# MiniMax and Alpha Beta Pruning

CMSC 250H

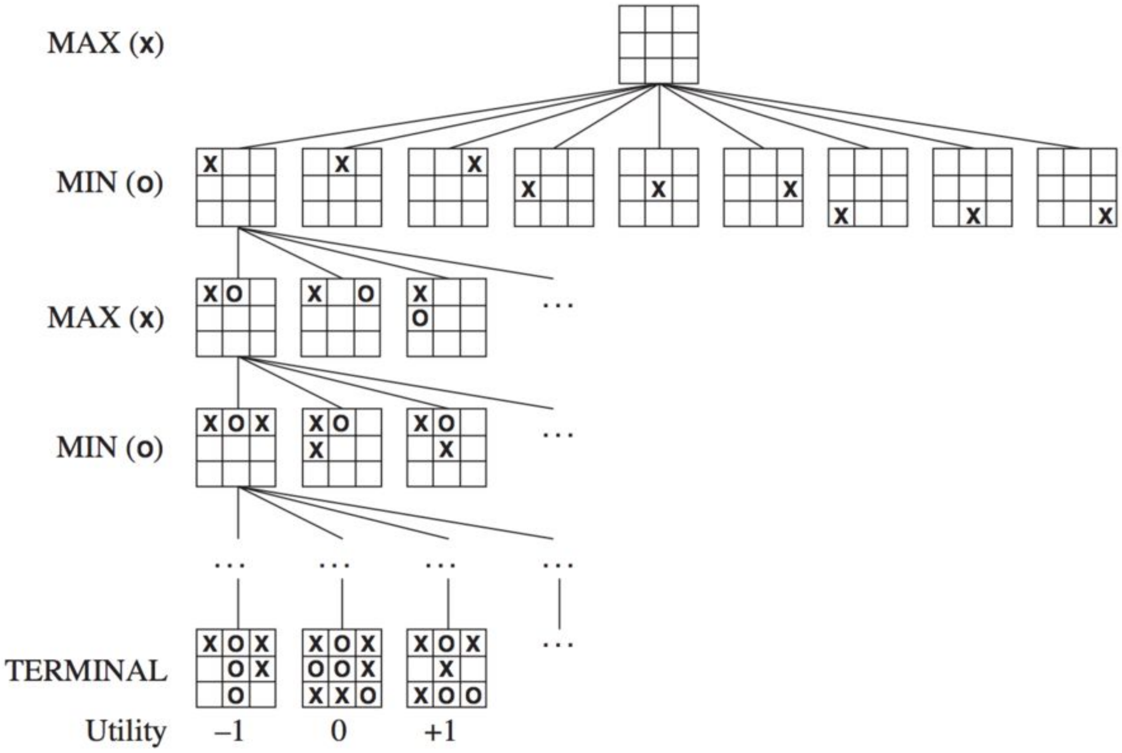
# Combinatorial Search

- Search algorithms that solve a particular problem by using large solution spaces
    - A* Search
    - Minimax
    - Alpha Beta pruning

# Combinatorial Search

- Search algorithms that solve a particular problem by using large solution spaces
  - A* Search
  - Minimax
  - Alpha Beta pruning
- At each step, the algorithm looks at all possible combinations of decisions

# Game Tree



# Tic Tac Toe

- How many ways can you make the first move?

# Tic Tac Toe

- How many ways can you make the first move?
  - 9

# Tic Tac Toe

- How many ways can you make the first move?
  - 9
- How many ways can a game of Tic-Tac-Toe be played?

# Tic Tac Toe

- How many ways can you make the first move?
  - 9
- How many ways can a game of Tic-Tac-Toe be played?
  - 255,168

# Tic Tac Toe

- How many ways can you make the first move?
  - 9
- How many ways can a game of Tic-Tac-Toe be played?
  - 255,168
- The game tree will have 255,168 leaves

# MiniMax

- Algorithm used in AI, Decision Theory, Game Theory, Stats, and Philosophy
  - Combinatorial Game Theory: Gives Game Solutions

# MiniMax

- Algorithm used in AI, Decision Theory, Game Theory, Stats, and Philosophy
  - Combinatorial Game Theory: Gives Game Solutions
- Idea: Minimize Loss in Worst Case

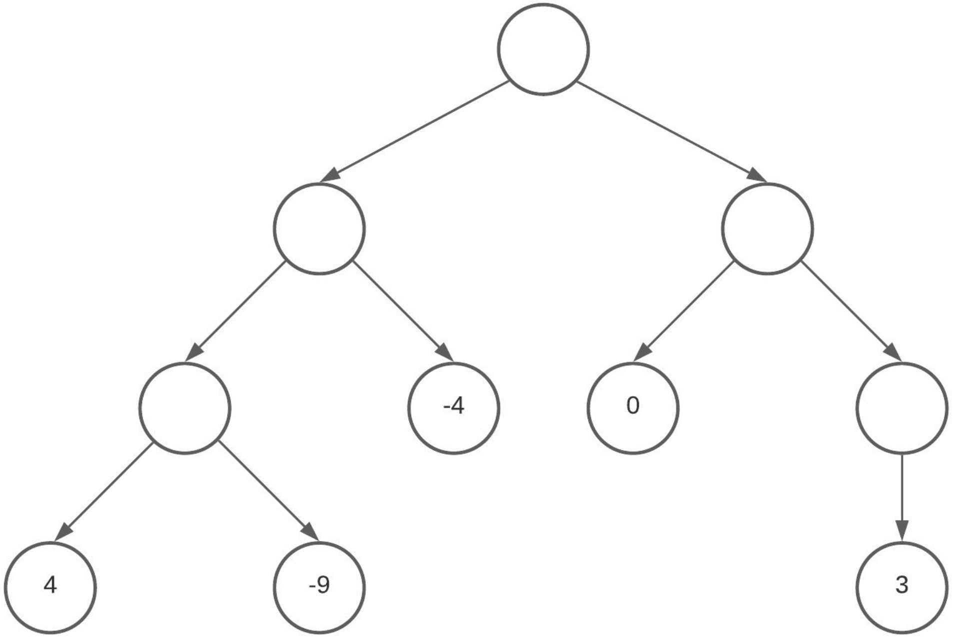
# MiniMax

- Algorithm used in AI, Decision Theory, Game Theory, Stats, and Philosophy
  - Combinatorial Game Theory: Gives Game Solutions
- Idea: Minimize Loss in Worst Case
- Uses Recursion or Backtracking to make a Perfect Choice

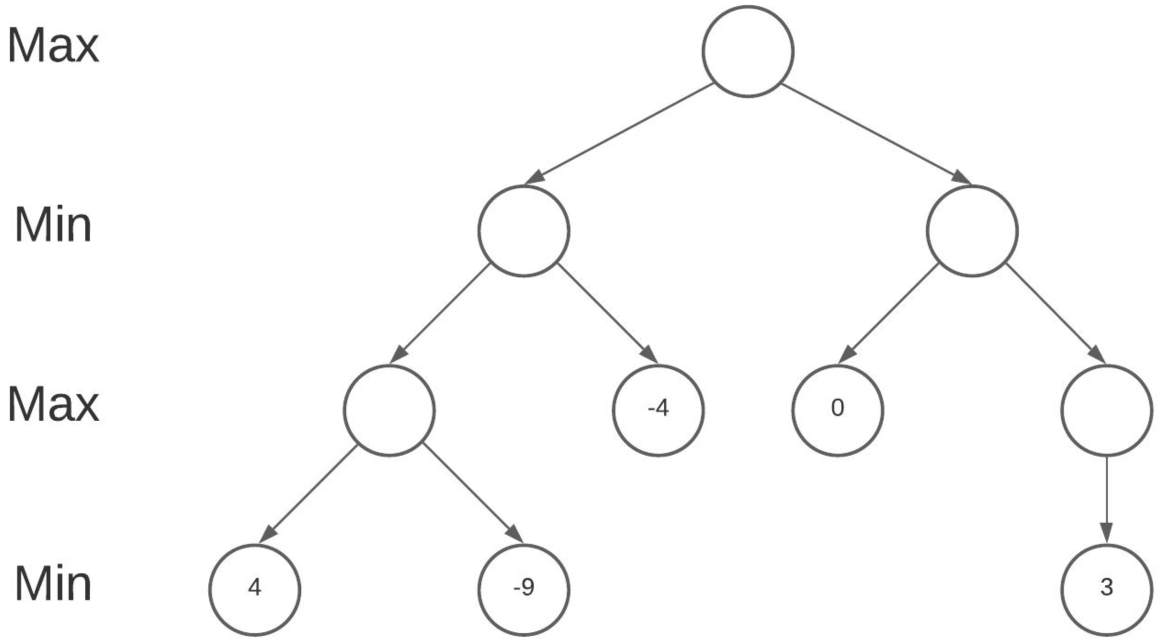
# MiniMax

- Algorithm used in AI, Decision Theory, Game Theory, Stats, and Philosophy
  - Combinatorial Game Theory: Gives Game Solutions
- Idea: Minimize Loss in Worst Case
- Uses Recursion or Backtracking to make a Perfect Choice
- Slow!
  - Needs to visit every node

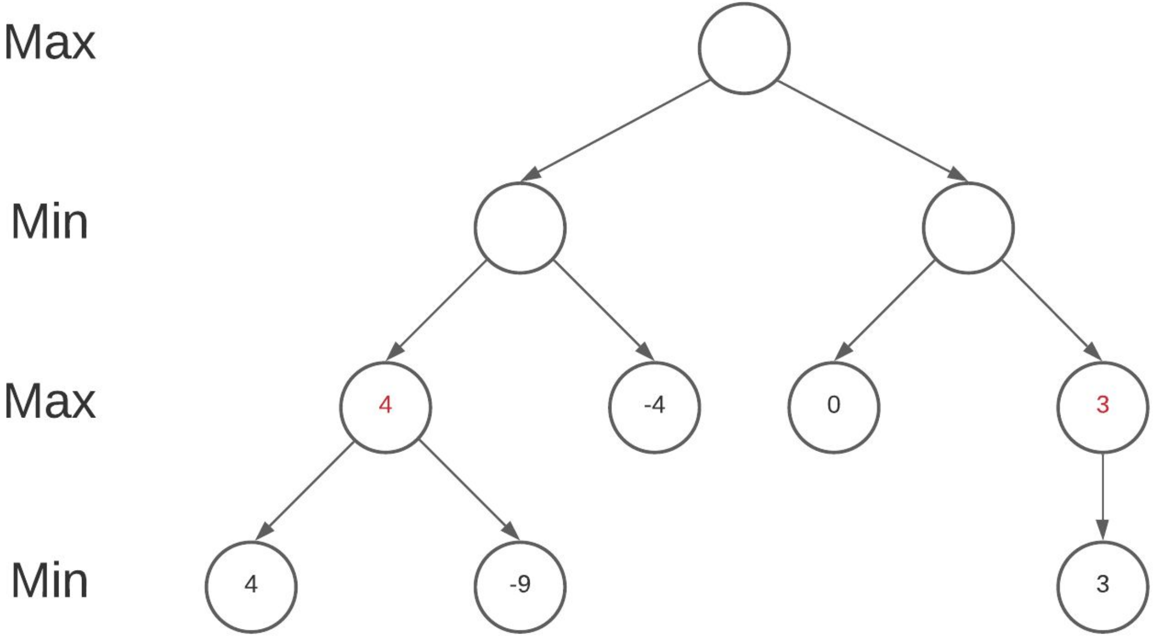
# MiniMax



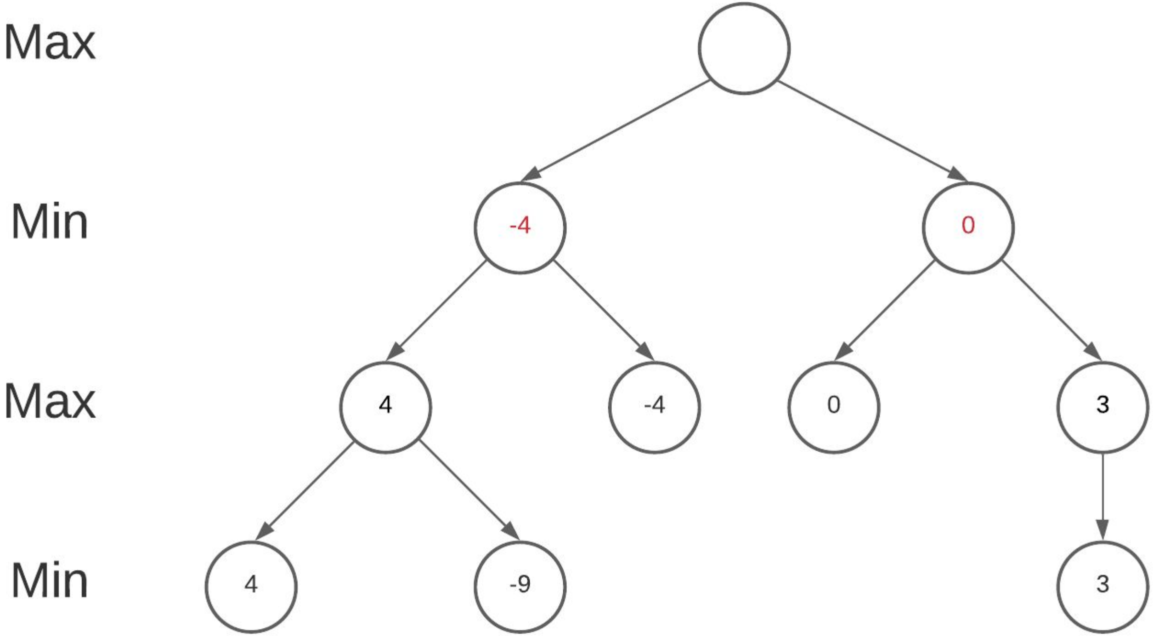
# MiniMax



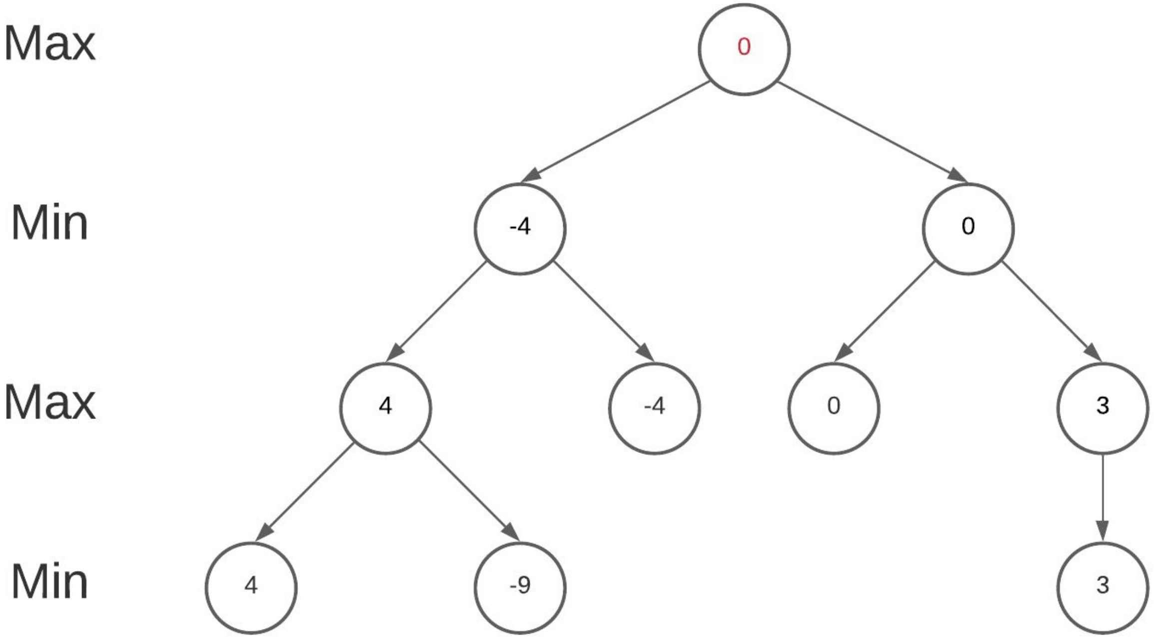
# MiniMax



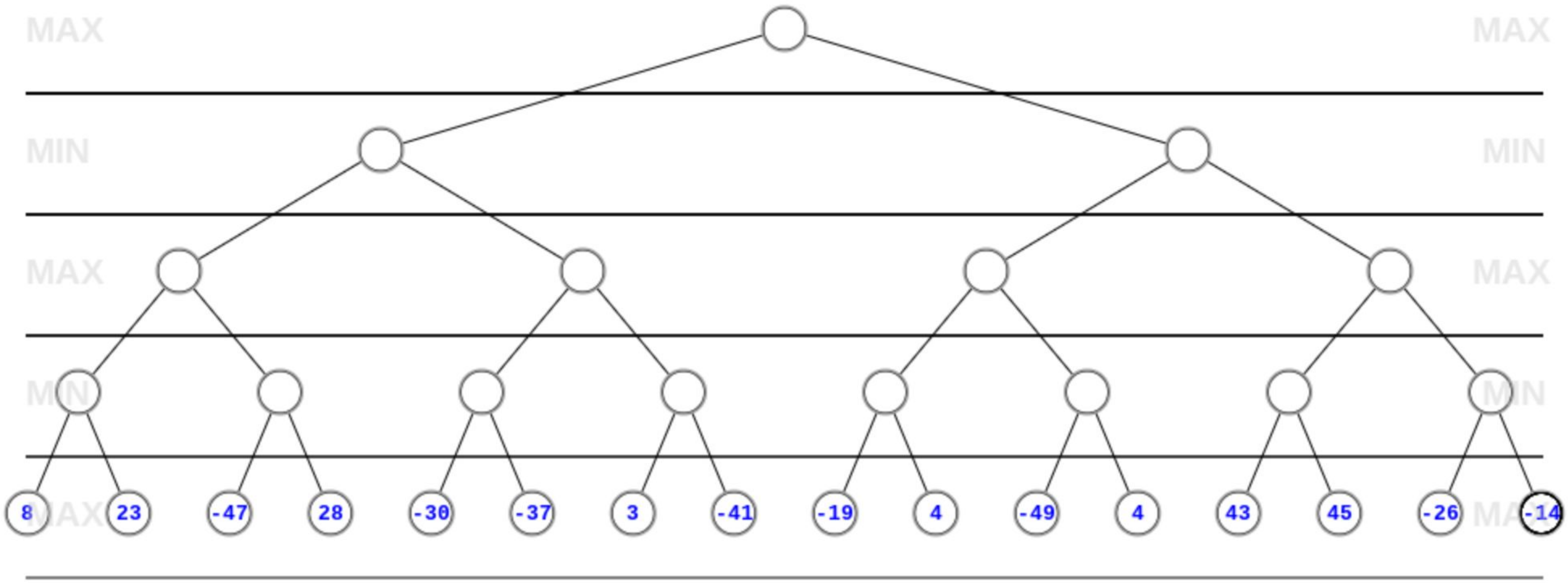
# MiniMax



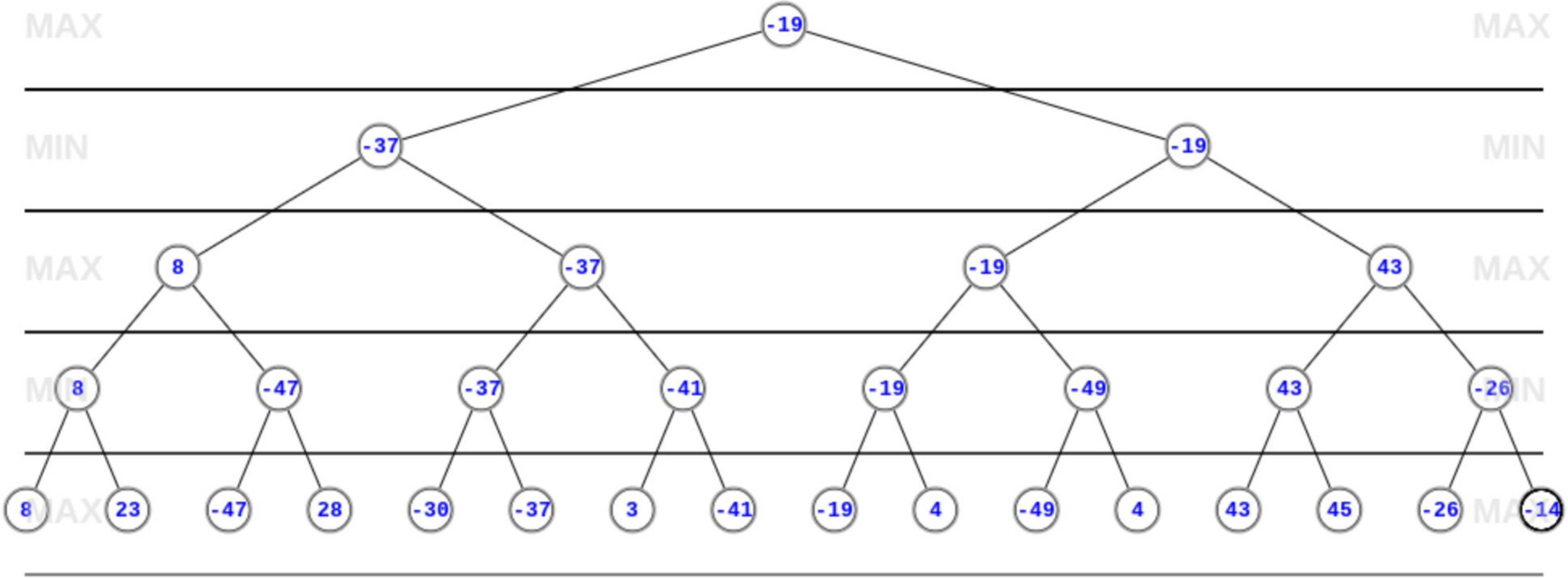
# MiniMax



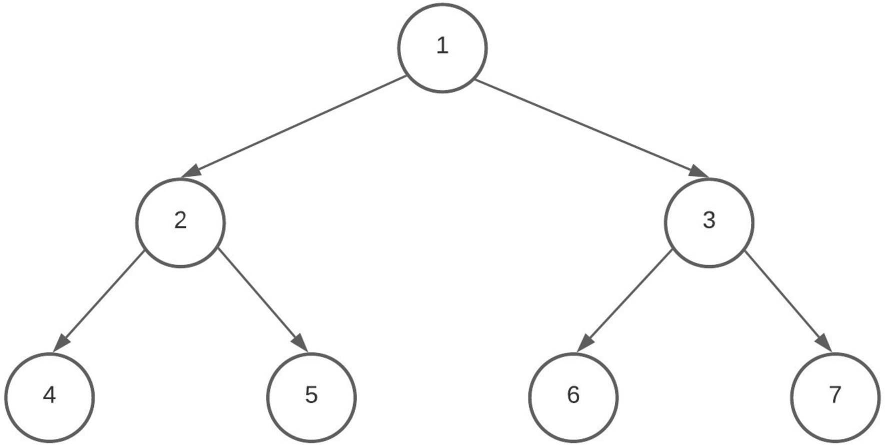
# Bigger Example



# Bigger Example

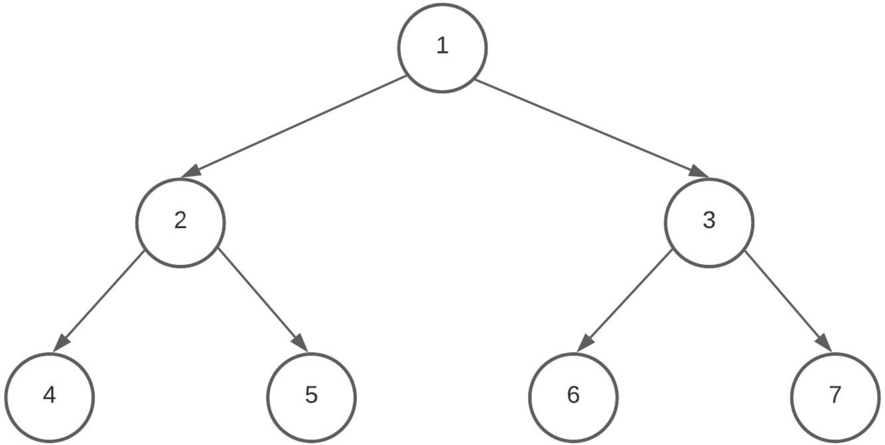


# Tree Traversal




**Pre-Order:** Left Side of Bubble

# Tree Traversal

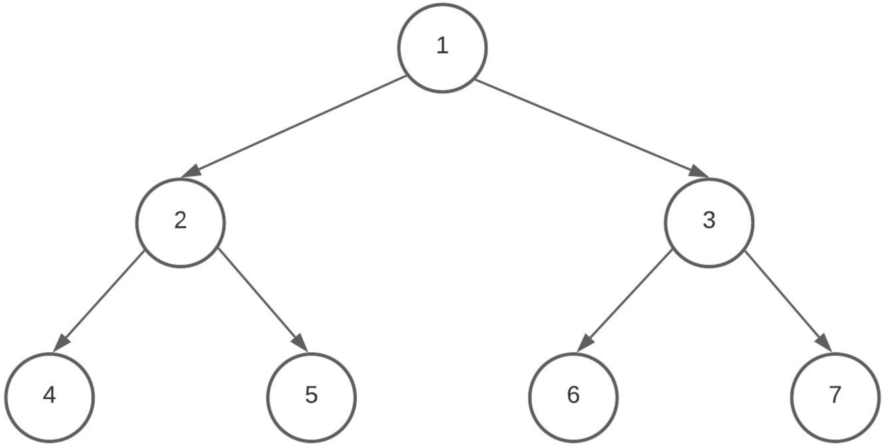



**Pre-Order:** Left Side of Bubble

{1, 2, 4, 5, 3, 6, 7}

}

# Tree Traversal

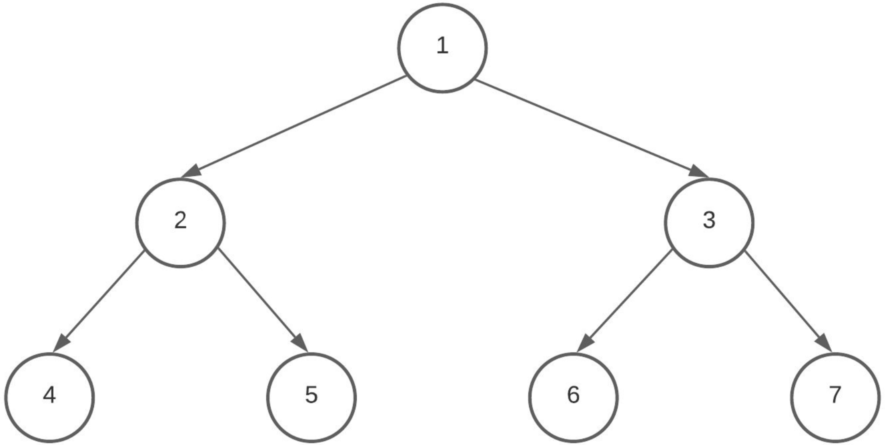



**Pre-Order:** Left Side of Bubble

{1, 2, 4, 5, 3, 6, 7}

**In-Order:** Bottom of Bubble

# Tree Traversal



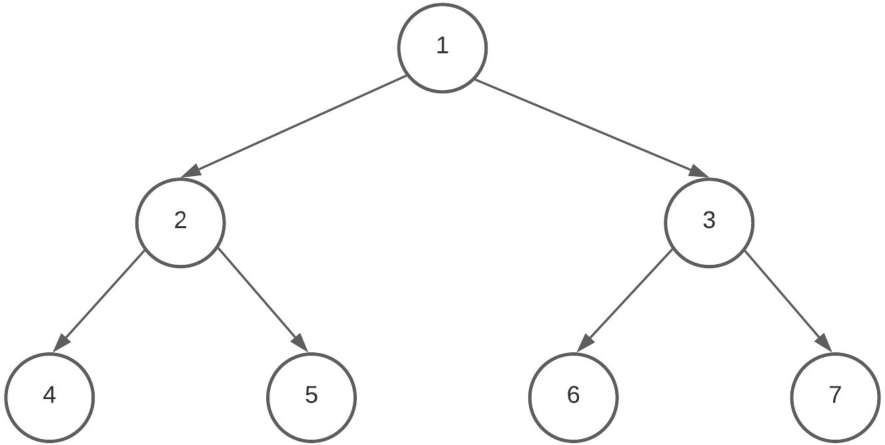

**Pre-Order:** Left Side of Bubble

{1, 2, 4, 5, 3, 6, 7}

**In-Order:** Bottom of Bubble

{4, 2, 5, 1, 6, 3, 7}

# Tree Traversal




**Pre-Order:** Left Side of Bubble

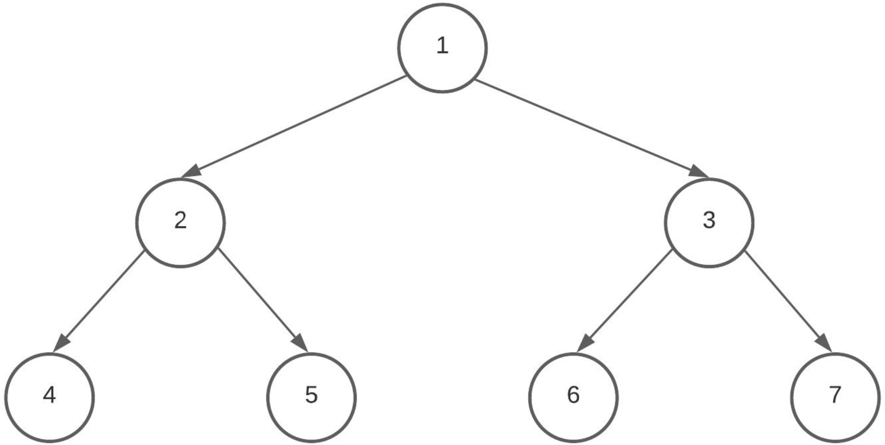
{1, 2, 4, 5, 3, 6, 7}

**In-Order:** Bottom of Bubble

{4, 2, 5, 1, 6, 3, 7}

**Post-Order:** Right Side of Bubble

# Tree Traversal




**Pre-Order:** Left Side of Bubble

{1, 2, 4, 5, 3, 6, 7}

**In-Order:** Bottom of Bubble

{4, 2, 5, 1, 6, 3, 7}

**Post-Order:** Right Side of Bubble

{4, 5, 2, 6, 7, 3, 1}

# Alpha Beta Pruning

- Makes MiniMax more efficient

# Alpha Beta Pruning

- Makes MiniMax more efficient
- If we search down the whole tree, the number of states is exponential to the depth of the tree

# Alpha Beta Pruning

- Makes MiniMax more efficient
- If we search down the whole tree, the number of states is exponential to the depth of the tree
- Alpha Beta Pruning cuts away leaves when traversing tree

# Alpha Beta Pruning

- Makes MiniMax more efficient
- If we search down the whole tree, the number of states is exponential to the depth of the tree
- Alpha Beta Pruning cuts away leaves when traversing tree
- Stops evaluating a state when at least one possibility has been found to prove worse then a previous found move

# Alpha Beta Pruning

- Makes MiniMax more efficient
- If we search down the whole tree, the number of states is exponential to the depth of the tree
- Alpha Beta Pruning cuts away leaves when traversing tree
- Stops evaluating a state when at least one possibility has been found to prove worse then a previous found move
- Returns the same value that MiniMax would produce

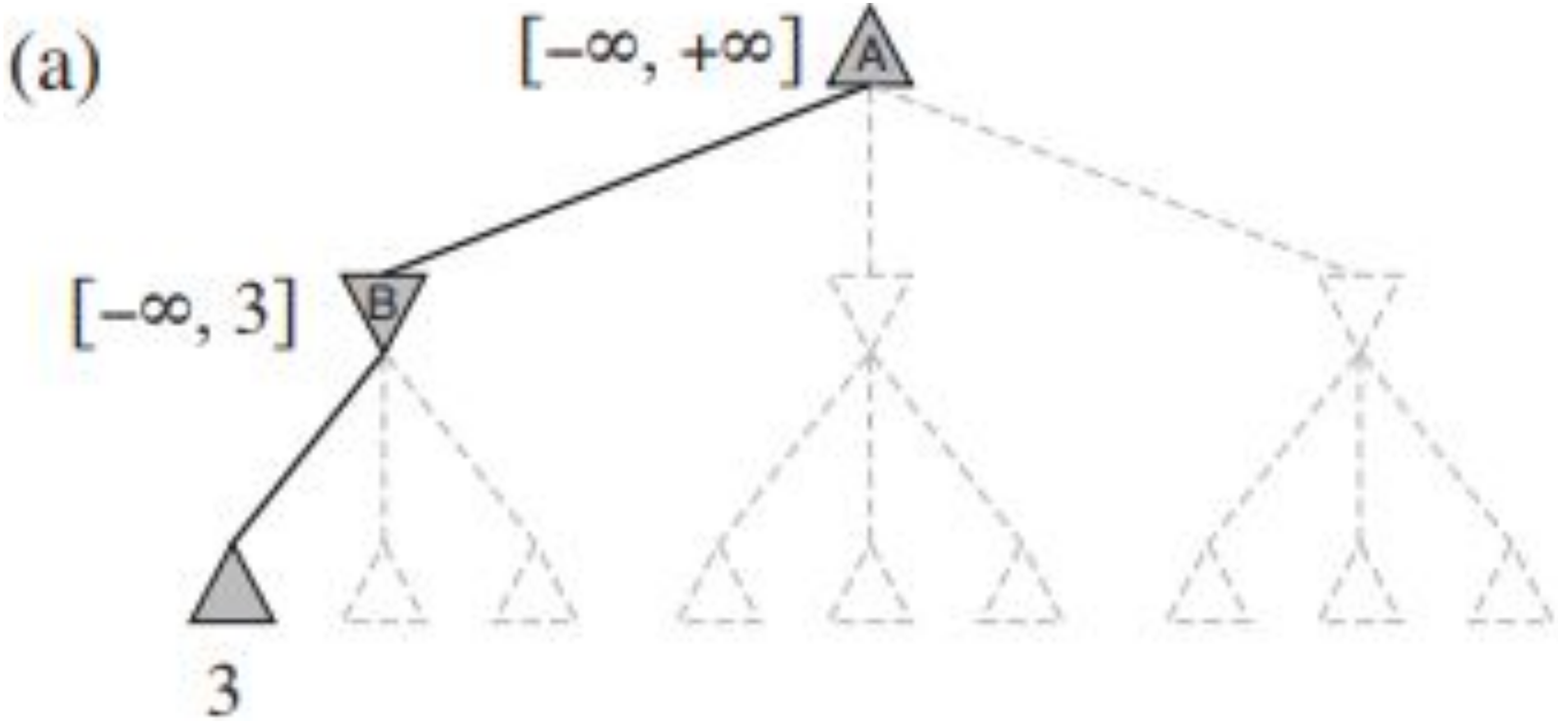
# Alpha Beta Pruning

- Makes MiniMax more efficient
- If we search down the whole tree, the number of states is exponential to the depth of the tree
- Alpha Beta Pruning cuts away leaves when traversing tree
- Stops evaluating a state when at least one possibility has been found to prove worse then a previous found move
- Returns the same value that MiniMax would produce
- Prunes away branches that do not influence final decision

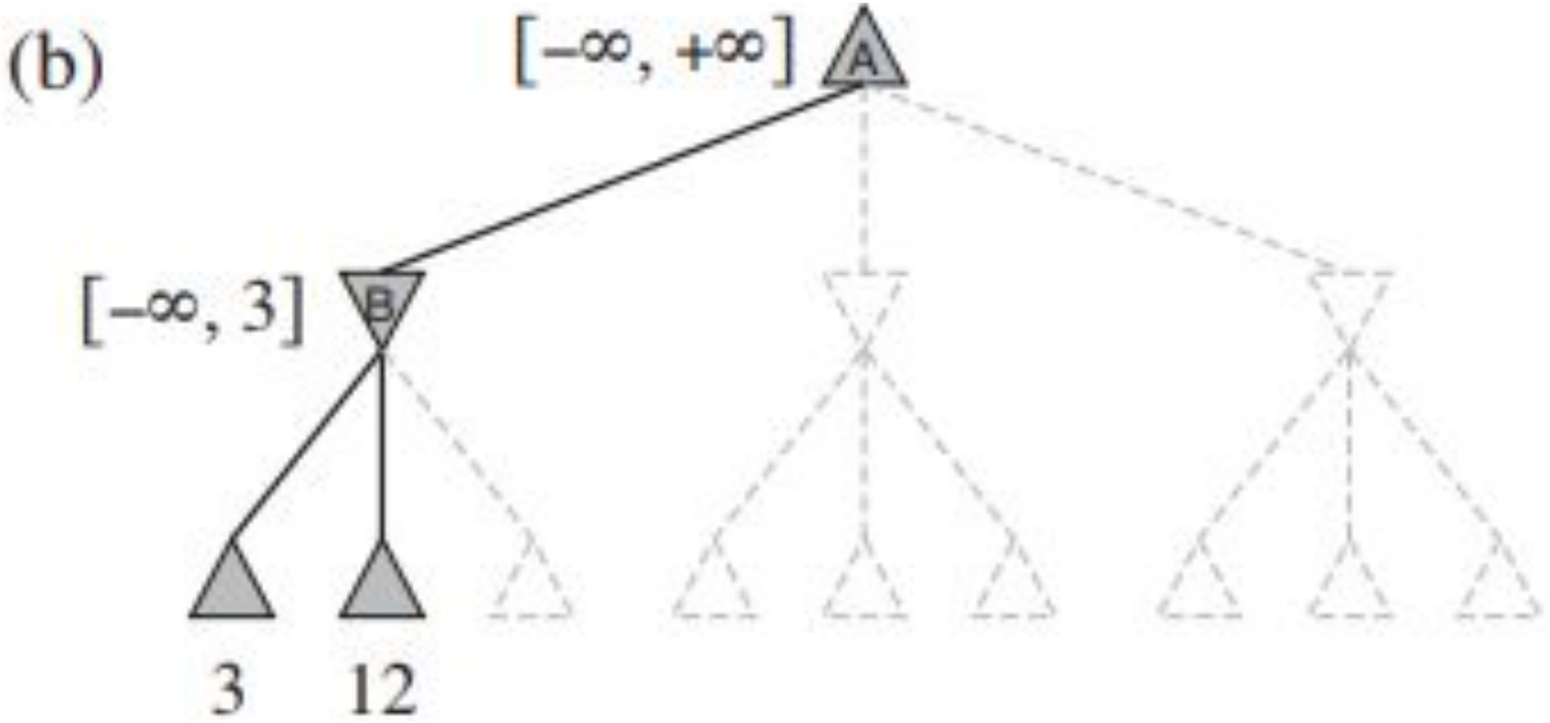
# Alpha Beta Pruning

- Makes MiniMax more efficient
- If we search down the whole tree, the number of states is exponential to the depth of the tree
- Alpha Beta Pruning cuts away leaves when traversing tree
- Stops evaluating a state when at least one possibility has been found to prove worse then a previous found move
- Returns the same value that MiniMax would produce
- Prunes away branches that do not influence final decision
- In the tuple [$\alpha$ , $\beta$]
    - Maximize $\alpha$
    - Minimize $\beta$

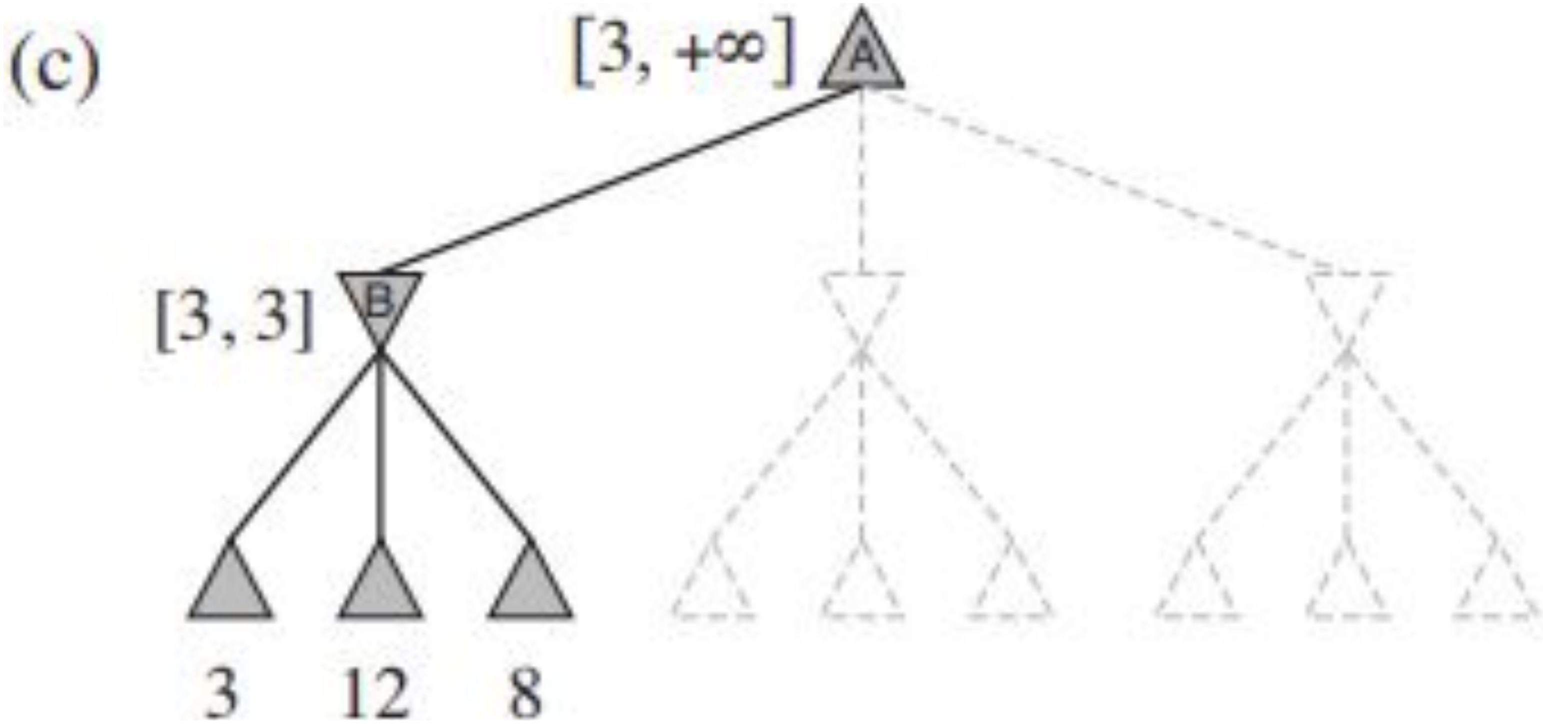
# Alpha Beta Pruning



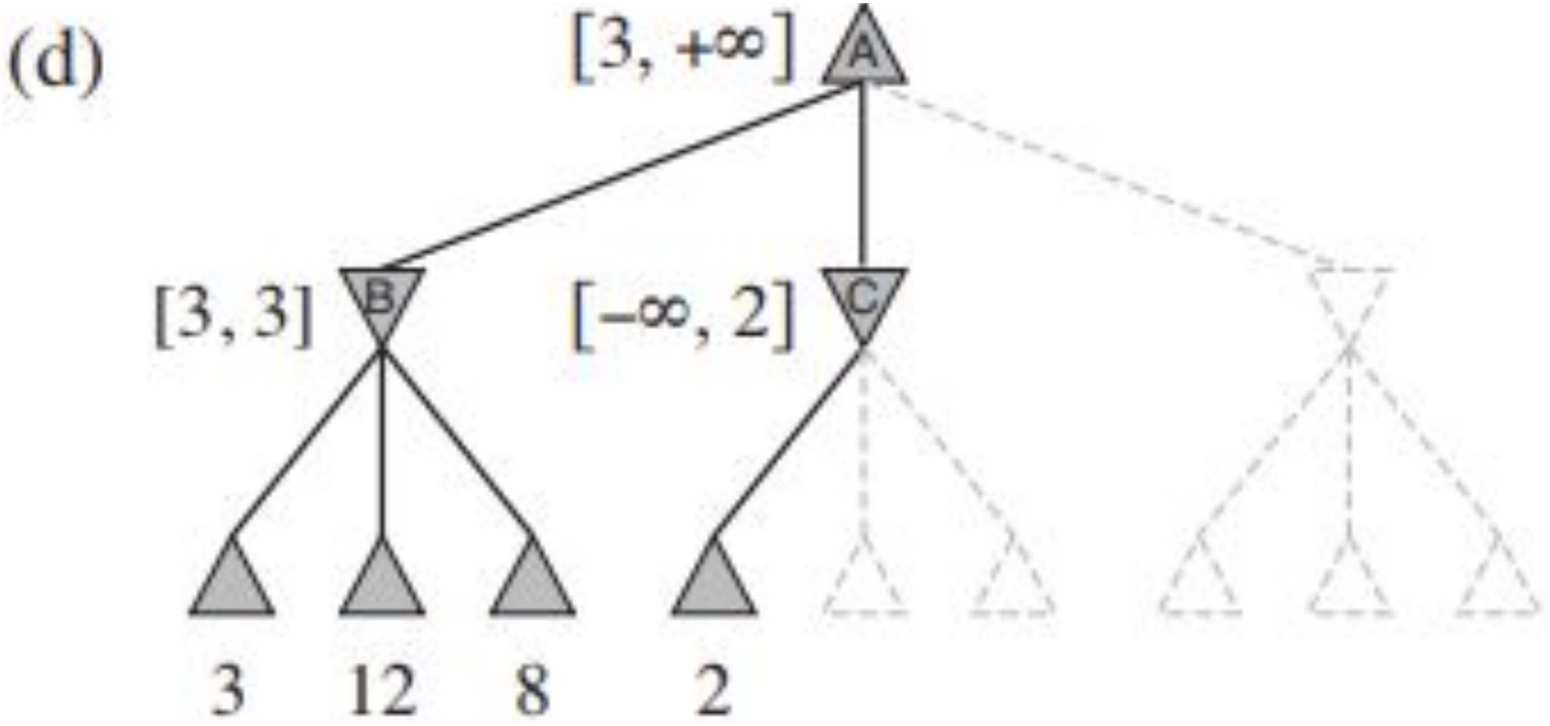
# Alpha Beta Pruning



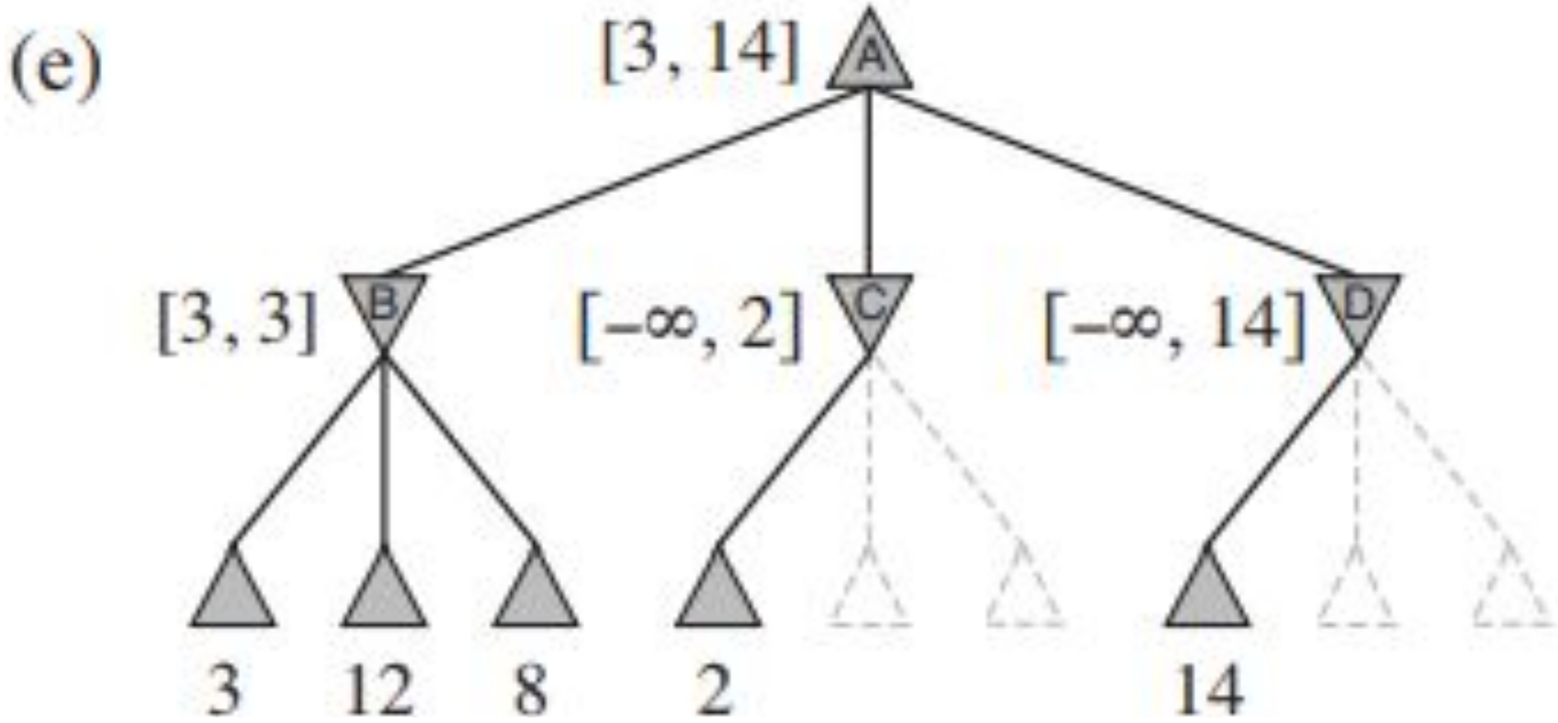
# Alpha Beta Pruning



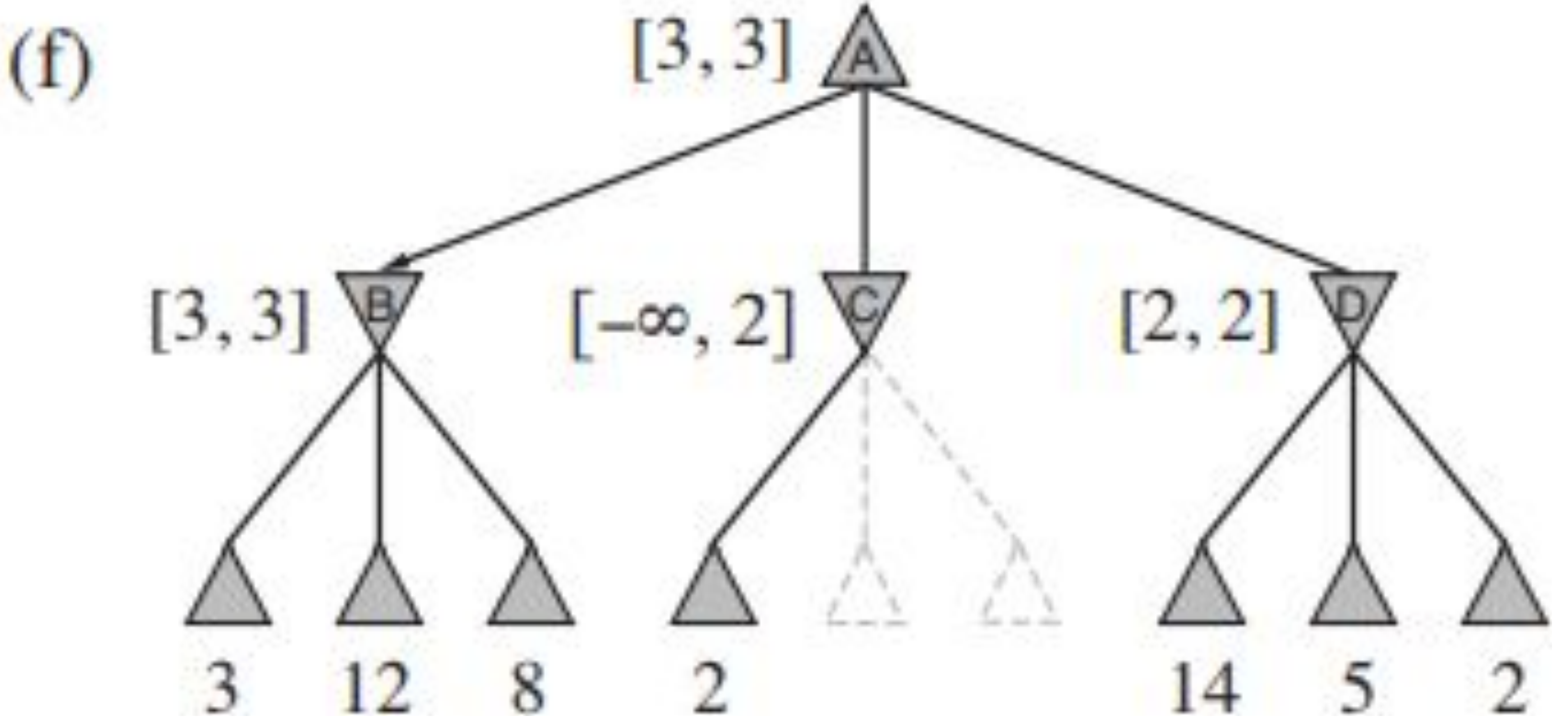
# Alpha Beta Pruning



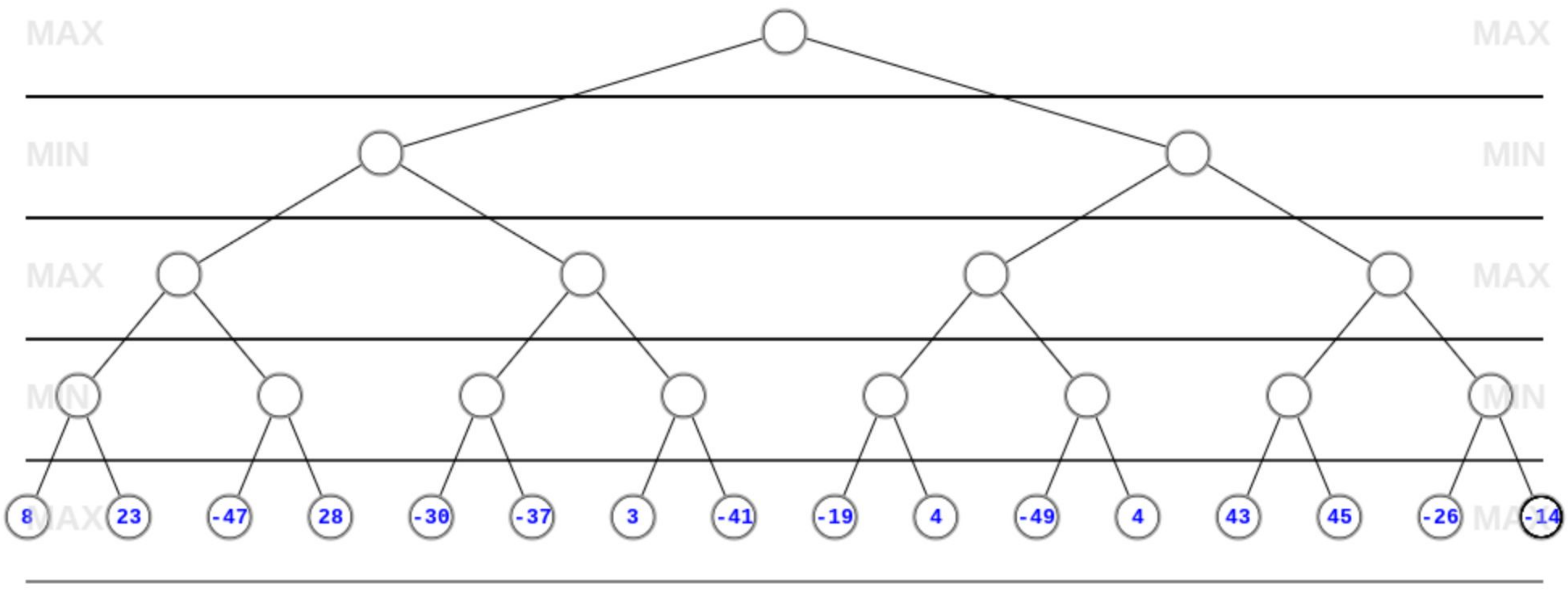
# Alpha Beta Pruning



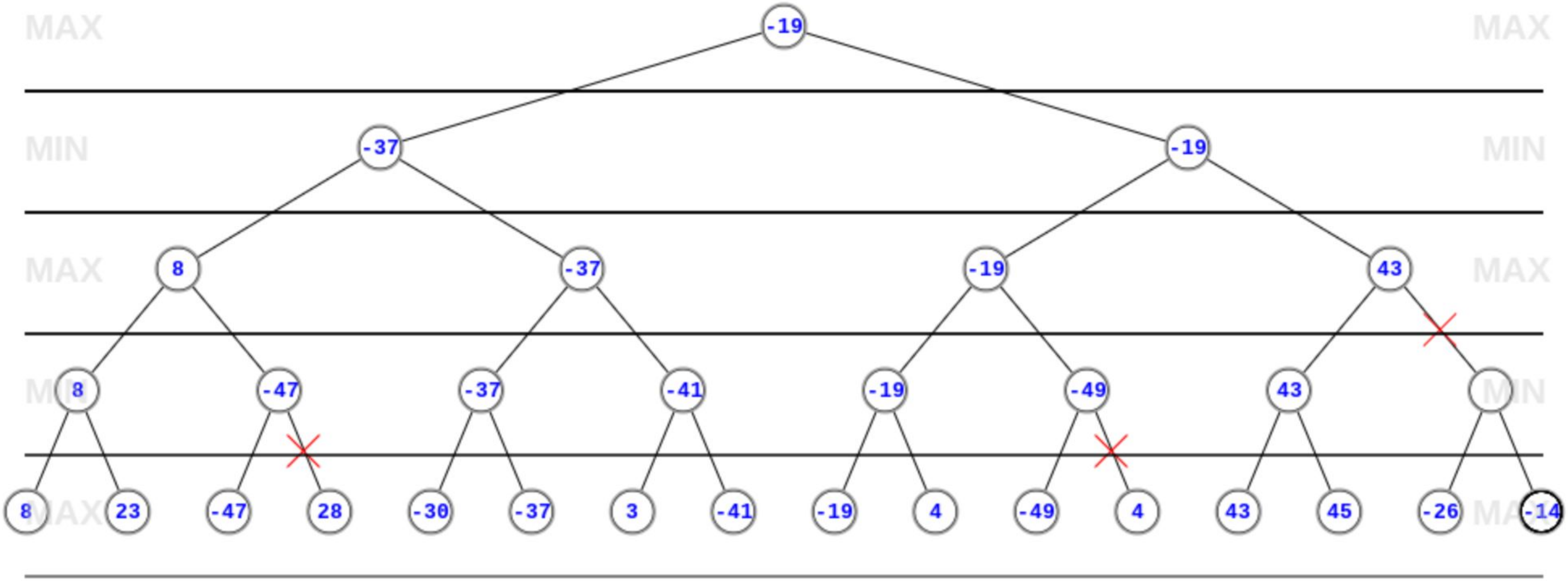
# Alpha Beta Pruning



# Bigger Example



# Bigger Example



# Pseudo Code

```
maxValue(state, α, β)
    If (Terminal State)
        Return value
    Else
        For each child
            If (Player 2's turn)
                α = max(α, minValue(state,α, β))
                If (α ≥ β)
                    return β
            Else
                β = min(β, maxValue(state,α, β))
                Return β
        Return α
```

# Pseudo Code

```
maxValue(state, α, β)
    If (Terminal State)
        Return value
    Else
        For each child
            If (Player 2's turn)
                α = max(α, minValue(state,α, β))
                If (α ≥ β)
                    return β
            Else
                β = min(β, maxValue(state,α, β))
                Return β
        Return α
```

```
minValue(state, α, β)
    If (Terminal State)
        Return value
    Else
        For each child
            If (Player 1's turn)
                β = min(β, maxValue(state,α, β))
                If (β ≤ α)
                    return α
            Else
                α = max(α, minValue(state,α, β))
                Return α
        Return β
```

# MiniMax vs. Alpha Beta Pruning Runtime

- MiniMax
  - Runtime: $O(b^h)$
  - Space: $O(bh)$

b = Branching Factor
h = Height of the Tree

# MiniMax vs. Alpha Beta Pruning Runtime

- MiniMax
  - Runtime: $O(b^h)$
  - Space: $O(bh)$
- Alpha Beta Pruning
  - Runtime:
    - Worst-Case: $O(b^h)$
    - Best-Case: $O(b^{h/2})$
  - Space: $O(bh)$

b = Branching Factor
h = Height of the Tree

# MiniMax vs. Alpha Beta Pruning Runtime

- MiniMax
  - Runtime: $O(b^h)$
  - Space: $O(bh)$
- Alpha Beta Pruning
  - Runtime:
    - Worst-Case: $O(b^h)$
    - Best-Case: $O(b^{h/2})$
  - Space: $O(bh)$

Why is the Worst-Case Runtime equal to MiniMax?

b = Branching Factor
h = Height of the Tree

# MiniMax vs. Alpha Beta Pruning Runtime

- MiniMax
  - Runtime: $O(b^h)$
  - Space: $O(bh)$
- Alpha Beta Pruning
  - Runtime:
    - Worst-Case: $O(b^h)$
    - Best-Case: $O(b^{h/2})$
  - Space: $O(bh)$

Why is the Worst-Case Runtime equal to MiniMax?

In the Worst-Case, your Alpha Beta is running MiniMax!

b = Branching Factor
h = Height of the Tree

# Alpha Beta for 2 Player Games

- Game Trees get really big really fast
  - Grows exponentially
  - Alpha Beta Pruning is more efficient than Minimax

# Alpha Beta for 2 Player Games

- Game Trees get really big really fast
  - Grows exponentially
  - Alpha Beta Pruning is more efficient than Minimax
- Used for many games
  - Tic-Tac-Toe
  - Chess
  - Go

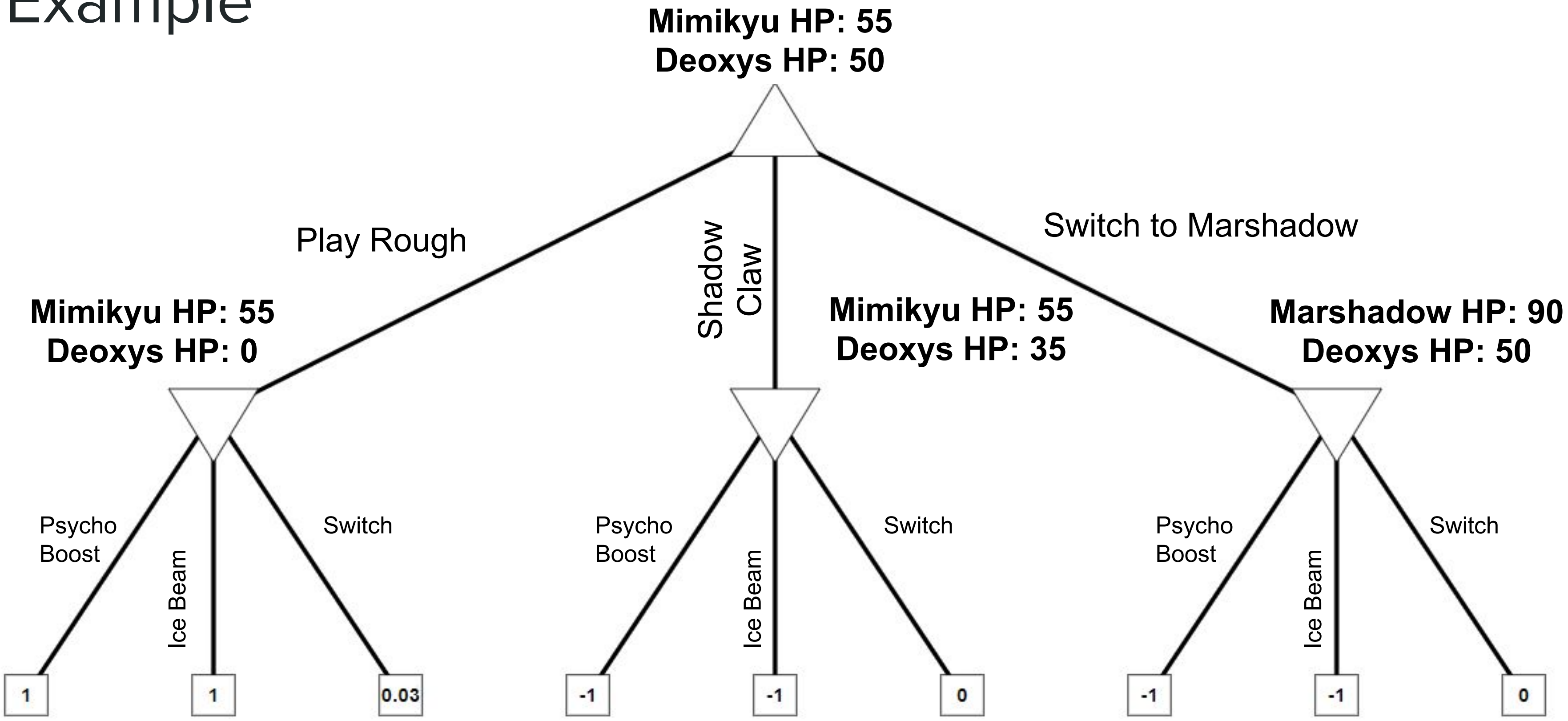
# Alpha Beta for 2 Player Games

- Game Trees get really big really fast
  - Grows exponentially
  - Alpha Beta Pruning is more efficient than Minimax
- Used for many games
  - Tic-Tac-Toe
  - Chess
  - Go
- Heuristic is easily incorporated
  - A Heuristic is a mapping from a game state to a value
    - Ex: In Chess, White Pieces - Black Pieces = Value
      - This is a bad heuristic to use
  - We use heuristics when we do not want calculate every end game state

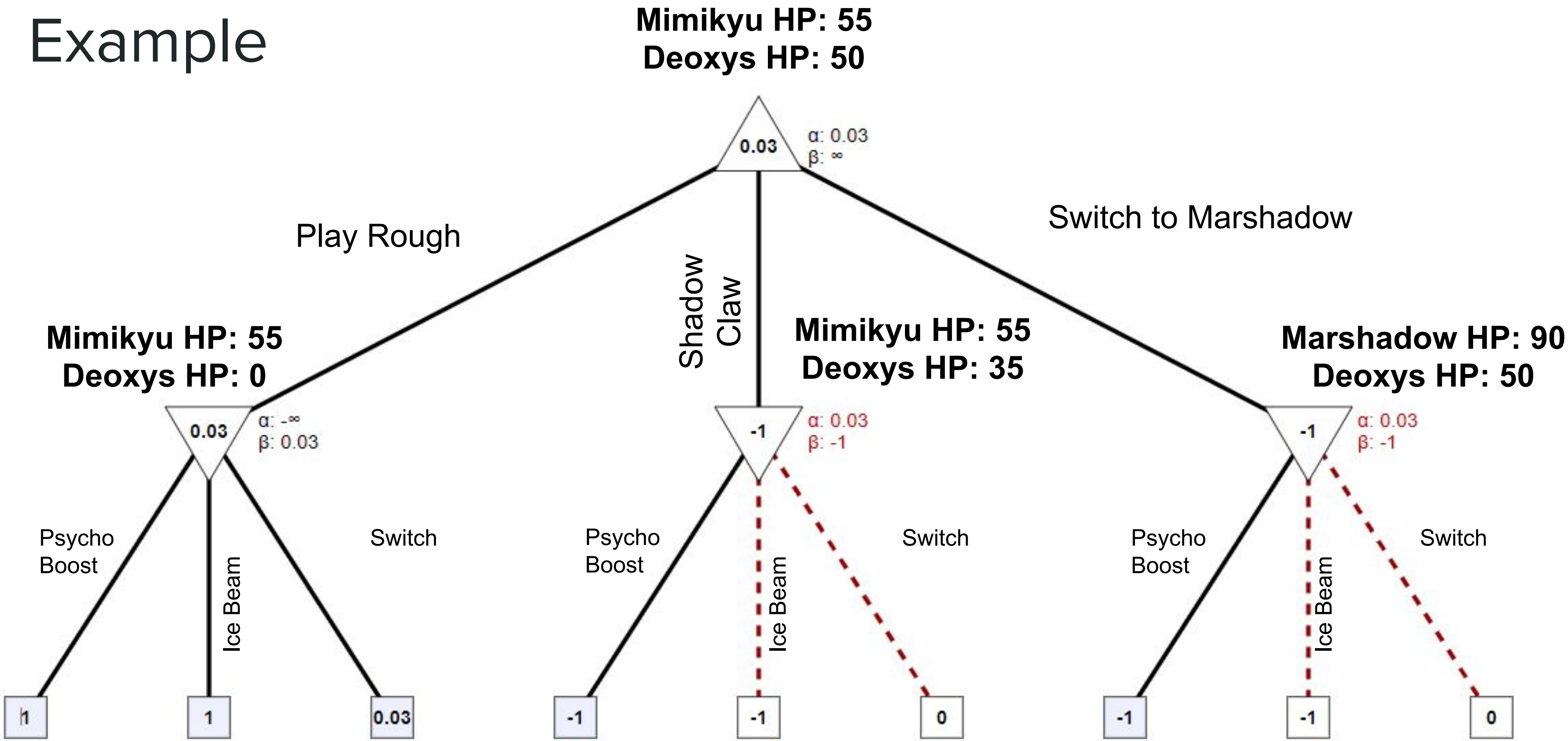
# Real Life Use: Pokemon

- I created a AI simulation that simulates a competitive battling scenario
  - Used Java
  - Dictionary of Pokemon
  - Dictionary of Moves
  - Battle Game Tree
  - Alpha Beta Pruning to Traverse tree
  - Minimax to Check Alpha Beta
  - 12 different classes

# Example


Mimikyu HP: 55
Deoxys HP: 50
Play Rough
Shadow Claw
Switch to Marshadow
Mimikyu HP: 55
Deoxys HP: 0
Mimikyu HP: 55
Deoxys HP: 35
Marshadow HP: 90
Deoxys HP: 50
Psycho Boost
Ice Beam
Switch
Psycho Boost
Ice Beam
Switch
Psycho Boost
Ice Beam
Switch
1
1
0.03
-1
-1
0
-1
-1
0

# Example



Mimikyu HP: 55
Deoxys HP: 50
0.03
α: 0.03
β: ∞
Play Rough
Shadow Claw
Switch to Marshadow
Mimikyu HP: 55
Deoxys HP: 0
0.03
α: -∞
β: 0.03
Mimikyu HP: 55
Deoxys HP: 35
-1
α: 0.03
β: -1
Marshadow HP: 90
Deoxys HP: 50
-1
α: 0.03
β: -1
Psycho Boost
Ice Beam
Switch
Psycho Boost
Ice Beam
Switch
Psycho Boost
Ice Beam
Switch
1
1
0.03
-1
-1
0
-1
-1
0