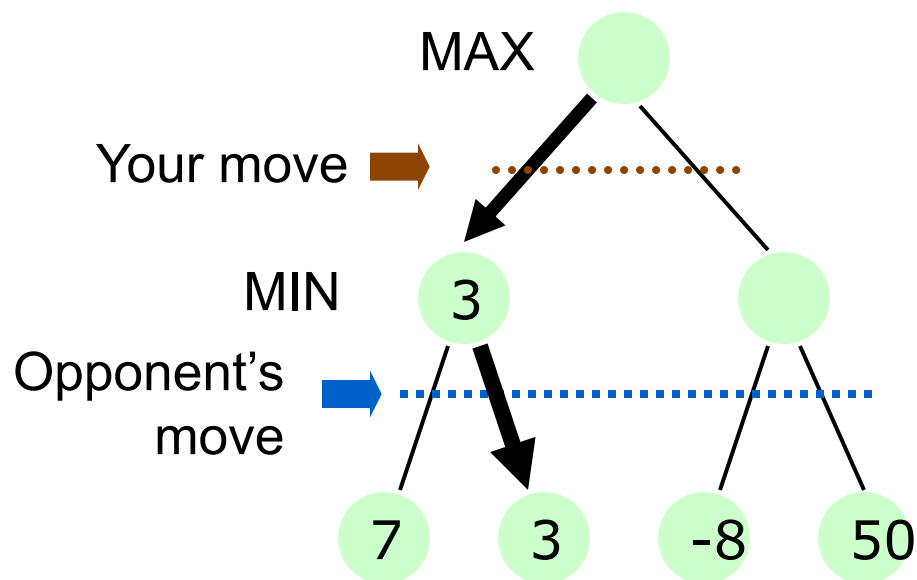


# Minimaxing



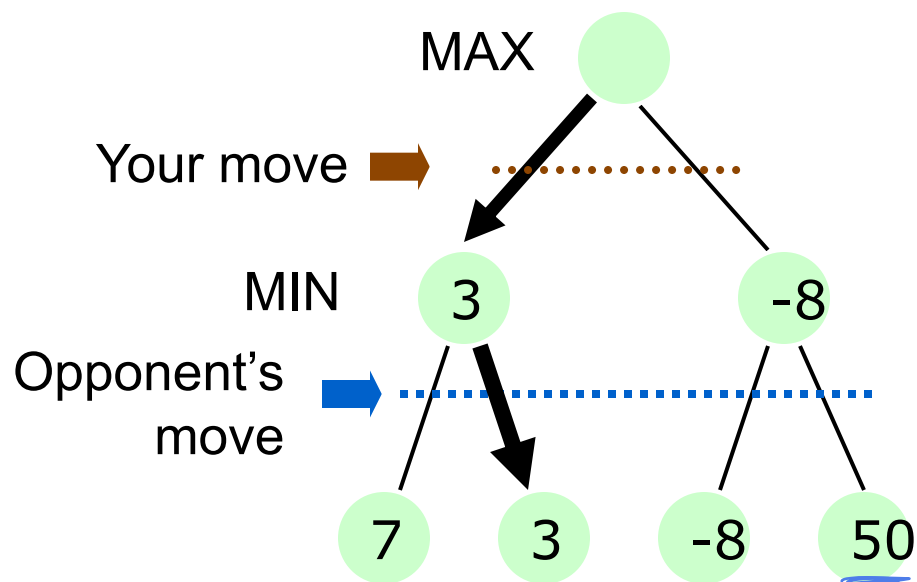
- Your opponent will choose smaller numbers
- If you move left, your opponent will choose 3
- If you move right, your opponent will choose -8
- Thus your choices are only 3 or -8
- You should move left

# Minimaxing



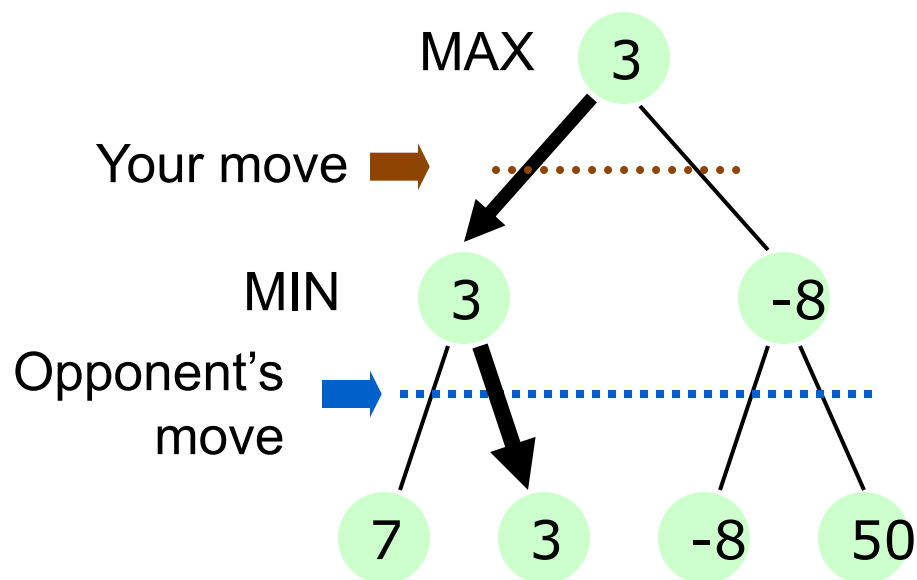
- Your opponent will choose smaller numbers
- If you move left, your opponent will choose 3
- If you move right, your opponent will choose -8
- Thus your choices are only 3 or -8
- You should move left

# Minimaxing



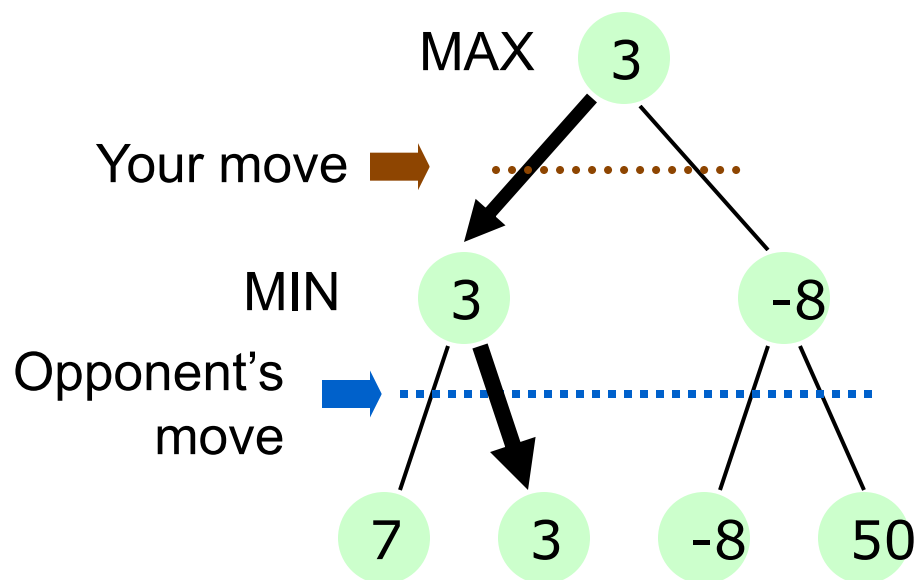
- Your opponent will choose smaller numbers
- If you move left, your opponent will choose 3
- If you move right, your opponent will choose -8
- Thus your choices are only 3 or -8
- You should move left

# Minimaxing



- Your opponent will choose smaller numbers
- If you move left, your opponent will choose 3
- If you move right, your opponent will choose -8
- Thus your choices are only 3 or -8
- You should move left

# Minimaxing

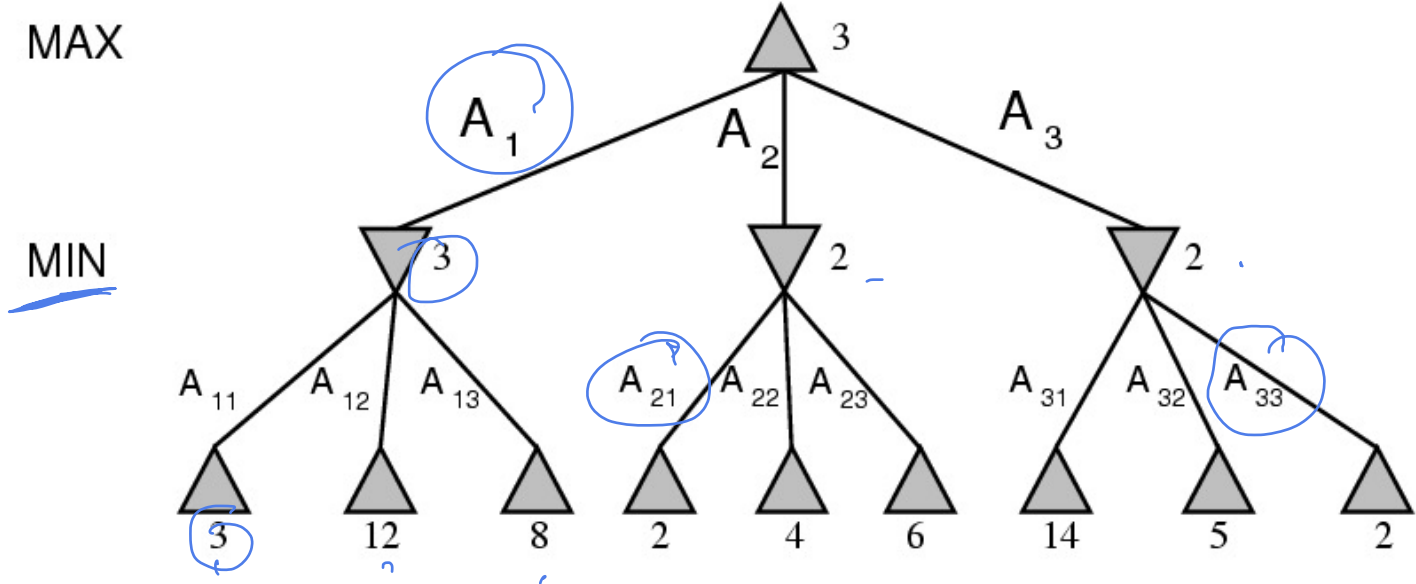


- Your opponent will choose smaller numbers
- If you move left, your opponent will choose 3
- If you move right, your opponent will choose -8
- Thus your choices are only 3 or -8
- You should move left

Each move is called a “ply”. One round is K-ply for a K-player game.

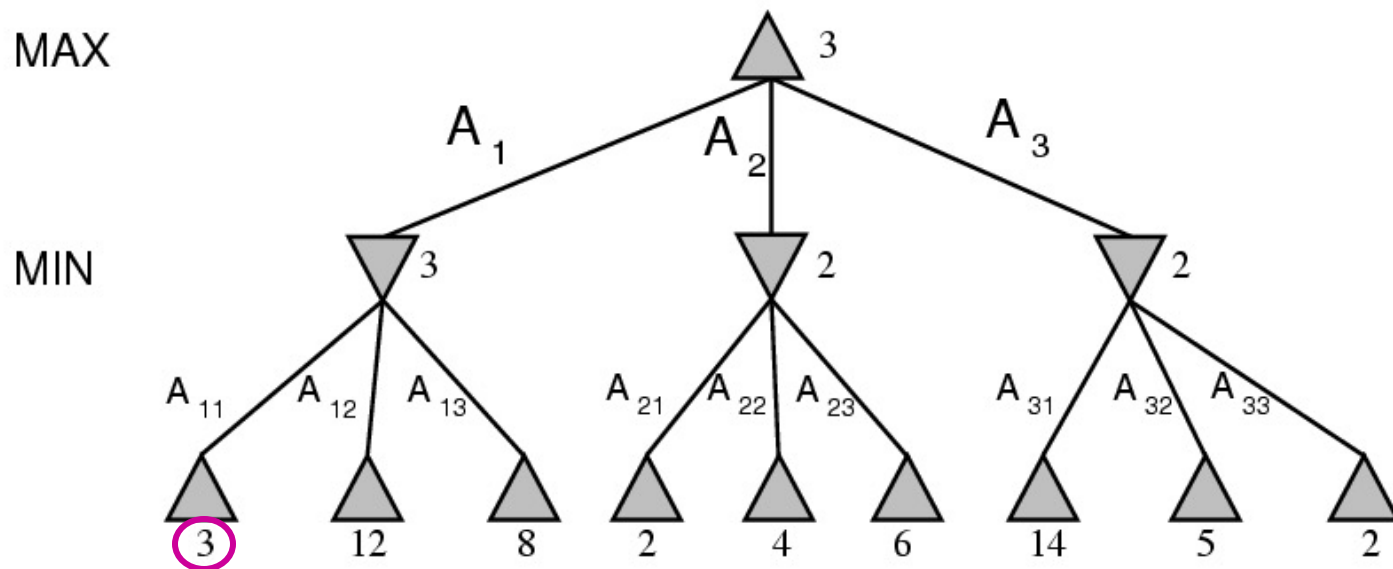
# Minimax example

Which move to choose?



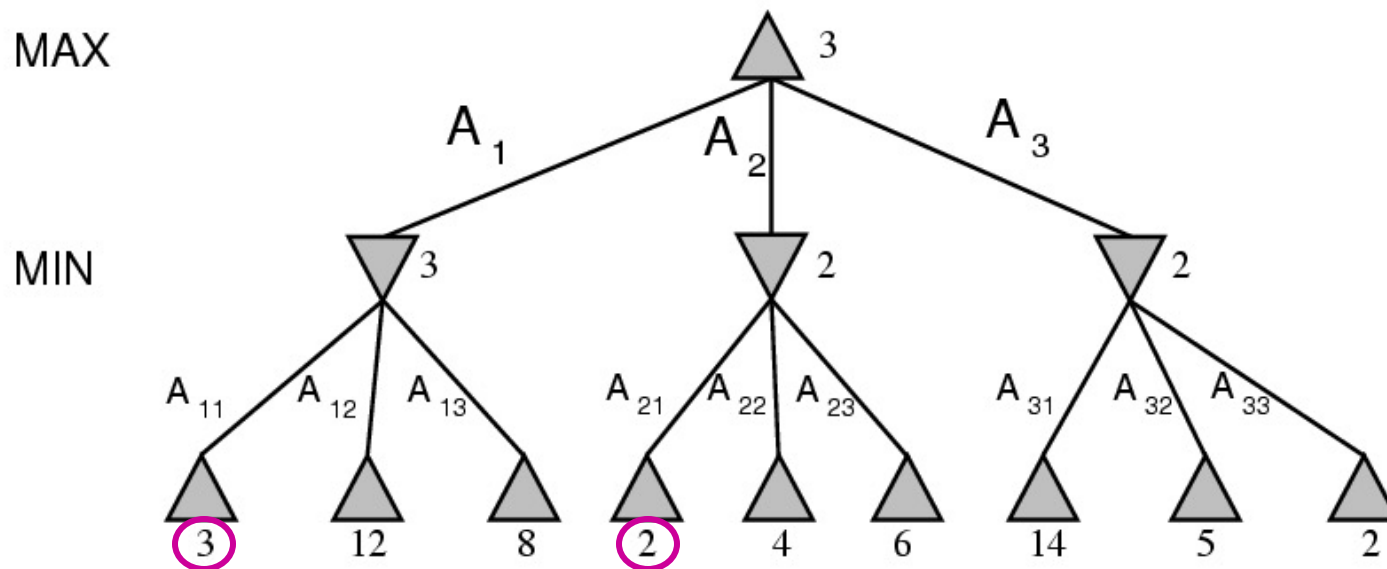
# Minimax example

Which move to choose?



# Minimax example

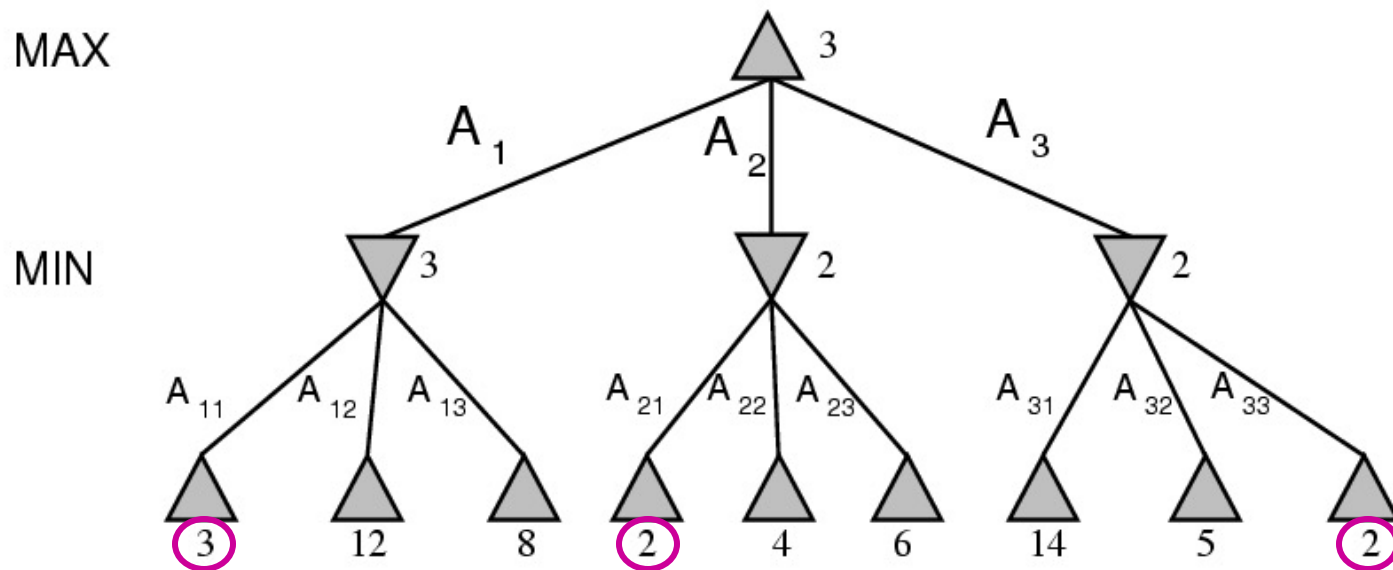
Which move to choose?





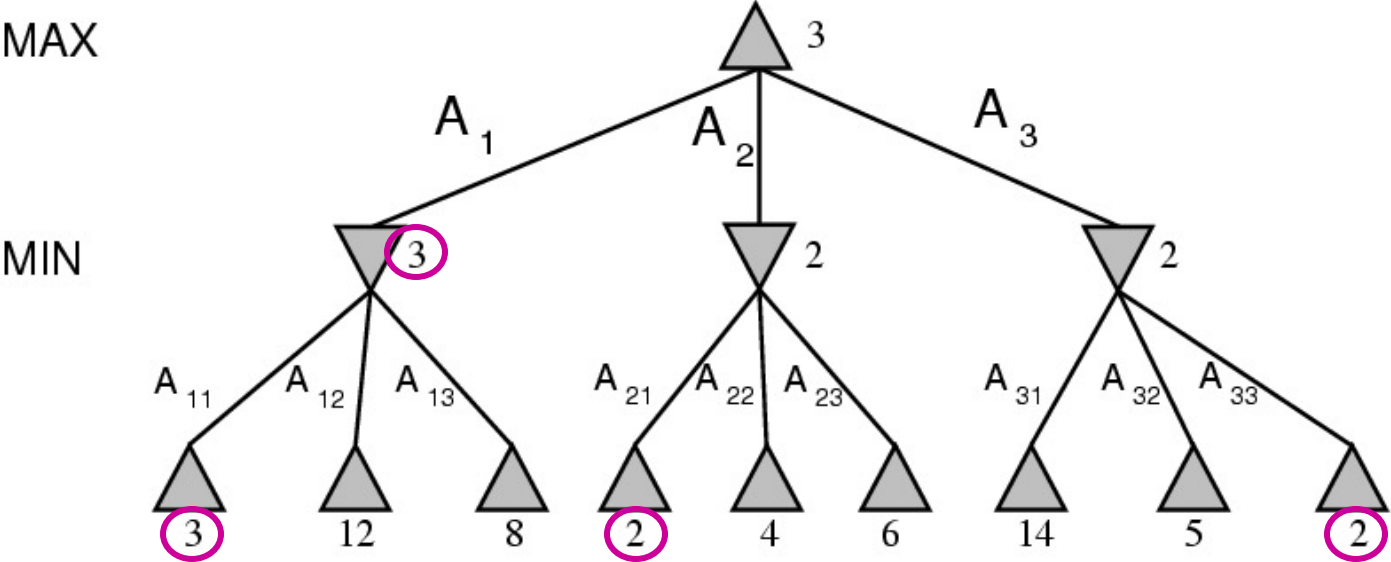
# Minimax example

Which move to choose?



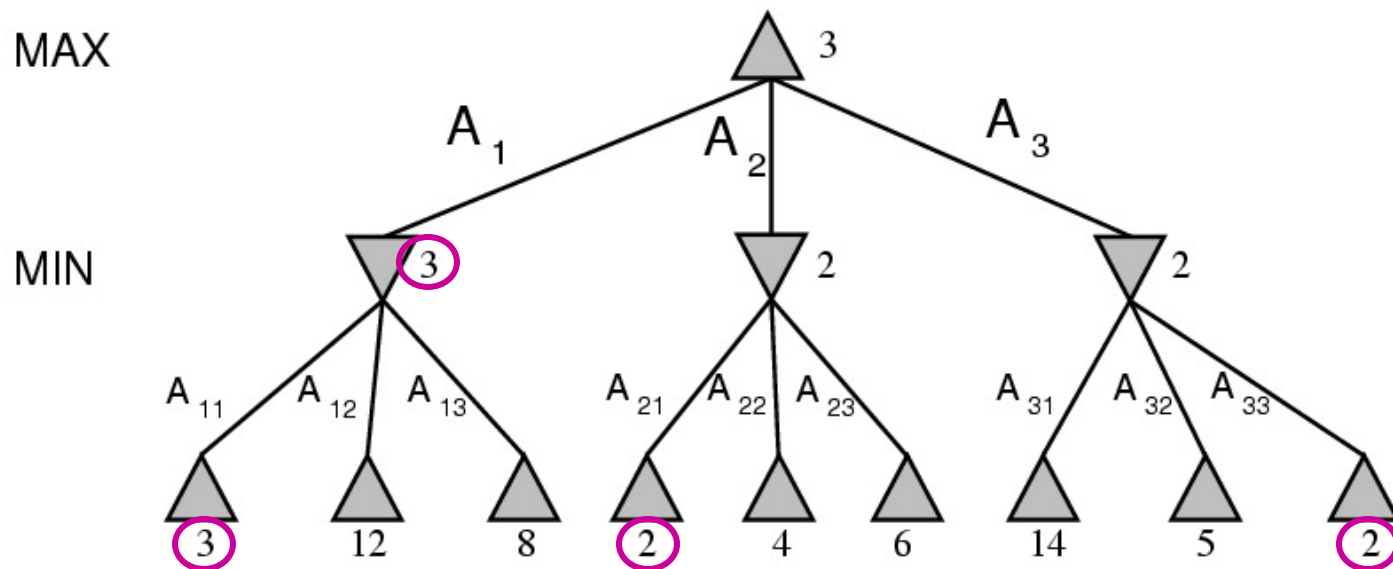
# Minimax example

Which move to choose?



# Minimax example

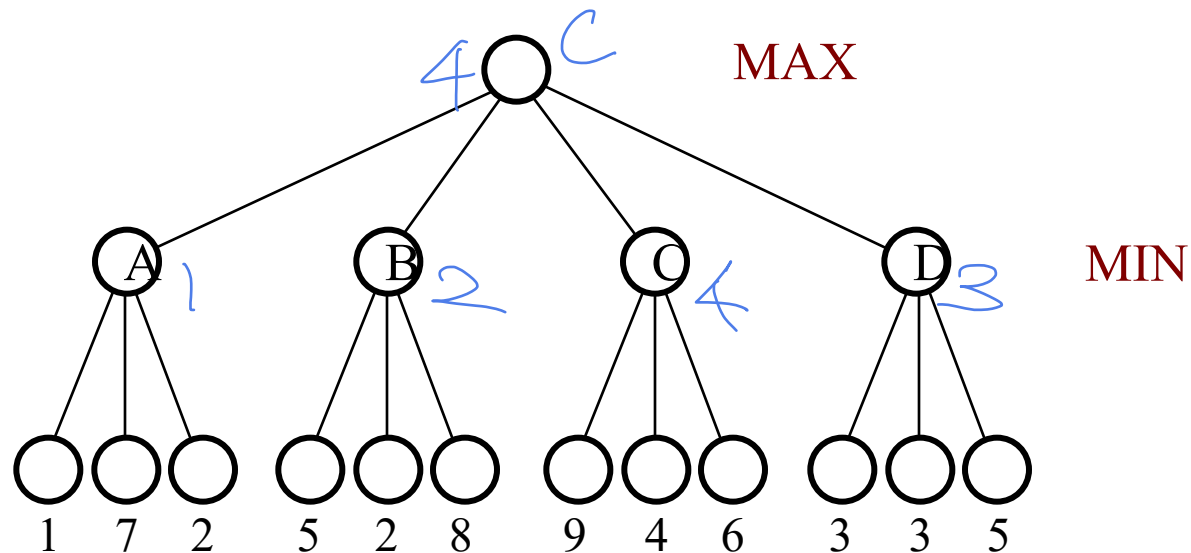
Which move to choose?



The **minimax decision** is move  $A_1$

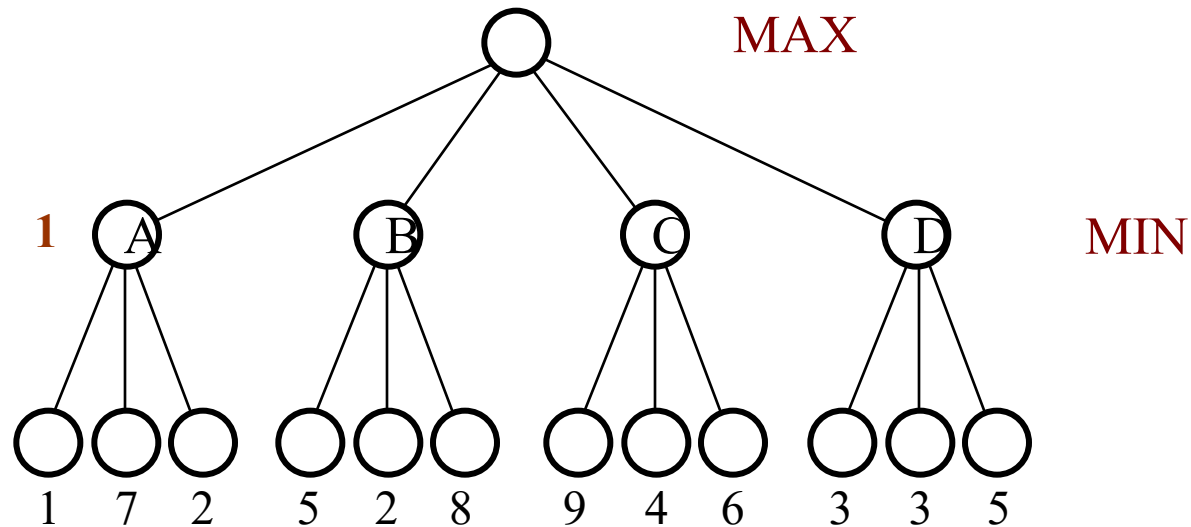
# Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



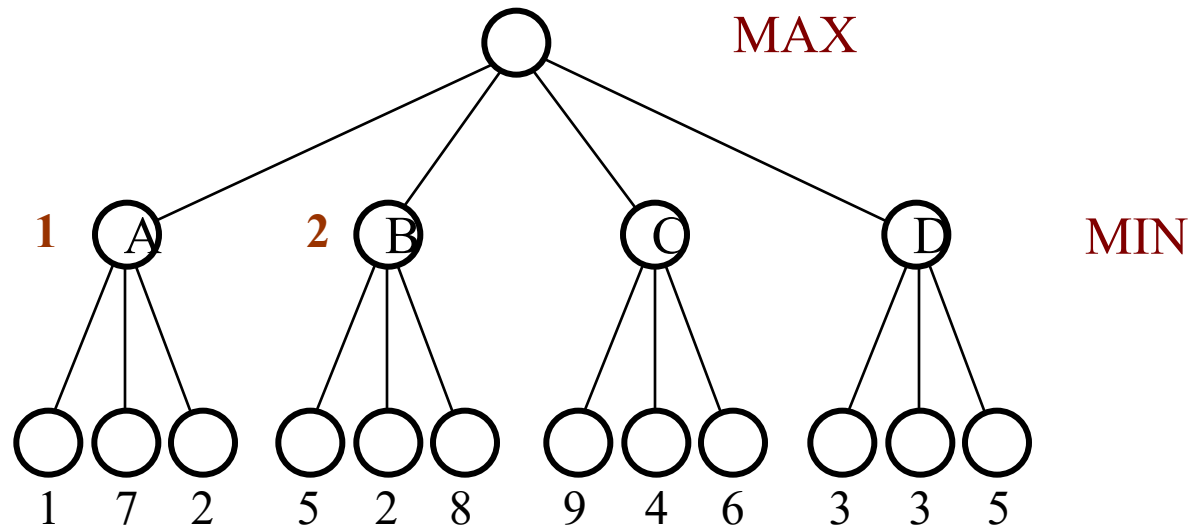
# Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



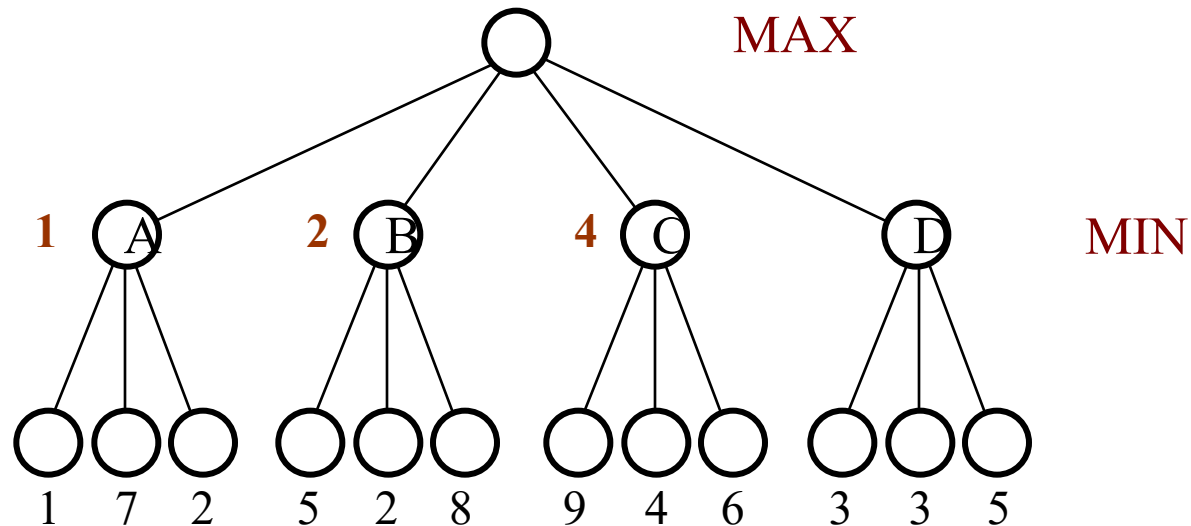
# Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



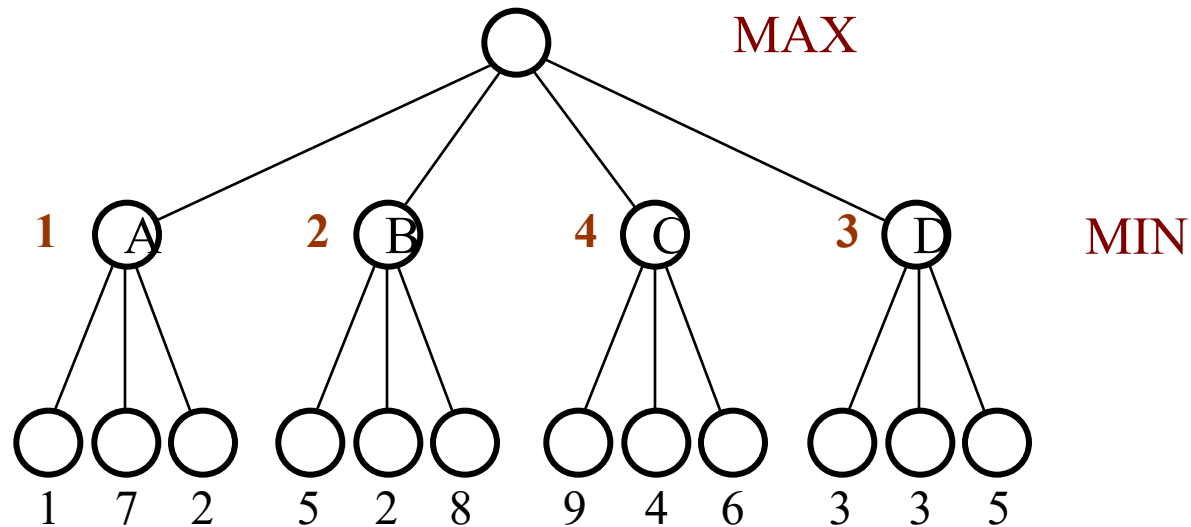
# Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



# Another example

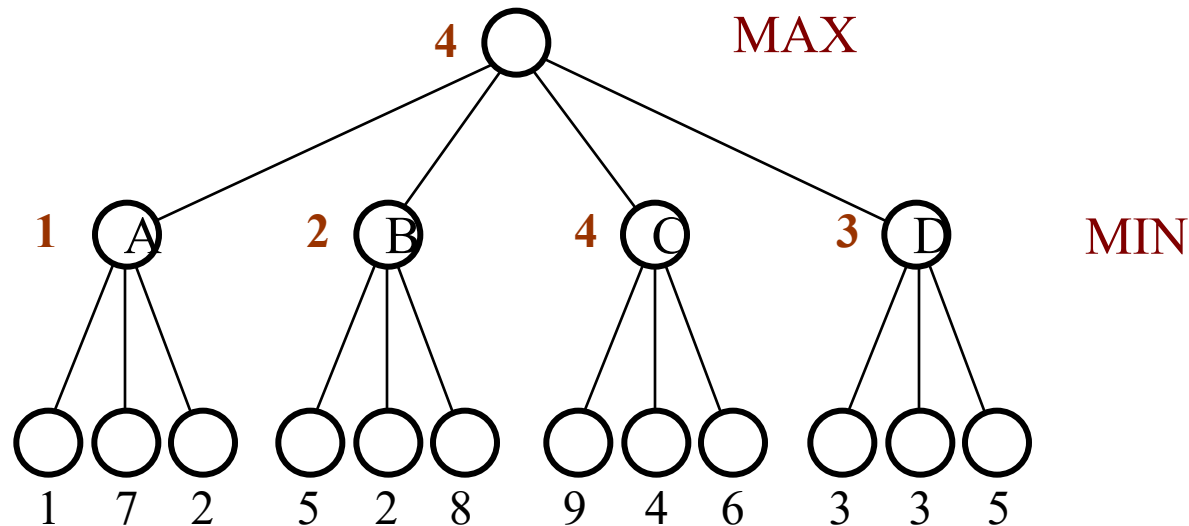
- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?





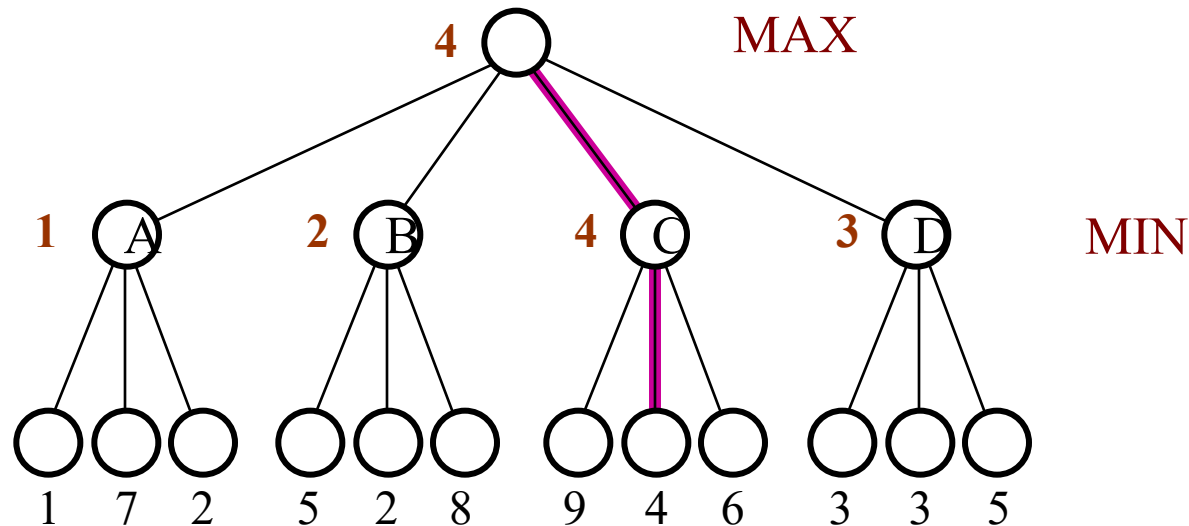
# Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



# Another example

- In the game, it's your move. Which move will the minimax algorithm choose – A, B, C, or D? What is the minimax value of the root node and nodes A, B, C, and D?



# Minimax search

# Minimax search

- The *minimax decision* maximizes the utility under the assumption that the opponent seeks to minimize it (if it uses the same evaluation function)

# Minimax search

- The *minimax decision* maximizes the utility under the assumption that the opponent seeks to minimize it (if it uses the same evaluation function)
- Generate the tree of minimax values
  - Then choose best (maximum) move
  - Don't need to keep all values around
    - Good memory property

# Minimax search

- The *minimax decision* maximizes the utility under the assumption that the opponent seeks to minimize it (if it uses the same evaluation function)
- Generate the tree of minimax values
  - Then choose best (maximum) move
  - Don't need to keep all values around
    - Good memory property
- Depth-first search is used to implement minimax
  - Expand all the way down to leaf nodes
  - Recursive implementation

# Minimax properties

- Optimal? Yes, against an optimal opponent, **if** the tree is finite
- Complete? Yes, **if** the tree is finite
- Time complexity? Exponential:  **$O(b^m)$**
- Space complexity? Polynomial:  **$O(bm)$**

# But this could take forever...

- Exact search is intractable
  - Tic-Tac-Toe is  $9! = 362,880$
  - For chess,  $b \approx 35$  and  $m \approx 100$  for “reasonable” games
  - Go is  $361! \approx 10^{750}$



# But this could take forever...

- Exact search is intractable
  - Tic-Tac-Toe is  $9! = 362,880$
  - For chess,  $b \approx 35$  and  $m \approx 100$  for “reasonable” games
  - Go is  $361! \approx 10^{750}$
- Idea 1: Pruning

# But this could take forever...

- Exact search is intractable
  - Tic-Tac-Toe is  $9! = 362,880$
  - For chess,  $b \approx 35$  and  $m \approx 100$  for “reasonable” games
  - Go is  $361! \approx 10^{750}$
- Idea 1: Pruning
- Idea 2: Cut off early and use a heuristic function

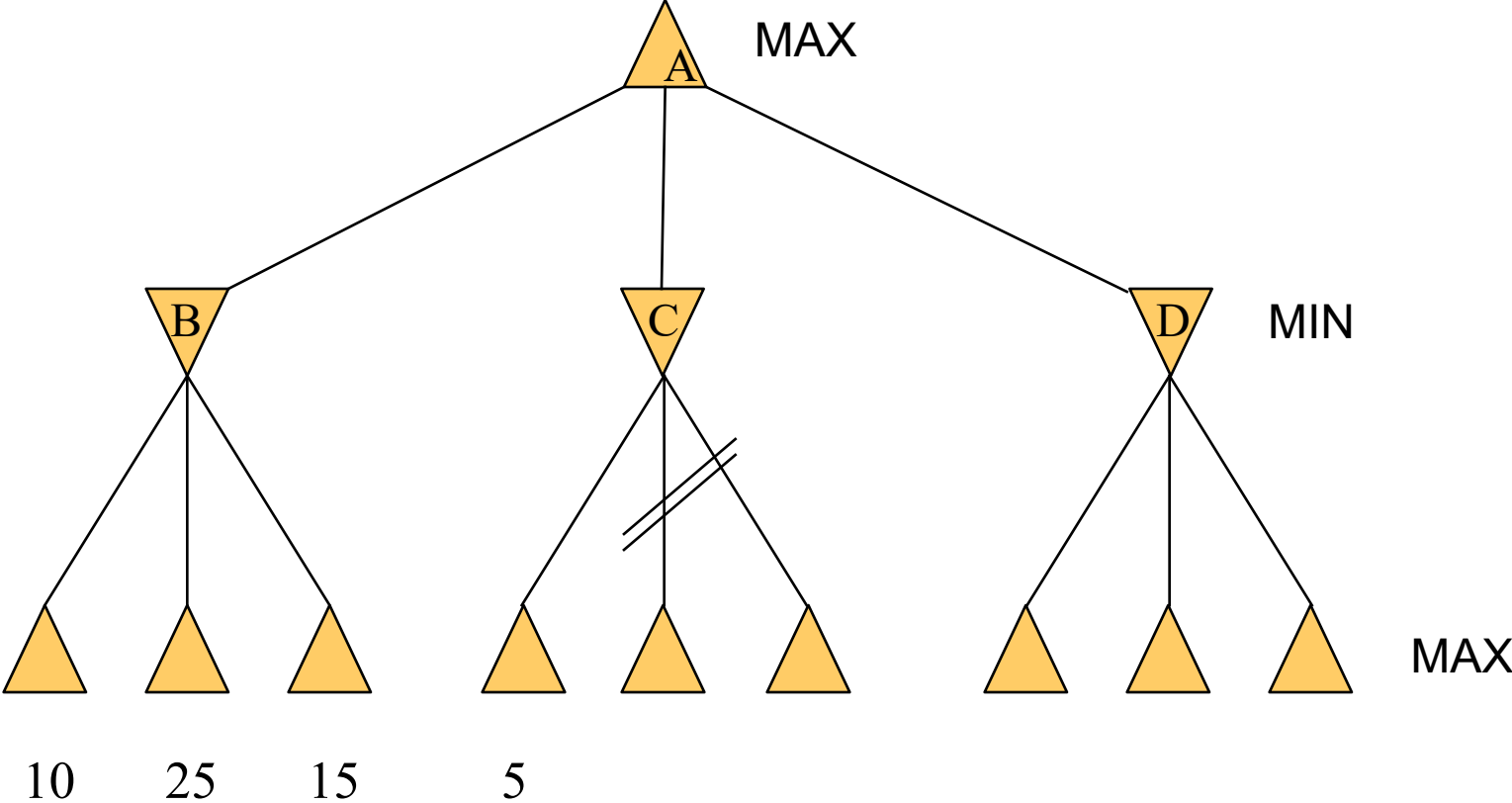
# Pruning

- What's really needed is “smarter,” more efficient search
  - Don't expand “dead-end” nodes!
- **Pruning** – eliminating a branch of the search tree from consideration

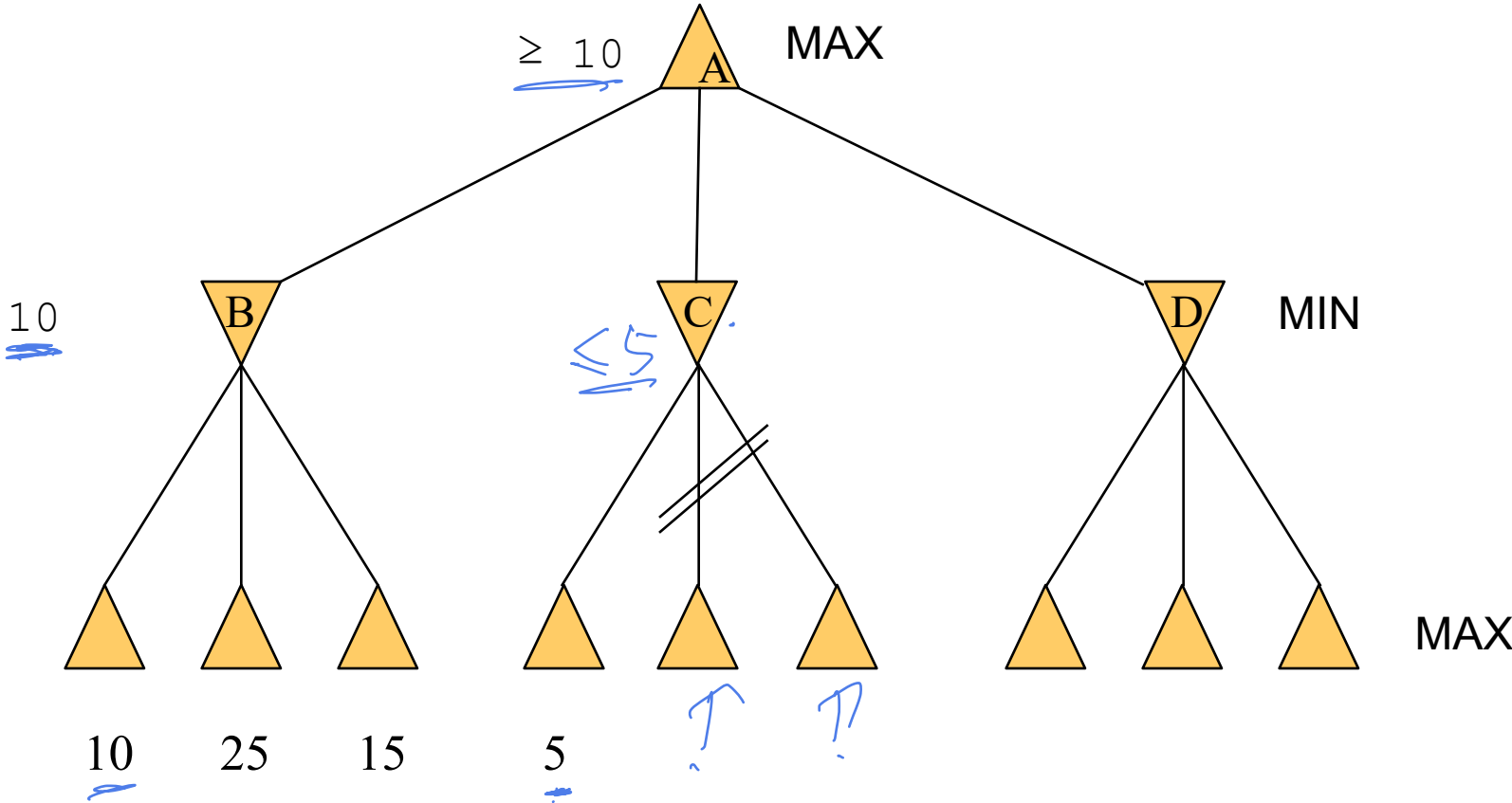
# Pruning

- What's really needed is “smarter,” more efficient search
  - Don't expand “dead-end” nodes!
- **Pruning** – eliminating a branch of the search tree from consideration
- **Alpha-beta pruning**, applied to a minimax tree, returns the same “best” move, while pruning away unnecessary branches
  - Many fewer nodes might be expanded
  - Hence, smaller effective branching factor
  - ...and deeper search
  - ...and better performance
    - Remember, minimax is *depth-first* search

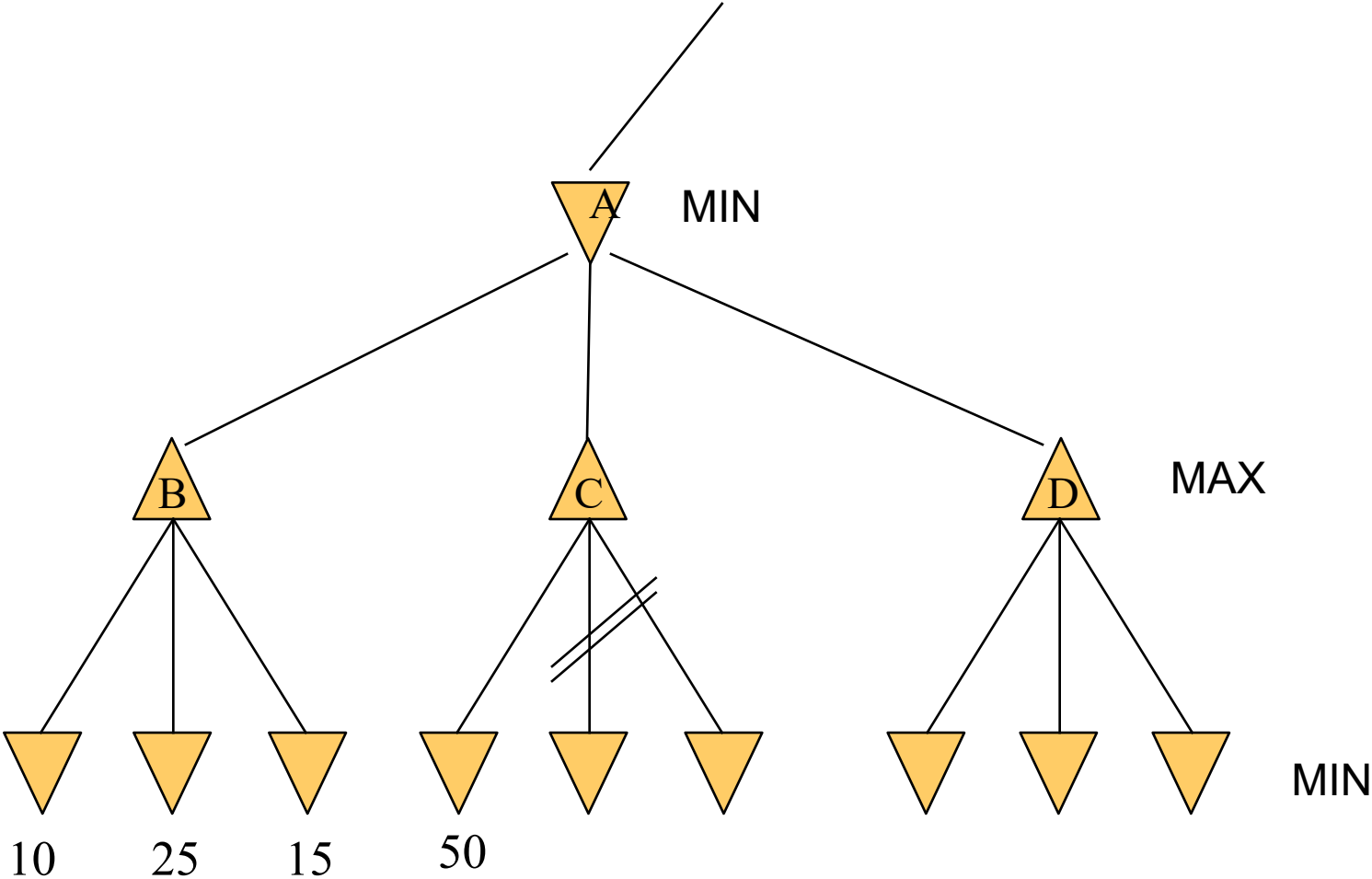
# Alpha pruning



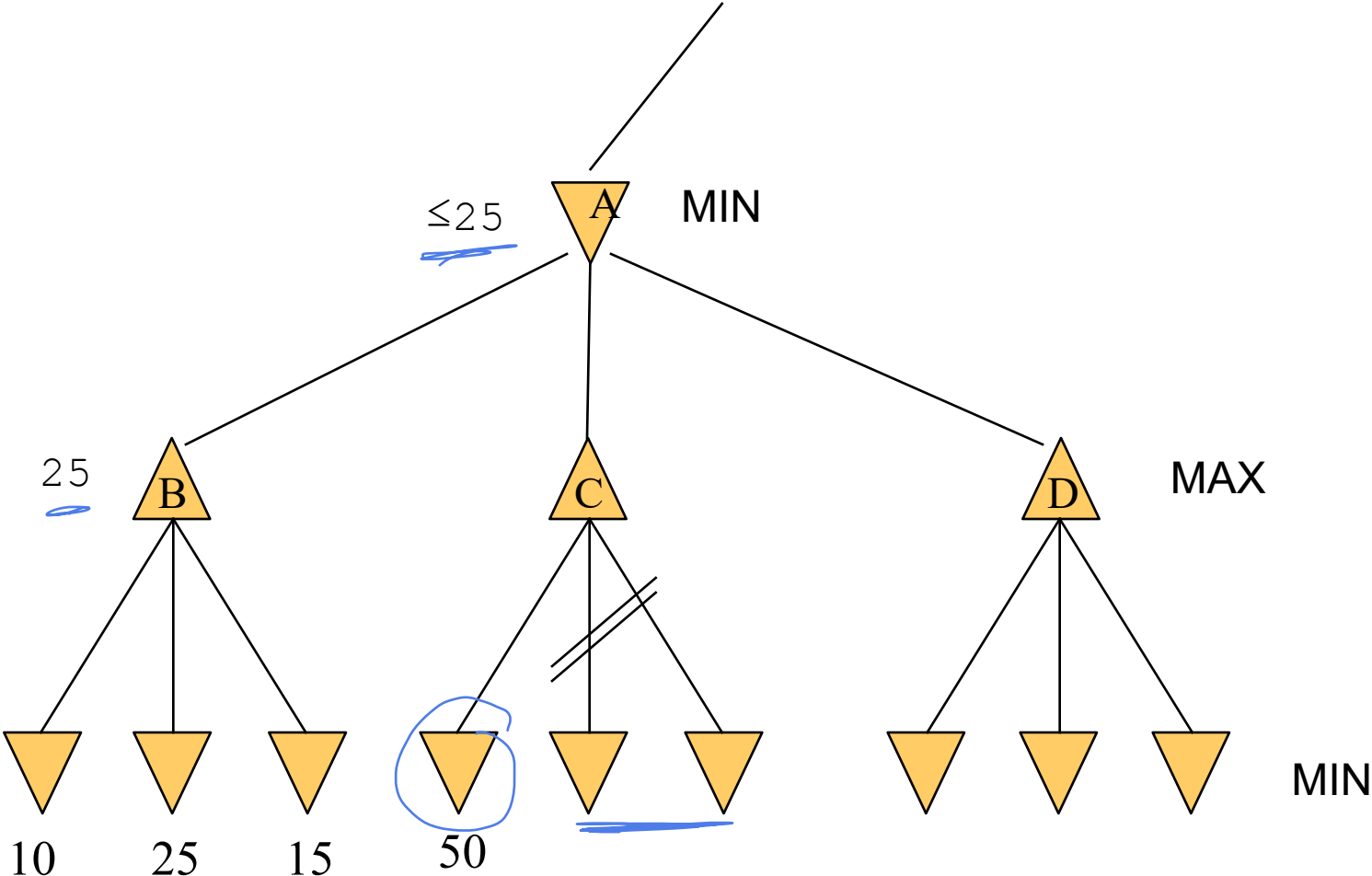
# Alpha pruning



# Beta pruning



# Beta pruning





# Improvements via alpha/beta pruning

# Improvements via alpha/beta pruning

- Depends on the ordering of expansion

# Improvements via alpha/beta pruning

- Depends on the ordering of expansion
- Perfect ordering  $O(b^{m/2})$

# Improvements via alpha/beta pruning

- Depends on the ordering of expansion
- Perfect ordering  $O(b^{m/2})$
- Random ordering  $O(b^{3m/4})$

# Improvements via alpha/beta pruning

- Depends on the ordering of expansion
- Perfect ordering  $O(b^{m/2})$
- Random ordering  $O(b^{3m/4})$
- For specific games like Chess, you can get to almost perfect ordering.

# Heuristic (Evaluation function)

- It is usually impossible to solve games completely
- Rather, cut the search off early and apply a heuristic evaluation function to the leaves
  - $h(s)$  estimates the expected utility of the game from a given position (node/state)  $s$
  - like depth bounded depth first, lose completeness
  - Explore game tree using combination of evaluation function and search

# Heuristic (Evaluation function)

- It is usually impossible to solve games completely
- Rather, cut the search off early and apply a heuristic evaluation function to the leaves
  - $h(s)$  estimates the expected utility of the game from a given position (node/state)  $s$
  - like depth bounded depth first, lose completeness
  - Explore game tree using combination of evaluation function and search
- The performance of a game-playing program depends on the quality (and speed!) of its evaluation function

# Heuristics (Evaluation function)

- Typical evaluation function for game: weighted linear function
  - $h(s) = w_1f_1(s) + w_2f_2(s) + \dots + w_df_d(s)$
  - *weights* • *features* [dot product]



# Heuristics (Evaluation function)

- Typical evaluation function for game: weighted linear function
  - $h(s) = w_1f_1(s) + w_2f_2(s) + \dots + w_df_d(s)$
  - *weights* · *features* [dot product]
- For example, in chess
  - $W = \{ \underline{1}, \underline{3}, \underline{3}, \underline{5}, \underline{8} \}$
  - $F = \{ \# \text{ pawns advantage, } \# \text{ bishops advantage, } \# \text{ knights advantage, } \# \text{ rooks advantage, } \# \text{ queens advantage} \}$

# Heuristics (Evaluation function)

- Typical evaluation function for game: weighted linear function
  - $h(s) = w_1f_1(s) + w_2f_2(s) + \dots + w_df_d(s)$
  - *weights* · *features* [dot product]
- For example, in chess
  - $W = \{ 1, 3, 3, 5, 8 \}$
  - $F = \{ \# \text{ pawns advantage, } \# \text{ bishops advantage, } \# \text{ knights advantage, } \# \text{ rooks advantage, } \# \text{ queens advantage} \}$
  - Is this what Deep Blue used?

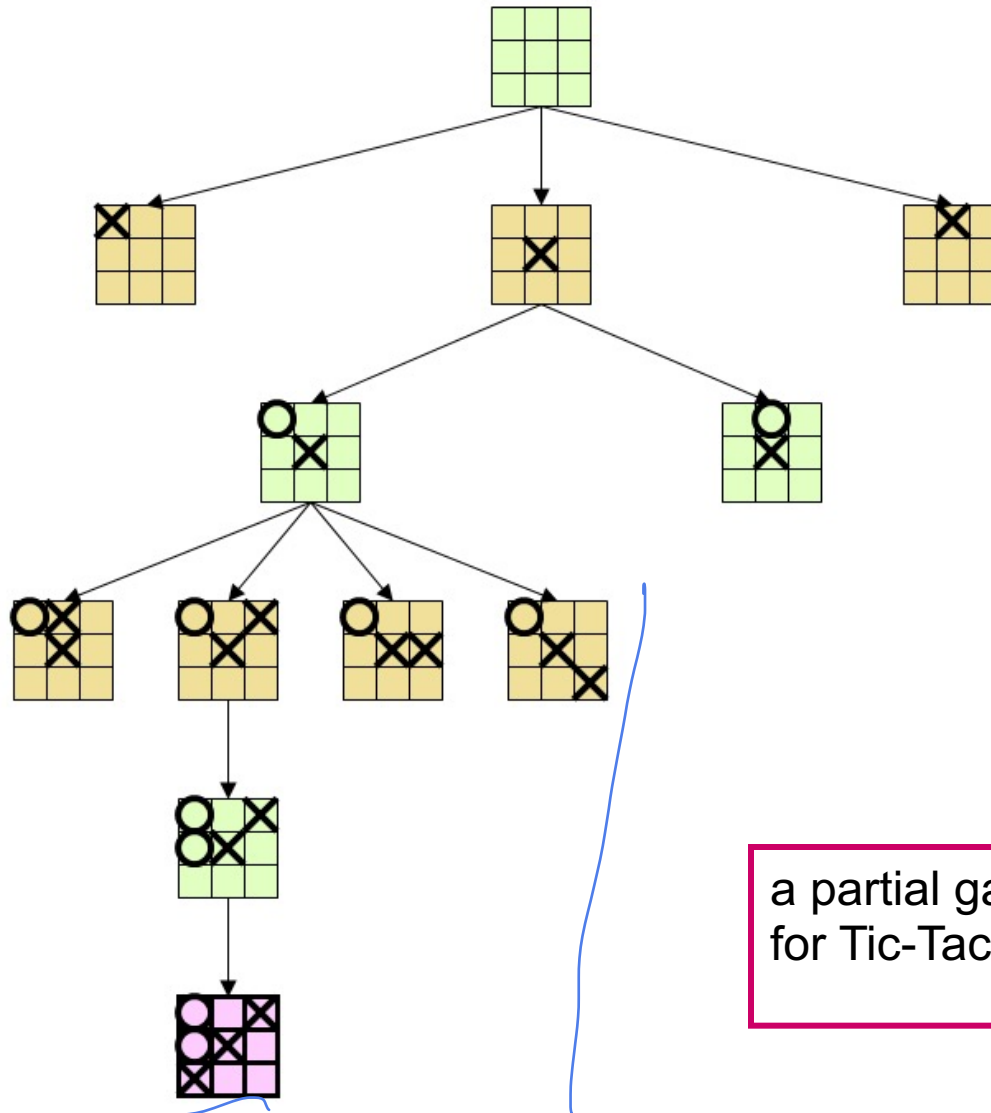
# Heuristics (Evaluation function)

- Typical evaluation function for game: weighted linear function
  - $h(s) = w_1f_1(s) + w_2f_2(s) + \dots + w_df_d(s)$
  - *weights* · *features* [dot product]
- For example, in chess
  - $W = \{ 1, 3, 3, 5, 8 \}$
  - $F = \{ \# \text{ pawns advantage, } \# \text{ bishops advantage, } \# \text{ knights advantage, } \# \text{ rooks advantage, } \# \text{ queens advantage} \}$
  - Is this what Deep Blue used?
  - What are some problems with this?

# Heuristics (Evaluation function)

- Typical evaluation function for game: weighted linear function
  - $h(s) = w_1f_1(s) + w_2f_2(s) + \dots + w_df_d(s)$
  - *weights* · *features* [dot product]
- For example, in chess
  - $W = \{ 1, 3, 3, 5, 8 \}$
  - $F = \{ \# \text{ pawns advantage, } \# \text{ bishops advantage, } \# \text{ knights advantage, } \# \text{ rooks advantage, } \# \text{ queens advantage} \}$
  - Is this what Deep Blue used?
  - What are some problems with this?
- More complex evaluation functions may involve learning
  - Adjusting weights based on outcomes
  - Perhaps non-linear functions
  - How to choose the *features*?

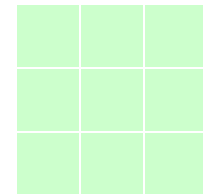
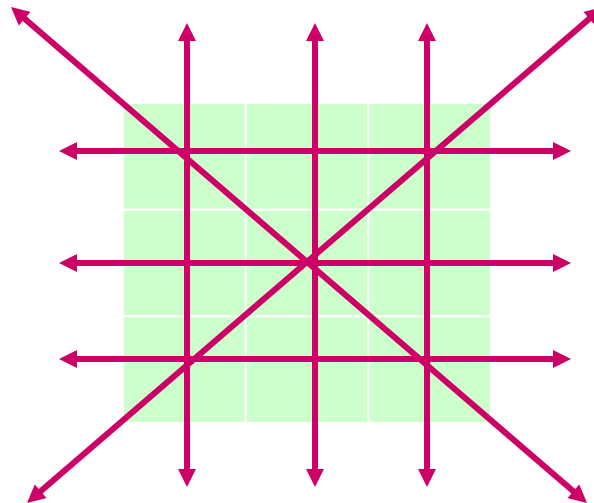
# Tic-Tac-Toe revisited



a partial game tree  
for Tic-Tac-Toe

# Evaluation function for Tic-Tac-Toe

- A simple evaluation function for Tic-Tac-Toe
  - count the number of rows where **X** can win
  - subtract the number of rows where **O** can win
- Value of evaluation function at start of game is zero
  - on an empty game board there are 8 possible winning rows for both **X** and **O**



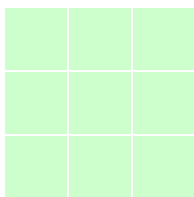
$$8 - 8 = 0$$

*# of ways*  
*X can win*  
*# of ways*  
*O can win*

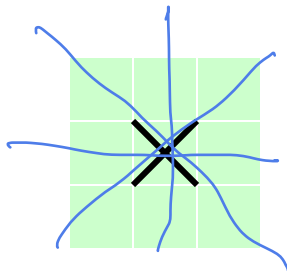
# Evaluating Tic-Tac-Toe

$\text{eval}_X = (\text{number of rows where } X \text{ can win}) - (\text{number of rows where } O \text{ can win})$

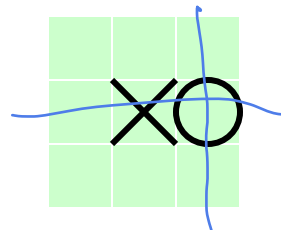
- After **X** moves in center, score for **X** is +4
- After **O** moves, score for **X** is +2
- After **X**'s next move, score for **X** is +4



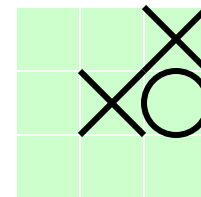
$$8-8 = 0$$



$$8-4 = 4$$



$$6-4 = 2$$

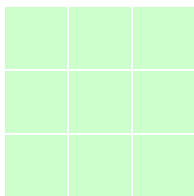


$$6-2 = 4$$

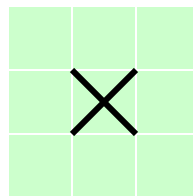
# Evaluating Tic-Tac-Toe

$\text{evalO} = (\text{number of rows where } \textcircled{O} \text{ can win}) - (\text{number of rows where } \text{X} \text{ can win})$

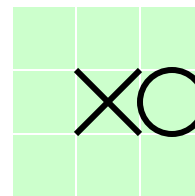
- After **X** moves in center, score for **O** is -4
- After **O** moves, score for **O** is +2
- After **X**'s next move, score for **O** is -4



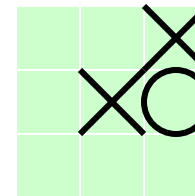
$$8-8 = 0$$



$$4-8 = -4$$



$$4-6 = -2$$

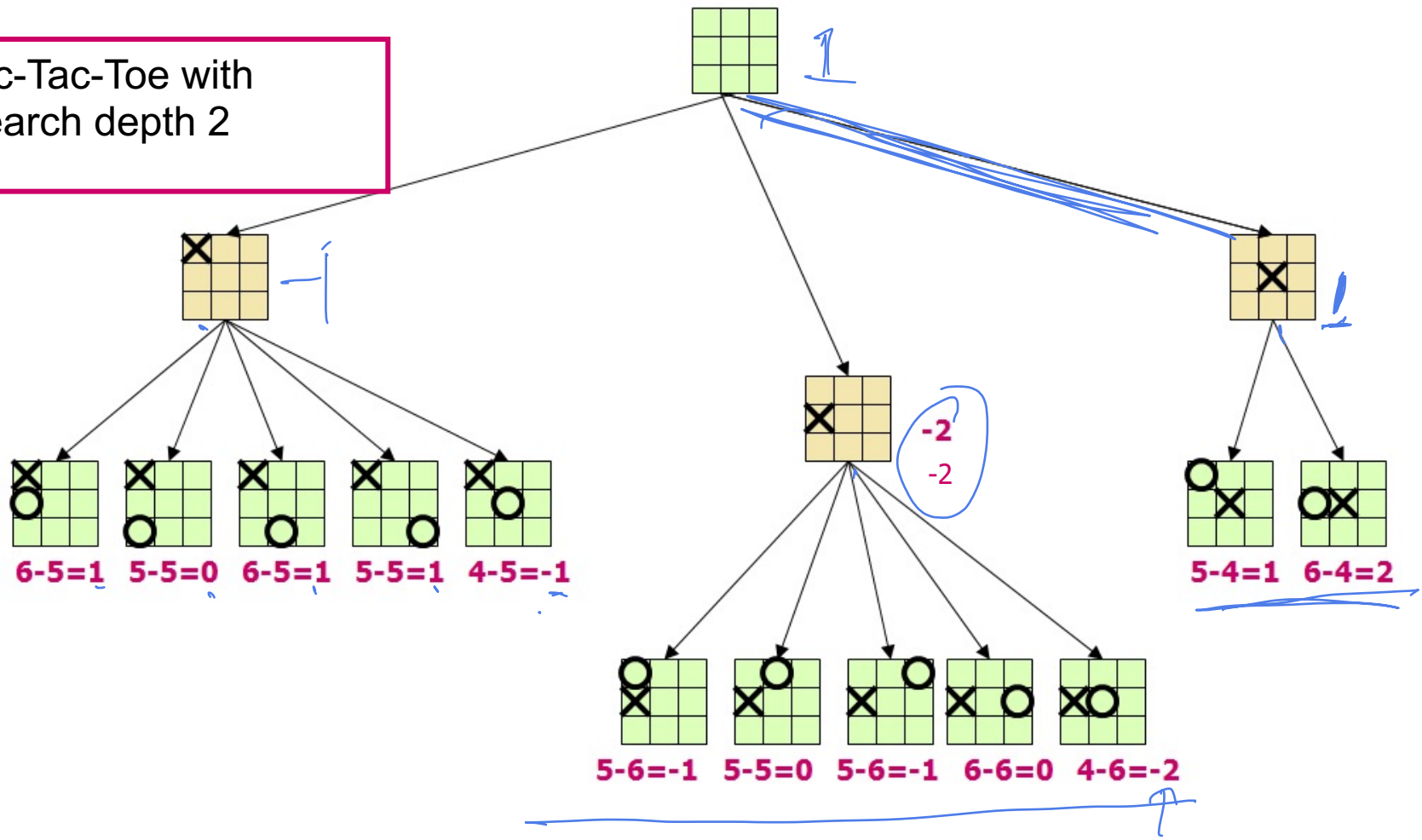


$$2-6 = -4$$



# Search depth cutoff

Tic-Tac-Toe with search depth 2

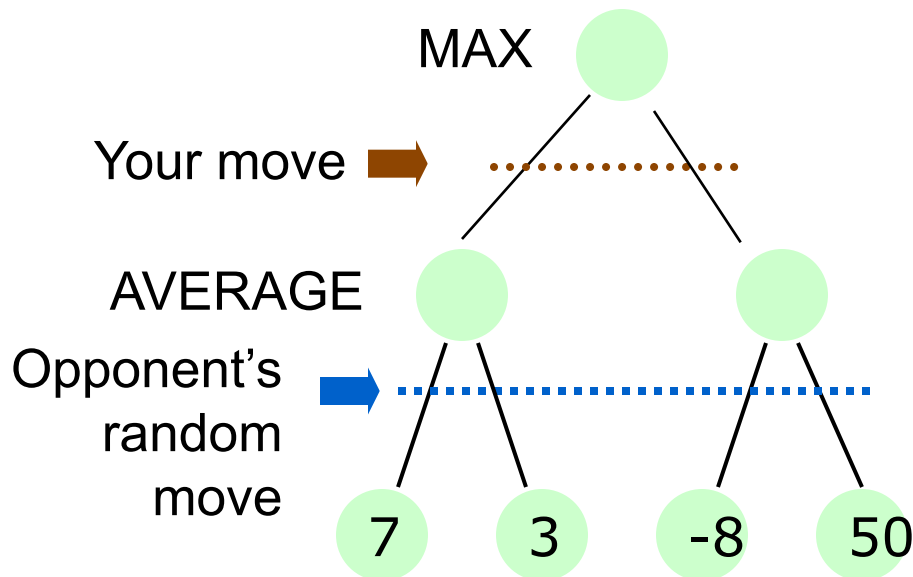


Evaluations shown for X

# Expectimax: Playing against a benign opponent

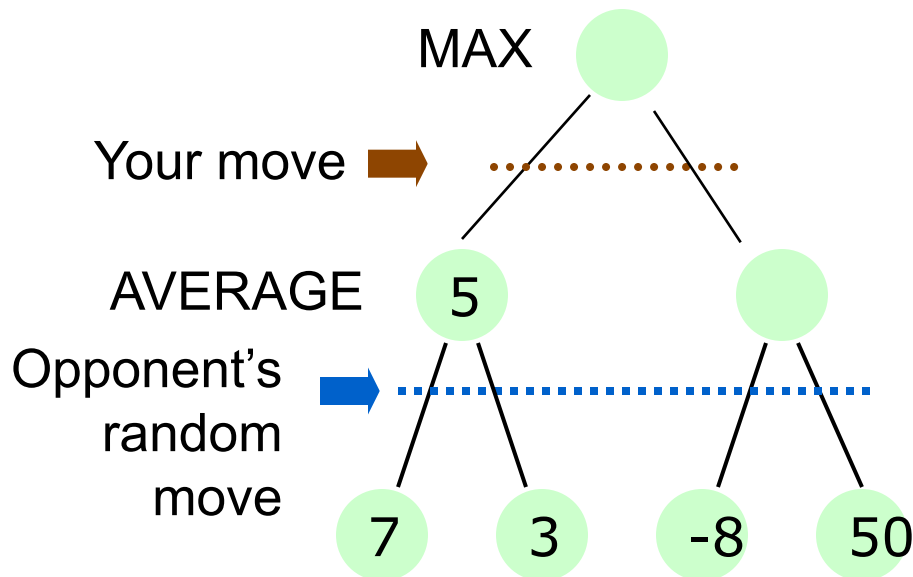
- Sometimes your opponents are not clever.
  - They behave randomly.
  - You can take advantage of that by modeling your opponent.
- Example of game of chance:
  - Slot machines
  - Tetris

# Expectimax example



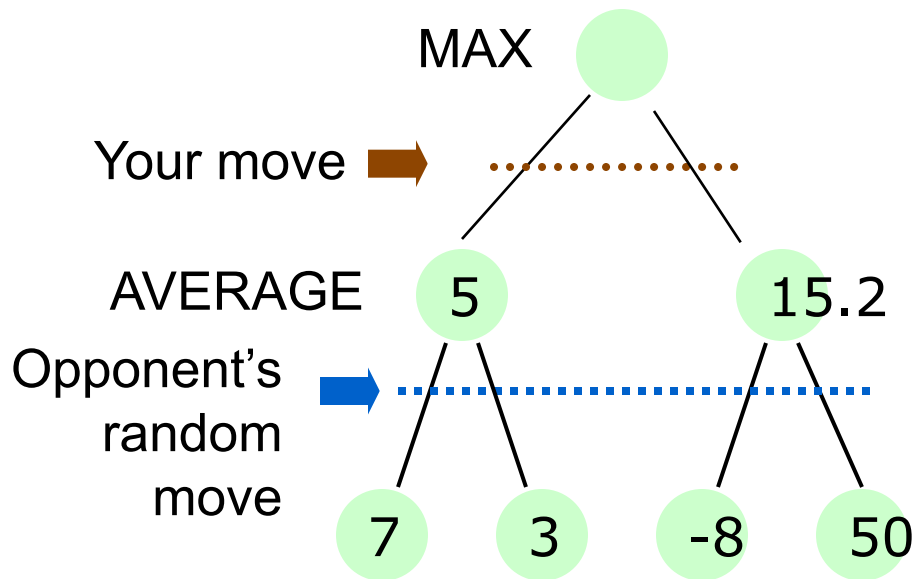
- Your opponent behaves randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability  $[0.5, 0.5]$
- If you move right, your opponent will select actions with  $[0.6, 0.4]$

# Expectimax example



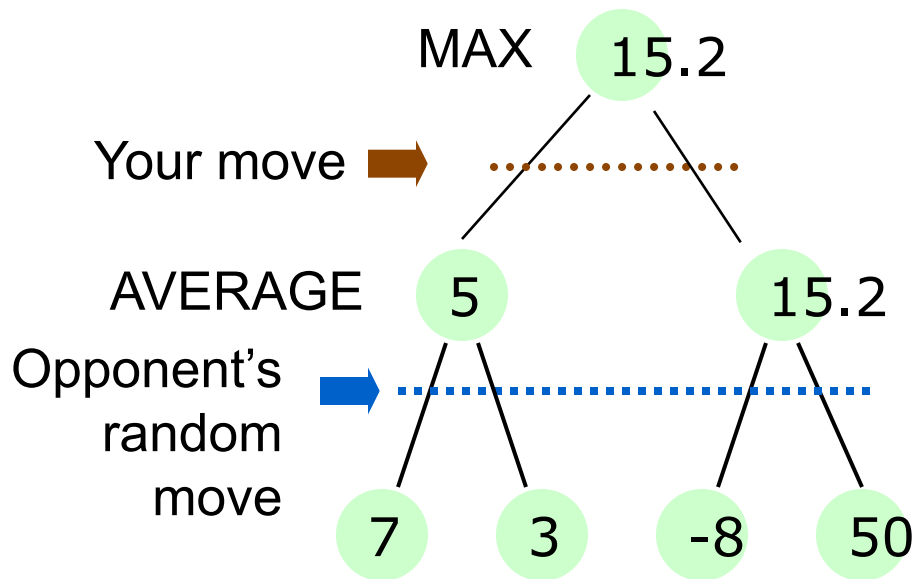
- Your opponent behaves randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability  $[0.5, 0.5]$
- If you move right, your opponent will select actions with  $[0.6, 0.4]$

# Expectimax example



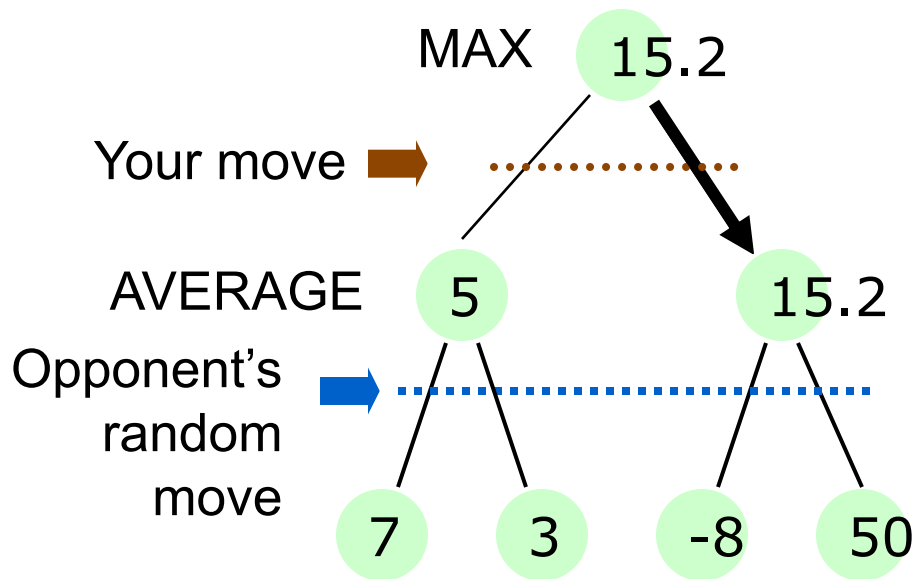
- Your opponent behaves randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability  $[0.5, 0.5]$
- If you move right, your opponent will select actions with  $[0.6, 0.4]$

# Expectimax example



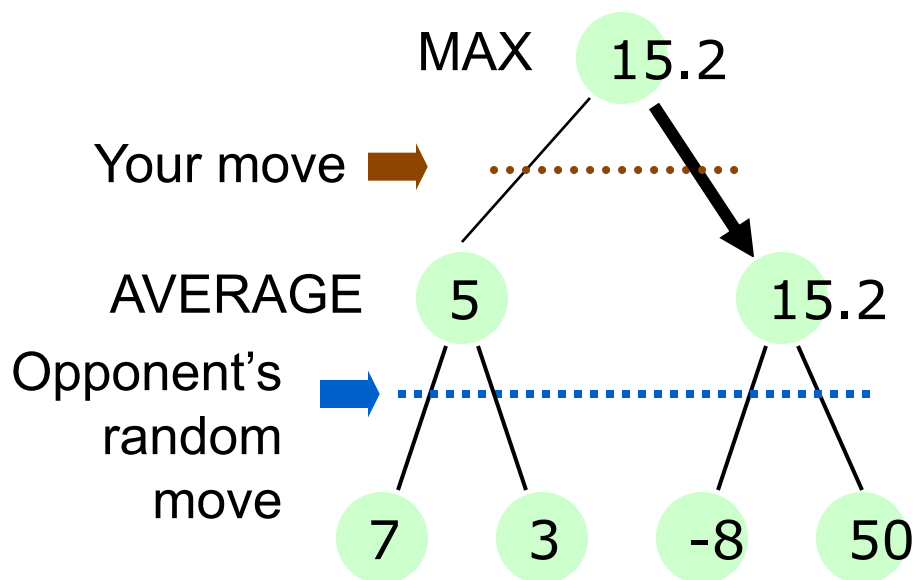
- Your opponent behaves randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability  $[0.5, 0.5]$
- If you move right, your opponent will select actions with  $[0.6, 0.4]$

# Expectimax example



- Your opponent behaves randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability  $[0.5, 0.5]$
- If you move right, your opponent will select actions with  $[0.6, 0.4]$

# Expectimax example



- Your opponent behaves randomly with a given probability distribution,
- If you move left, your opponent will select actions with probability  $[0.5, 0.5]$
- If you move right, your opponent will select actions with  $[0.6, 0.4]$

**Note: pruning becomes tricky in expectimax... think about why.**



# Summary of game playing

- Minimax search
- Game tree
- Alpha-beta pruning
- Early stop with an evaluation function
- Expectimax

# More reading / resources about game playing

- Required reading: AIMA 5.1-5.3
- Stochastic game / Expectiminimax: AIMA 5.5
  - Backgammon. TD-Gammon
  - Blackjack, Poker
- Famous game AI: Read the “Historical notes” of the AIMA Chapter 5
  - Deep blue
  - TD Gammon
- AlphaGo: <https://www.nature.com/articles/nature16961>