Building a simple Chess Engine

Jens Mueller
Computer Science (Algorithms)
Bocconi University
June 12, 2020

Agenda

01 The Game of Chess

02 Setting up the Code

03 Building the Engine

- Minimax Algorithm
- Alpha-Beta Pruning
- Further Improvements

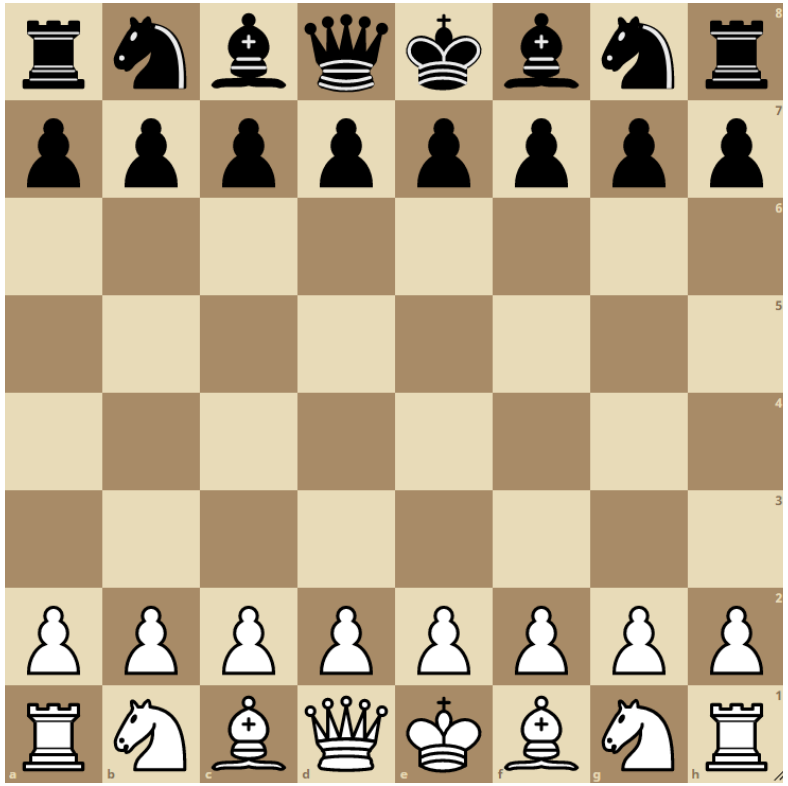
The Game of Chess

What is Chess?

- Two-player, zero-sum strategy game
- Turn-based
- Perfect information
- Played on 8x8 board
- Different piece types
- Objective: Checkmate opponents king
- Multiple ways to win or draw

Questions

- Infinitely many possible games?
- Is chess solved?



Initial position

The Game of Chess

What is Chess?

- Two-player, zero-sum strategy game
- Turn-based
- Perfect information
- Played on 8x8 board
- Different piece types
- Objective: Checkmate opponents king
- Multiple ways to win or draw

Questions

- Infinitely many possible games?
- Is chess solved?

Checkmate

The Game of Chess

What is Chess?

- Two-player, zero-sum strategy game
- Turn-based
- Perfect information
- Played on 8x8 board
- Different piece types
- Objective: Checkmate opponents king
- Multiple ways to win or draw

Questions

- Infinitely many possible games? **No** (drawing rules)
- Is chess solved? **No**, computer chess is booming!

Checkmate

Setting up the Code

Python-Chess library

- Rules implementation
- Objects:
  - Board
  - Piece
  - Move
- Methods:
  - Legal-move generation
  - Determine game end

Project files

games.py

Functions:
- game()
- match()
- tournament()

players.py

Classes:
- Human
- Engine

other

- play.py
- analysis.py
- utils.py

Simple Heuristics

Random moves

- Make a random legal move

Attacking moves

- Look for checkmate → check → capture

Limiting moves

- Minimize the opponent's legal moves

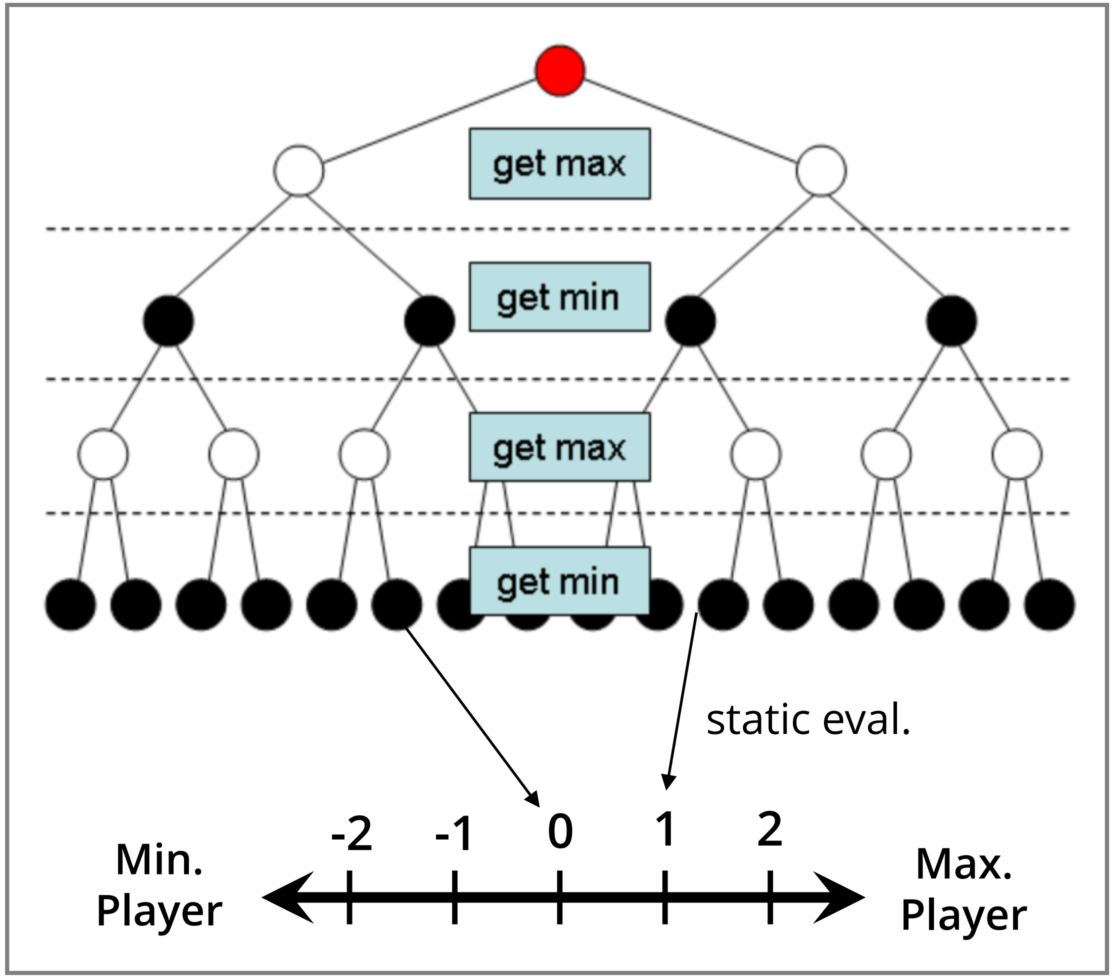
		Black		
		Random	Attacking	Limiting
White	Random	7:82:11	0:79:21	1:51:48
	Attacking	30:69:1	7:89:4	6:81:13
	Limiting	43:57:0	9:85:6	4:84:12

Problem: These engines only see one move ahead. → **Solution: Minimax**

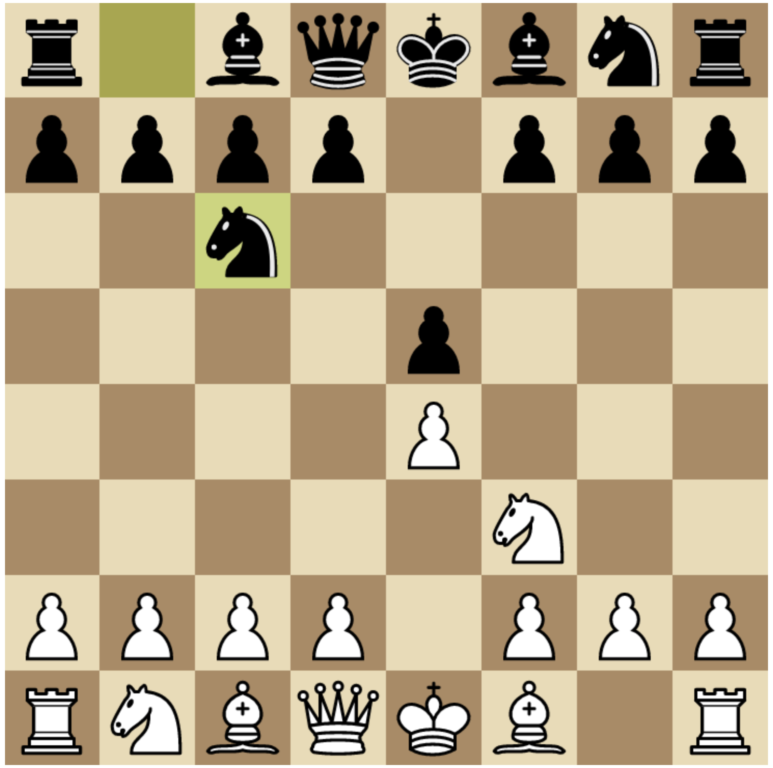
Minimax Algorithm

What is the Minimax Algorithm?

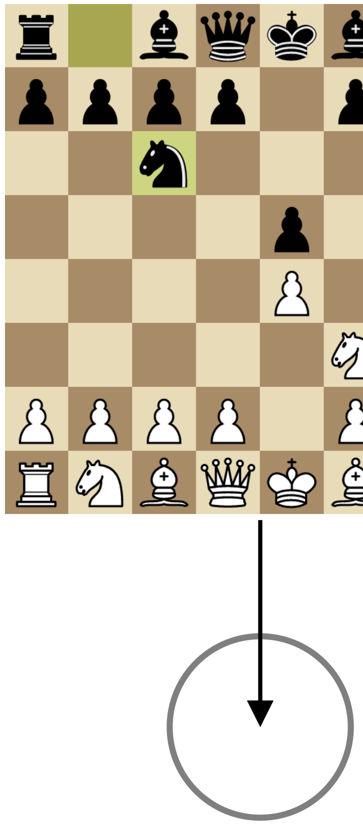
- Adversarial search method from game theory for minimizing the maximum possible loss
- Optimal? Yes, assuming best opponent play for deterministic, fully observable two-player-games → Nash equilibrium
- More complicated scenarios possible
- Requirements: Initial state, Operators, Terminal test, Evaluation function



Minimax - Demo

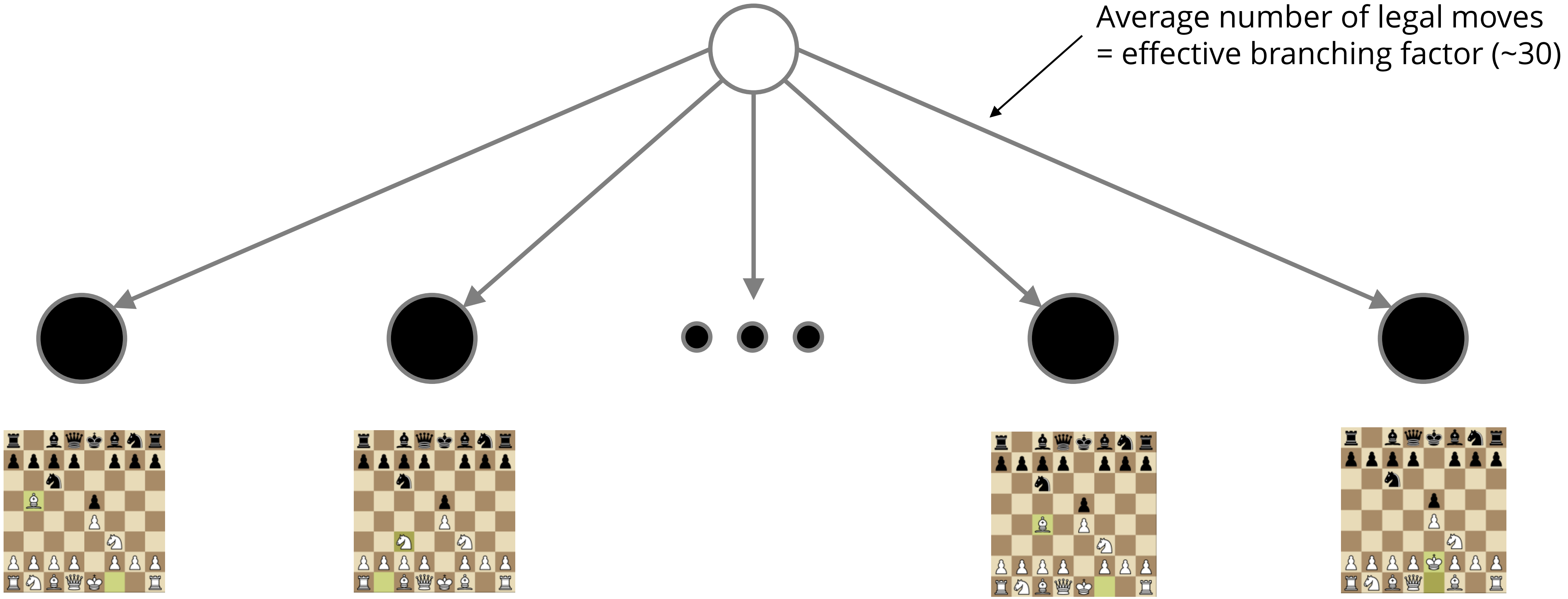


Minimax - Demo



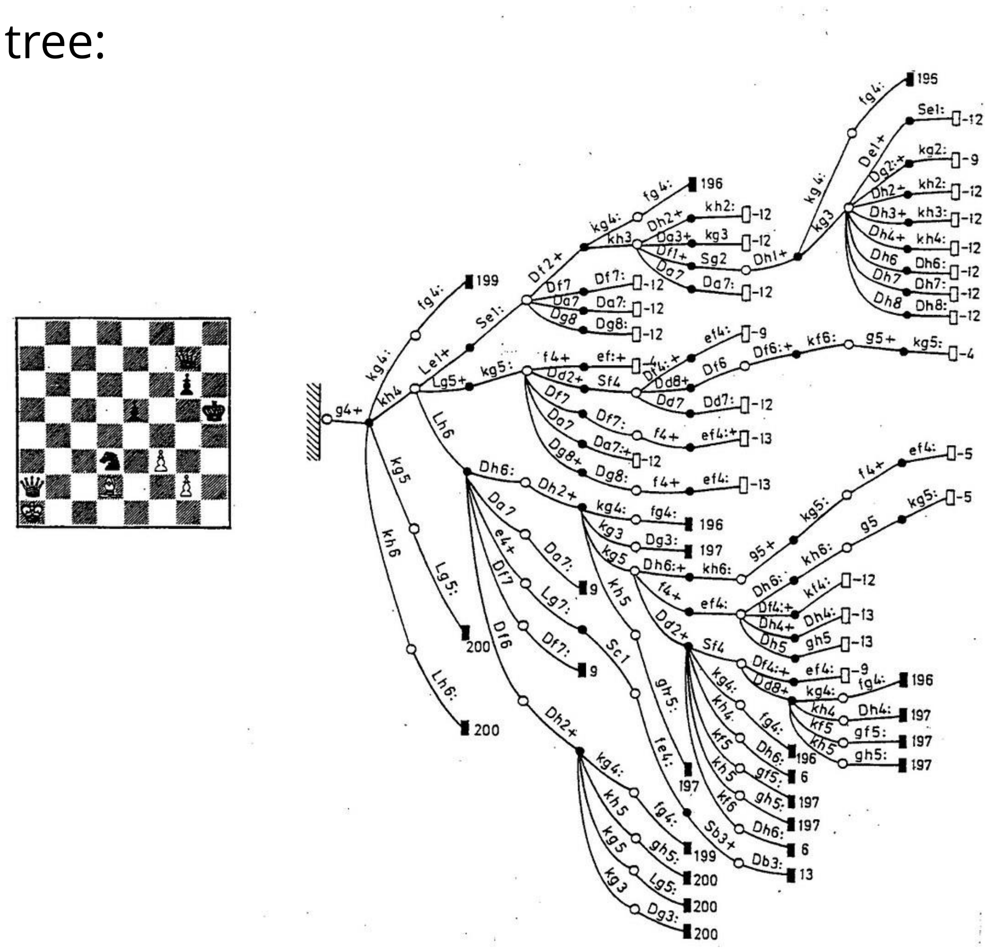
Represent position as node with the color indicating the side to move

Minimax - Demo

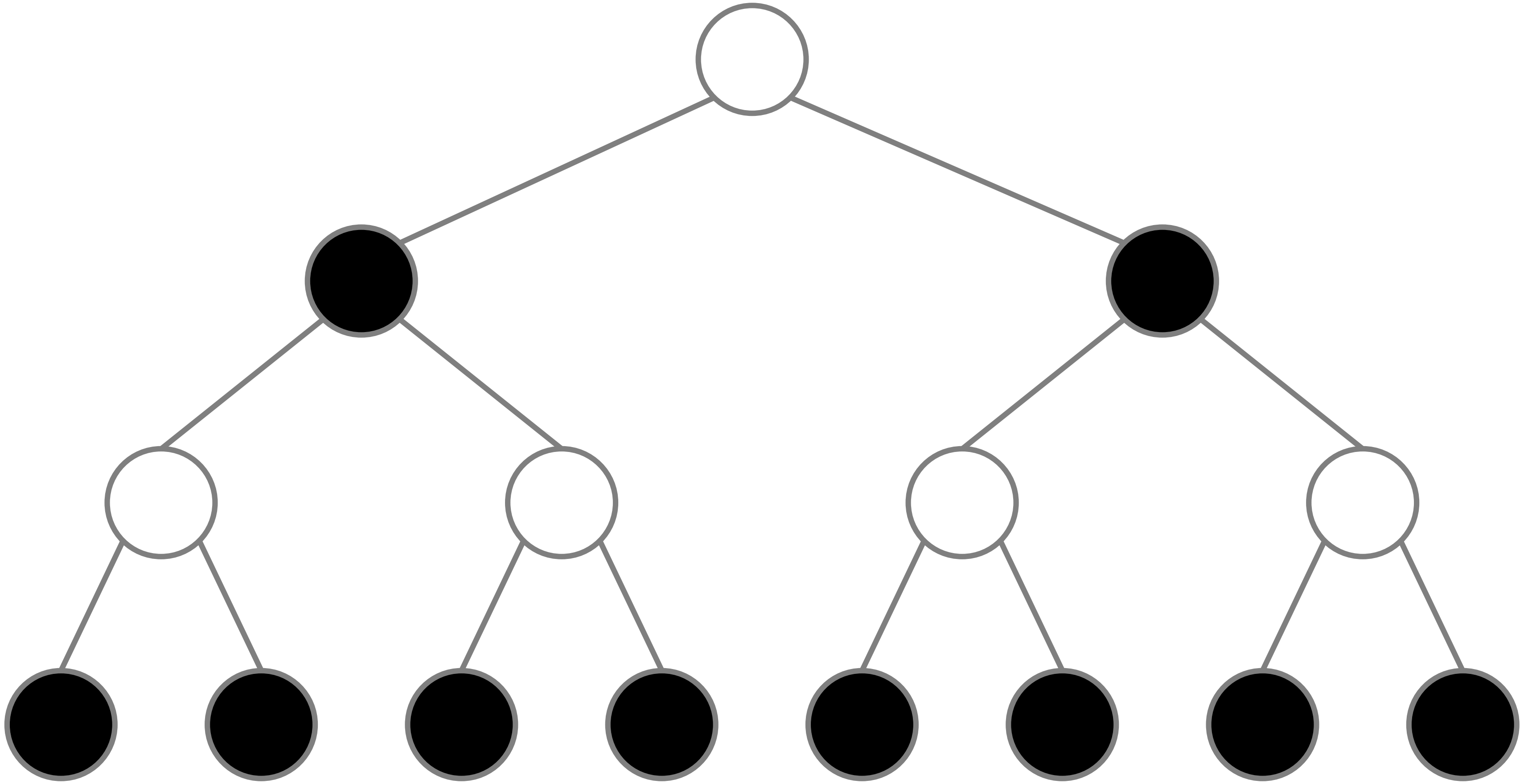


Minimax - Demo

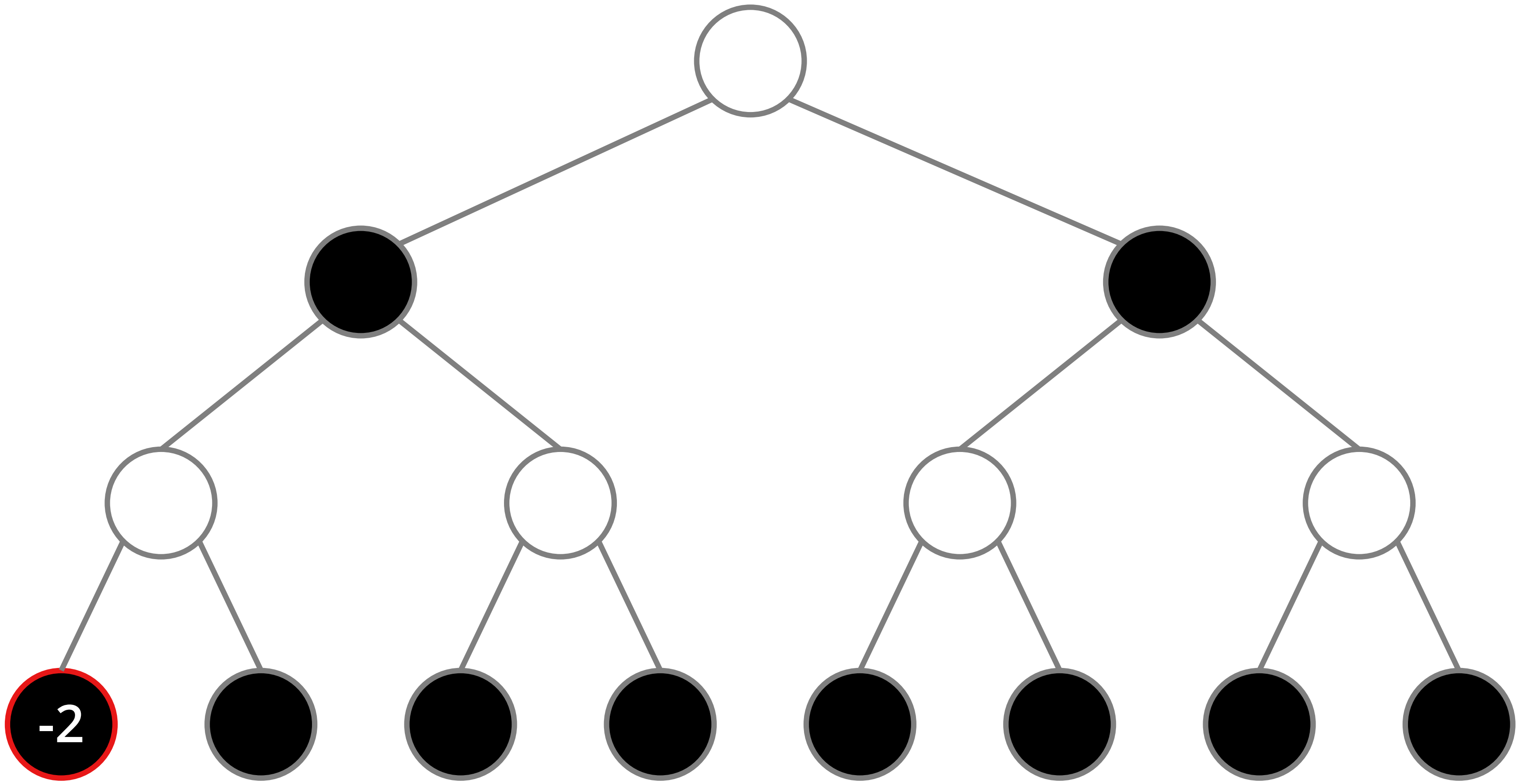
Exemplary chess game tree:



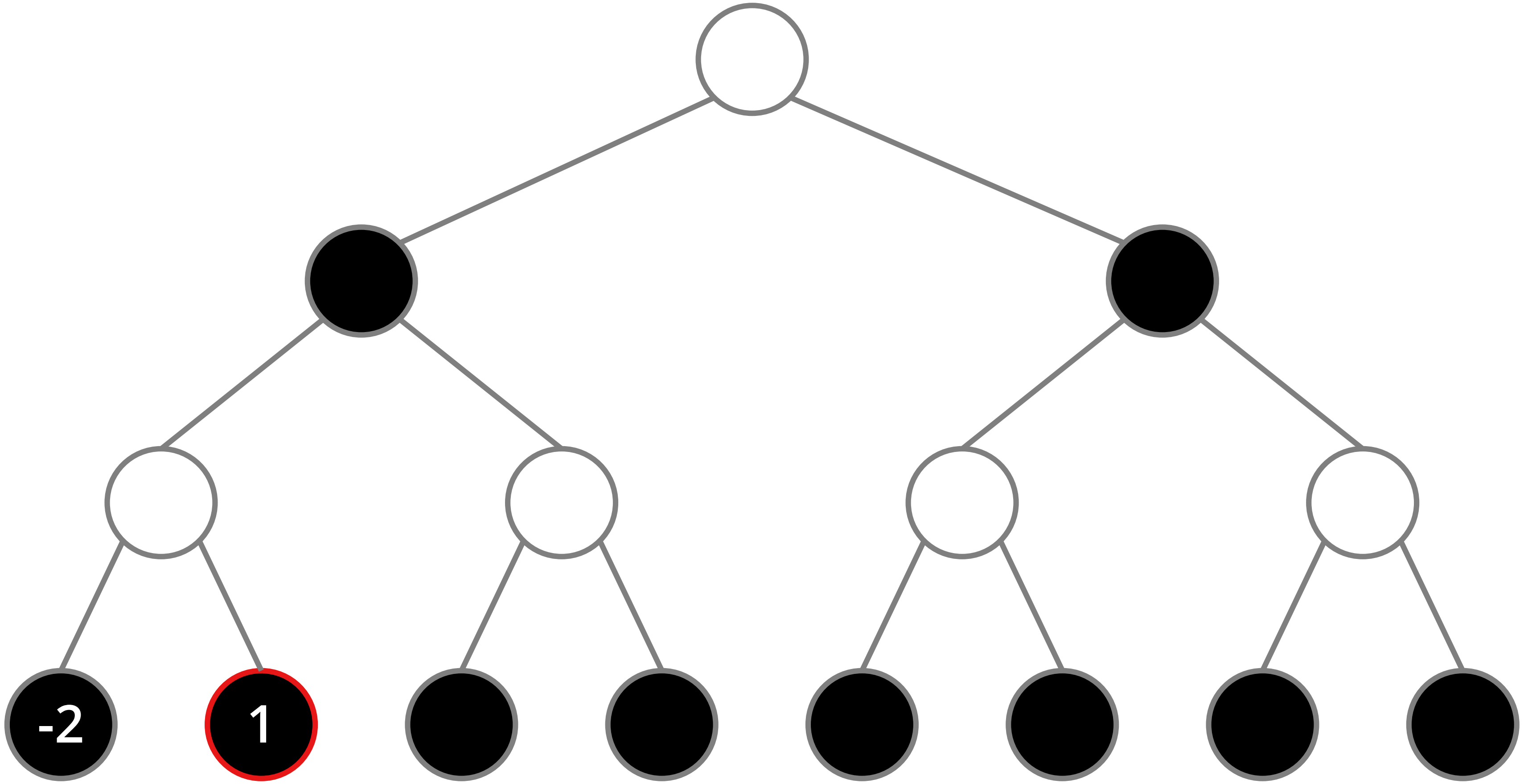
Minimax - Demo



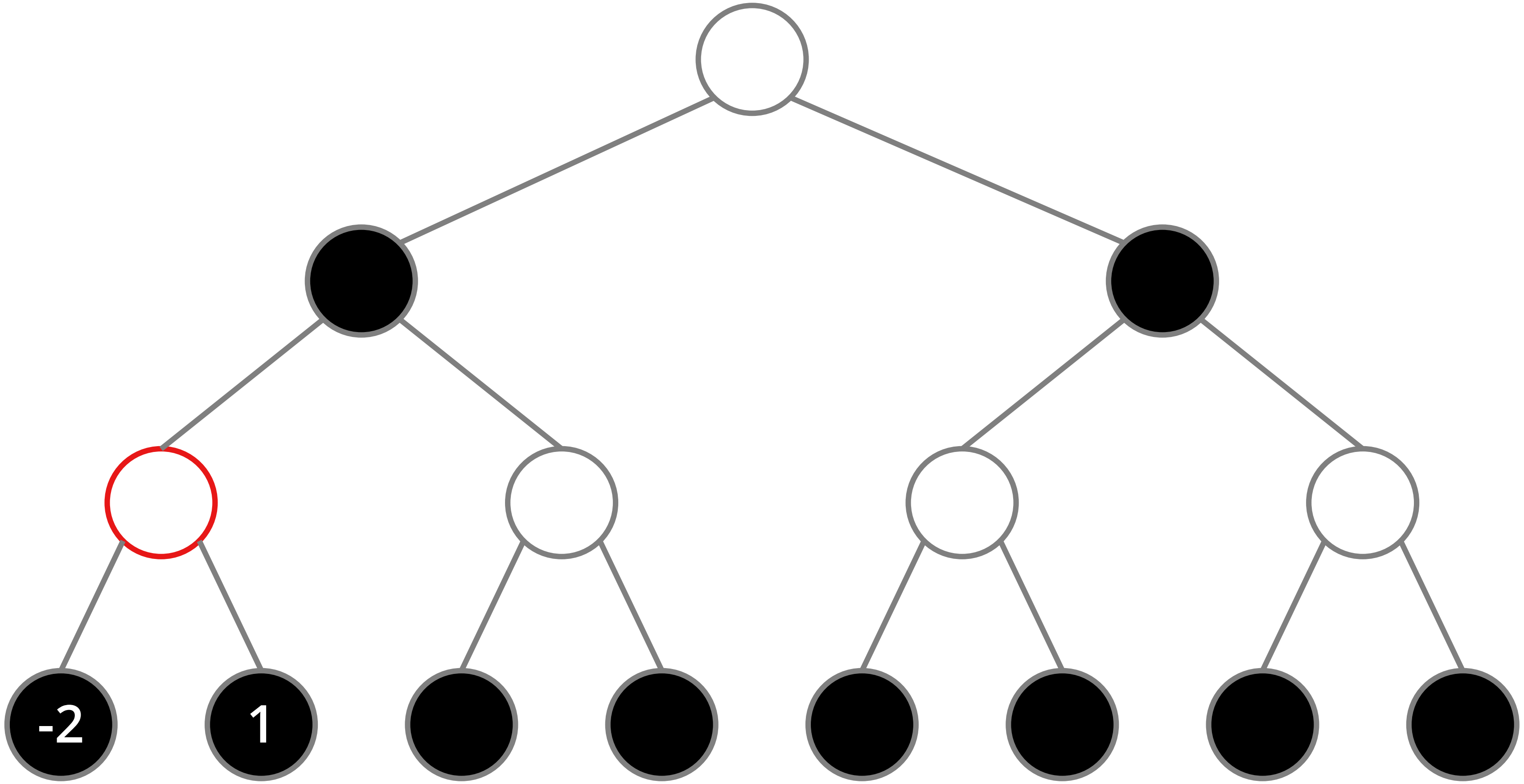
Minimax - Demo



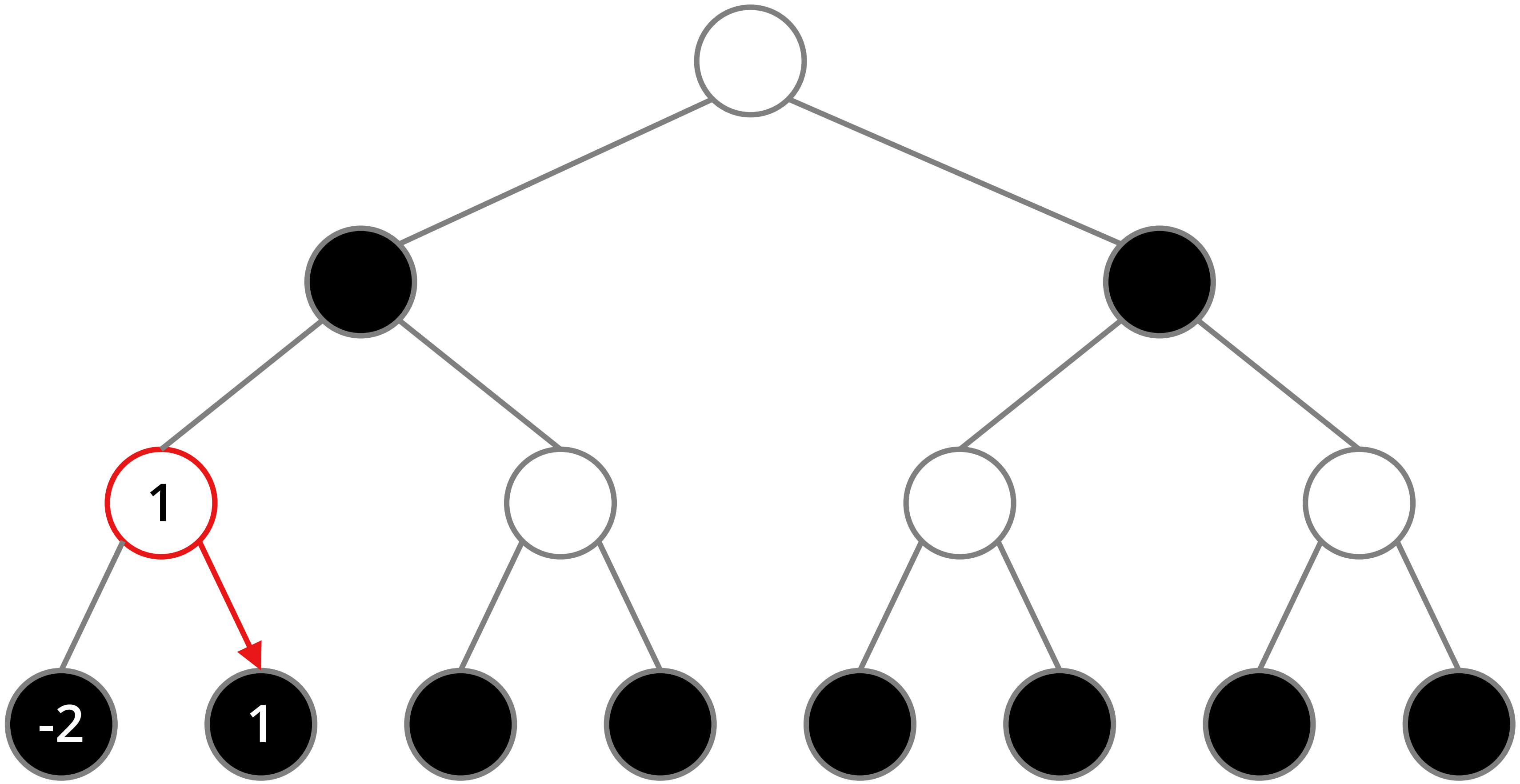
Minimax - Demo



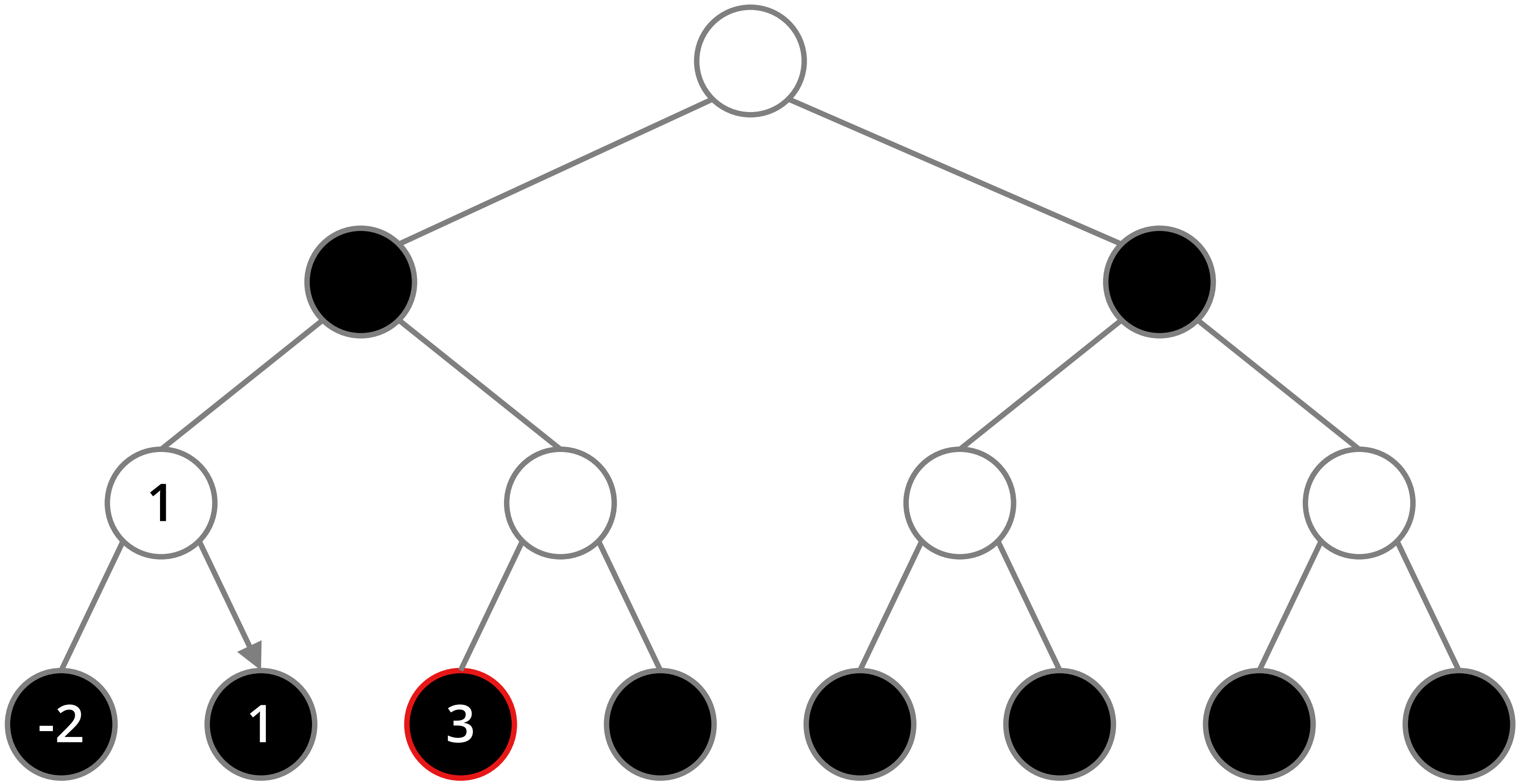
Minimax - Demo



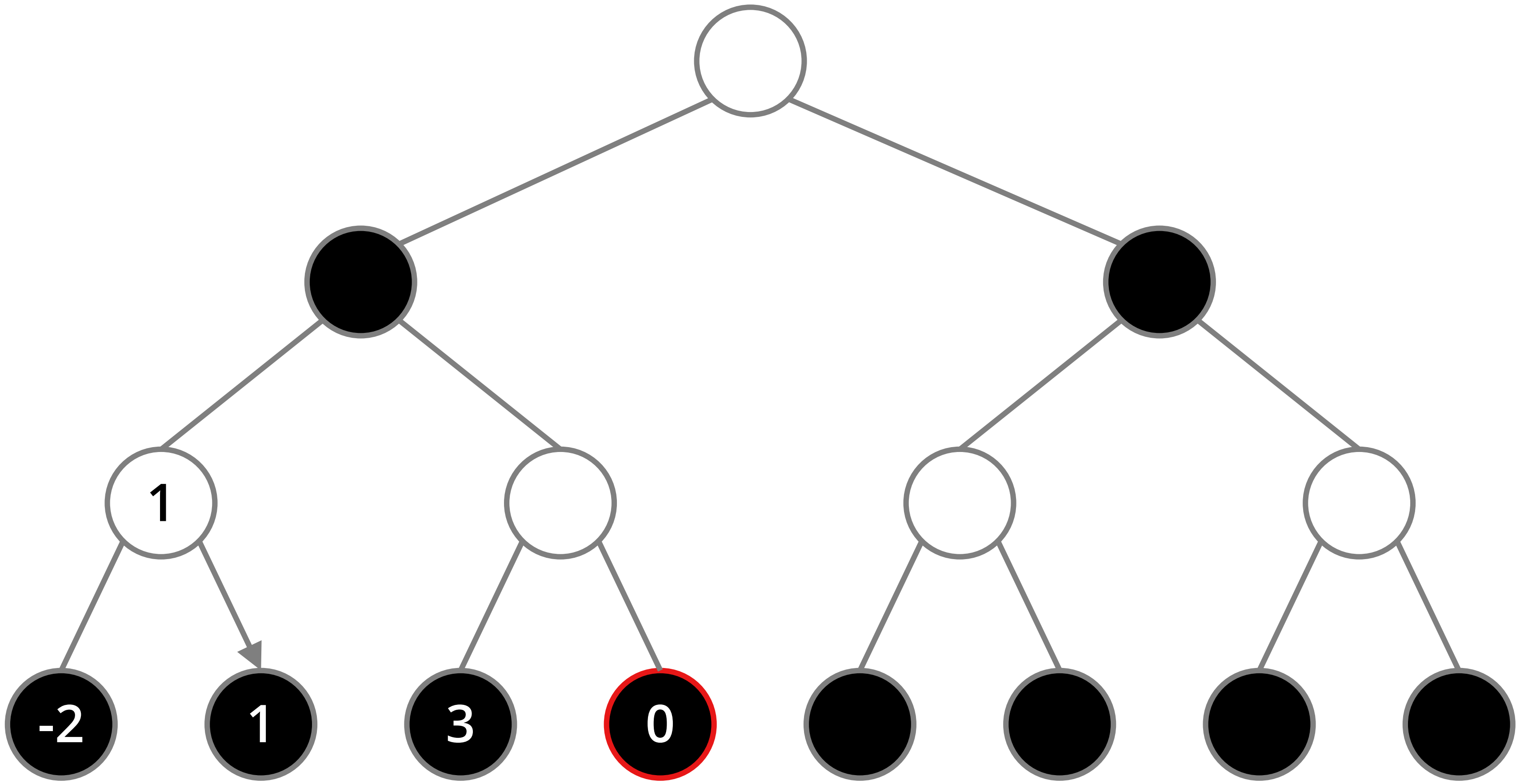
Minimax - Demo



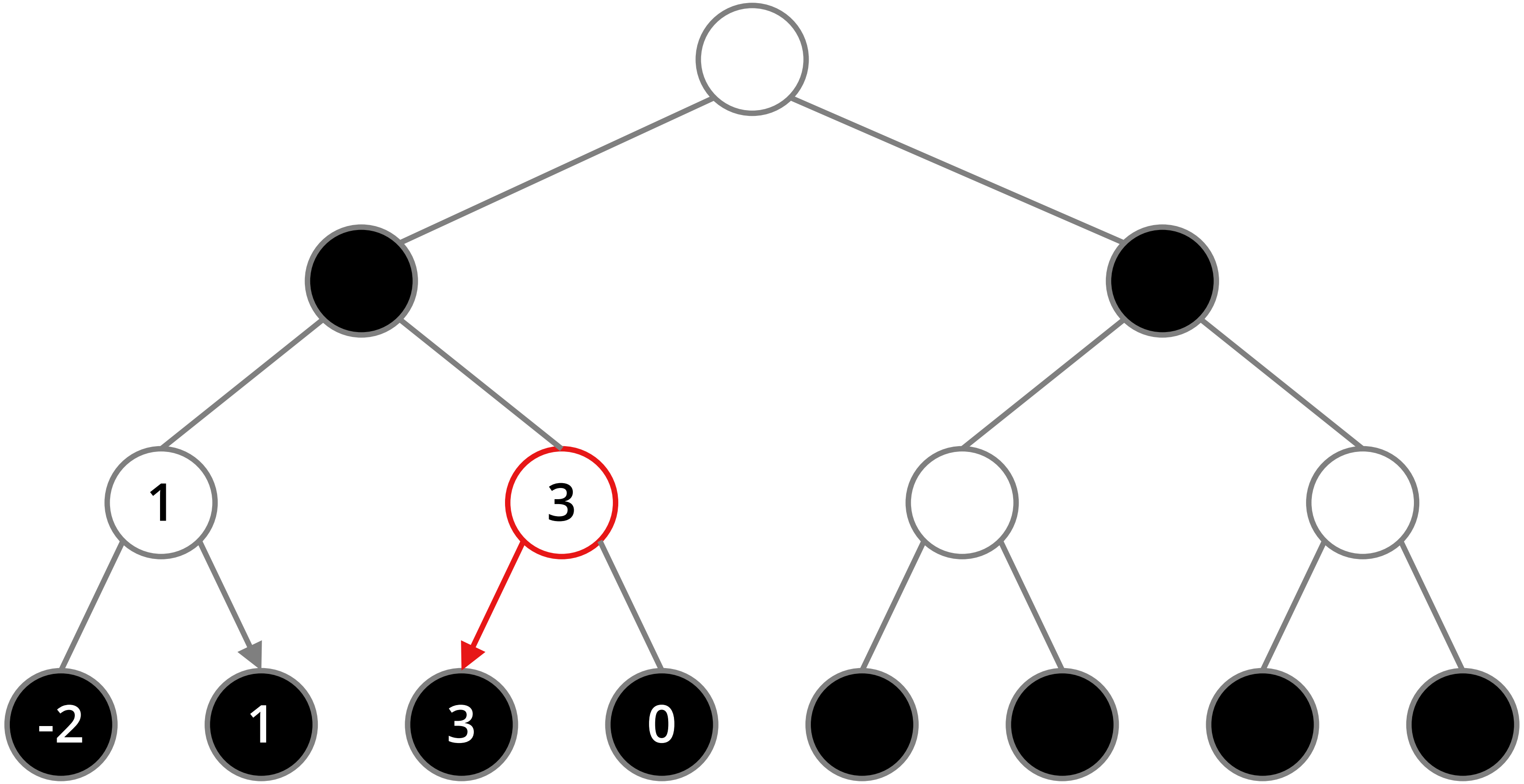
Minimax - Demo



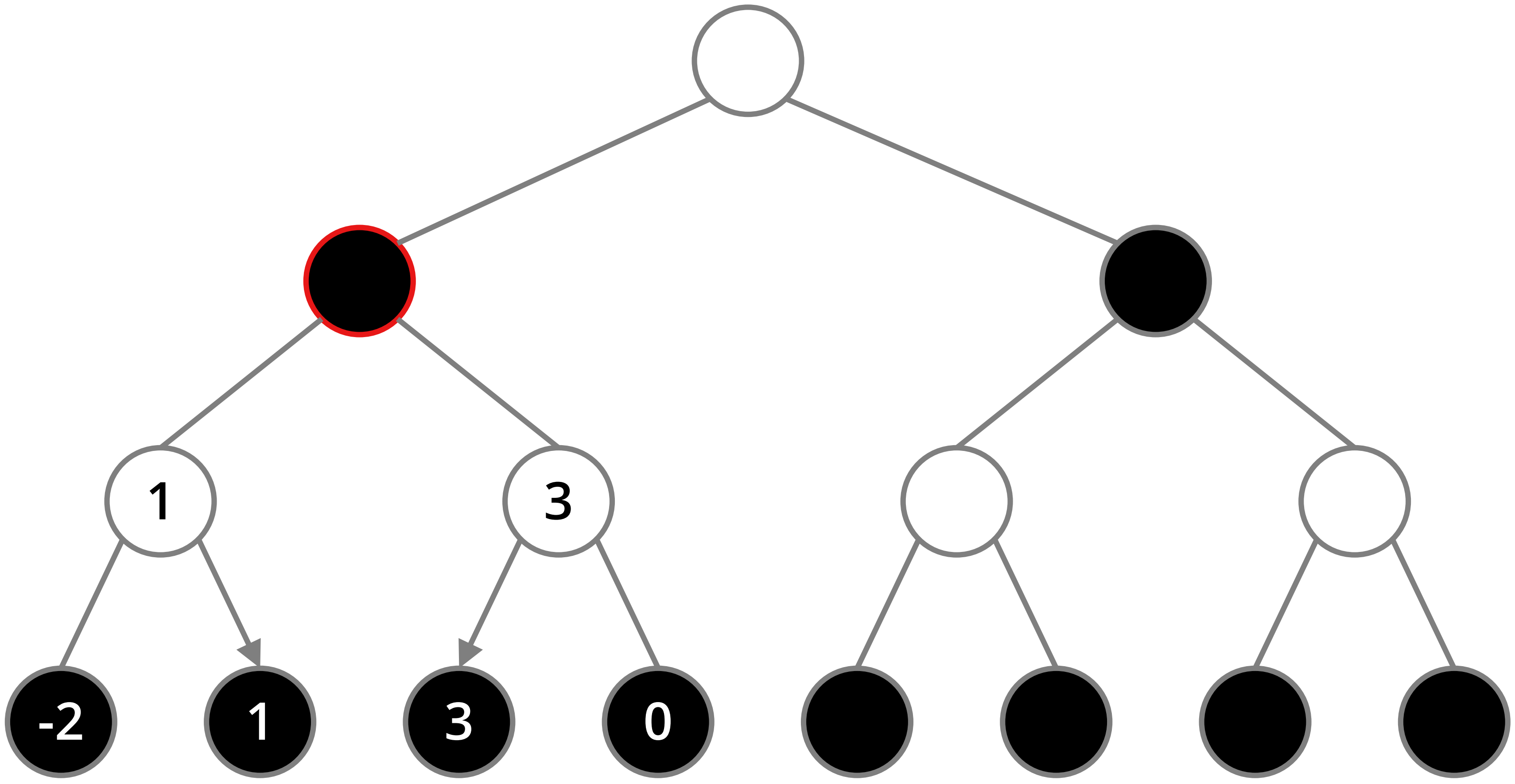
Minimax - Demo



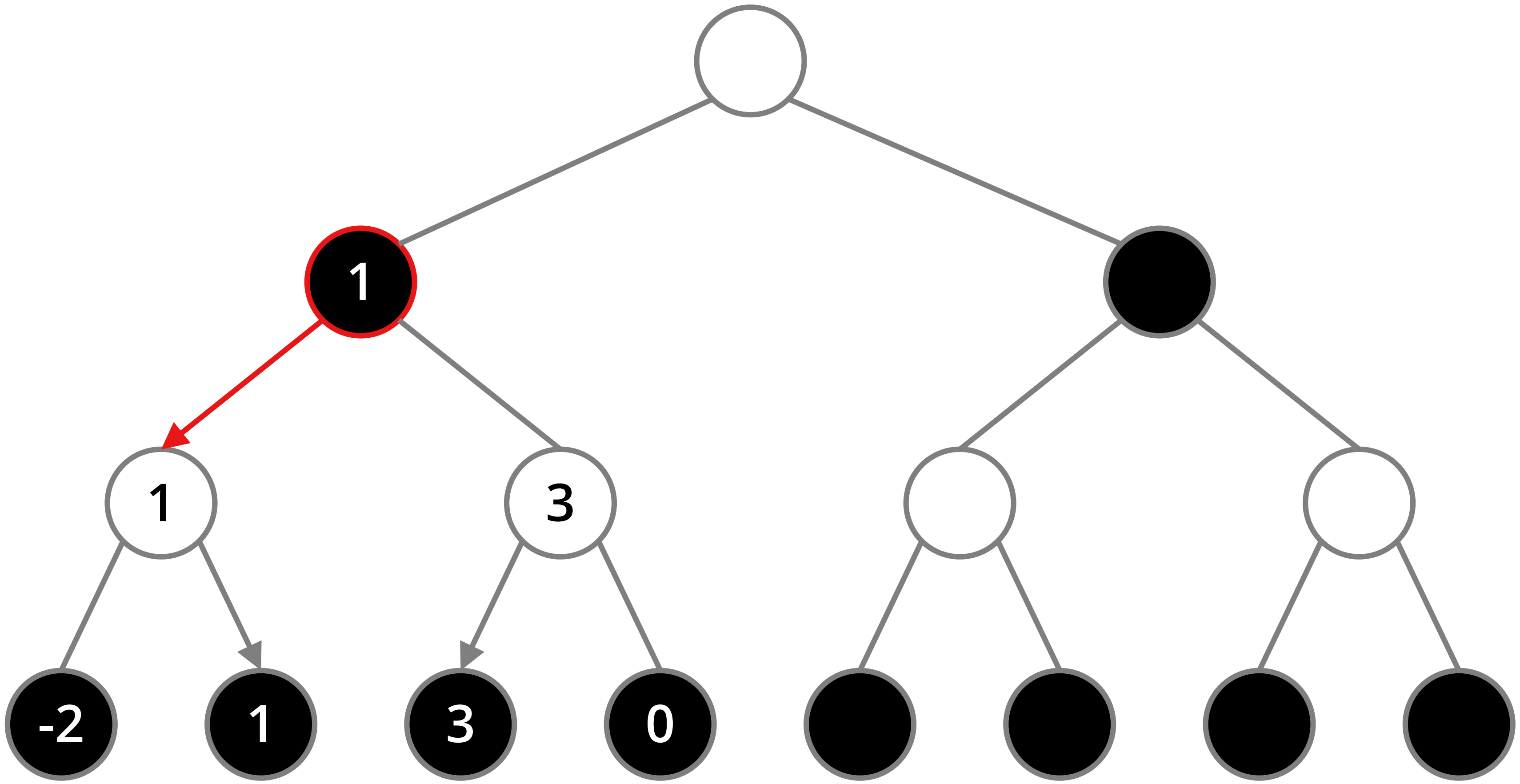
Minimax - Demo



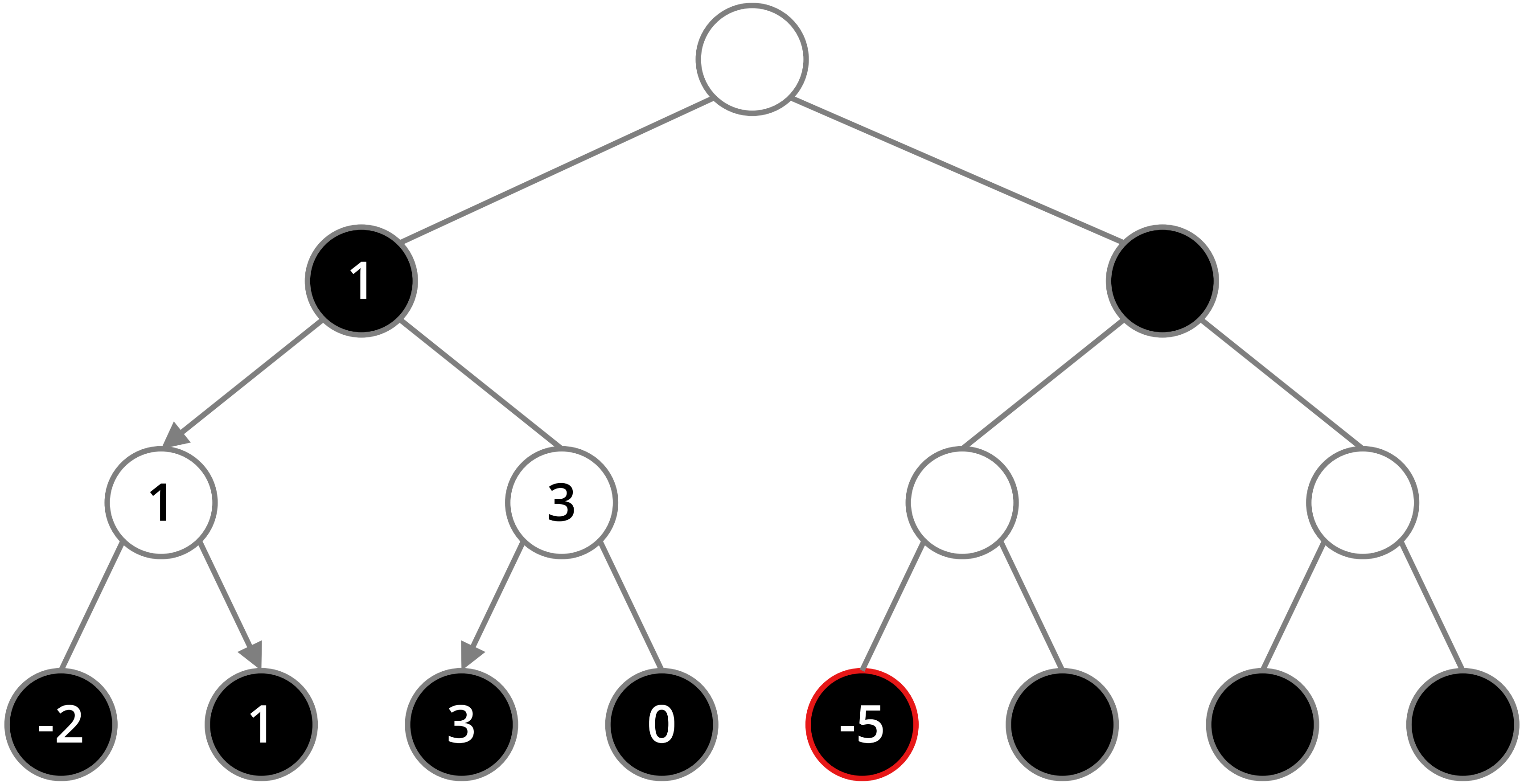
Minimax - Demo



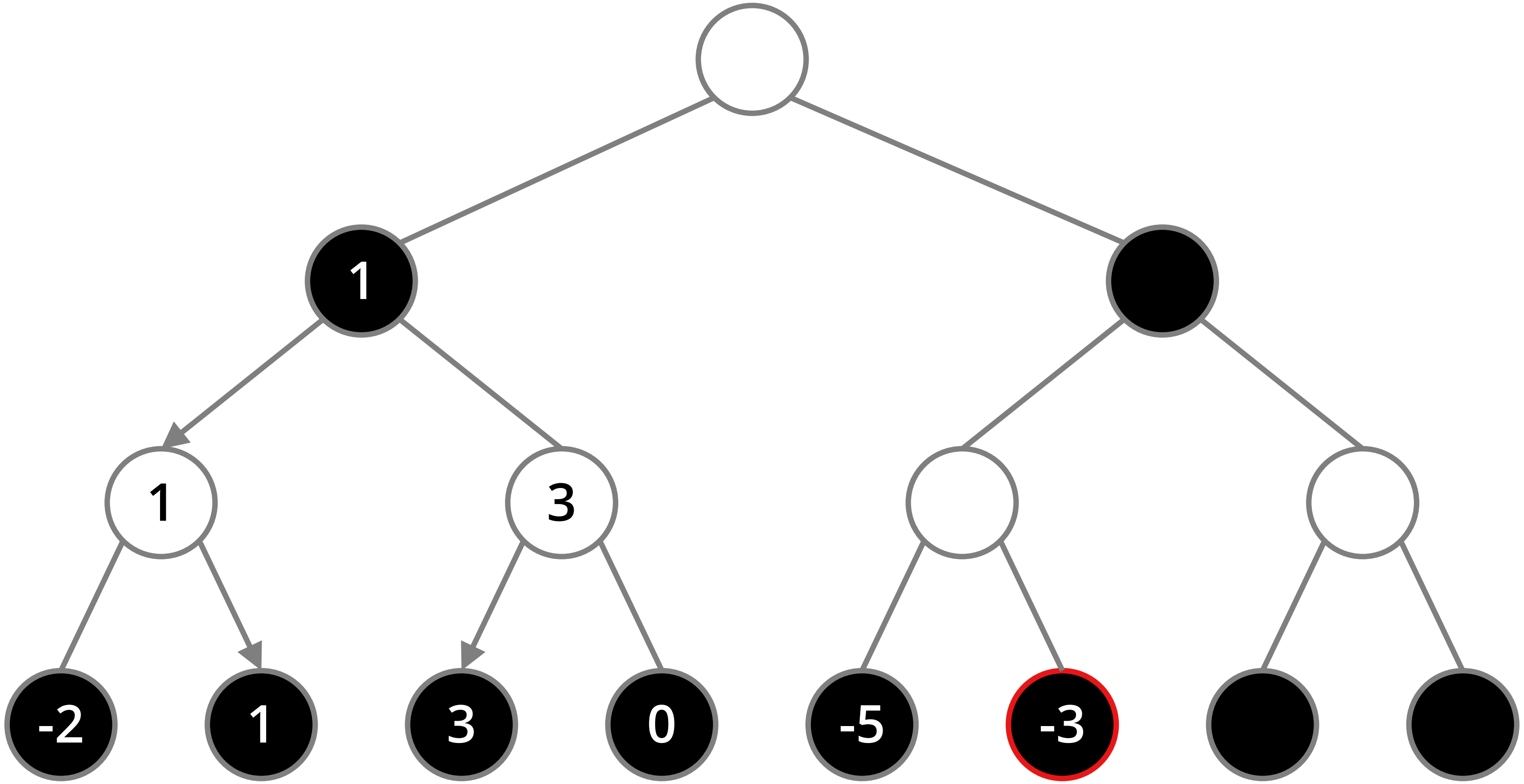
Minimax - Demo



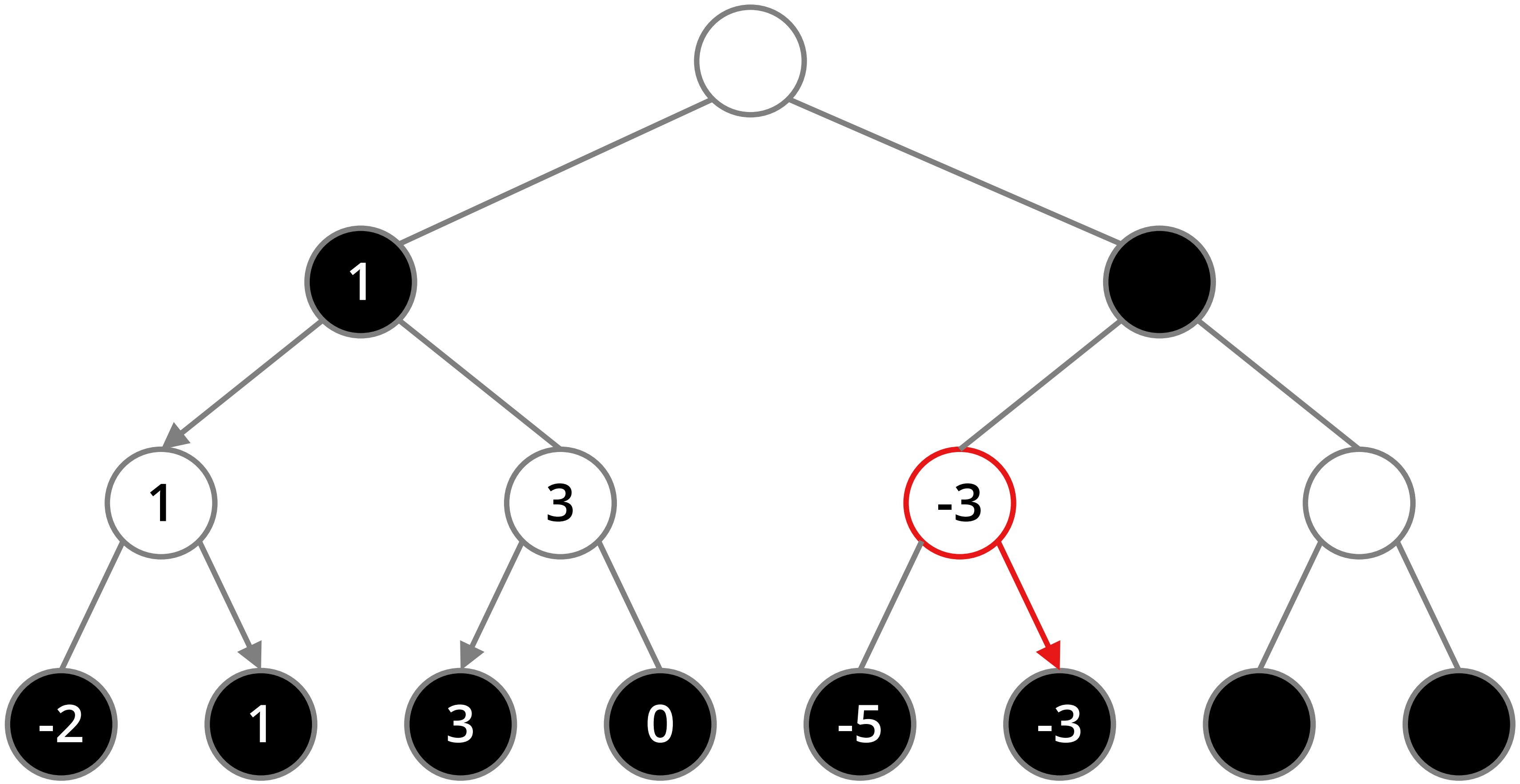
Minimax - Demo



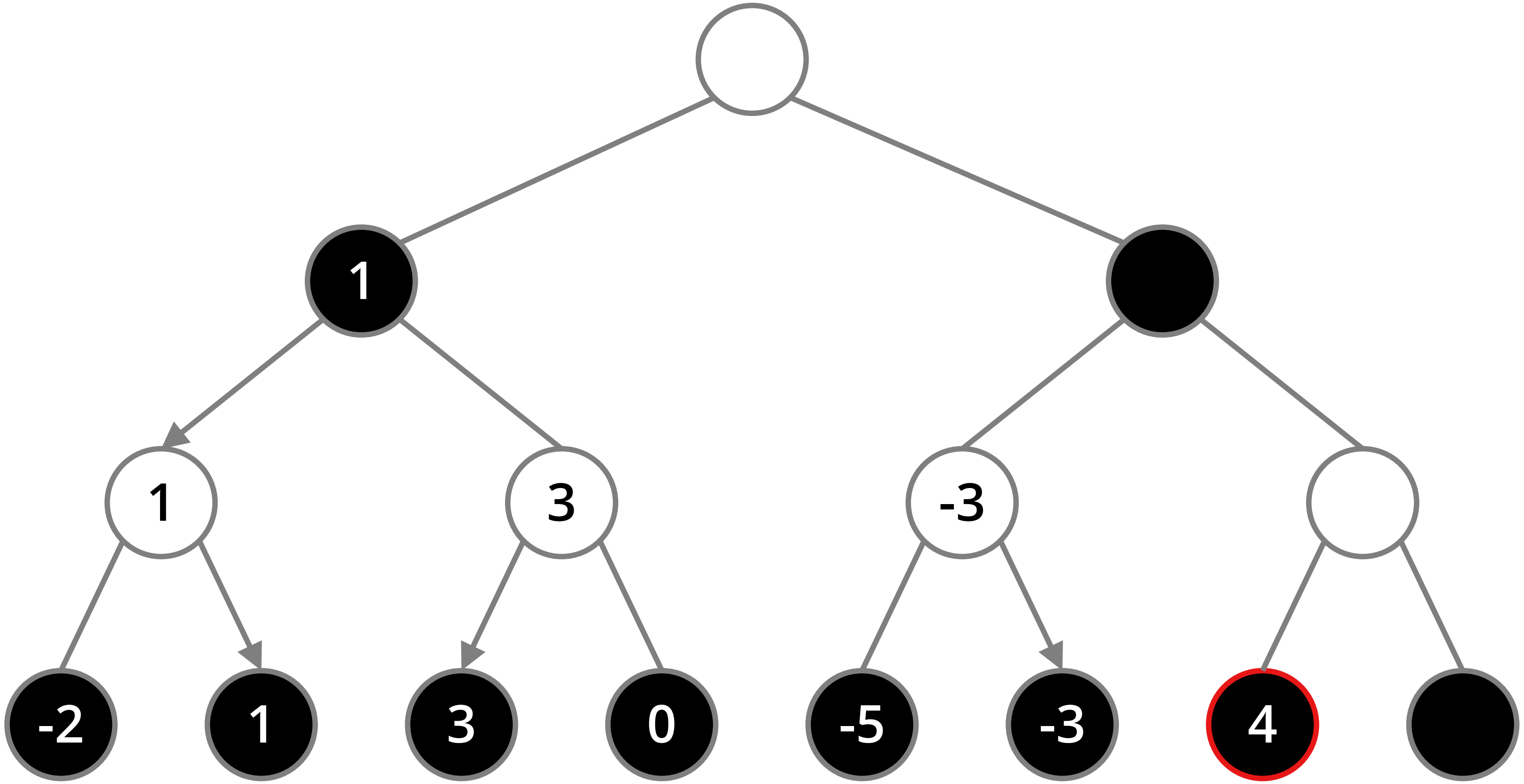
Minimax - Demo



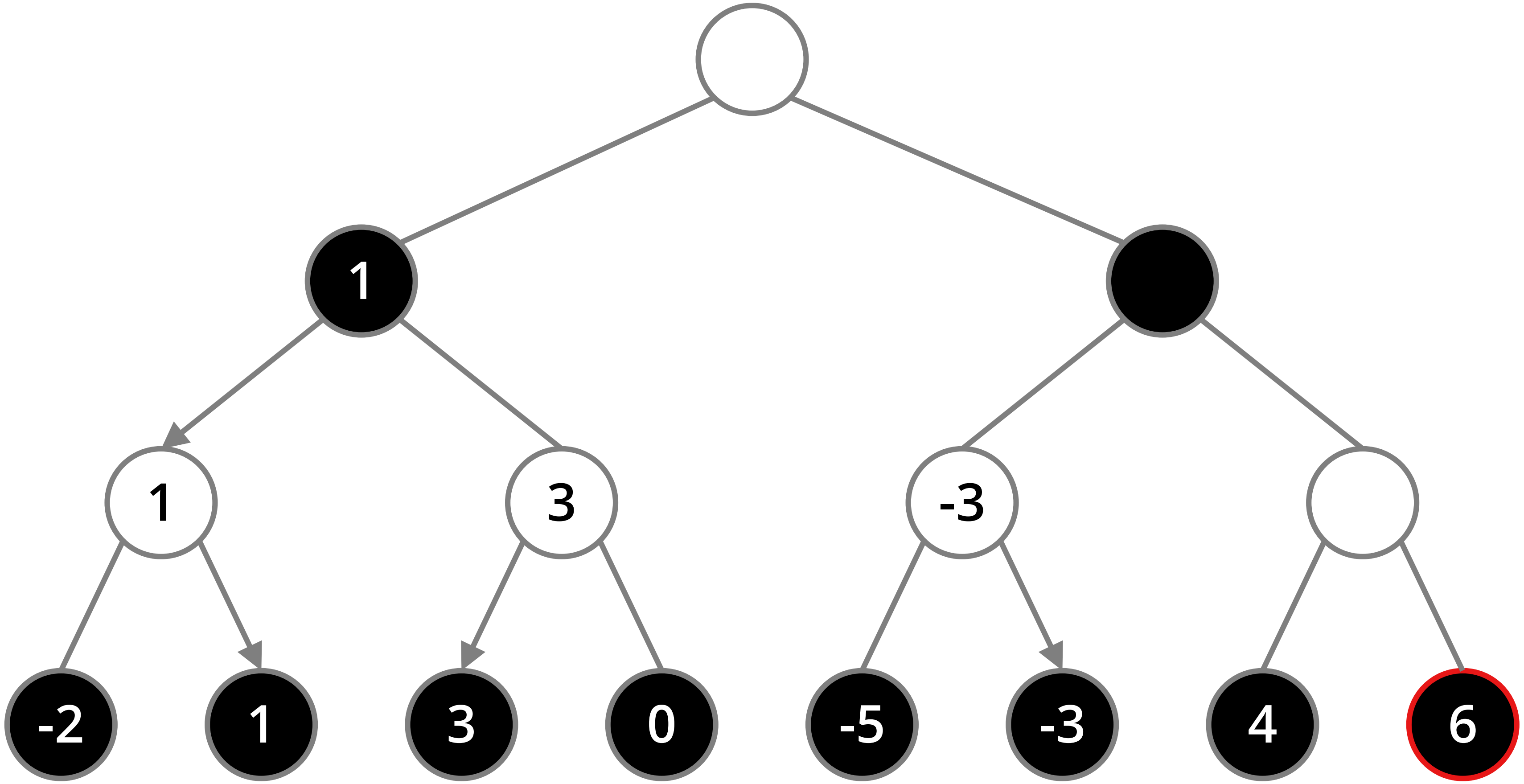
Minimax - Demo



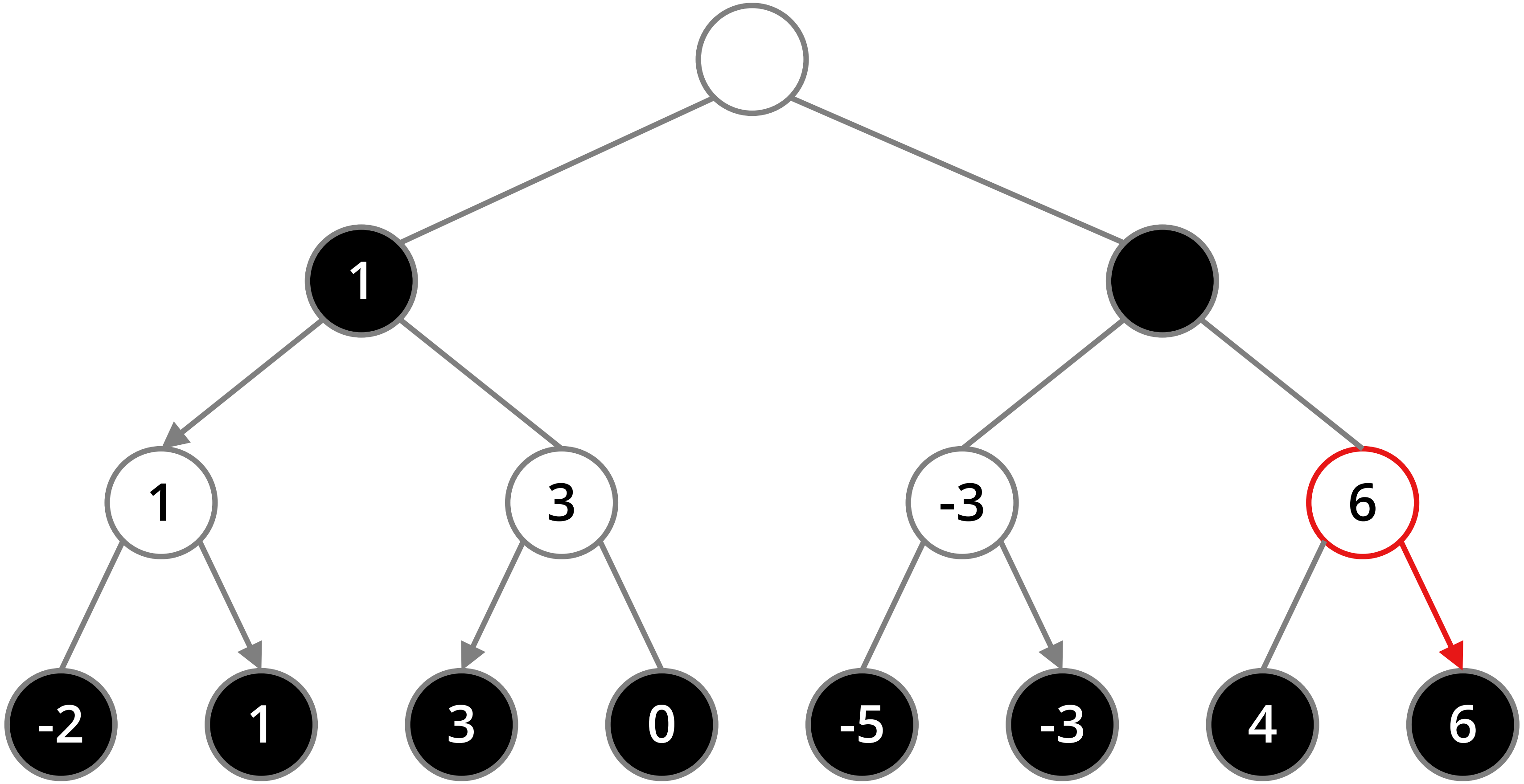
Minimax - Demo



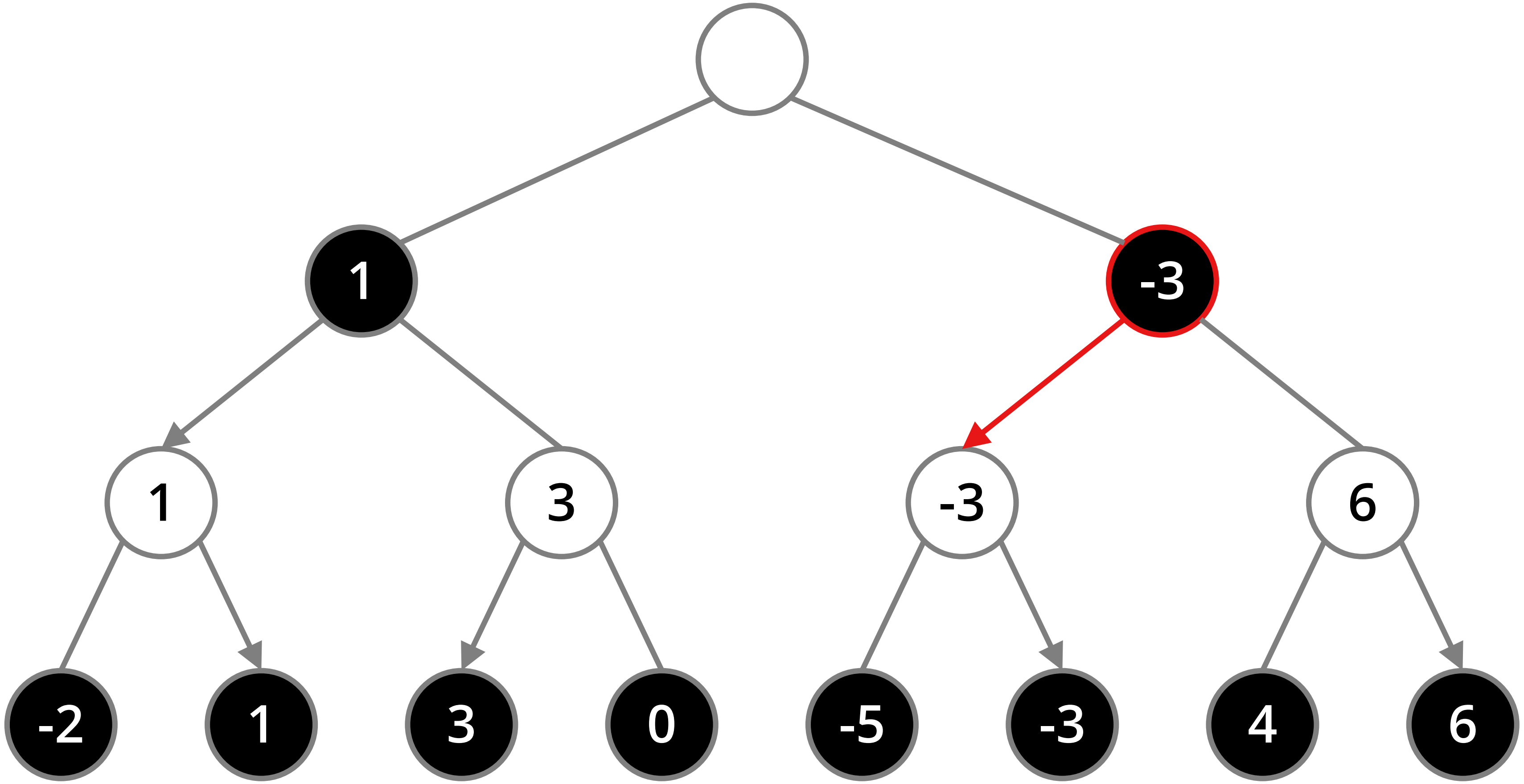
Minimax - Demo



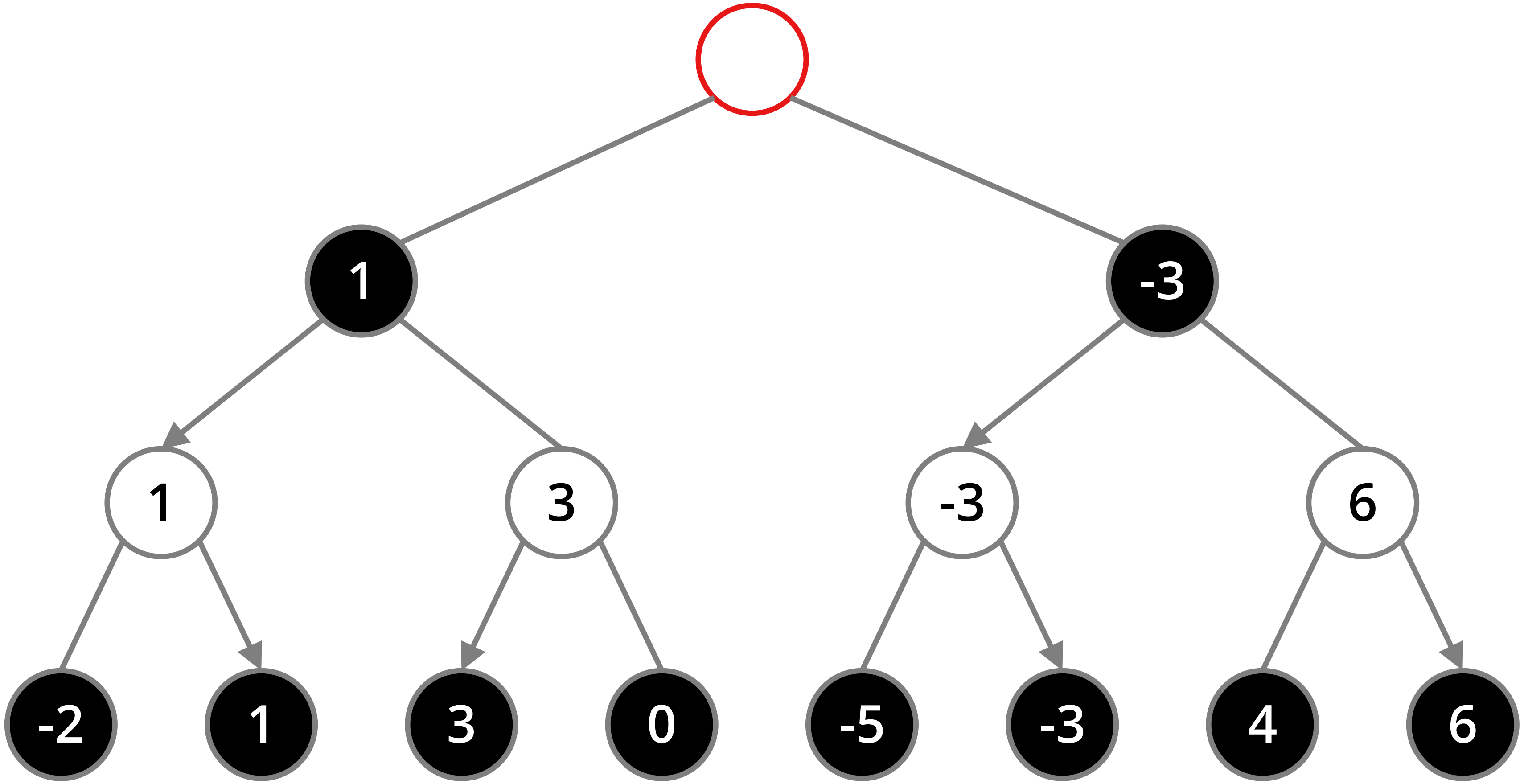
Minimax - Demo



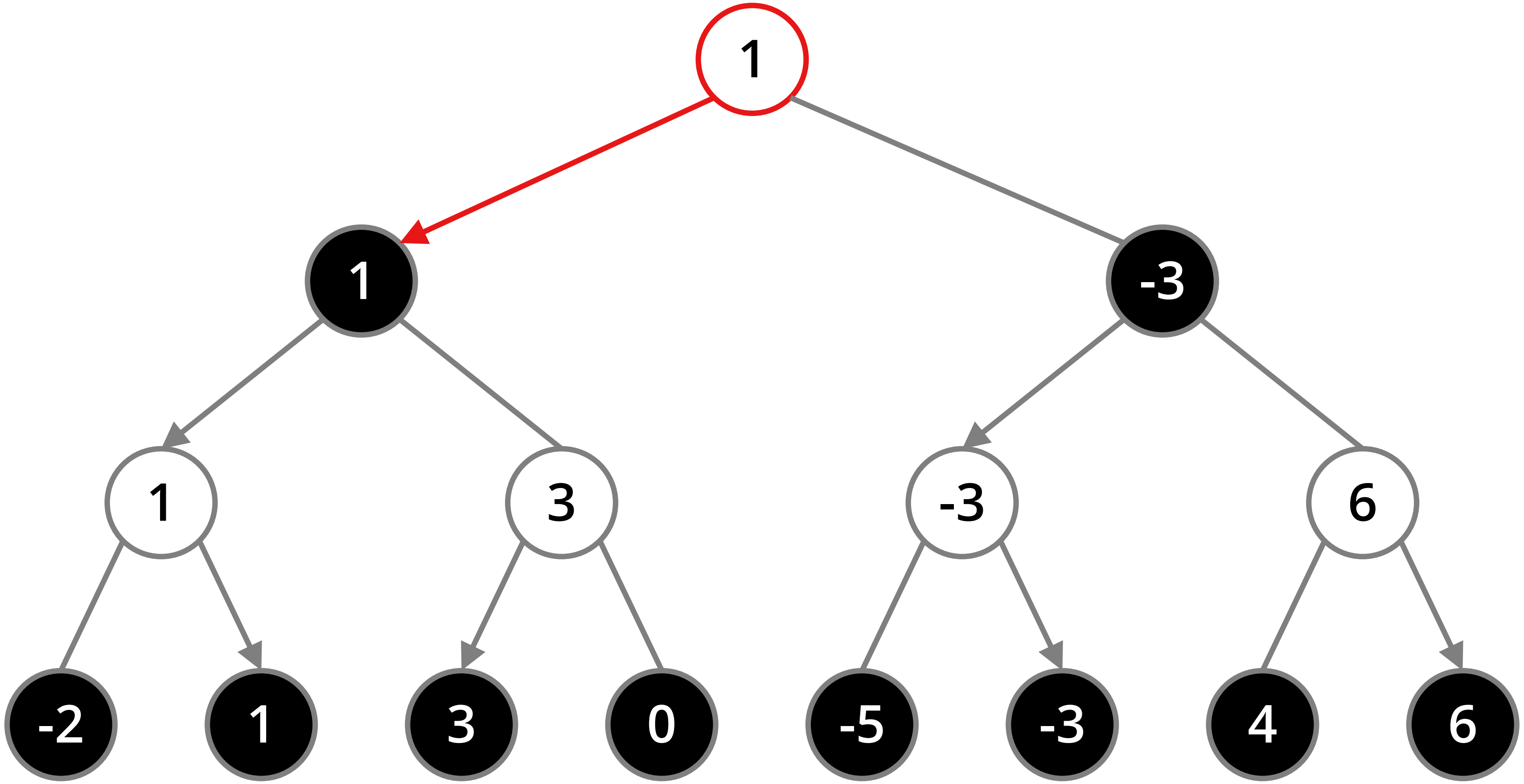
Minimax - Demo



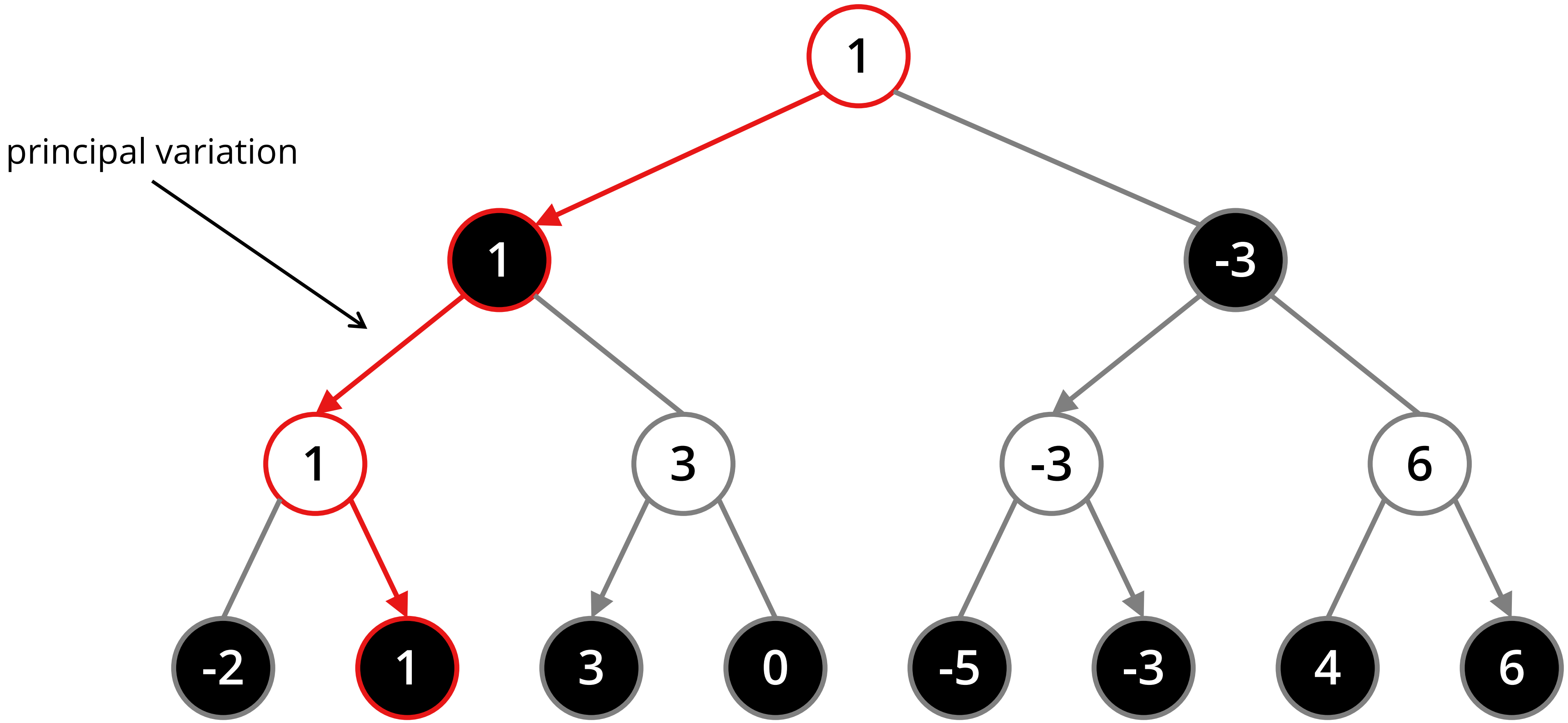
Minimax - Demo



Minimax - Demo



Minimax - Demo



Minimax - Implementation

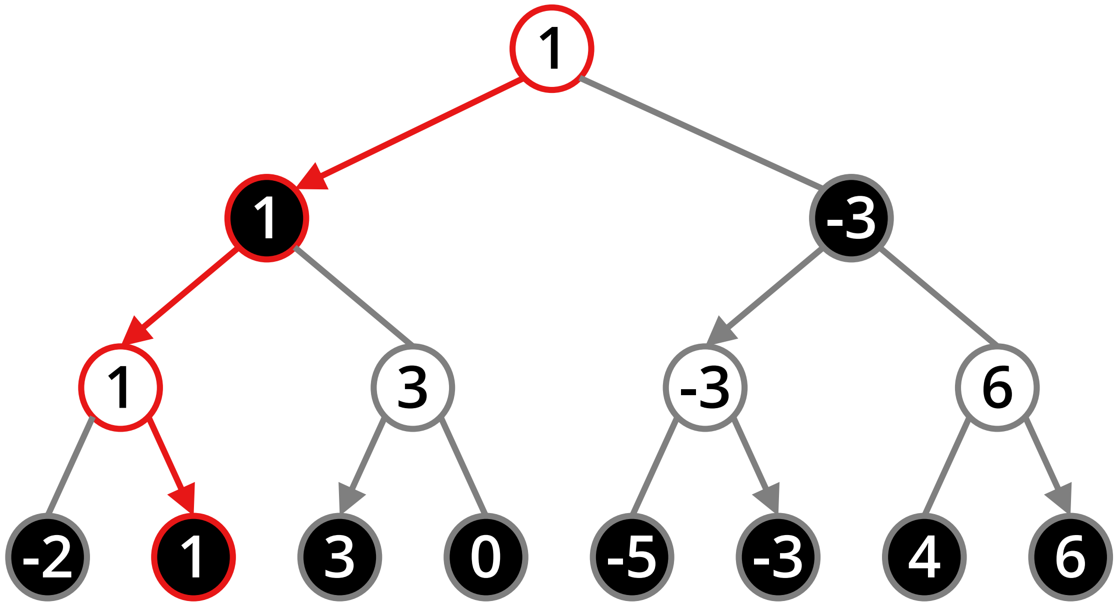
```
def minimax(board, depth):  *

        # check terminal condition
        if board.is_game_over() or depth == 0:
                return evaluate(board)

        #maximizing player
        if board.turn: # white to move
                max_eval = float('-Inf')
                for move in board.legal_moves:
                        evaluation = minimax(board, depth-1)
                        max_eval = max(max_eval, evaluation)
                return max_eval

        # minimizing player
        else: # black to move
                min_eval = float('Inf')
                for move in board.legal_moves:
                        evaluation =  minimax(board, depth-1)
                        min_eval = min(min_eval, evaluation)
                        return min_eval
```

* Pseudocode



Negamax Algorithm

```
def minimax(board, depth): *

    # check terminal condition
    if board.is_game_over() or depth == 0:
        return evaluate(board)

    #maximizing player
    if board.turn: # white to move
        max_eval = float('-Inf')
        for move in board.legal_moves:
            evaluation = minimax(board, depth-1)
            max_eval = max(max_eval, evaluation)
        return max_eval

    # minimizing player
    else: # black to move
        min_eval = float('Inf')
        for move in board.legal_moves:
            evaluation =  minimax(board, depth-1)
            min_eval = min(min_eval, evaluation)
            return min_eval
```

* Pseudocode

What is Negamax?

- Variant of Minimax that relies on zero-sum property
- Both players are maximizing, but sign is switching
- Simplified Implementation

```
def negamax(board, color, depth): *

    # check terminal condition
    if board.is_game_over() or depth == 0:
        return color * evaluate(board)

    # call negamax recursively with switched color
    max_eval = float('-Inf')
    for move in board.legal_moves:
        evaluation = negamax(board, -color, depth-1)
        max_eval = max(max_eval, -evaluation)
    return max_eval
```

Minimax - Analysis

Algorithmic complexity

- Number of nodes grows exponentially
- Impractical to search complete tree → Specify depth m
- Effective branching factor b ~ 30

  → Time complexity: $O(b^m)$

  → Space complexity: $O(bm)$

- Complexity is highly dependent on position
- Lower bound: Shannon number 10^{120}

Speed is determined by two factors:
Evaluation function * # evaluation calls

Search depth	Average # of terminal nodes
1	30
2	900
3	27,000
4	810,000
5	24 million
6	729 million
7	22 billion
8	656 billion
9	20 trillion
10	590 trillion

Evaluation Function

Considerations

Problem: accuracy vs. speed

My approach:

- Step 1: Convert board to Numpy array
- Step 2: Assign piece values and take sum
- The second step can be sped up with Numba

Possible improvements:

- Speed up also the first step (with Cython)
- Positional aspects (e.g. center control, doubled pawns, knight at the rim is dim)
- Dynamic aspects (e.g. centralize king in endgame)

Illustration

Step 1 ▶ **Numpy** Step 2 ▶ **Evaluation**

Speed Improvement with Numba

	Numba		Improve-ment
	No	Yes	
Step 1: Convert_to_numeric()	4.1	N/A	**N/A**
Step 2: Compute_evaluation()	2.9s	0.28s	**90%**
Total: evaluate_board()	7.5s	5.2s	**30%**

Alpha-Beta Pruning

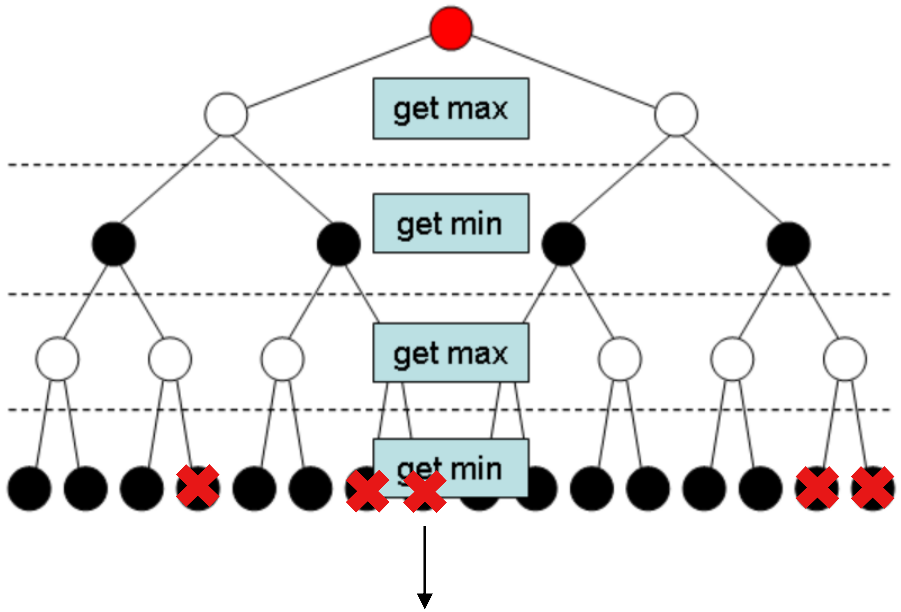
What is it?

- Avoid processing subtrees that have no effect on the search result
- Preserves completeness and optimality of minimax

How does it work?

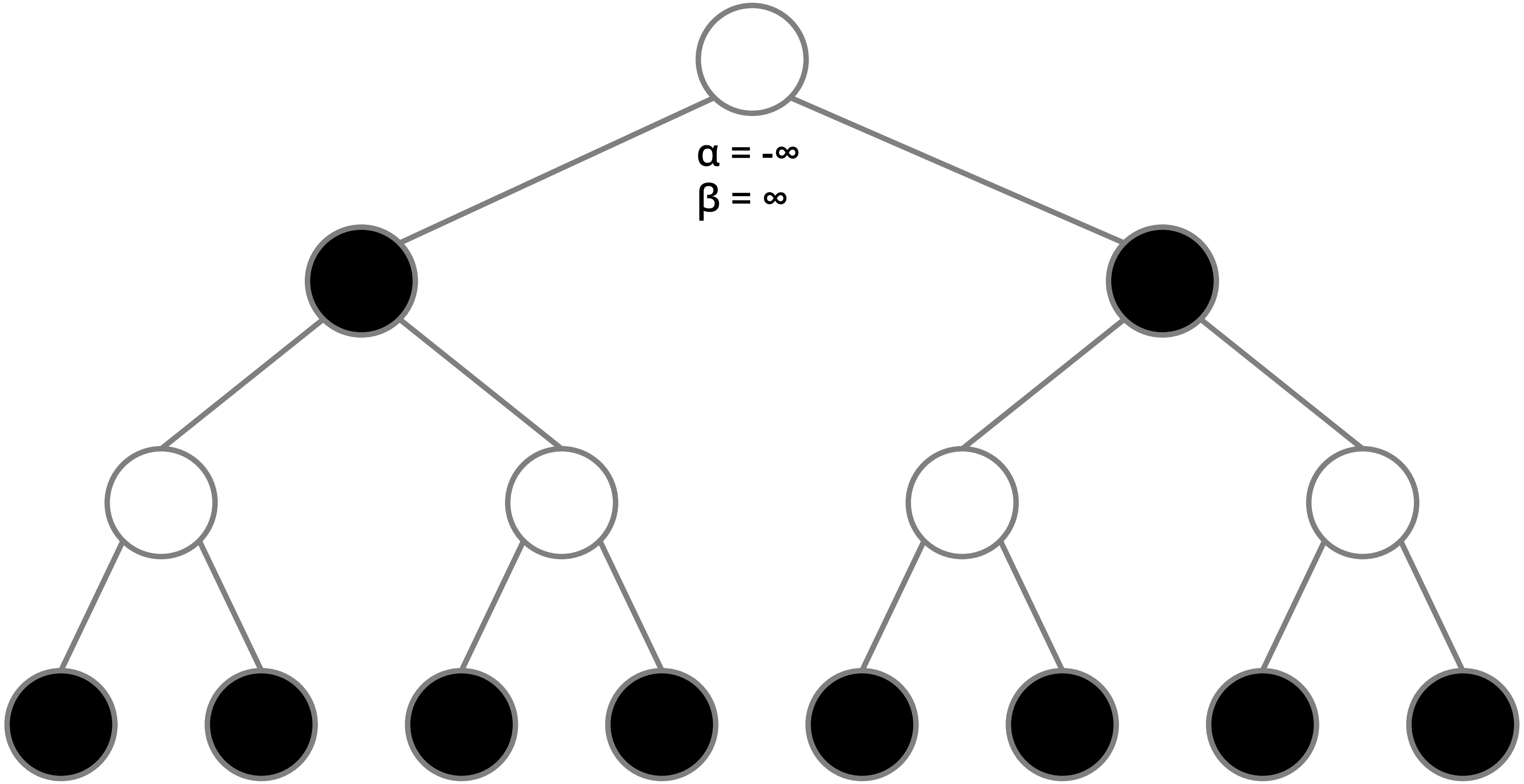
- Two new parameters:
  - **α**: the best value for MAX seen so far (used in MIN nodes and assigned to MAX nodes)
  - **β**: The best value for MIN seen so far (used in MAX nodes and assigned in MIN nodes)
- Prune whenever **α ≥ β**

Goal of pruning:

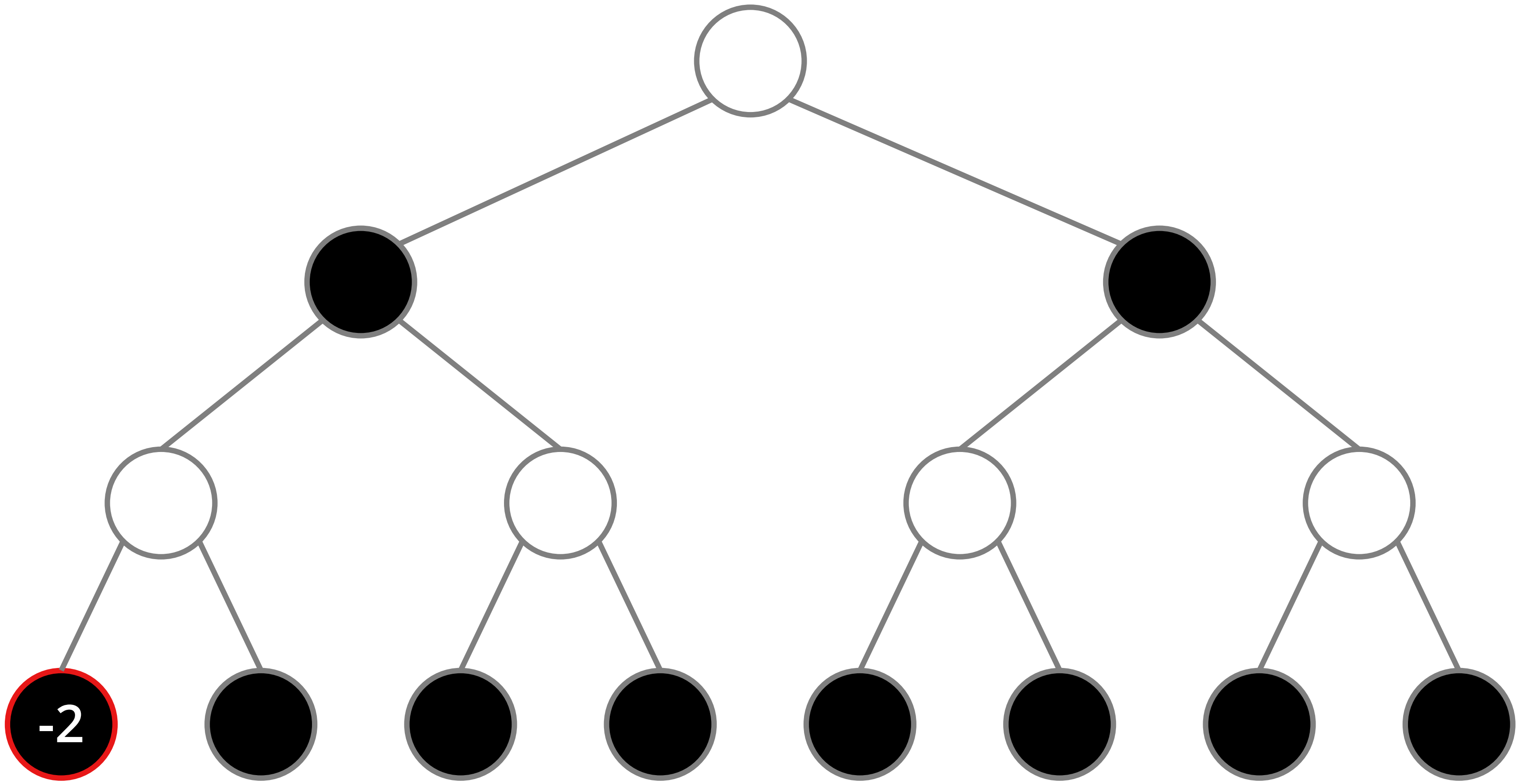


Reduce number of evaluations

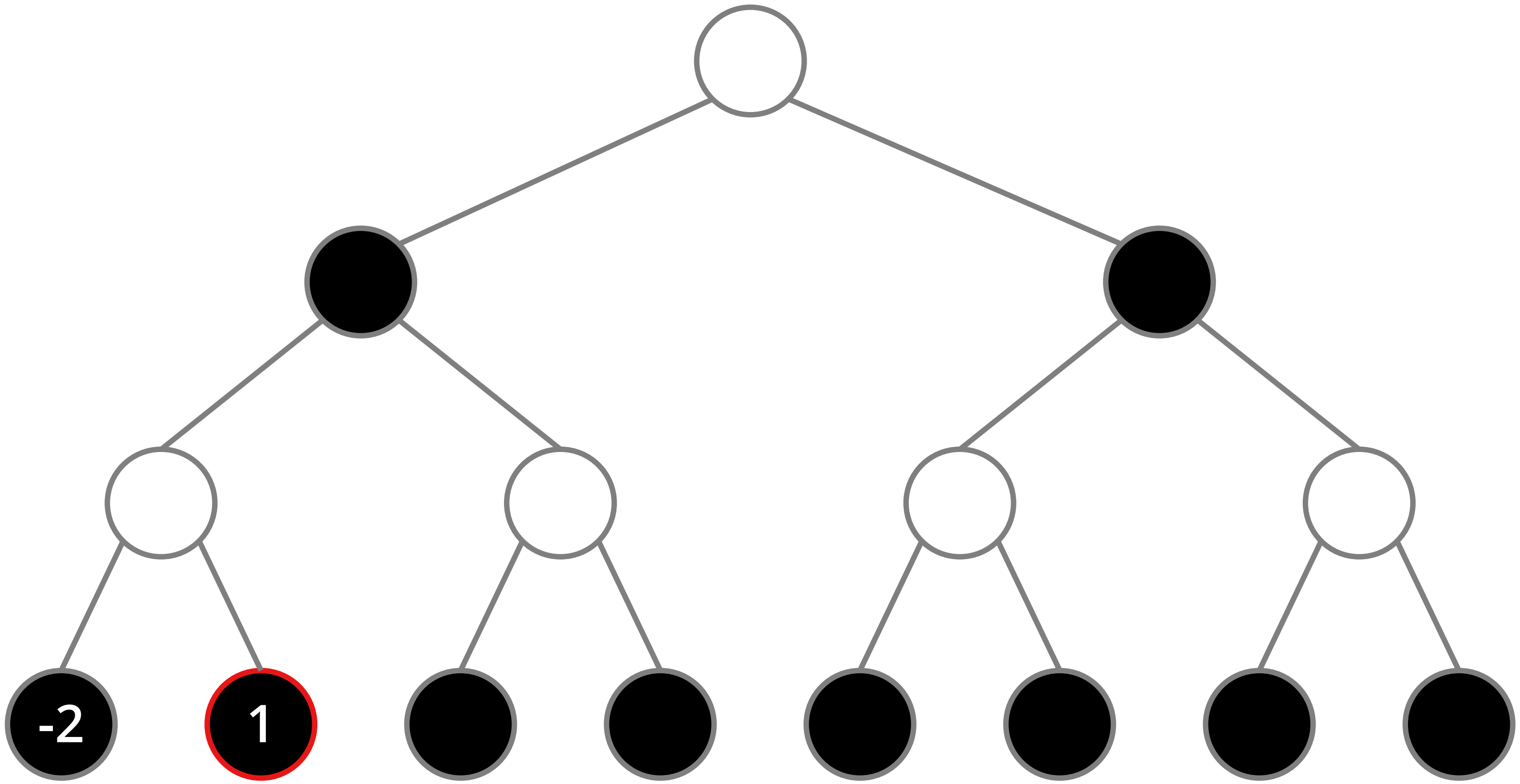
Alpha-Beta Pruning - Demo



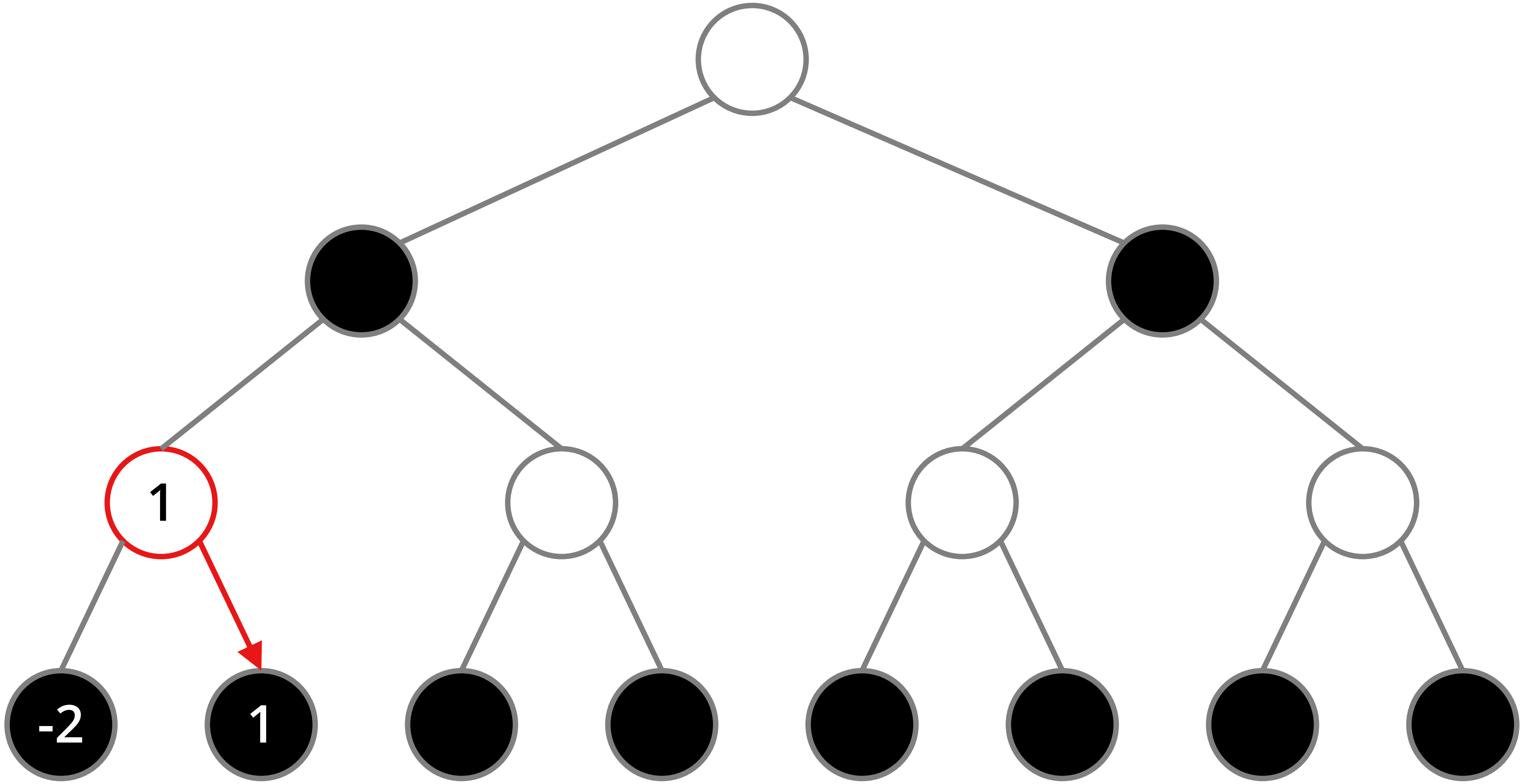
Alpha-Beta Pruning - Demo



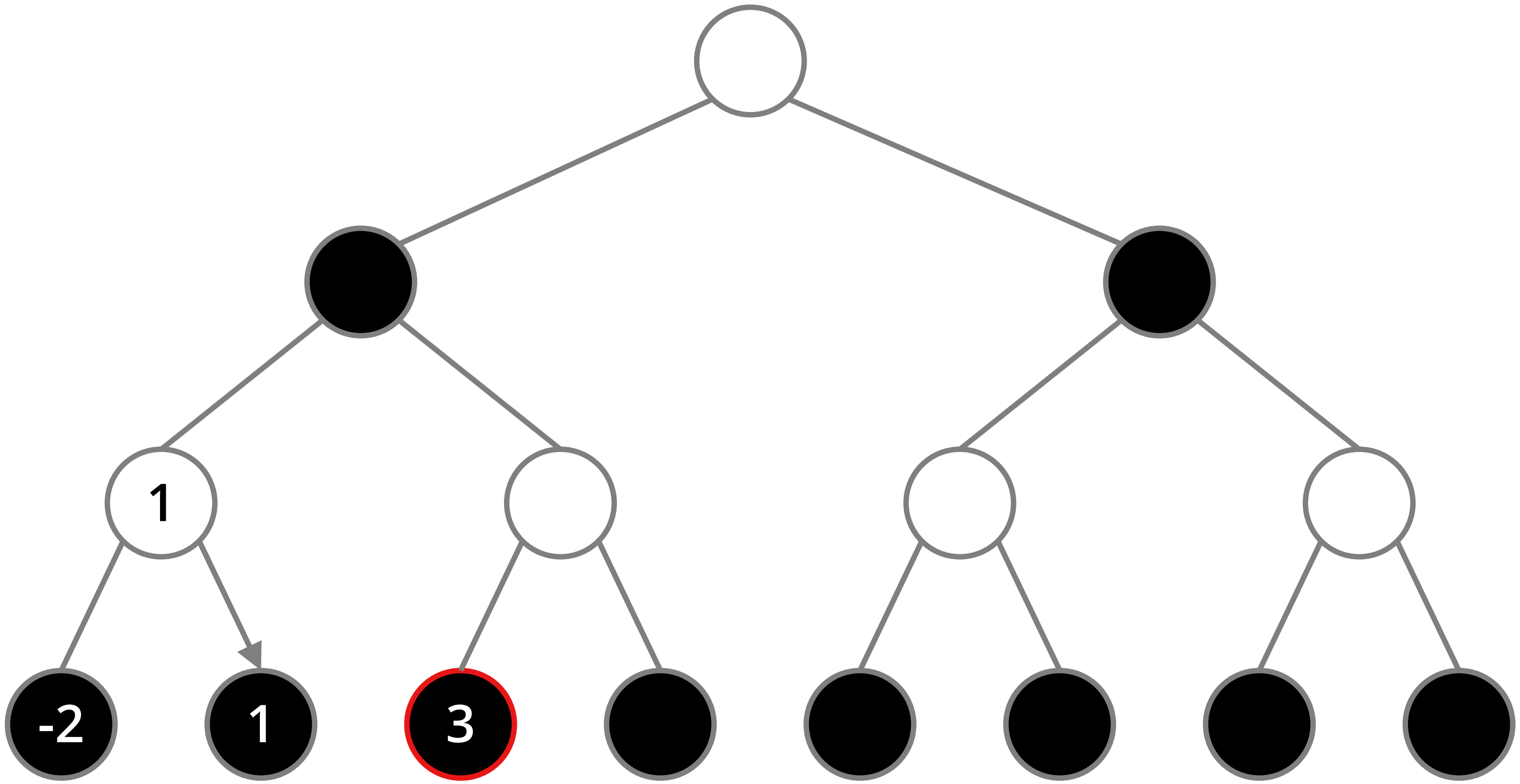
Alpha-Beta Pruning - Demo



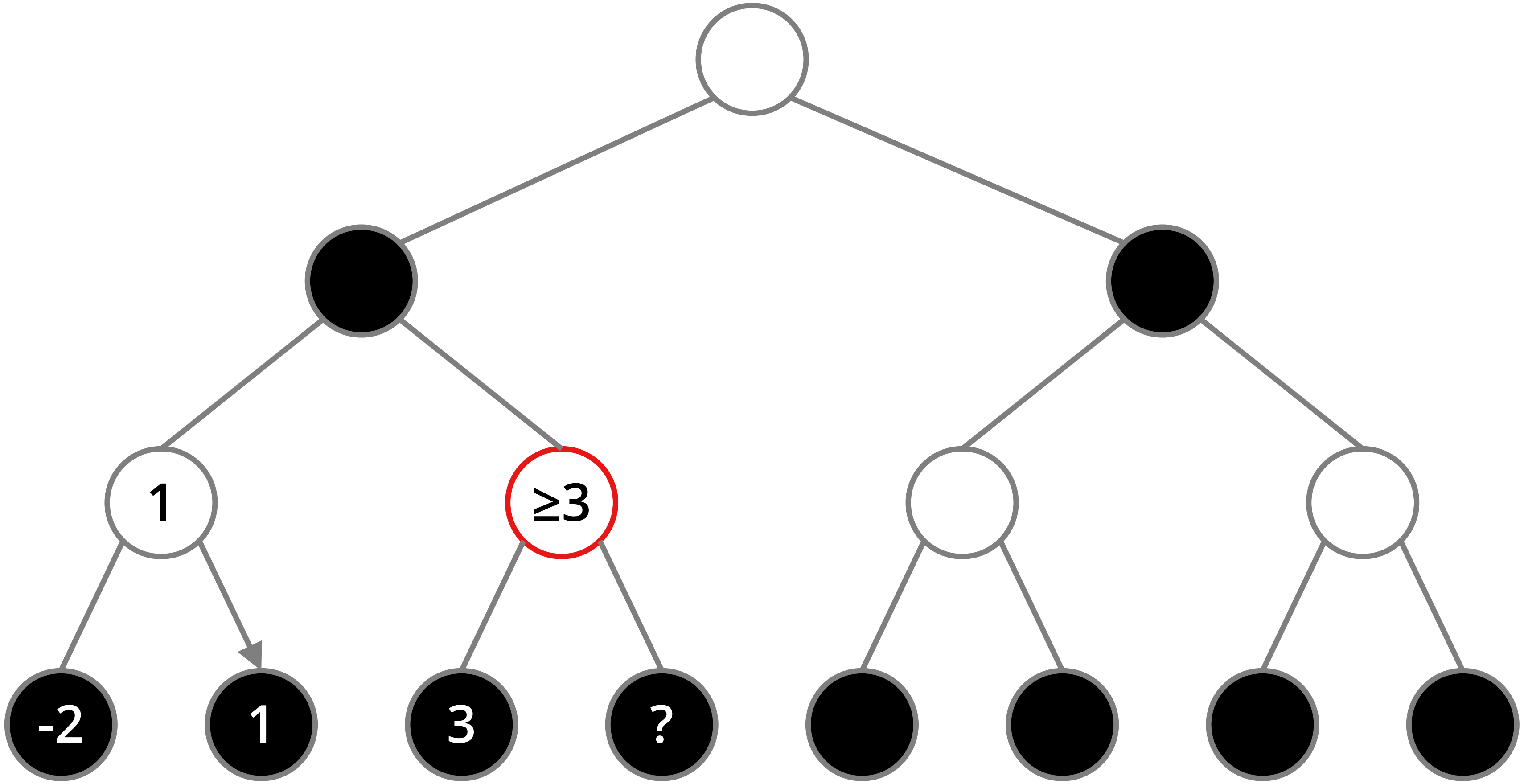
Alpha-Beta Pruning - Demo



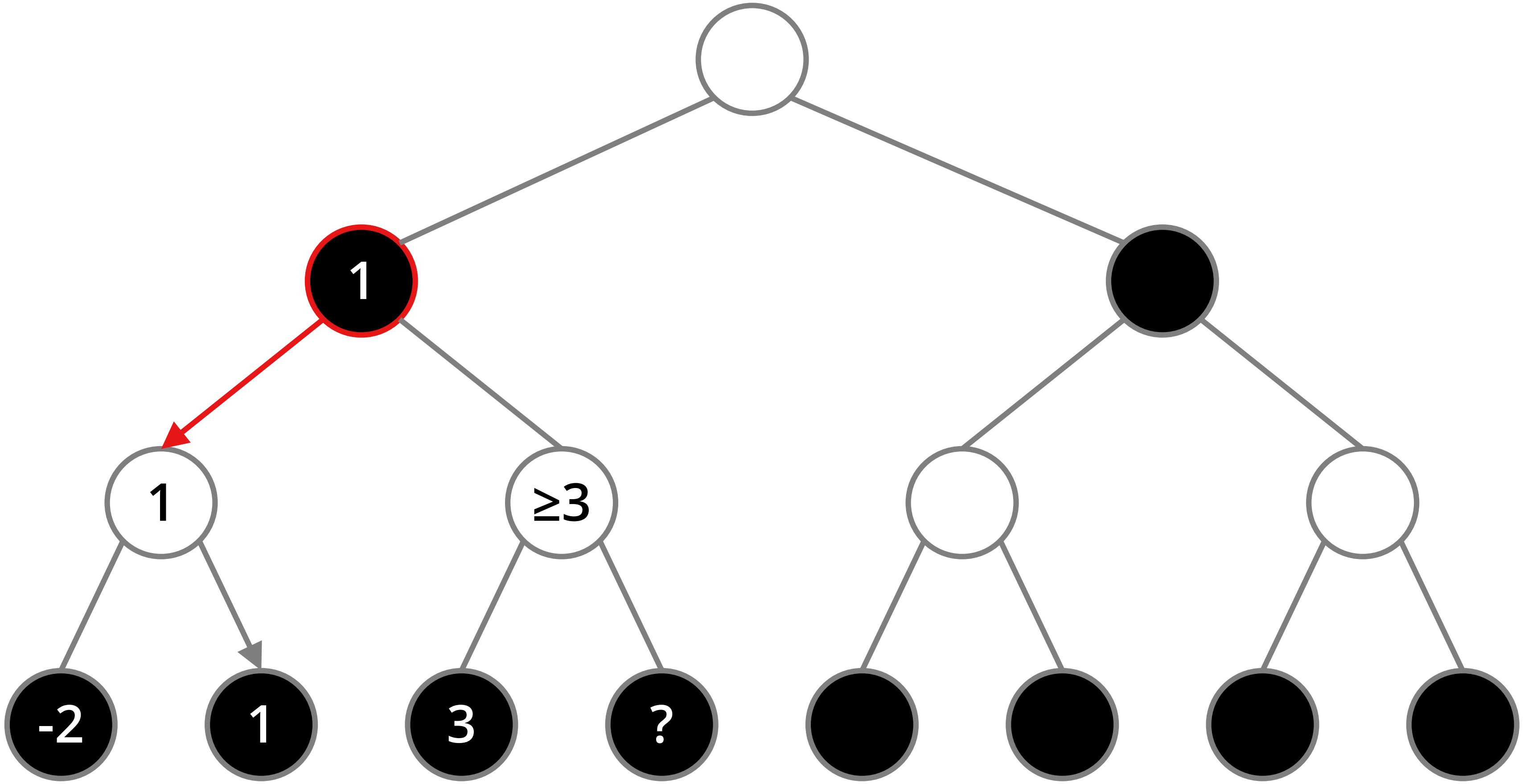
Alpha-Beta Pruning - Demo



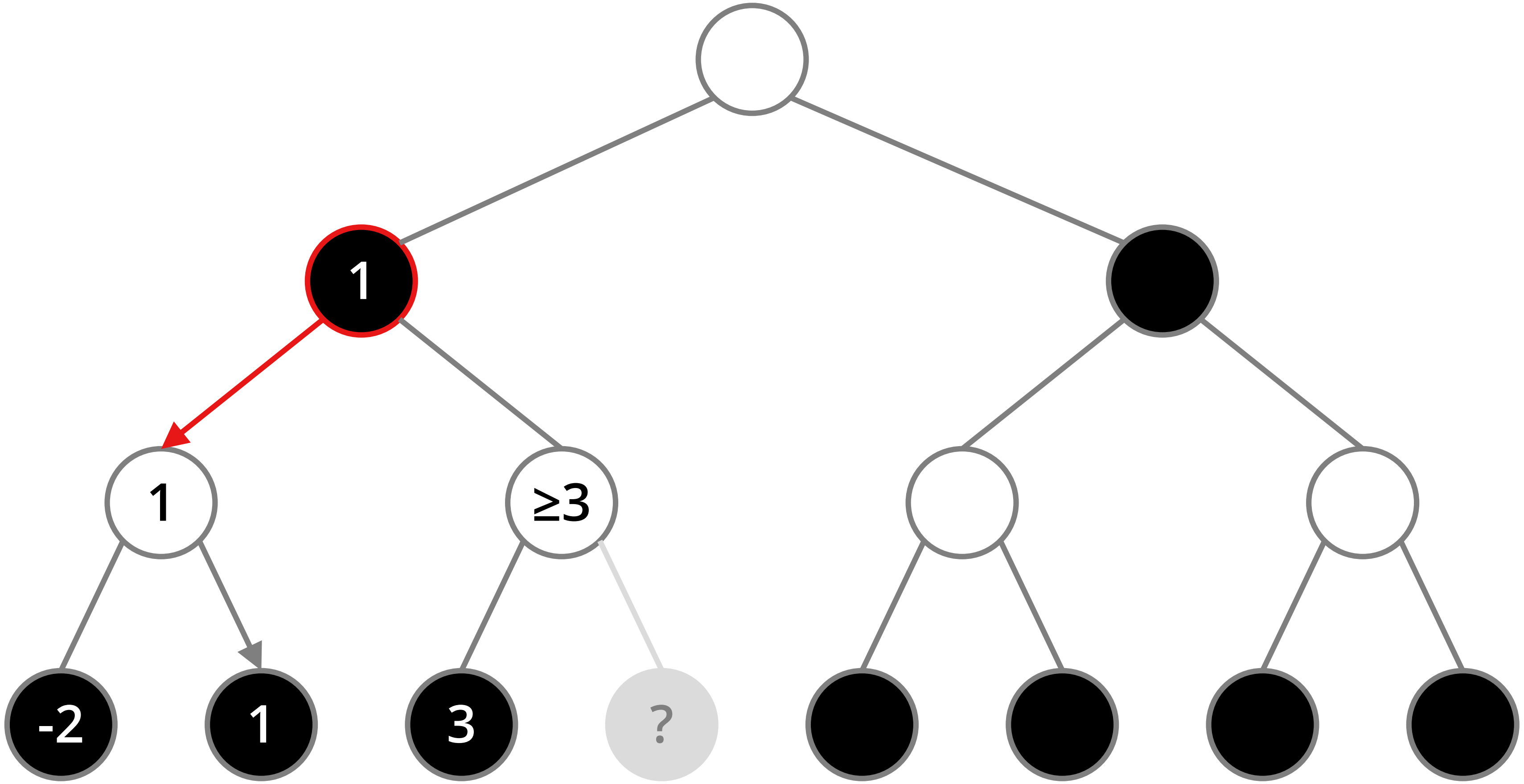
Alpha-Beta Pruning - Demo



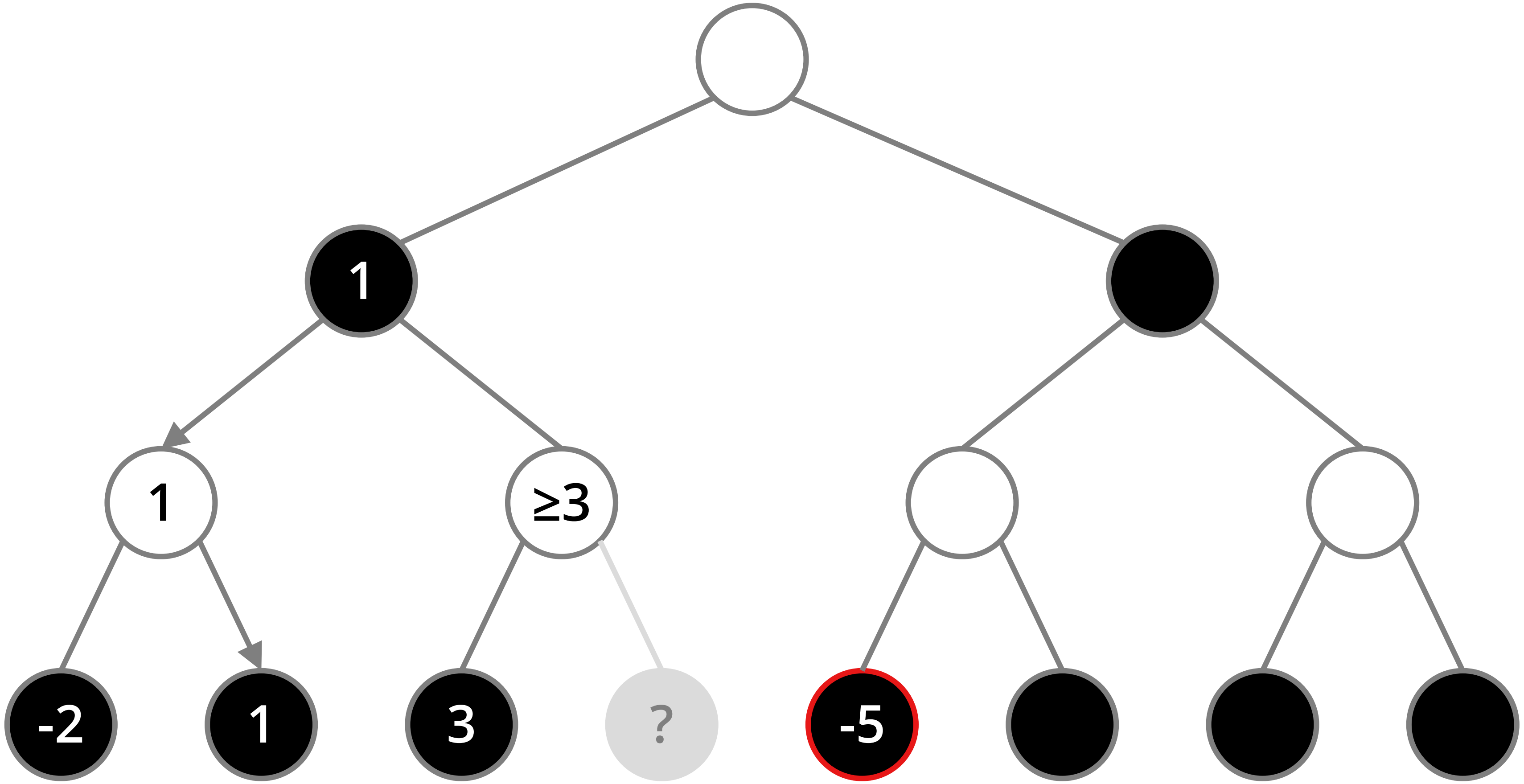
Alpha-Beta Pruning - Demo



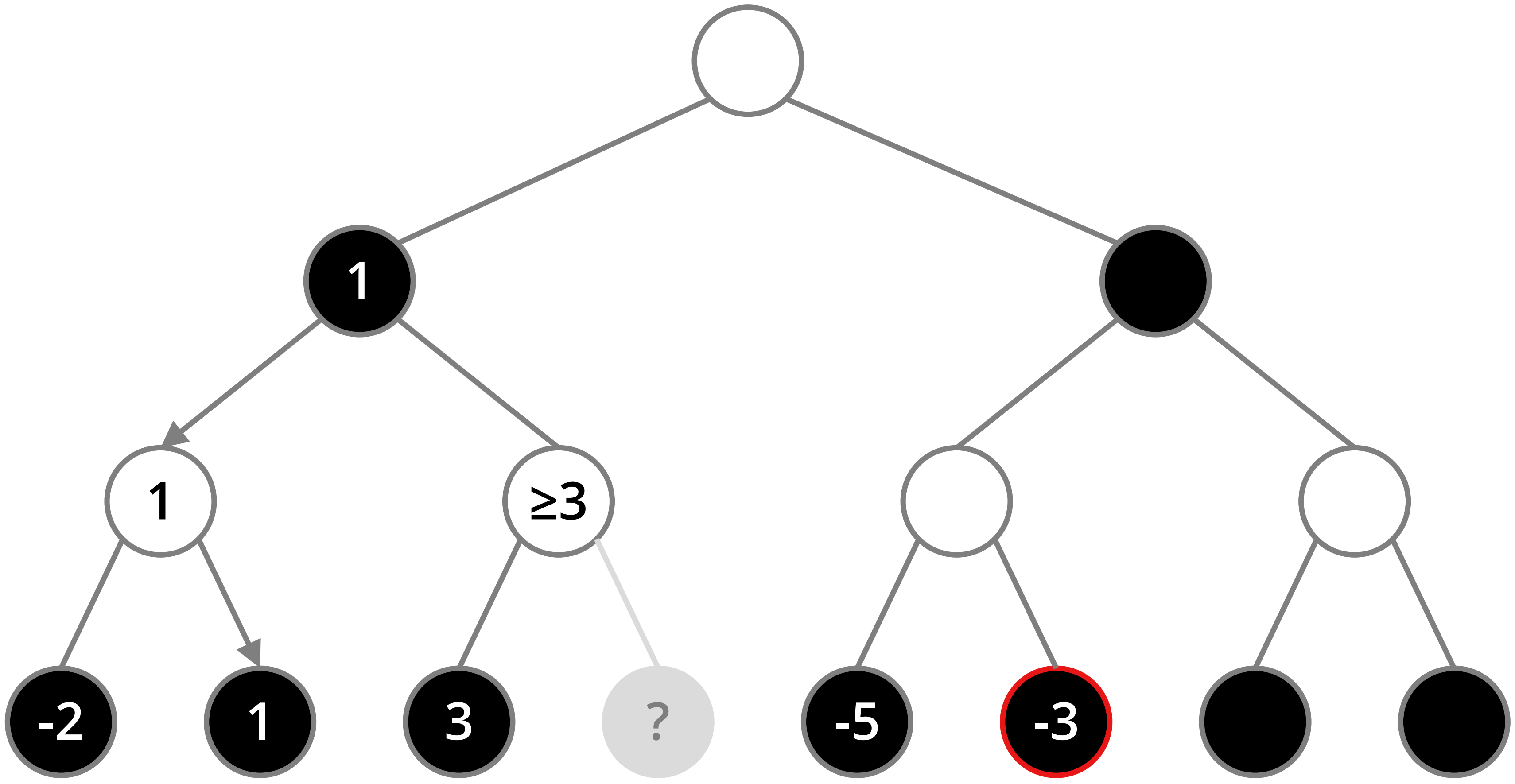
Alpha-Beta Pruning - Demo



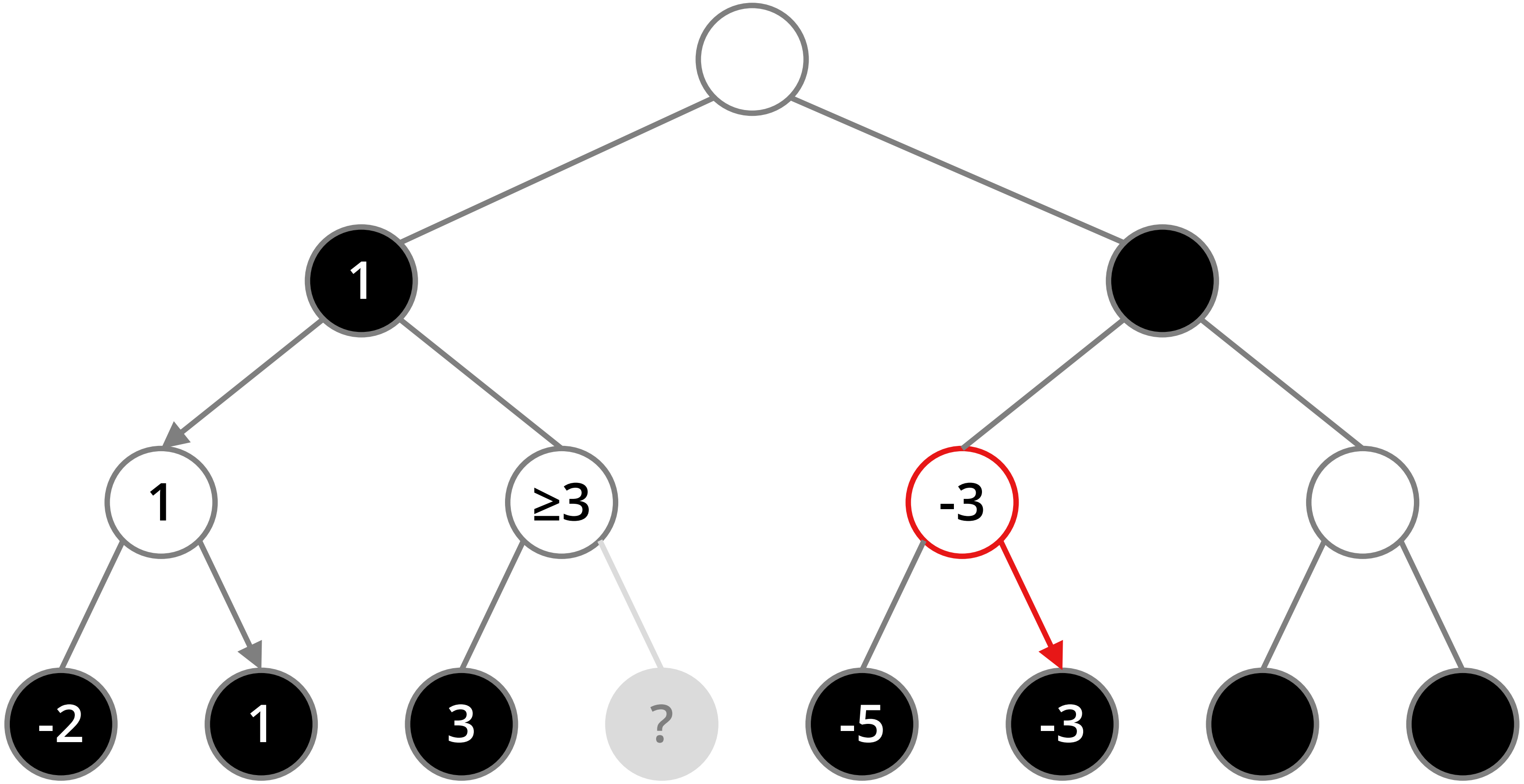
Alpha-Beta Pruning - Demo



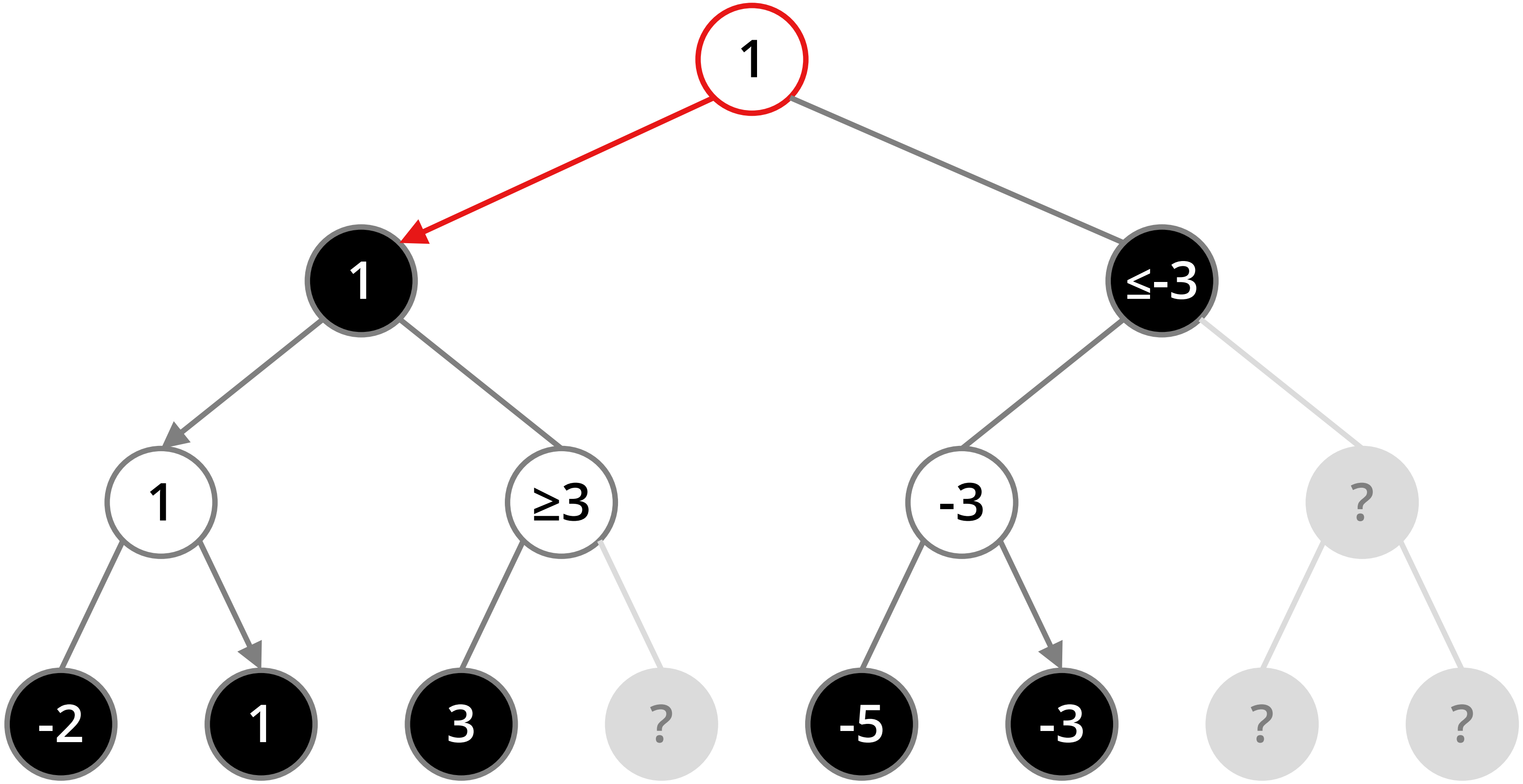
Alpha-Beta Pruning - Demo



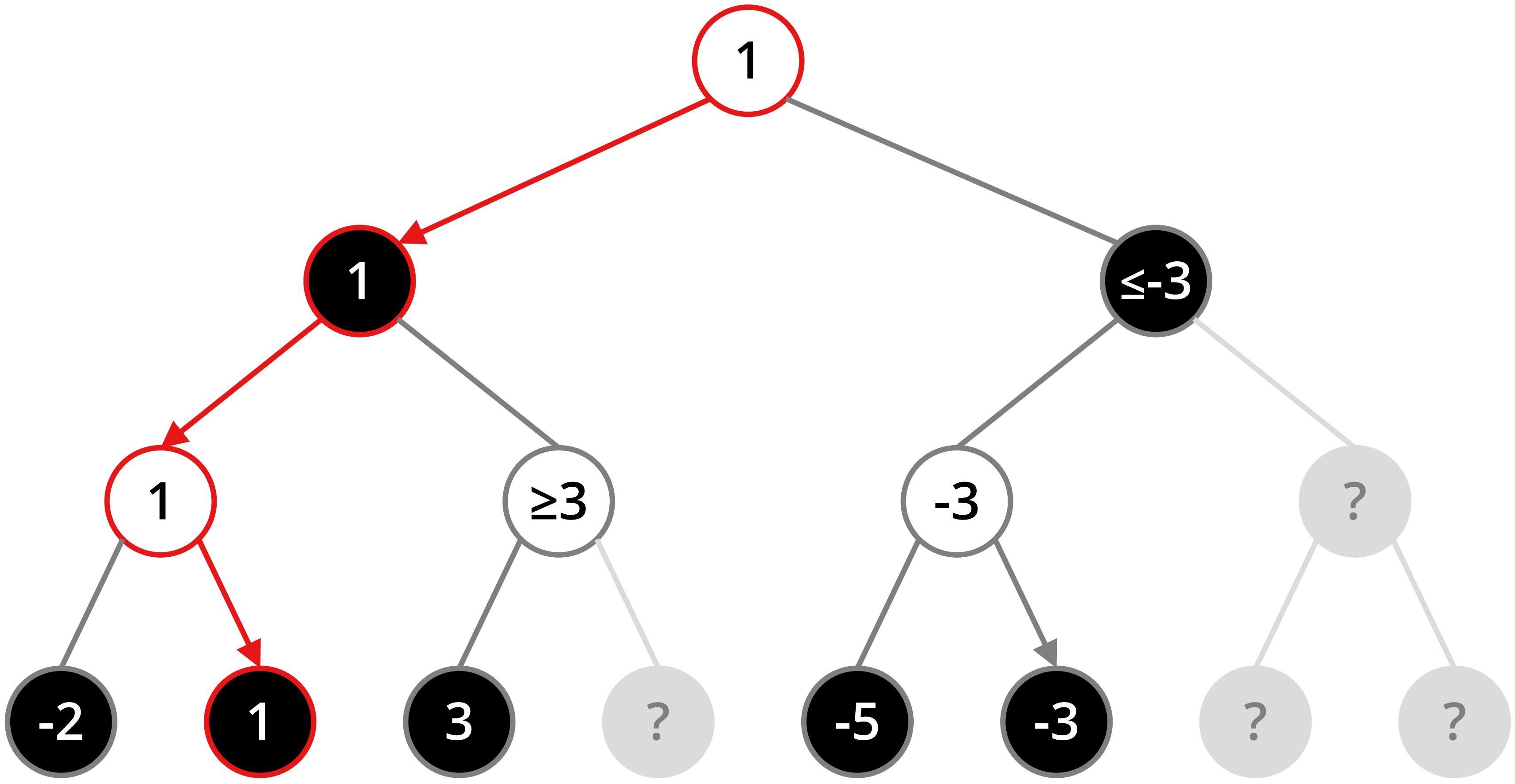
Alpha-Beta Pruning - Demo



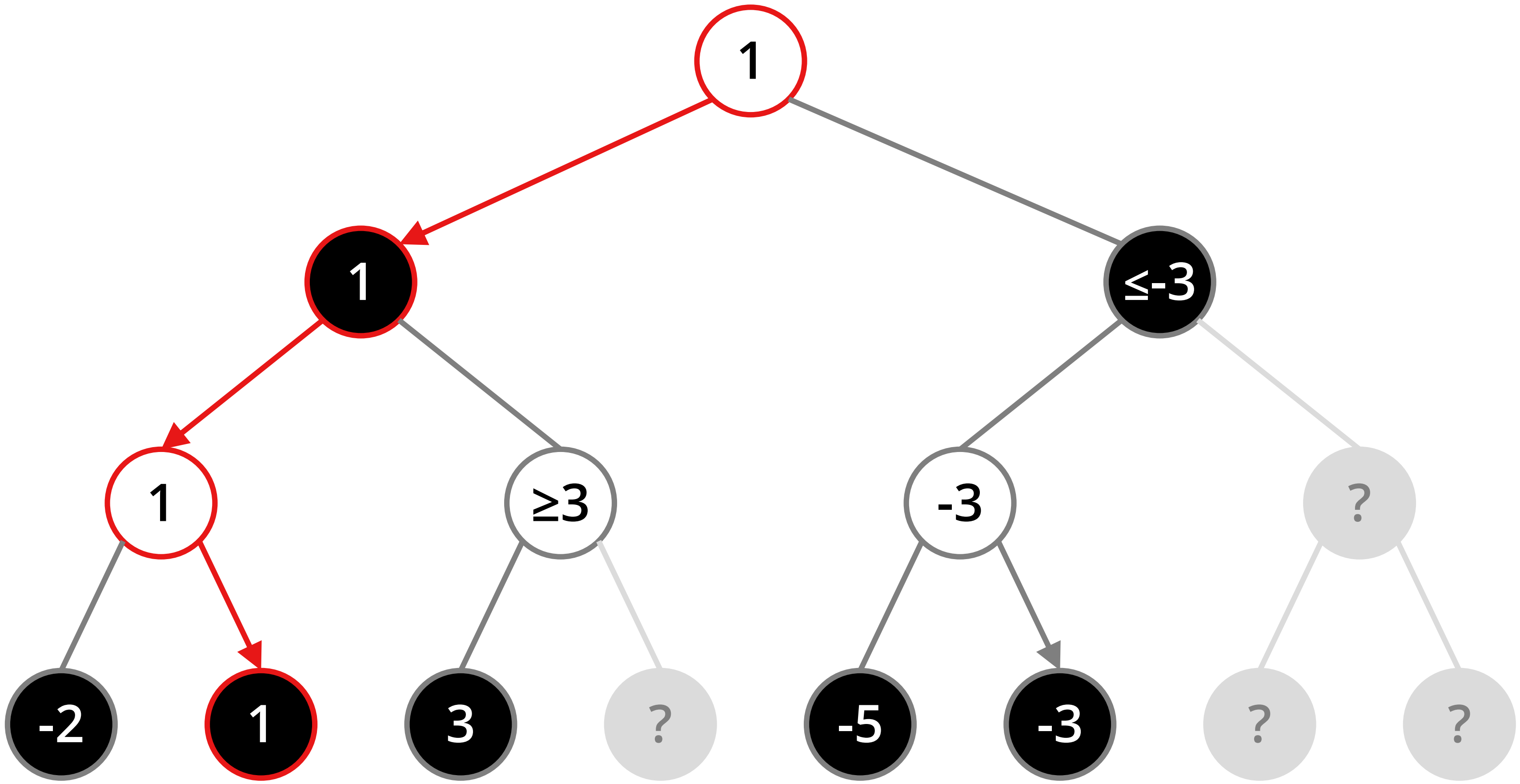
Alpha-Beta Pruning - Demo



Alpha-Beta Pruning - Demo

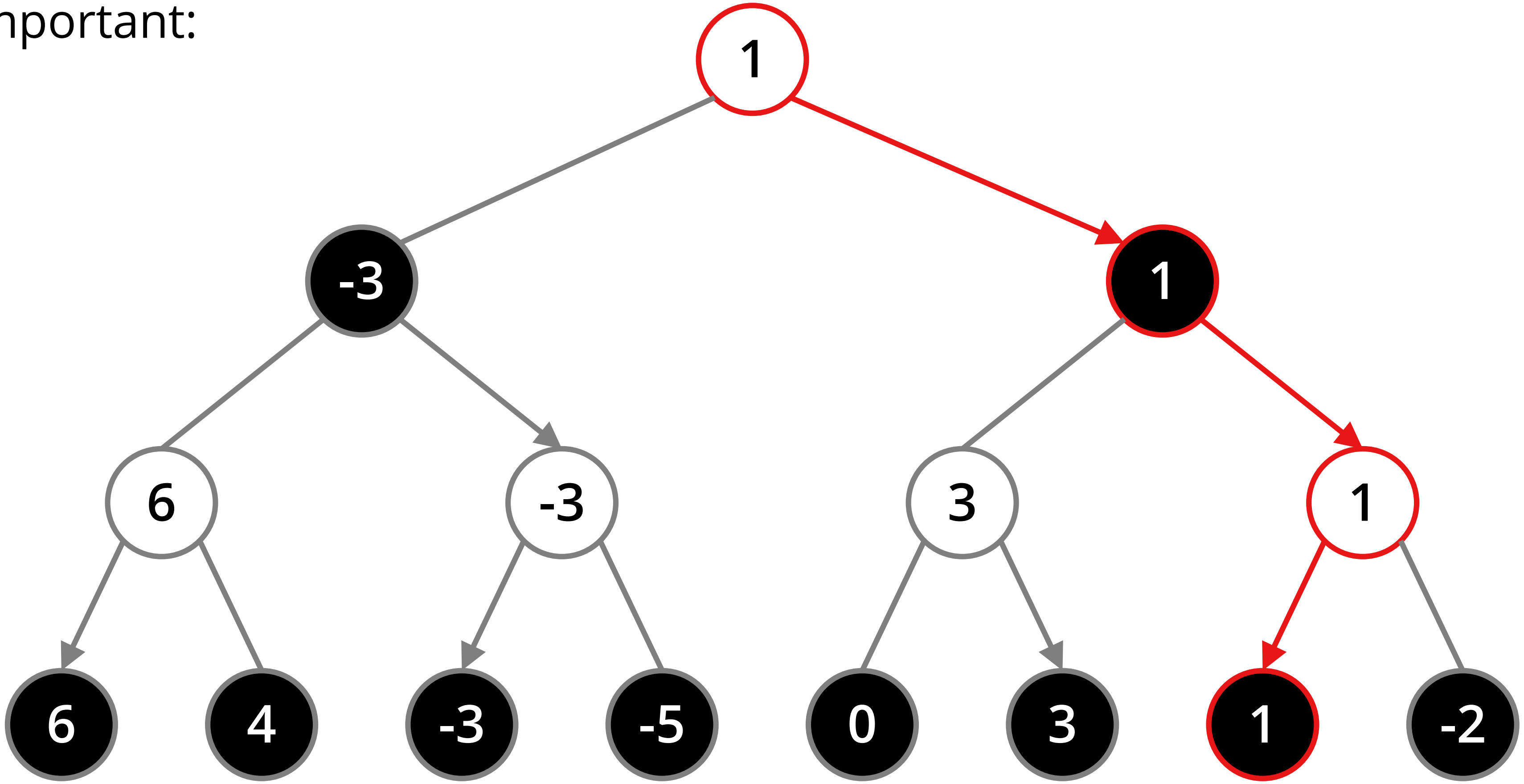


Alpha-Beta Pruning - Demo



Alpha-Beta Pruning - Demo

Order is important:



Alpha-Beta Pruning - Analysis

Algorithmic complexity

- Preserves completeness and optimality
- Complexity depends on move-ordering
- Worst-case: No improvement
- Best-case: time-complexity = $O(b^{m/2})$ → doubles search depth
- Good moves should come first

Move ordering

- Random ordering
- Captures first
- Killer Heuristic (move that caused beta cutoff in sibling node)

 # eval. calls by pruning and ordering:

Depth	No pruning	Pruning No ordering	Pruning Random	Pruning Captures 1st
1	39	39	39	39
2	1,689	243	166	117
3	63,455	5,927	5,762	1,514
4	2,625,675	24,831	38,427	8,762

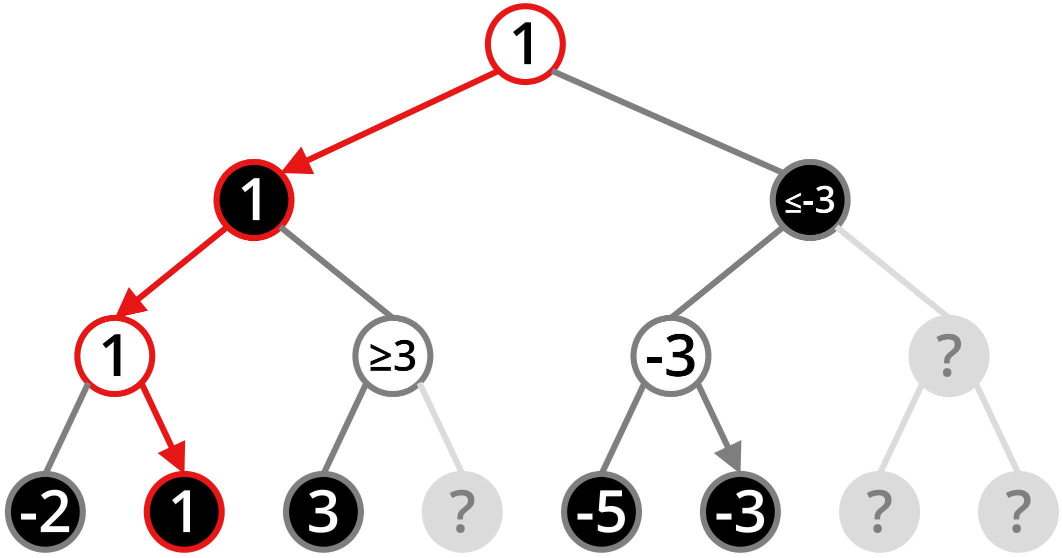
Alpha-Beta Pruning - Implementation

```
def negamax(self, board, color, depth, alpha, beta): *

        # check terminal condition
        if board.is_game_over() or depth == 0:
                return color * evaluate(board)

        # random move ordering
        legal_moves = list(board.legal_moves)
        random.shuffle(legal_moves)

        # call negamax recursively with switched sign
        max_eval = float('-Inf')
        for move in legal_moves:
                evaluation = negamax(board, -color, depth-1, -beta, -alpha)
                        max_eval = max(max_eval, -evaluation)
                alpha = max(alpha, max_eval)
                if alpha >= beta:
                        break
        return max_eval
```



* Pseudocode

Opening Book and Endgame Tablebase

Opening Book

- Many similar possibilities at beginning of game
- Chess openings are well developed theory
- Save time
- Especially beneficial vs humans

Endgame Tablebase

- Endgames are completely solved because it is manageable with limited number of pieces
- Steer into winning game
- Depth to mate (DTM) vs depth to zero (DTZ)

Further Ideas for Improvement

Possible Improvements:

- Compiled programming language
- Quiescence search (uneven tree development) to prevent horizon effect
- Transposition tables to avoid evaluating the same position multiple times
- More sophisticated evaluation function (positional, dynamic aspects)

Modern Chess Engines:

- Monte Carlo Tree search
- Neural networks
- Reinforcement Learning

Thank you!

Jens Mueller
Computer Science (Algorithms)
Bocconi University
June 12, 2020

Sources

- https://python-chess.readthedocs.io/en/latest/index.html
- https://www.chessprogramming.org/Minimax
- https://www.chessprogramming.org/Negamax
- https://www.chessprogramming.org/Alpha-Beta
- https://www.chessprogramming.org/Move_Ordering
- https://www.chessprogramming.org/Killer_Heuristic
- https://www.chessprogramming.org/Opening_Book
- https://www.chessprogramming.org/Endgame_Tablebases
- https://www.chessprogramming.org/Quiescence_Search
- https://www.freecodecamp.org/news/simple-chess-ai-step-by-step-1d55a9266977/
- https://byanofsky.com/2017/07/06/building-a-simple-chess-ai/
- https://www.youtube.com/watch?v=l-hh51ncgDI&list=WL&index=7&t=0s
- https://www.slideshare.net/RohitVaidya3/how-i-taught-a-computer-to-play-chess
- https://www.slideshare.net/myemon/aiminimax-algorithm-and-alpha-beta-reduction
- https://en.wikipedia.org/Minimax