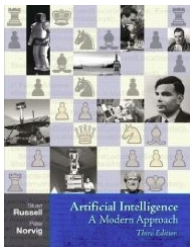
Outline

- Introduction
  - What are games and why are they interesting?
  - History and State-of-the-art in Game Playing
- Game-Tree Search
  - Minimax
  - Negamax
  - α-β pruning
- Real-time Game-Tree Search
  - NegaScout
  - evaluation functions
  - practical enhancements
  - selective search
- Multiplayer Game Trees



Many slides based on Russell & Norvig's slides Artificial Intelligence: A Modern Approach

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 search a.k.a. games

- Why study games?
  - Fun; historically entertaining
  - Interesting subject of study because they are hard
  - Easy to represent and agents restricted to small number of actions
  - Problem (and success) is easy to communicate

Relation of Games to Search

- Search – no adversary
  - Solution is method for finding goal
  - Heuristics and CSP techniques can find *optimal* solution
  - Evaluation function:
    - estimate of cost from start to goal through given node
  - Examples:
    - path planning, scheduling activities
- Games – adversary
  - Solution is strategy
    - 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, ...

Types of Games

- Zero-Sum Games
  - one player's gain is the other player's (or players') loss
- turn-taking
  - players alternate moves
- deterministic games vs. games of chance
  - do random components influence the progress of the game?
- perfect vs. imperfect information
  - does every player see the entire game situation?

	deterministic	chance
perfect information	chess, checkers, Go, Othello	backgammon, monopoly
imperfect information	battleship, kriegspiel, matching pennies, Roshambo	bridge, poker, scrabble

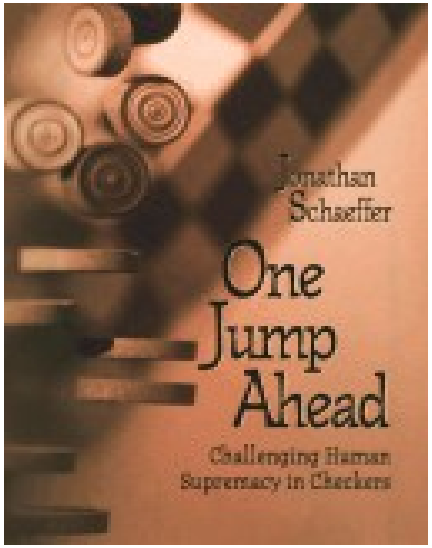
A Brief History of Search in Game Playing

- Computer considers possible lines of play
  (Babbage, 1846)
- Algorithm for perfect play
  (Zermelo, 1912; Von Neumann, 1944)
- Finite horizon, approximate evaluation
  (Zuse, 1945; Wiener, 1948; Shannon, 1950)
- First computer chess game
  (Turing, 1951)
- Machine learning to improve evaluation accuracy
  (Samuel, 1952-57)
- Selective Search Programs
  (Newell, Shaw, Simon 1958; Greenblatt, Eastake, Crocker 1967)
- Pruning to allow deeper search
  (McCarthy, 1956)
- Breakthrough of Brute-Force Programs
  (Atkin & Slate, 1970-77)

Checkers: Chinook vs. Tinsley

Name: Marion Tinsley
Profession: Teach mathematics
Hobby: Checkers
Record: Over 42 years loses only 3 (!) games of checkers

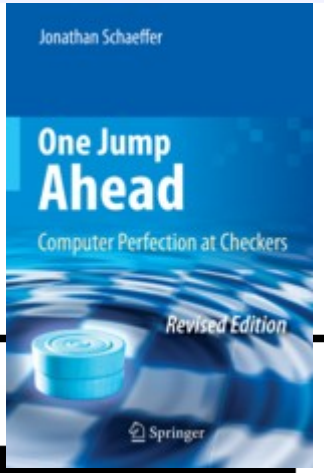
Chinook



First computer to win human world championship!

Visit http://www.cs.ualberta.ca/~chinook/ to play a version of Chinook over the Internet.

Chinook



- July 19 2007, after 18 years of computation:

http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.95.5393

Sciencexpress Research Article

Checkers Is Solved

Jonathan Schaeffer,* Neil Burch, Yngvi Björnsson, Akihiro Kishimoto, Martin Müller, Robert Lake, Paul Lu, Steve Sutphen

Department of Computing Science, University of Alberta, Edmonton, Alberta T6G 2E8, Canada.

*To whom correspondence should be addressed. E-mail: jonathan@cs.ualberta.ca

The game of checkers has roughly 500 billion billion possible positions (5×10^{20}). The task of solving the game, determining the final result in a game with no mistakes made by either player, is daunting. Since 1989, almost continuously, dozens of computers have been working on solving checkers, applying state-of-the-art artificial intelligence techniques to the proving process. This paper announces that checkers is now solved: perfect play by both sides leads to a draw. This is the most challenging popular game to be solved to date, roughly one million times more complex than Connect Four. Artificial intelligence technology has been used to generate strong heuristic-based game-playing programs, such as DEEP BLUE for chess. Solving a game takes this to the next level, by replacing the heuristics with perfection.

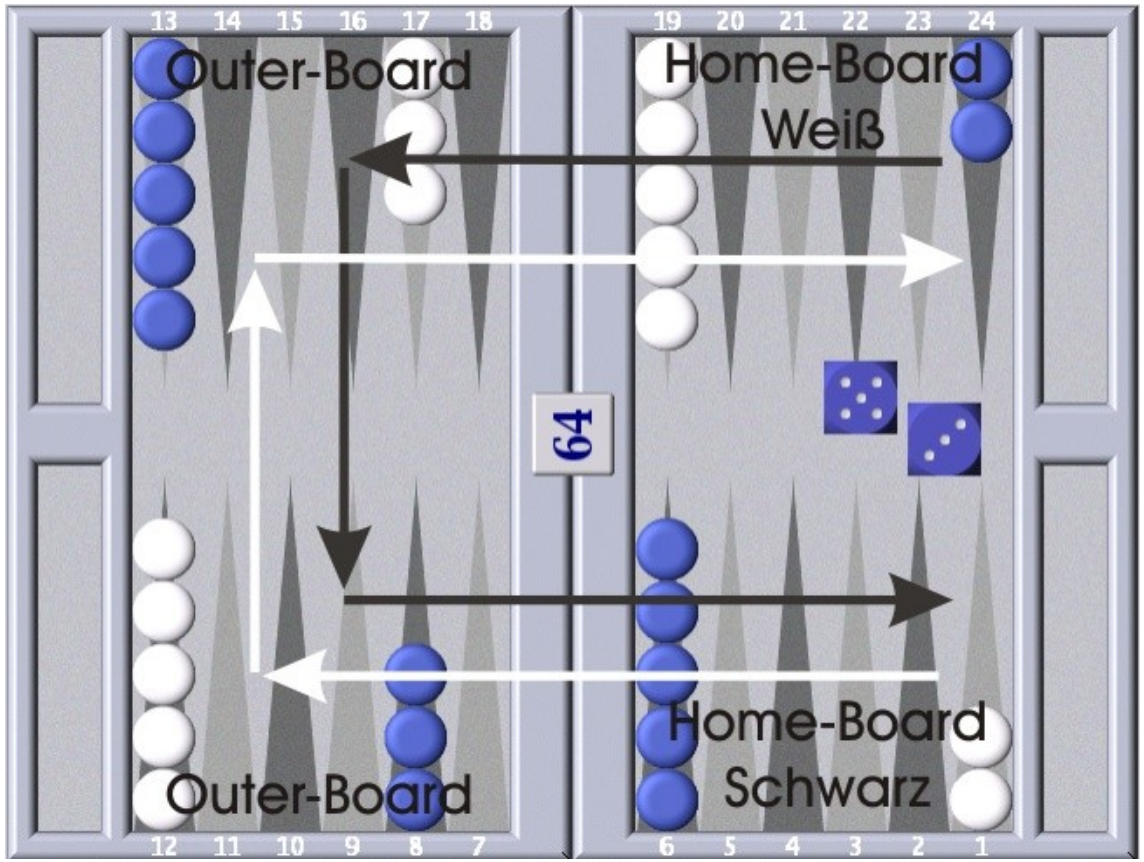
1992, over 200 processors were being used simultaneously. The end result is one of the longest running computations completed to date.

This paper announces that checkers has been weakly solved. From the starting position (Fig. 1A), we have a computational proof that checkers is a draw. The proof consists of an explicit strategy that never loses – the program can achieve at least a draw against *any* opponent, playing either the black or white pieces. That checkers is a draw is not a surprise; grandmaster players have conjectured this for decades.

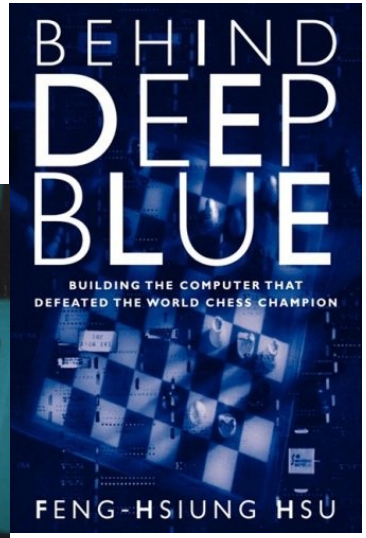
The checkers result pushes the boundary of artificial intelligence (AI). In the early days of AI research, the easiest path to achieving high performance was seen to be emulating the human approach. This was fraught with difficulty, especially the problems of capturing and encoding human knowledge. Human-like strategies are not

Backgammon

- branching factor several hundred
- TD-Gammon v1 – 1-step lookahead, learns to play games against itself
- TD-Gammon v2.1 – 2-ply search, does well against world champions
- TD-Gammon has changed the way experts play backgammon.



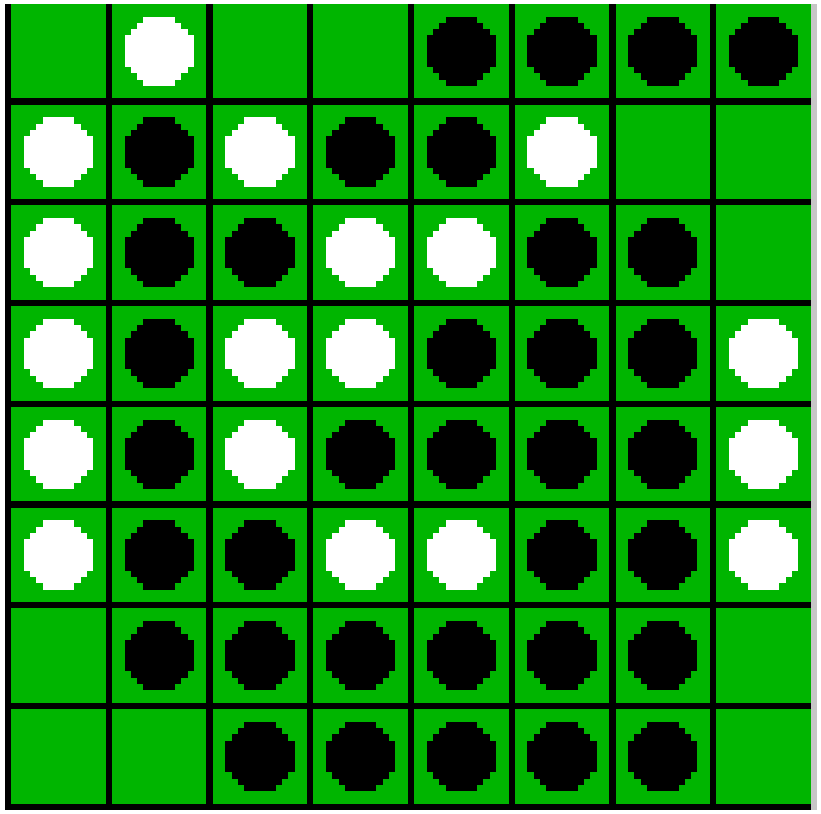
Chess



Kasparov	Name	Deep Blue
5'10"	Height	6' 5"
176 lbs	Weight	2,400 lbs
34 years	Age	4 years
50 billion neurons	Computers	512 processors
2 pos/sec	Speed	200,000,000 pos/sec
Extensive	Knowledge	Primitive
Electrical/chemical	Power Source	Electrical
Enormous	Ego	None

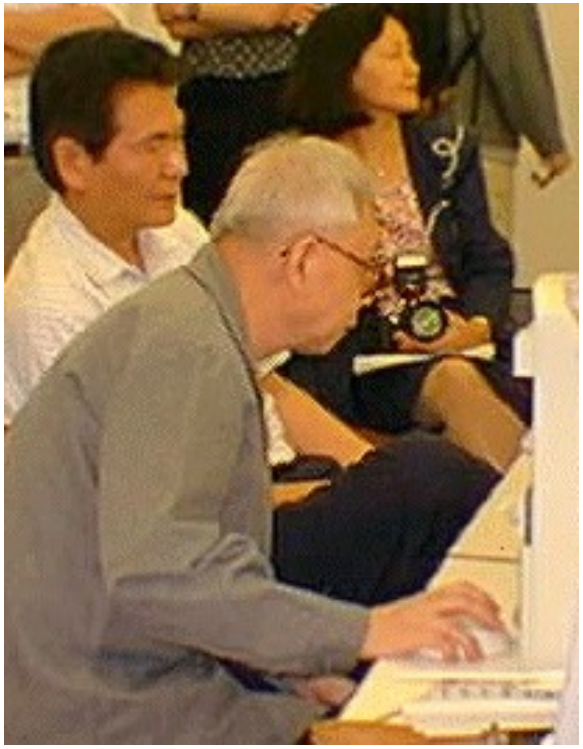
http://www.wired.com/wired/archive/9.10/chess.htm

Reversi/Othello



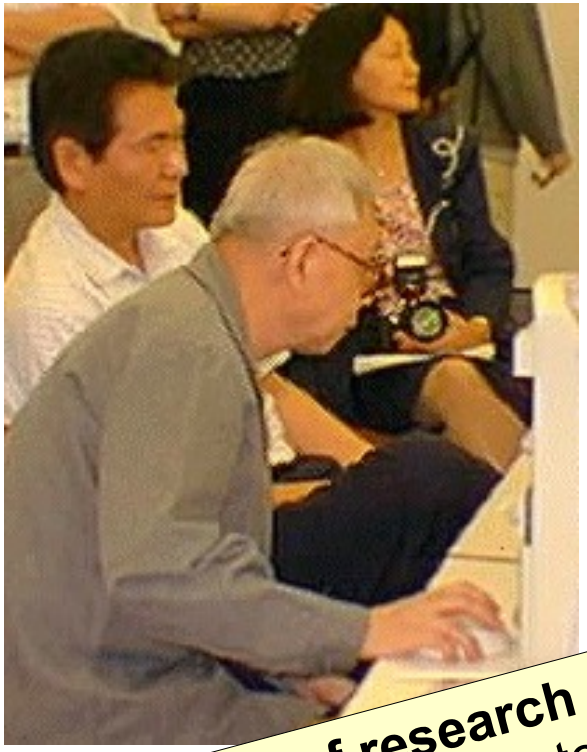
Name: Takeshi Murakami
Title: World Othello Champion
1997: Lost 6-0 against Othello Program Logistello

Computer Go



Name: Chen Zhixing
Author: Handtalk (Goemate)
Profession: Retired
Computer skills:
    Selftaught assembly
    language programmer
Accomplishments:
    dominated computer
    go for 4 years.

Computer Go, early 2000s



Name: Chen Zhixing
Author: Handtalk (Goemate)
Profession: Retired
Computer skills:
    Selftaught assembly
    language programmer
Accomplishments:
    dominated computer
    go for 4 years.

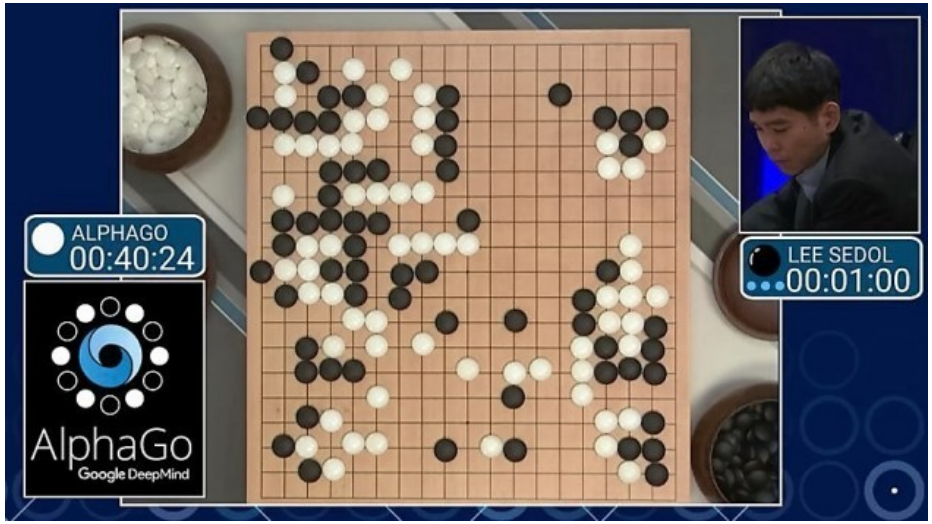
Active Area of research
Methods relying on Monte Carlo tree search gave a strong boost in performance, Best Humans are still out of reach on the 19x19 board.

Gave Handtalk a 9 stone handicap and still easily beat the program, thereby winning $15,000

Computer Go, 2016

- Oktober 2015:
  - AlphaGo beats European champion Fan Hui
  - First win of a computer against a professional Go player
  - https://gogameguru.com/alpha-go-fan-hui/

- March 2016:
  - AlphaGo beats Lee Sedol, one of the best professional players



- Techniques:
  - Combination of Deep Learning, Reinforcement Learning and Monte-Carlo Tree Search

AlphaZero

- Improved version of AlphaGo
  - Also successfully learned to play chess and Shogi (Japanese Chess)
- December 2017:
  - AlphaZero beats the strongest programs in all three games after hours (chess) or days (Go) of training

Game	White	Black	Win	Draw	Loss
Chess	*AlphaZero*	*Stockfish*	25	25	0
	Stockfish	*AlphaZero*	3	47	0
Shogi	*AlphaZero*	*Elmo*	43	2	5
	Elmo	*AlphaZero*	47	0	3
Go	*AlphaZero*	*AG0 3-day*	31	–	19
	AG0 3-day	*AlphaZero*	29	–	21

Table 1: Tournament evaluation of *AlphaZero* in chess, shogi, and Go, as games won, drawn or lost from *AlphaZero*'s perspective, in 100 game matches against *Stockfish*, *Elmo*, and the previously published *AlphaGo Zero* after 3 days of training. Each program was given 1 minute of thinking time per move.

https://arxiv.org/pdf/1712.01815.pdf

Outline

- Introduction
  - What are games?
  - History and State-of-the-art in Game Playing
- Game-Tree Search
  - Minimax
  - α-β pruning
  - NegaScout
- Real-time Game-Tree Search
  - evaluation functions
  - practical enhancements
  - selective search
- Games of imperfect information and games of chance
- Simulation Search
  - Monte-Carlo search
  - UCT search

Solving a Game

- Ultra-weak
  - prove whether the first player will win, lose, or draw from the initial position, given perfect play on both sides
  - could be a non-constructive proof, which does not help in play
  - could be done via a complete minimax or alpha-beta search
  - Example:
    - chess when first move may be a pass
- Weak
  - provide an algorithm which secures a win for one player, or a draw for either, against any possible moves by the opponent, from the initial position only
- Strong
  - provide an algorithm which can produce perfect play from any position
    - often in the form of a database for all positions

Retrograde Analysis

- Retrograde Analysis Algorithm (goes back to Zermelo 1912)
  - builds up a database if we want to strongly solve a game

0. Generate all possible positions
1. Find all positions that are won for player A
   i. mark all terminal positions that are won for A
   ii. mark all positions where A is to move and can make a move that leads to a marked position
   iii. mark all positions where B is to move and all moves lead to a marked position
   iv. if there are positions that have not yet been considered goto ii.
2. Find all positions that are won for B
   - analogous to 1.
3. All remaining positions are draw

Retrograde Analysis

- Several Games habe been solved completely using RA
  - Tic-Tac-Toe, Go-Moku, Connect-4, ...
- For other games, solutions for partial
  - Chess
    - All endgames with 7 pieces (=2 kings + 5 additional pieces) are solved since 2012
      - ca. 500 000 000 000 000 positions had to be stored, even when considering symmetries etc.
    - Accessible on-line http://tb7.chessok.com/
    - https://chessprogramming.wikispaces.com/Endgame+Tablebases
  - Checkers
    - In checkers, databases with up to 10 pieces were crucial for (weakly) solving the game
- Overall, RA is too complex for most games
  - Impossible to store all possible game states

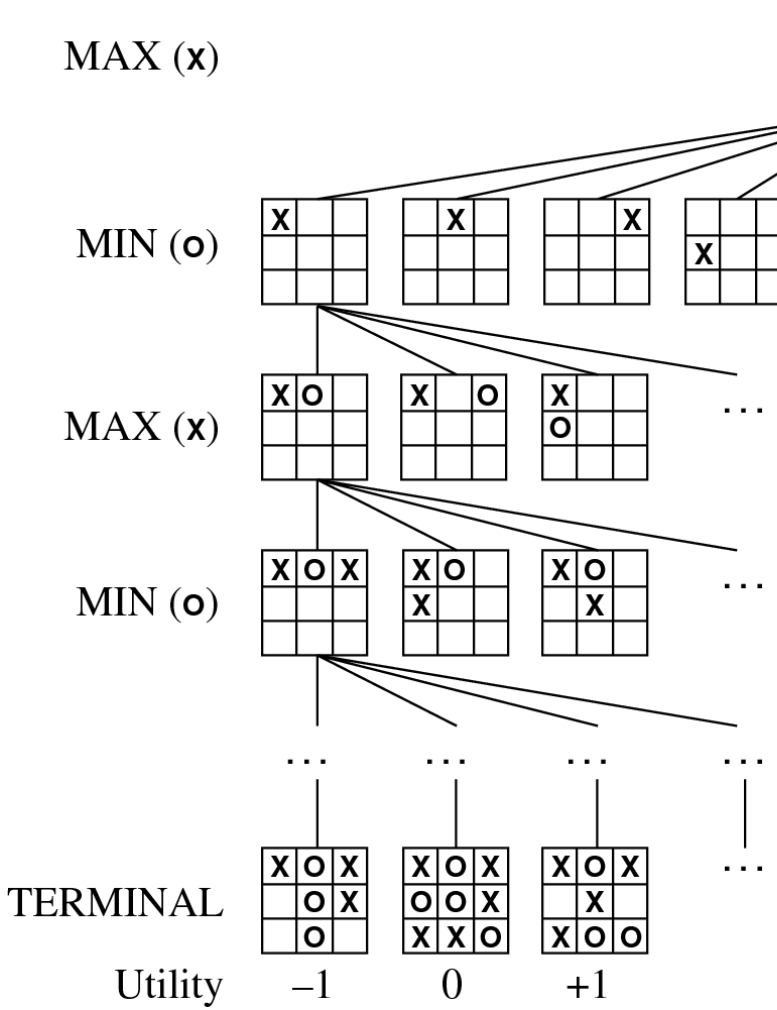
Status Quo in Game Playing

- Solved
  - Tic-Tac-Toe, Connect-4, Go-Moku, 9-men Morris
  - Most recent addition: Checkers is a draw
    - Solved with 18 years of computation time
      (first endgame databases were computed in 1989)
  - http://www.sciencemag.org/cgi/content/abstract/1144079
- Partly solved
  - Chess
    - all 6-men endgames, some 7-men endgames
    - longest win: position in KQN vs. KRBN after 517 moves
      - http://timkr.home.xs4all.nl/chess2/diary_16.htm
- World-Championship strength
  - Chess, Backgammon, Scrabble, Othello, Go, Shogi
- Human Supremacy
  - Bridge, Poker

Game setup

- Two players: MAX and MIN
  - MAX moves first and they take turns until the game is over.
    - ply: a half-move by one of the players
    - move: two plies, one by MAX and one by MIN
  - Winner gets award, looser gets penalty.
- Games as search:
  - Initial state:
    - e.g., board configuration of chess
  - Successor function:
    - list of (move,state) pairs specifying legal moves.
  - Terminal test:
    - Is the game finished?
  - Utility function (*objective function, payoff function*)
    - Gives numerical value of terminal states
    - E.g. win (+1), loose (-1) and draw (0) in tic-tac-toe (next)
    - typically from the point of view of MAX

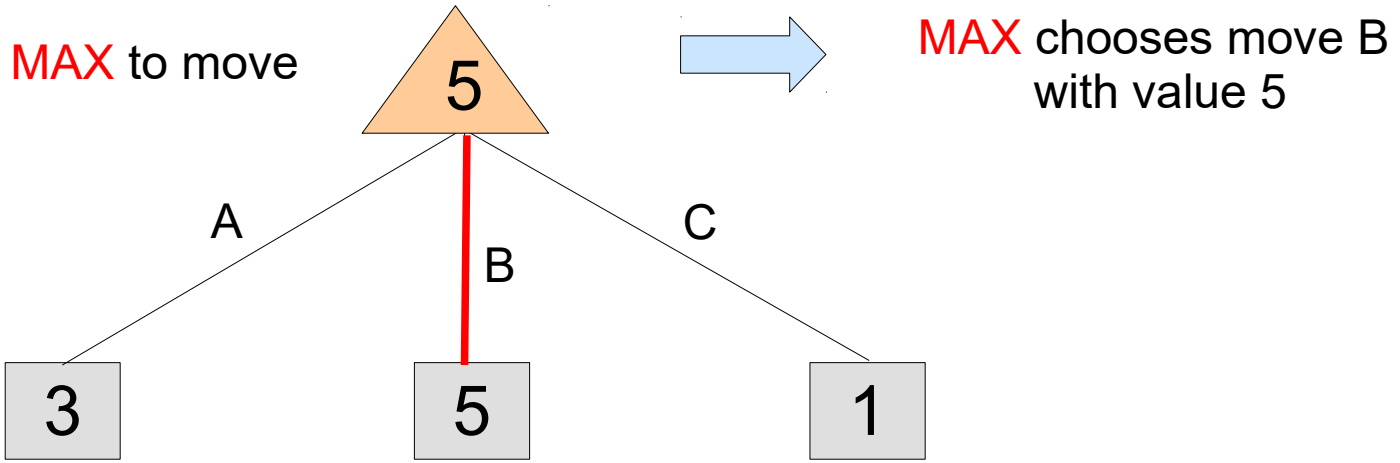
Partial Game Tree for Tic-Tac-Toe



- MAX is to move at odd depths
- MIN is to move at even depths
- Terminal nodes are evaluated from MAX's point of view

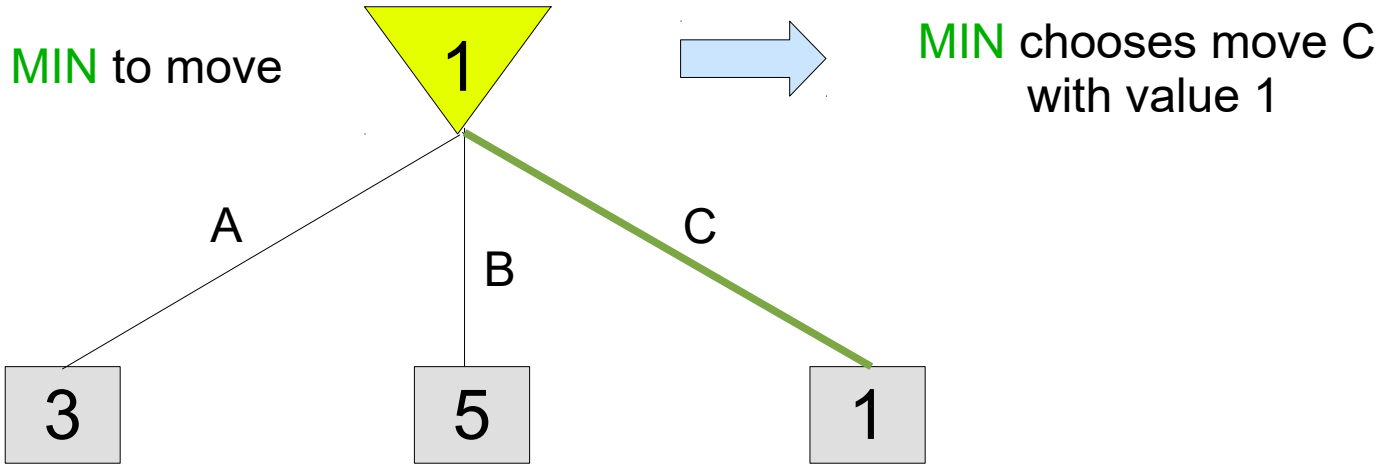
Optimal strategies

- Perfect play for deterministic, perfect-information games
  - Find the best strategy for MAX assuming an infallible MIN opponent.
  - Assumption: Both players play optimally
- Basic idea:
  - the terminal positions are evaluated form MAX's point of view
  - MAX player tries to maximize the evaluation of the position



Optimal strategies

- Perfect play for deterministic, perfect-information games
  - Find the best strategy for MAX assuming an infallible MIN opponent.
  - Assumption: Both players play optimally
- Basic idea:
  - the terminal positions are evaluated form MAX's point of view
  - MAX player tries to maximize the evaluation of the position
  - MIN player tries to minimize MAX's evaluation of the position

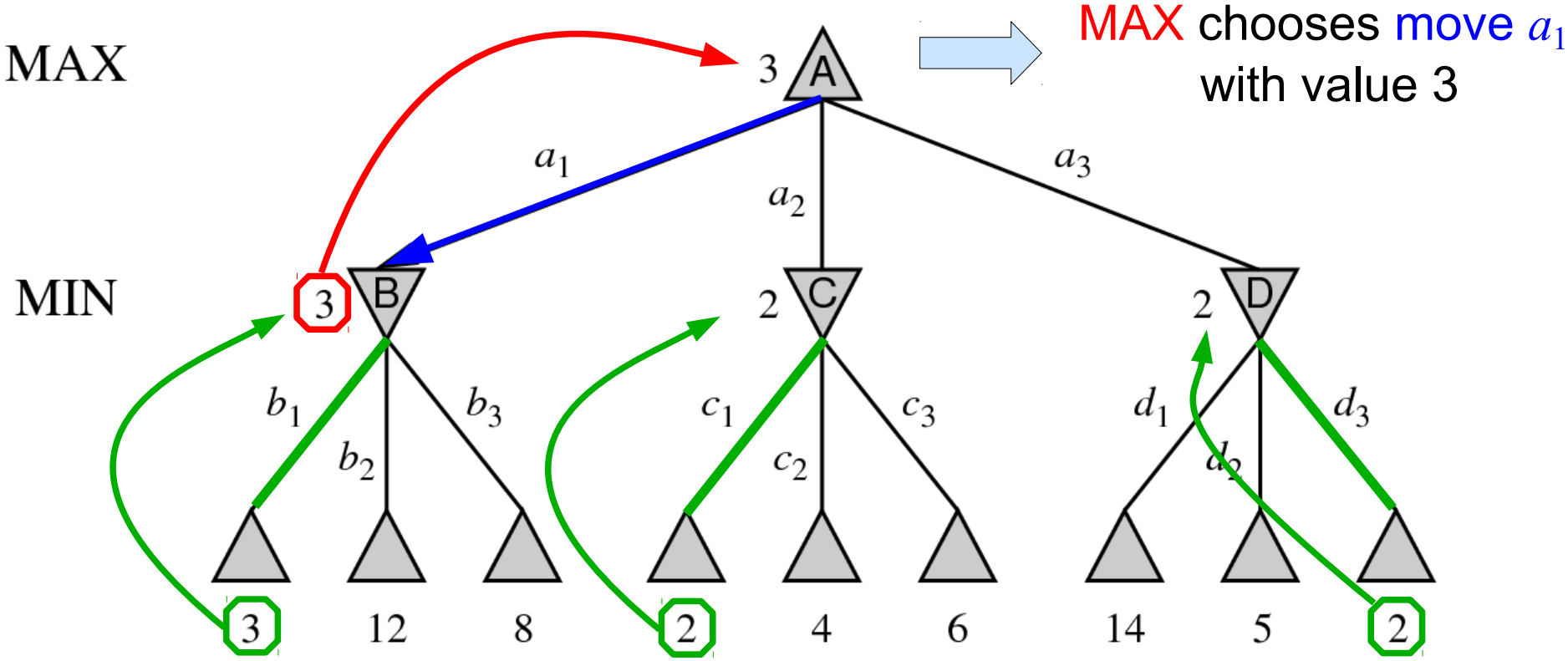


Optimal strategies

- Perfect play for deterministic, perfect-information games
  - Find the best strategy for MAX assuming an infallible MIN opponent.
  - Assumption: Both players play optimally
- Basic idea:
  - the terminal positions are evaluated form MAX's point of view
  - MAX player tries to maximize the evaluation of the position
  - MIN player tries to minimize MAX's evaluation of the position
- Minimax value
  - Given a game tree, the optimal strategy can be determined by using the minimax value of each node:

$$\text{MINIMAX}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in SUCCESSORS(n)} \text{MINIMAX}(s) & \text{if } n \text{ is a MAX node} \\ \min_{s \in SUCCESSORS(n)} \text{MINIMAX}(s) & \text{if } n \text{ is a MIN node} \end{cases}$$

Depth-Two Minimax Search Tree



Minimax maximizes the worst-case outcome for MAX.

Minimax Algorithm

```
function MINIMAX-DECISION(state) returns an action
    inputs: state, current state in game
    v ← MAX-VALUE(state)
    return action a which has value v and a, s is in SUCCESSORS(state)
```

```
function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do v ← MAX(v, MIN-VALUE(s))
    return v
```

```
function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do v ← MIN(v, MAX-VALUE(s))
    return v
```

NegaMax Formulation

- The minimax algorithm can be reformulated in a simpler way
  - for evaluation functions that are symmetric around 0 (zero-sum)
- Basic idea:
  - Assume that evaluations in all nodes (and leaves) are always from the point of view of the player that is to move
    - the MIN-player now also maximizes its value
  - As the values are zero-sum, the value of a position for MAX is equal to minus the value of position for MIN

    → NegaMax = Negated Maximum

$$\text{NEGAMAX}(n) = \begin{cases} \text{UTILITY}(n) & \text{if } n \text{ is a terminal state} \\ \max_{s \in SUCCESSORS(n)}(-\text{NEGAMAX}(s)) & \text{if } n \text{ is an internal node} \end{cases}$$

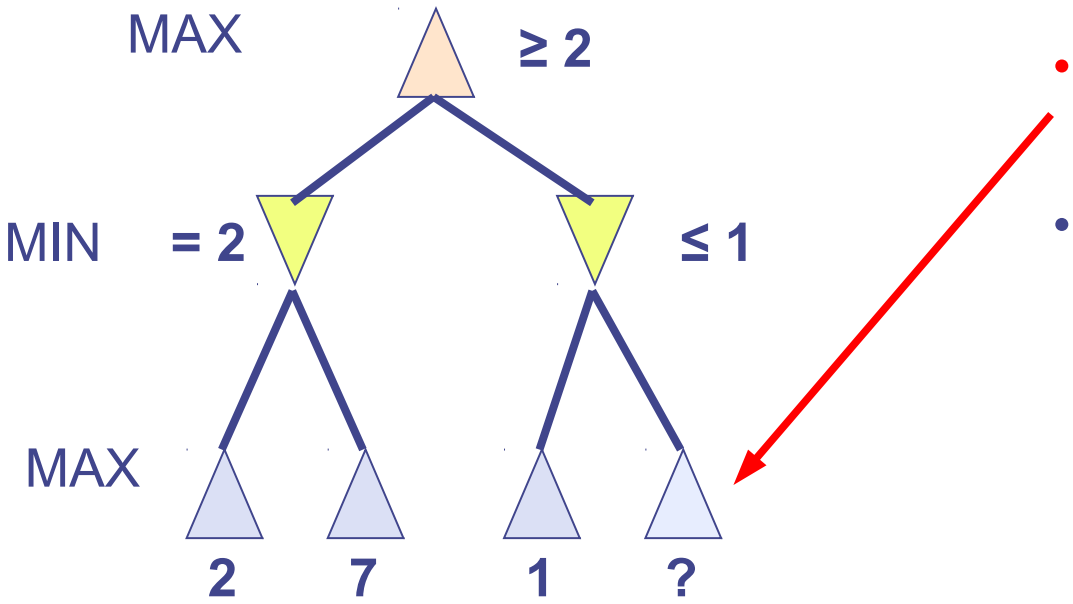
Properties of Minimax Search

- Completeness
  - Yes, if tree is finite
    - e.g., chess guarantees this through separate rules (3-fold repetition or 50 moves w/o irreversible moves are draw)
  - Note that there might also be finite solutions in infinite trees
- Optimality
  - Yes, if the opponent also plays optimally
    - If not, there might be better strategies (→ *opponent modeling*)
- Time Complexity
  - $O(b^m)$
  - has to search all nodes up to maximum depth (i.e., until terminal positions are reached)
  - for many games unfeasible (e.g., chess: $b \approx 35, m \approx 60$)
- Space Complexity
  - search proceeds depth-first → $O(m \cdot b)$

Alpha-Beta Pruning

- Minimax needs to search an exponential number of states
- Possible solution:
  - Do not examine every node
  - remove nodes that can not influence the final decision

> *"If you have an idea that is surely bad, don't take the time to see how truly awful it is."* -- Pat Winston



- We don't need to compute the value at this node.
- No matter what it is, it can't affect the value of the root node.

Alpha-Beta Pruning

Maintains two values $[\alpha,\beta]$ for all nodes in the current path

- Alpha:
  - the value of the best choice (i.e., highest value) for the MAX player at any choice node for MAX in the current path

  → MAX can obtain a value of at least α
- Beta:
  - the value of the best choice (i.e., lowest value) for the MIN player at any choice node for MIN in the current path

  → MIN can make sure that MAX obtains a value of at most β

The values are initialized with $[-\infty, +\infty]$

Alpha-Beta Pruning

Alpha and Beta are used for pruning the search tree:

- Alpha-Cutoff:
  - if we find a move with value $\leq \alpha$ at a MIN node, we do not examine alternatives to this move
  - we already know that MAX can achieve a better result in a different variation
- Beta-Cutoff:
  - if we find a move with value $\geq \beta$ at a MAX node, we do not examine alternatives to this move
  - we already know that MIN can achieve a better result in a different variation

Alpha-Beta Algorithm

function ALPHA-BETA-DECISION(*state*) **returns** an action
  $v \leftarrow$ MAX-VALUE(*state*, $-\infty$, $+\infty$)
  return action *a* which has value *v* and *a*, *s* is in SUCCESSORS(*state*)

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for *a*, *s* in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

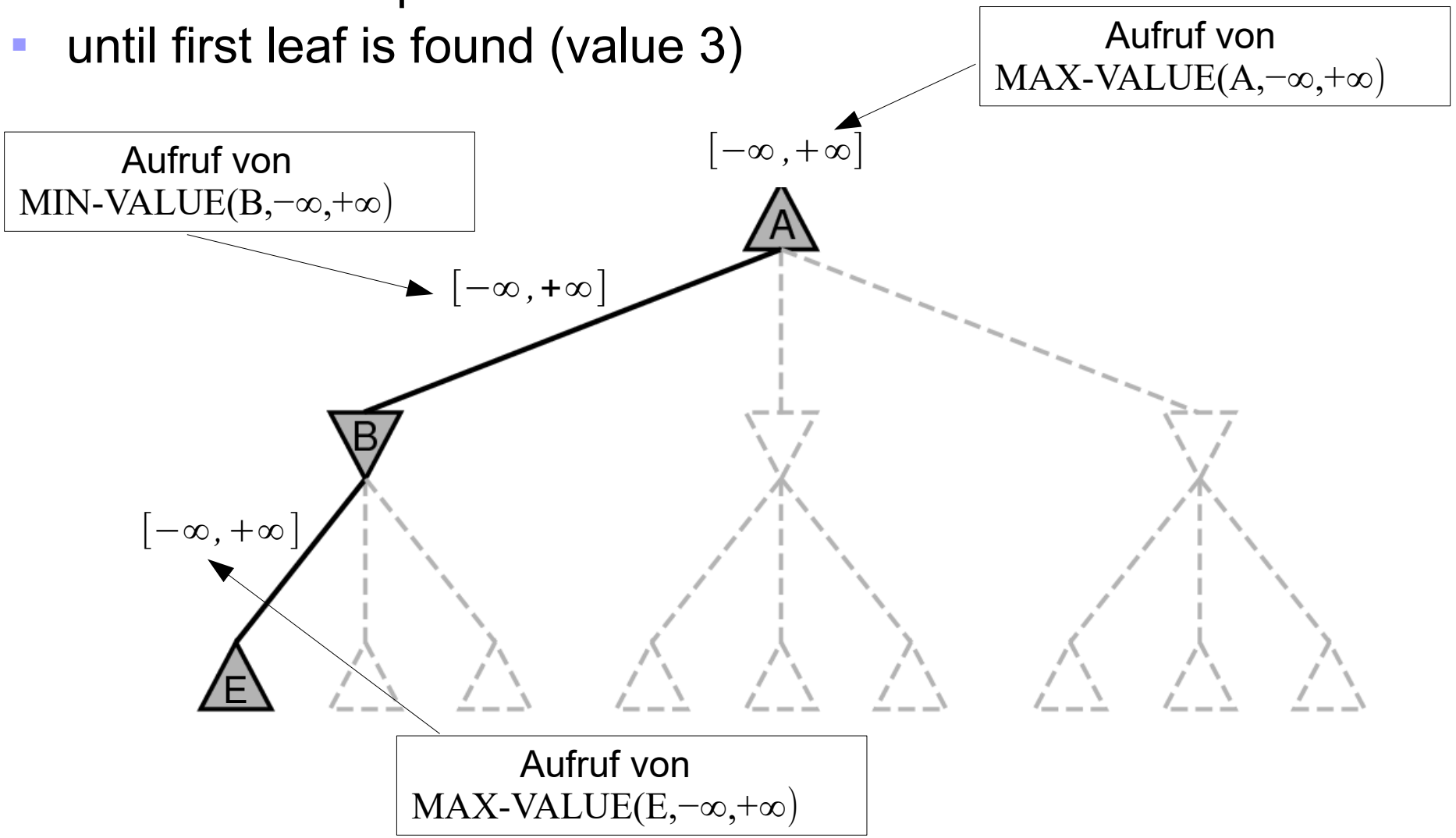
if TERMINAL-TEST(*state*) **return** UTILITY(*state*)
 $v \leftarrow +\infty$
for *a*, *s* in SUCCESSORS(*state*) **do**
 $v \leftarrow$ MIN(v,MAX-VALUE(s, α , β))
 if $v \leq \alpha$ **then return** v
 $\beta \leftarrow$ MIN(β ,v)
return v

---

function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  same as MAX-VALUE but with roles of α, β reversed

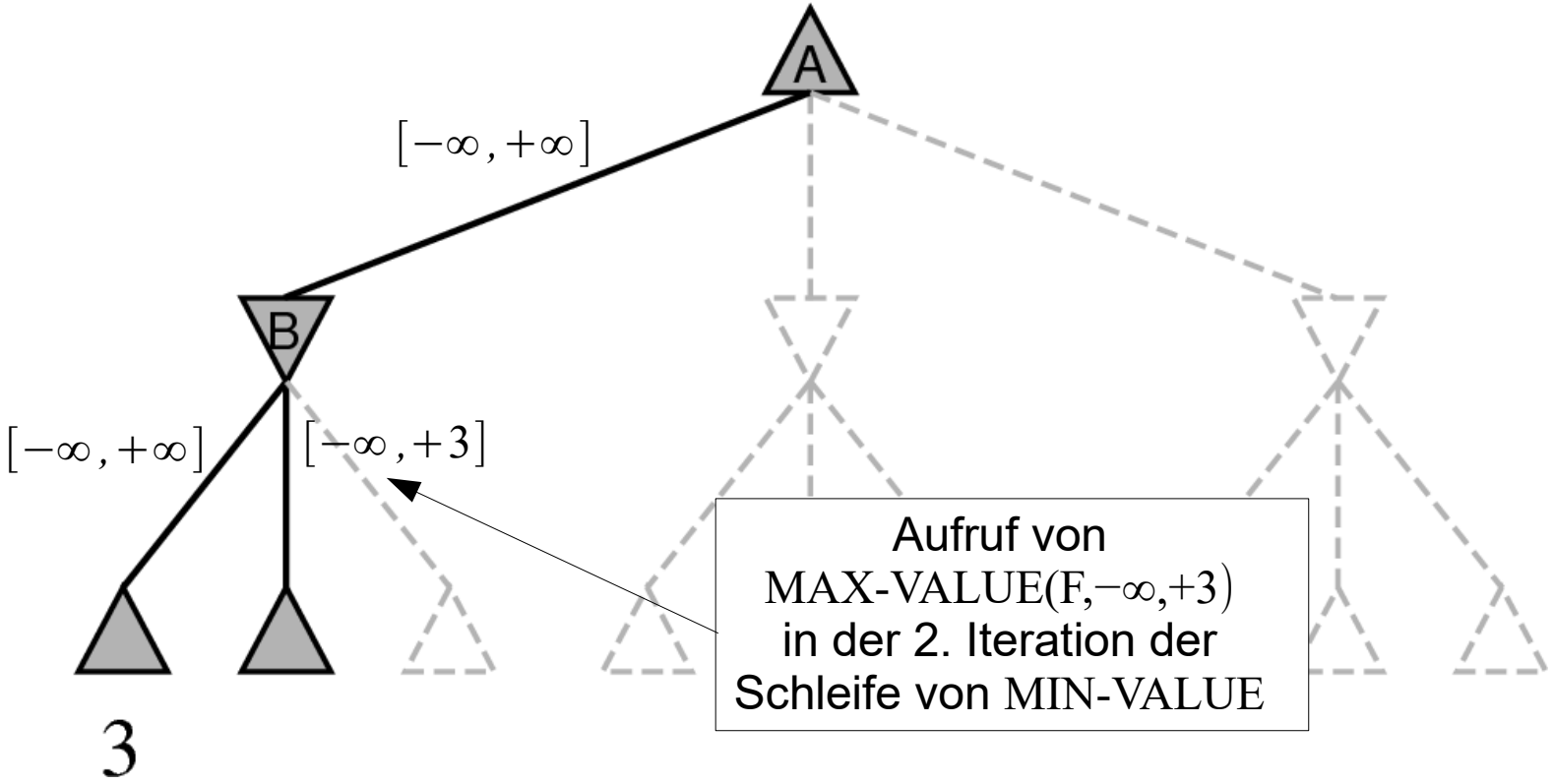
Example: Alpha-Beta

- The window is initialized with $[-\infty, +\infty]$
- search runs depth-first
- until first leaf is found (value 3)



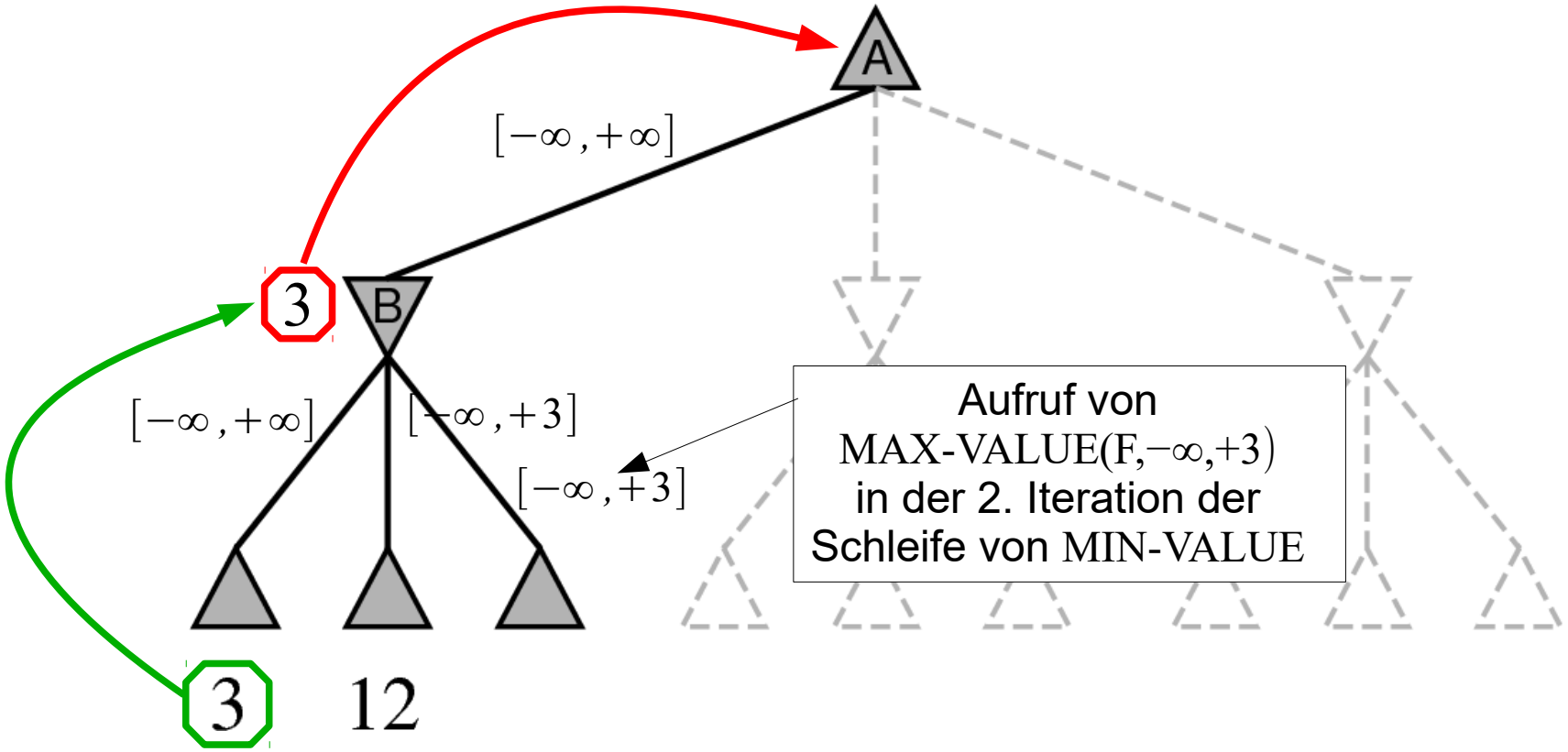
Example: Alpha-Beta

- It is followed that at node B, MIN can obtain at least 3
- Subsequent search below B is now initialized with $[-\infty, +3]$
- The leaf node (value 12) is worse for MIN (higher value for MAX)



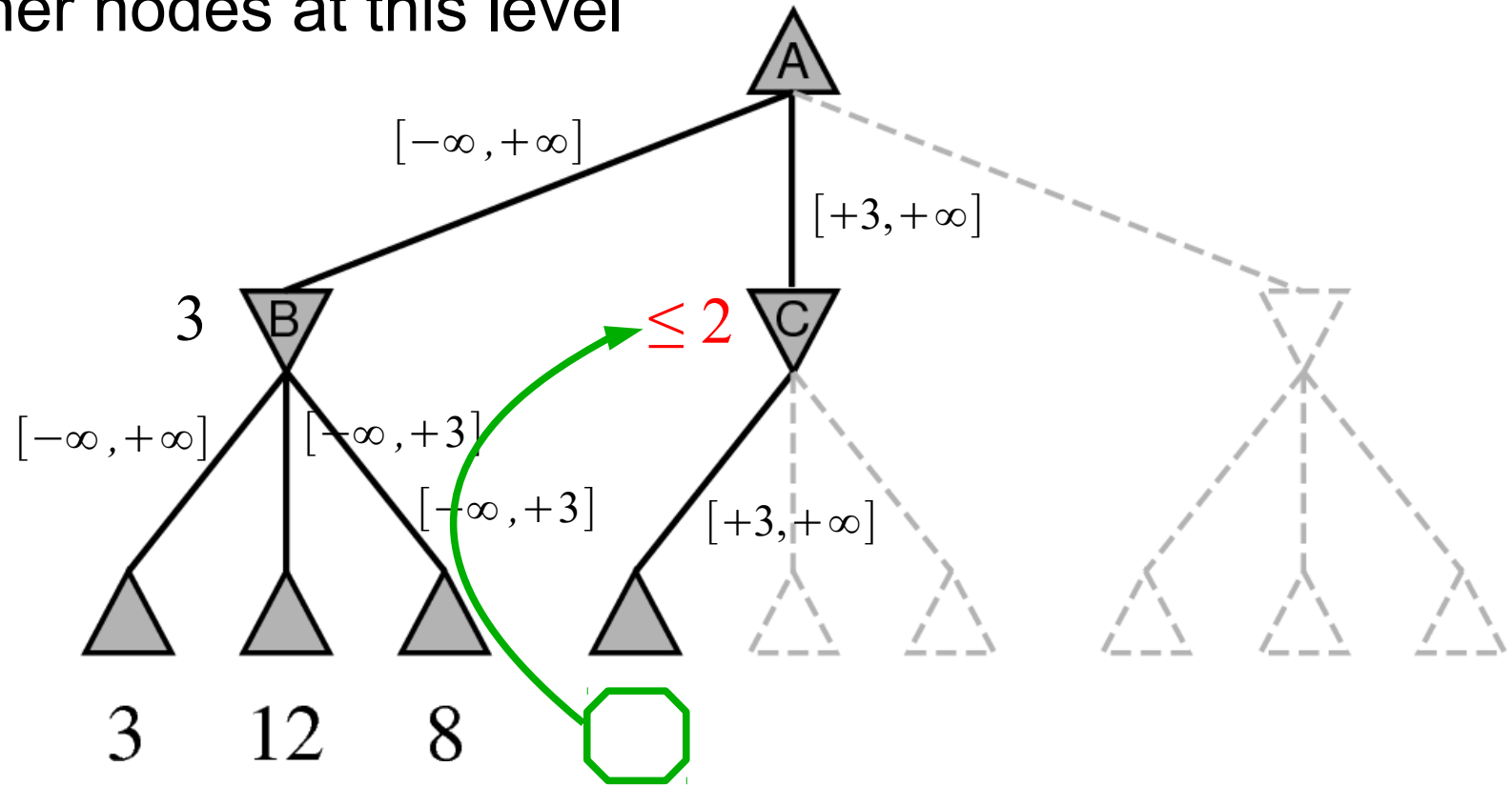
Example: Alpha-Beta

- The next leaf is also worse for MIN (value 8)
- Node B is now completed, and evaluated with 3
- The value is propagated up to A as a new minimum for MAX



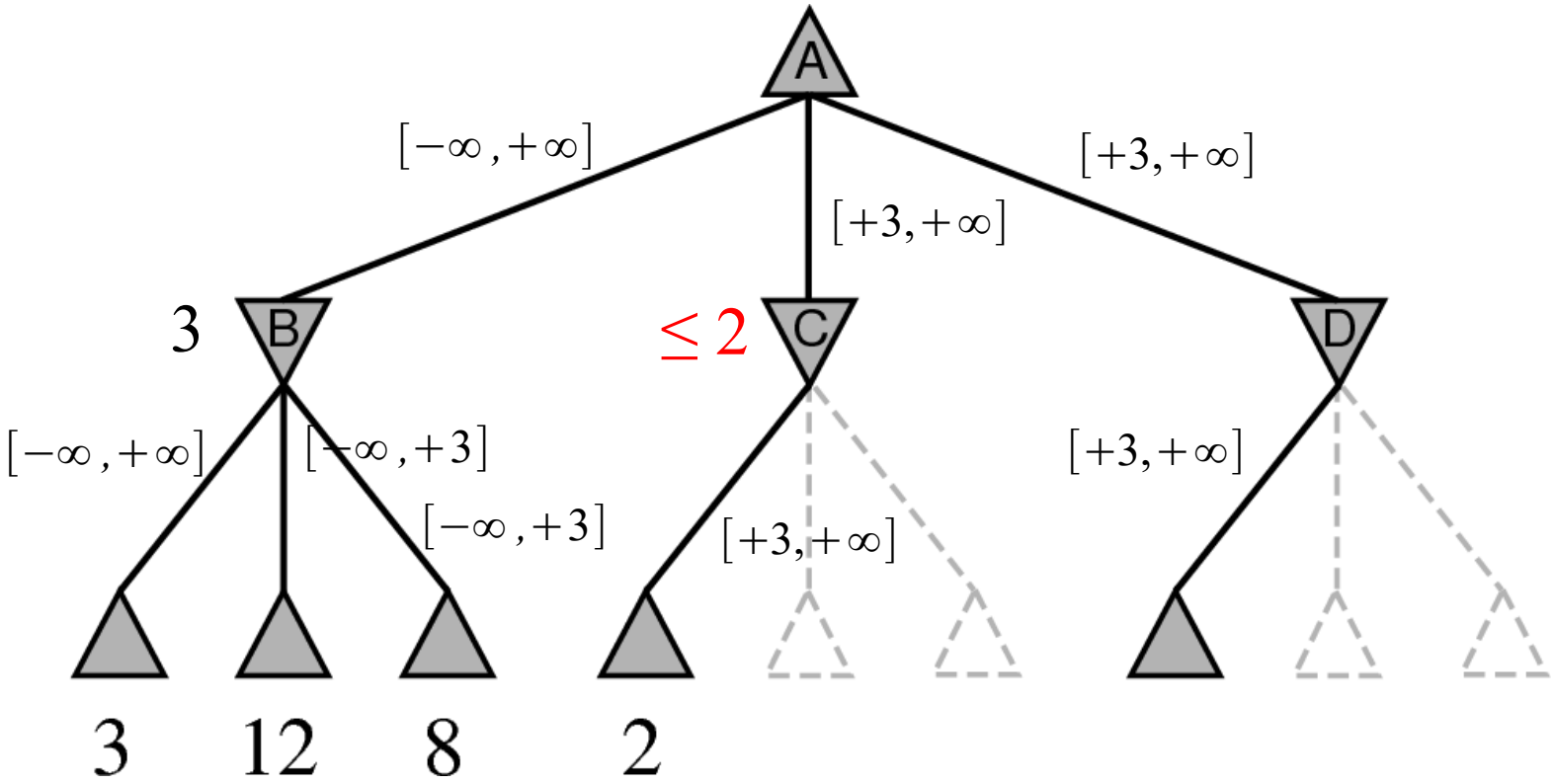
Example: Alpha-Beta

- Subsequent searches now know that MAX can achieve at least 3, i.e., the alpha-beta window is $[+3, +\infty]$
- The value 2 is found below the min node
- As the value is outside the window ($2 < 3$), we can prune all other nodes at this level



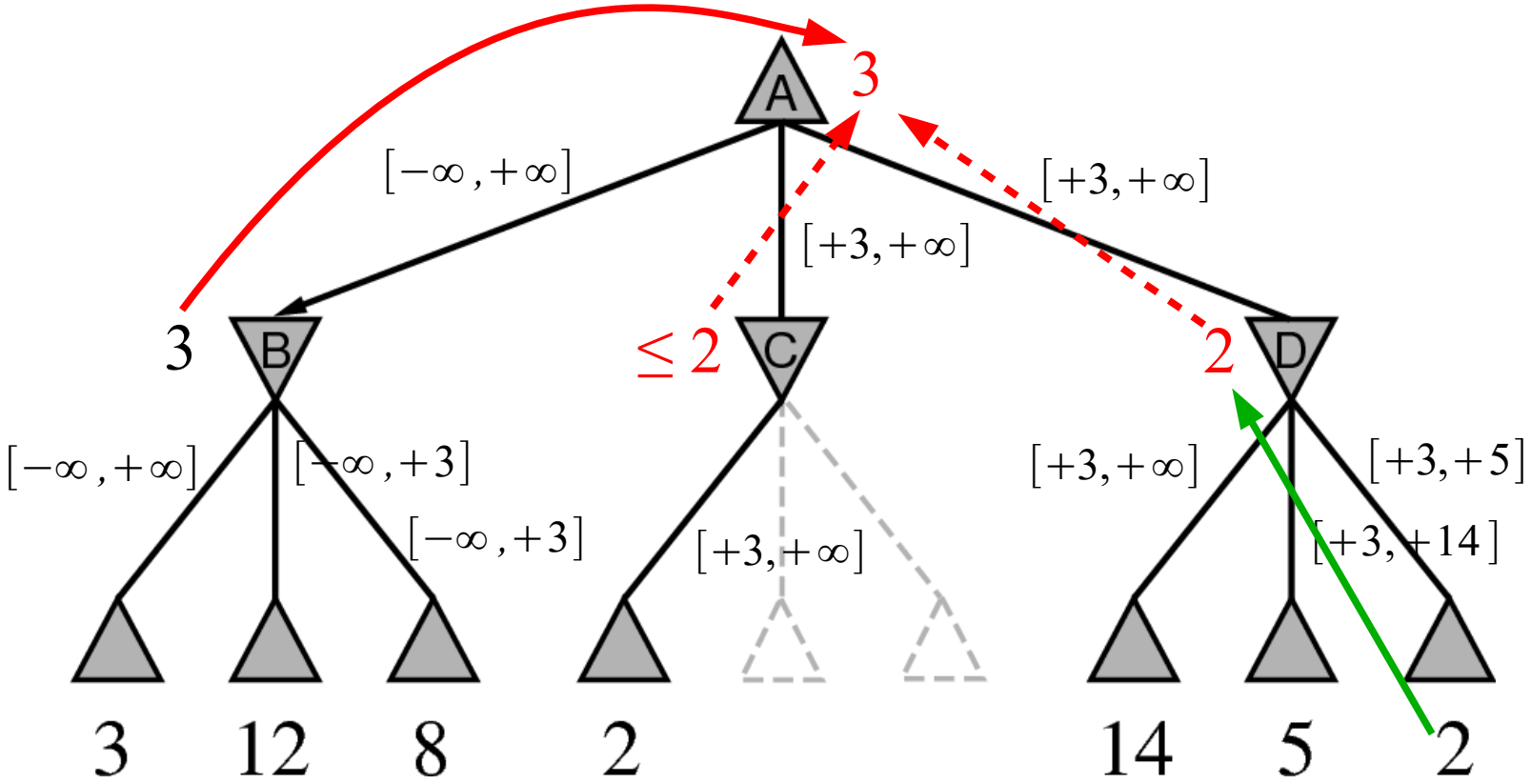
Example: Alpha-Beta

- Subsequent searches now know that MAX can achieve at least 3, i.e., the alpha-beta window is $[+3, +\infty]$
- The value 14 is found below the min node



Example: Alpha-Beta

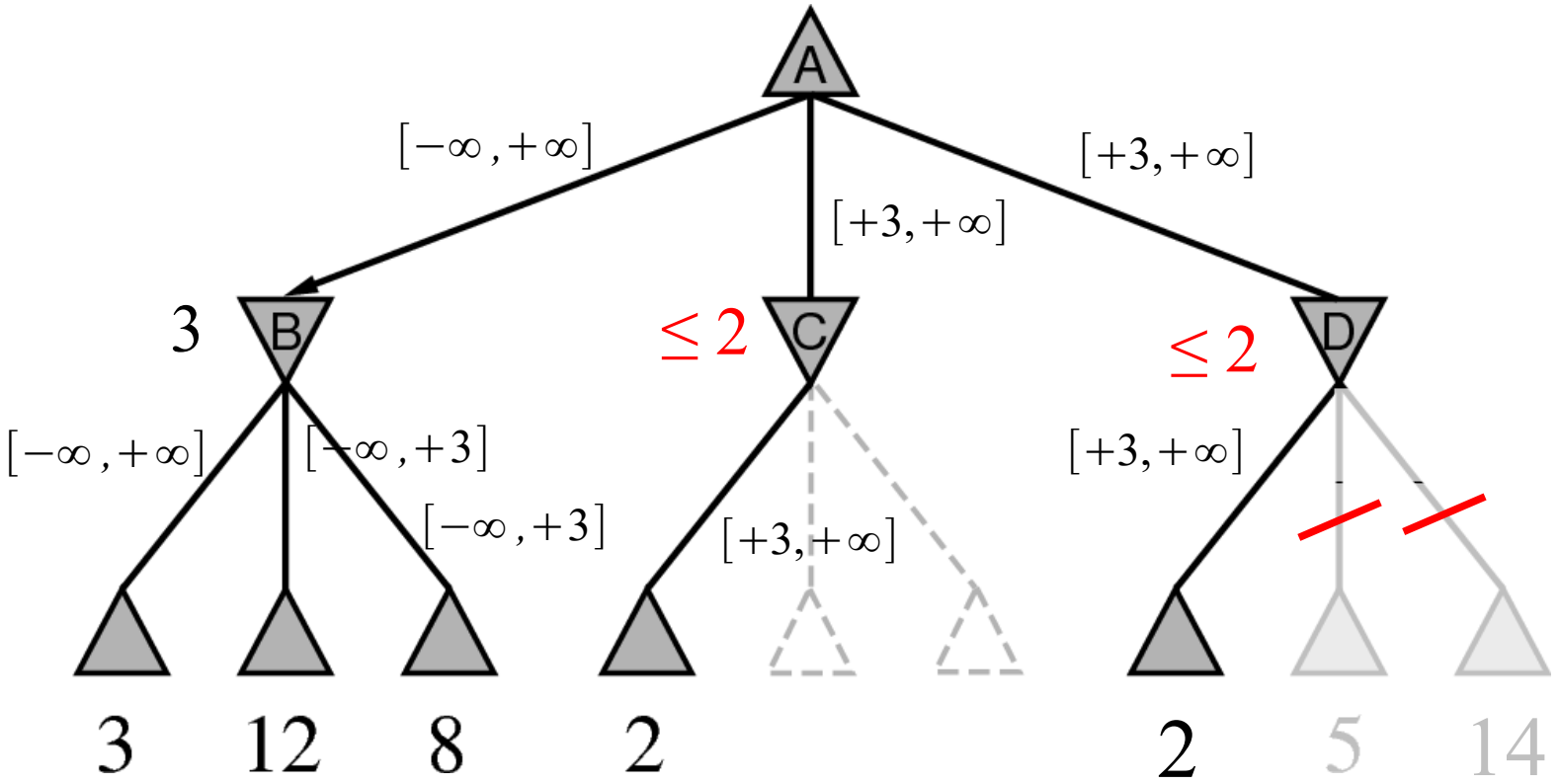
- The next search now knows that MAX can achieve at least 3 but MIN can hold him down to 14
- i.e., the alpha-beta window is $[+3, +14]$
- For the final node the window is $[+3, +5]$



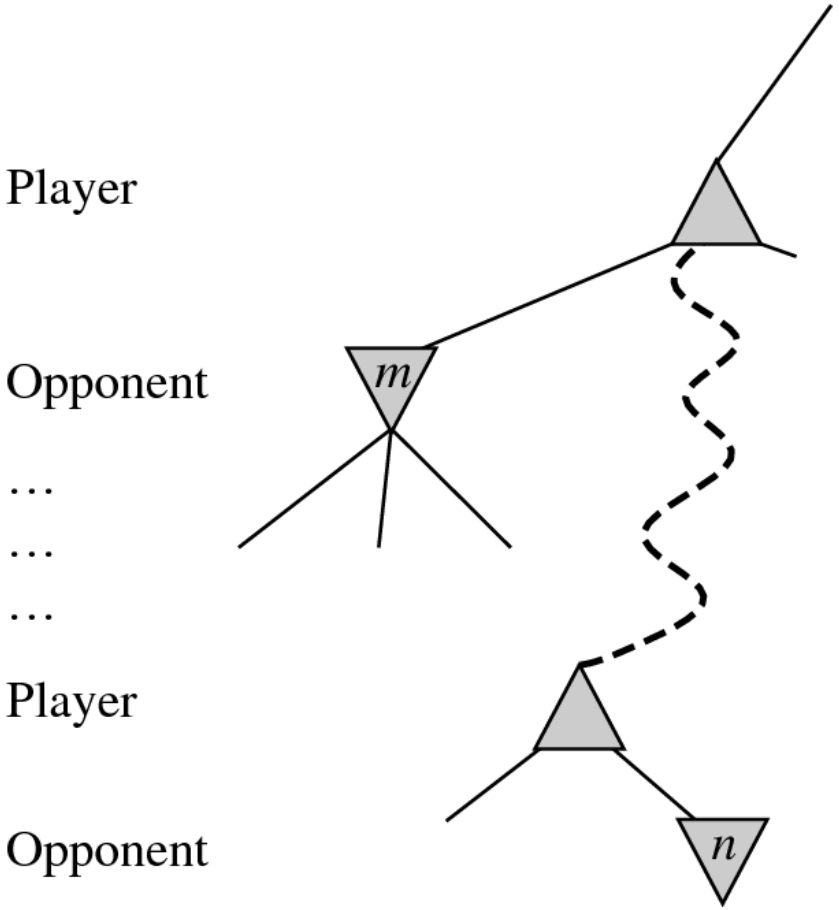
Evaluation Order

- Note that the order of the evaluation of the nodes is crucial
- e.g., if in node D, the node with evaluation 2 is seached first, another cutoff would have been possible

→ good move order is crucial for good performance



General Alpha-Beta Pruning

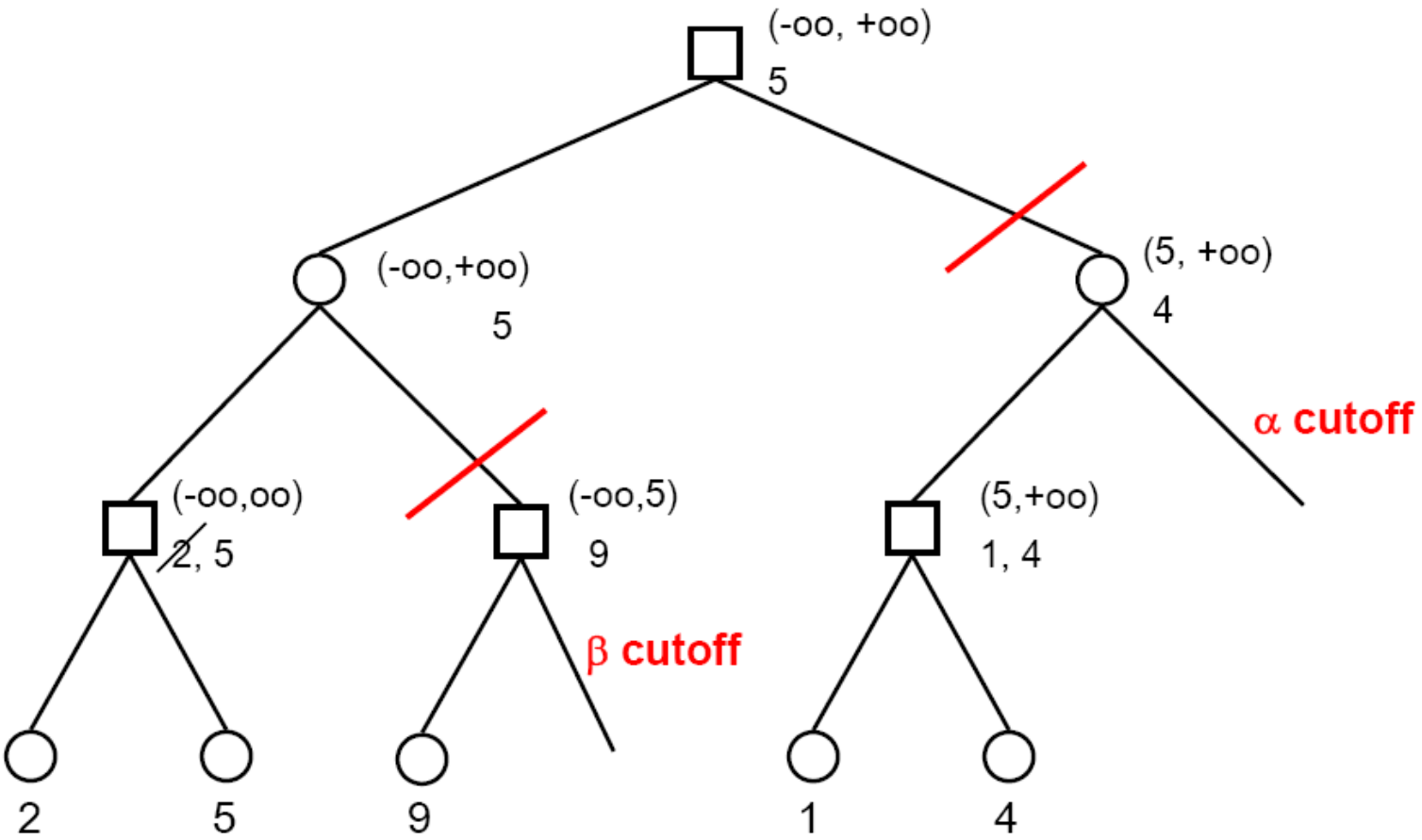


- Consider a node n somewhere in the tree
- If Player has a better choice
  - at parent node of n
  - or at any choice point further up

  n will never be reached in actual play.
- Hence we can prune n
  - as soon as we can establish that there is a better choice

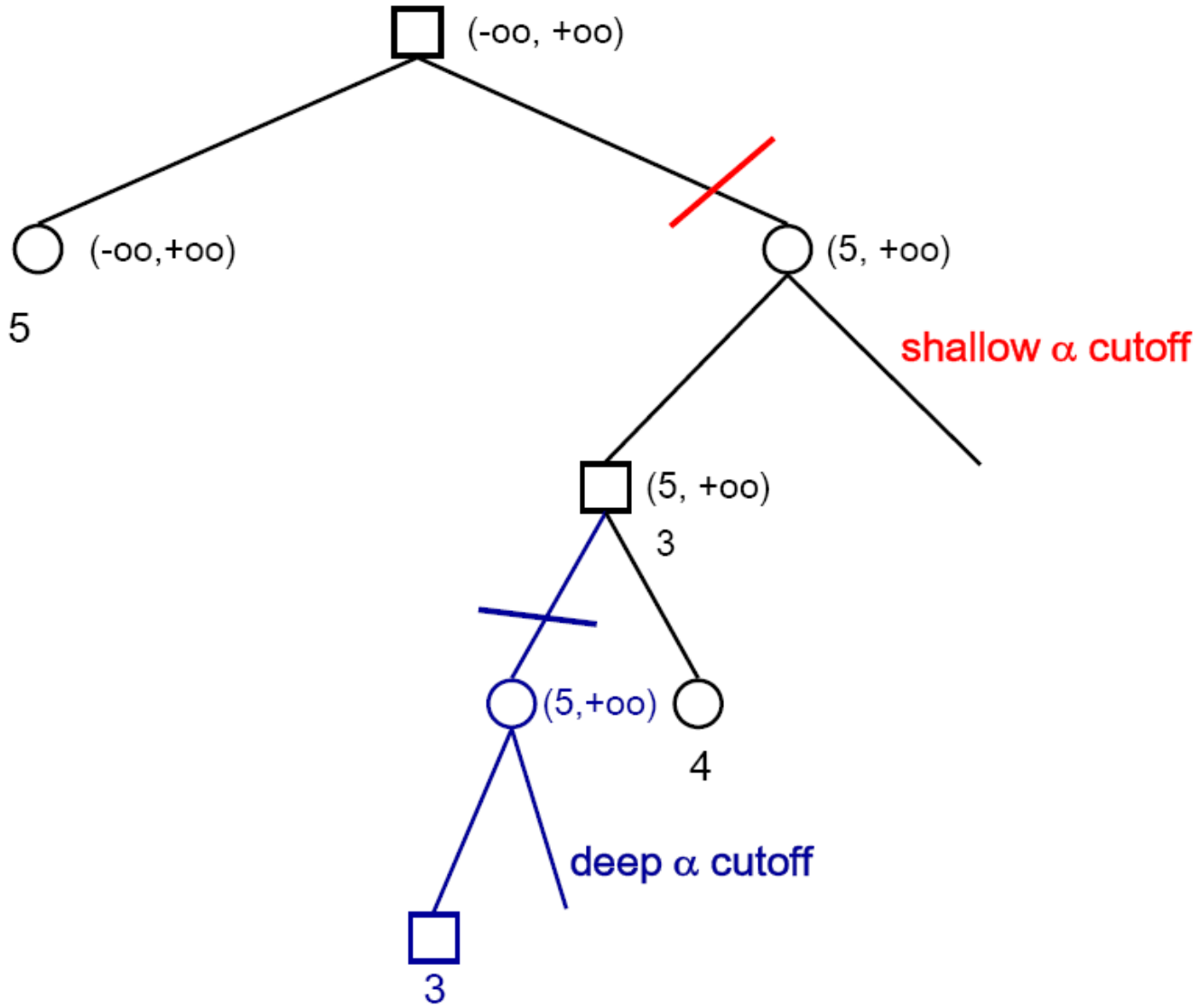
Alpha-Cutoff vs. Beta-Cutoff

- Of course, cutoffs can also occur at MAX-nodes

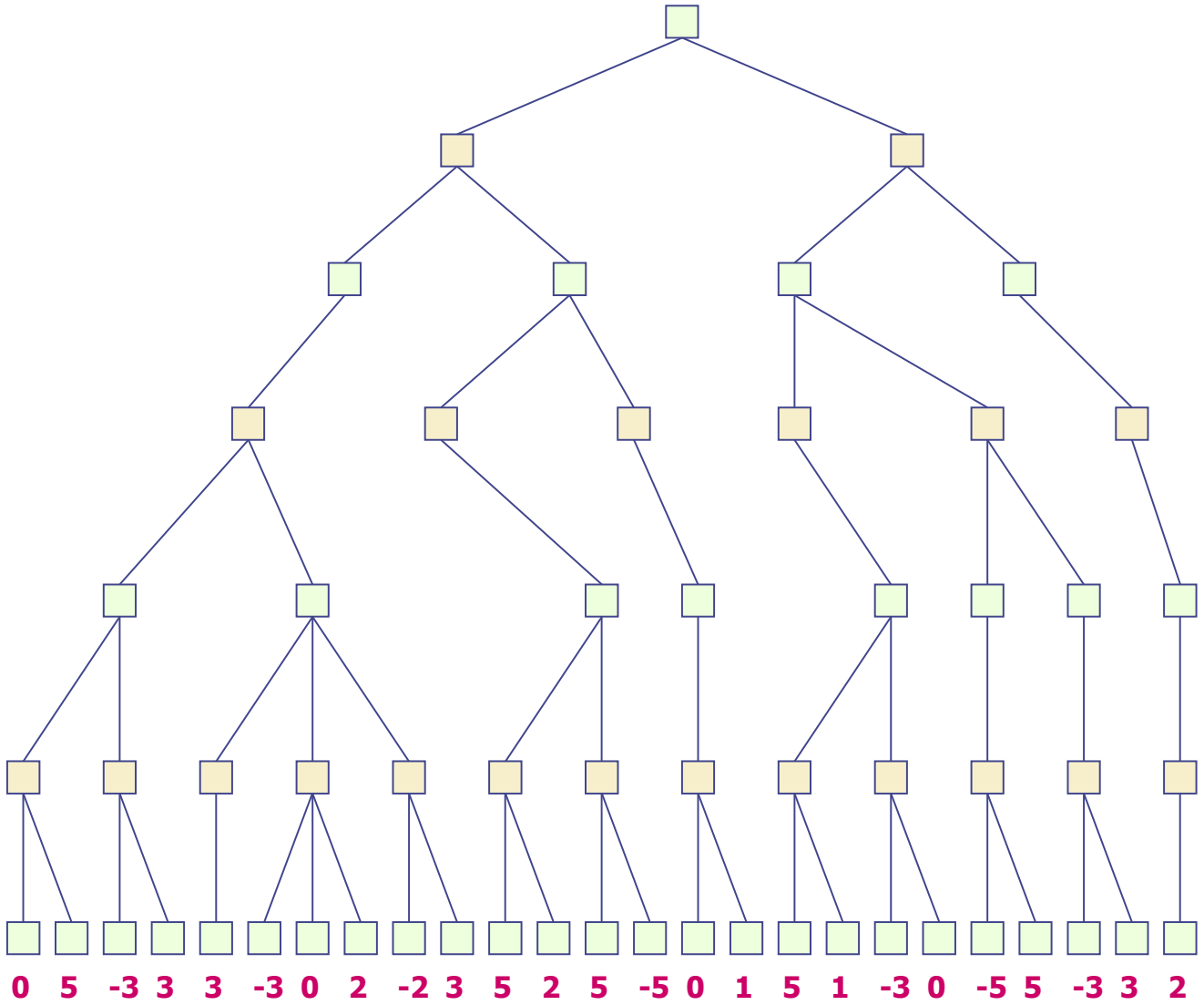


Shallow vs. Deep Cutoffs

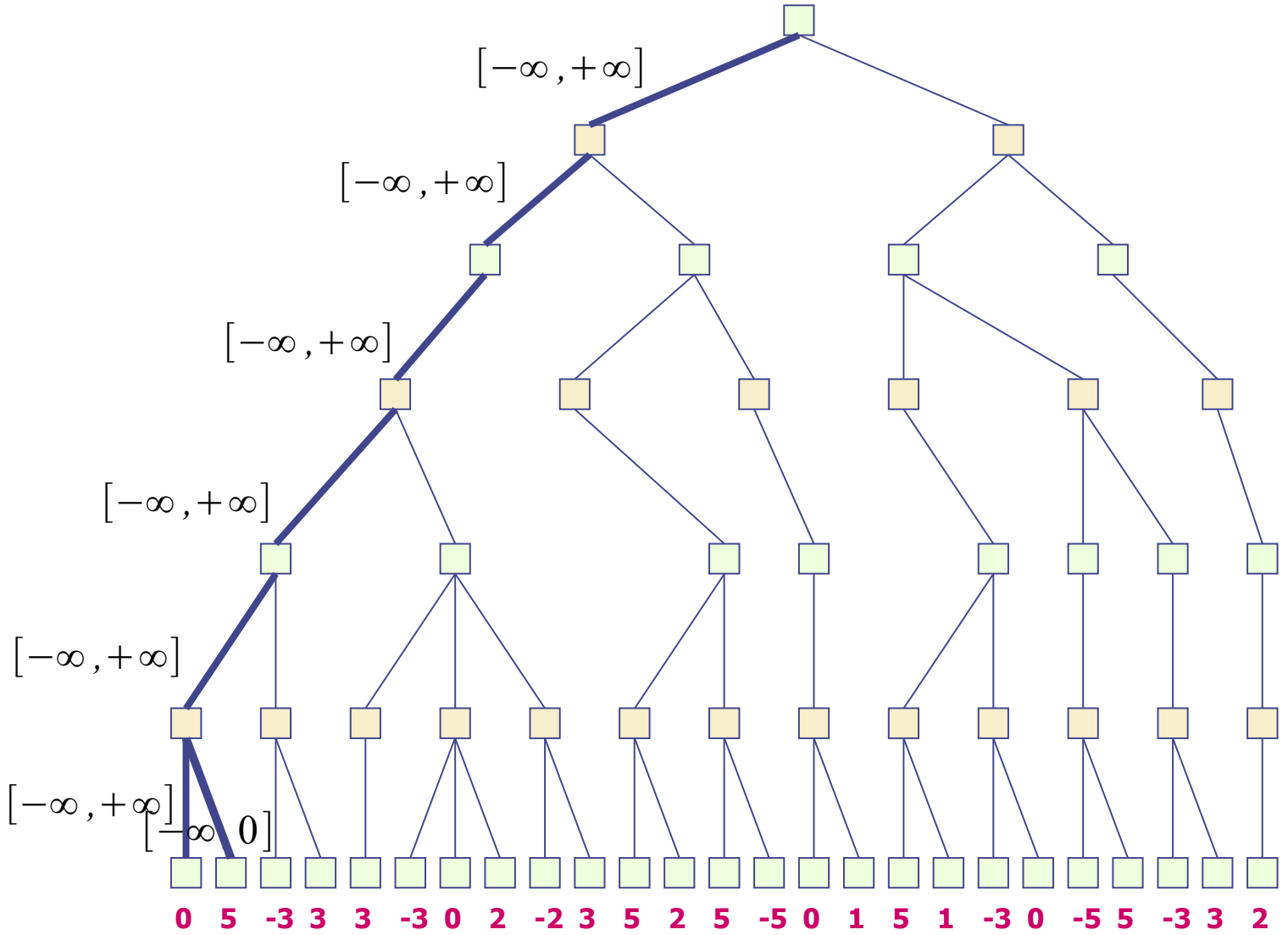
- Cutoffs may occur arbitrarily deep in (sub-)trees



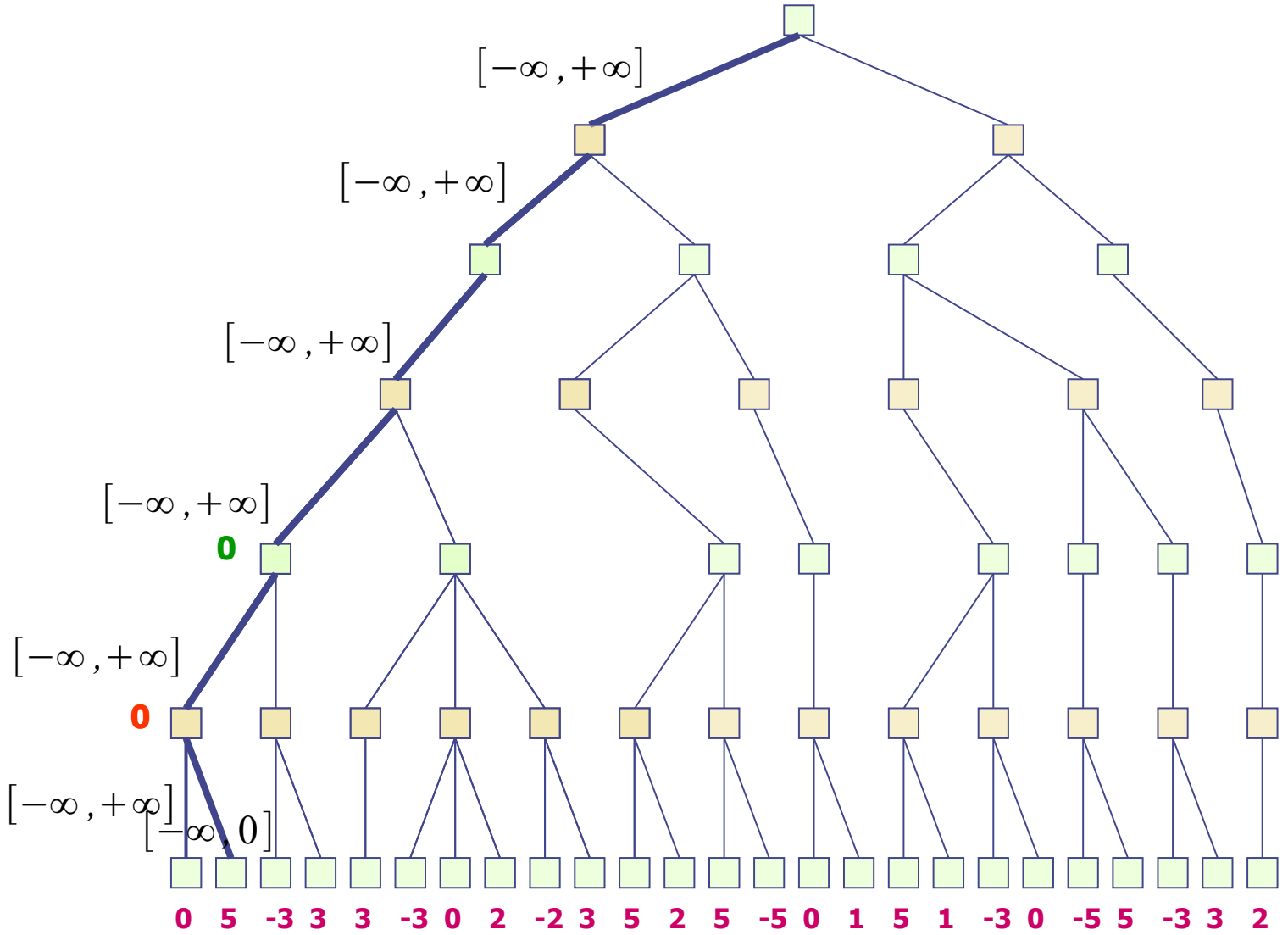
Alpha-Beta Example



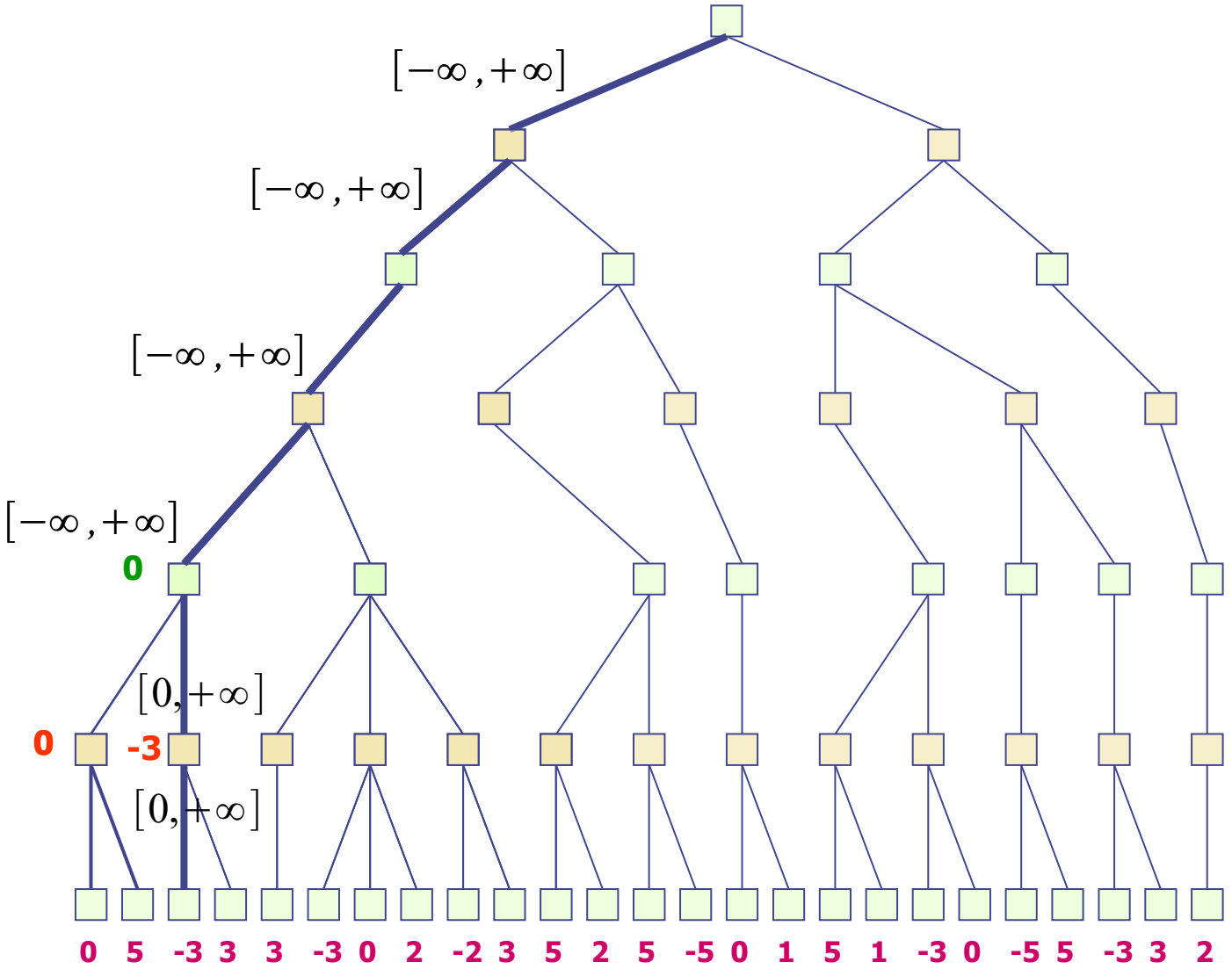
Alpha-Beta Example



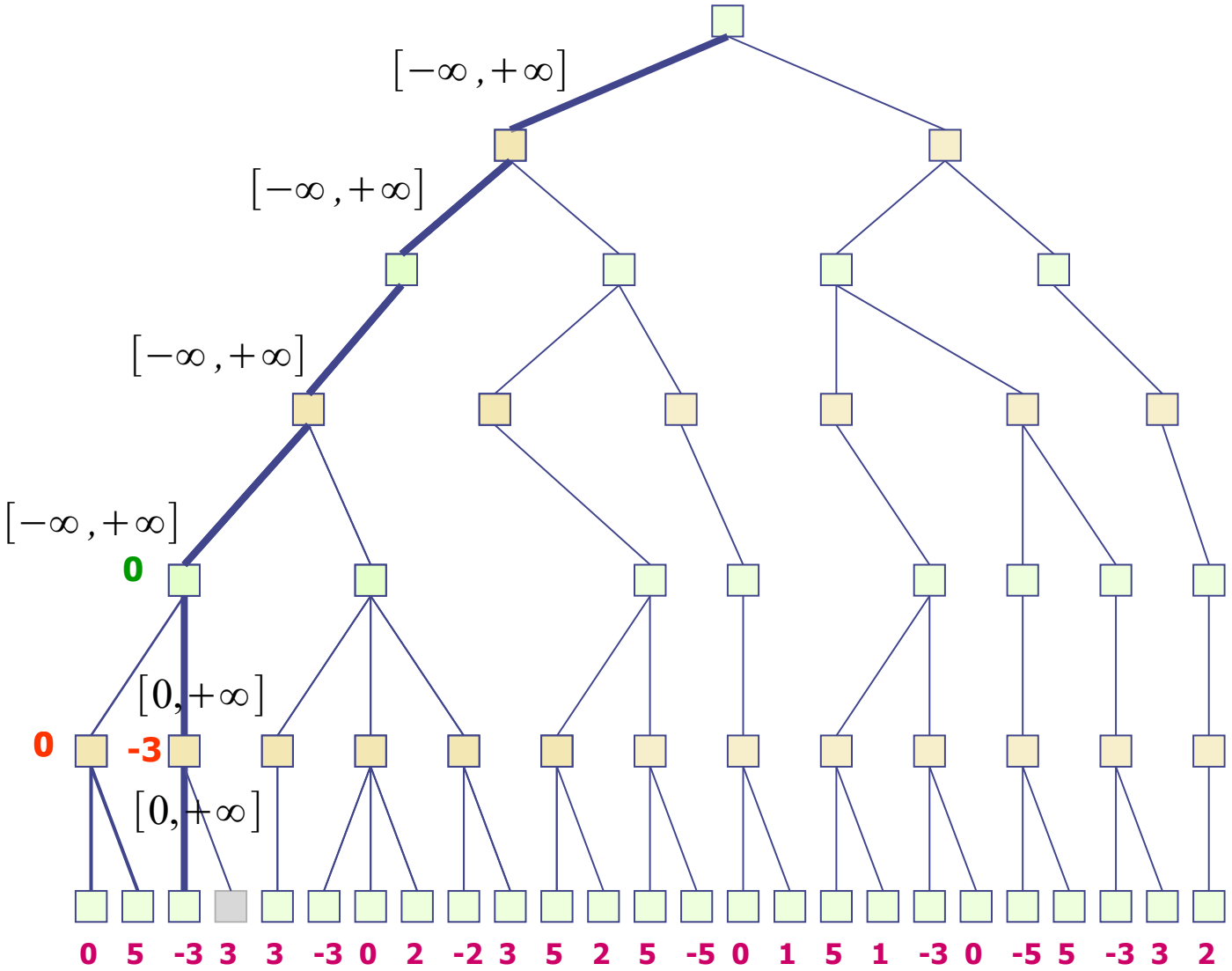
Alpha-Beta Example



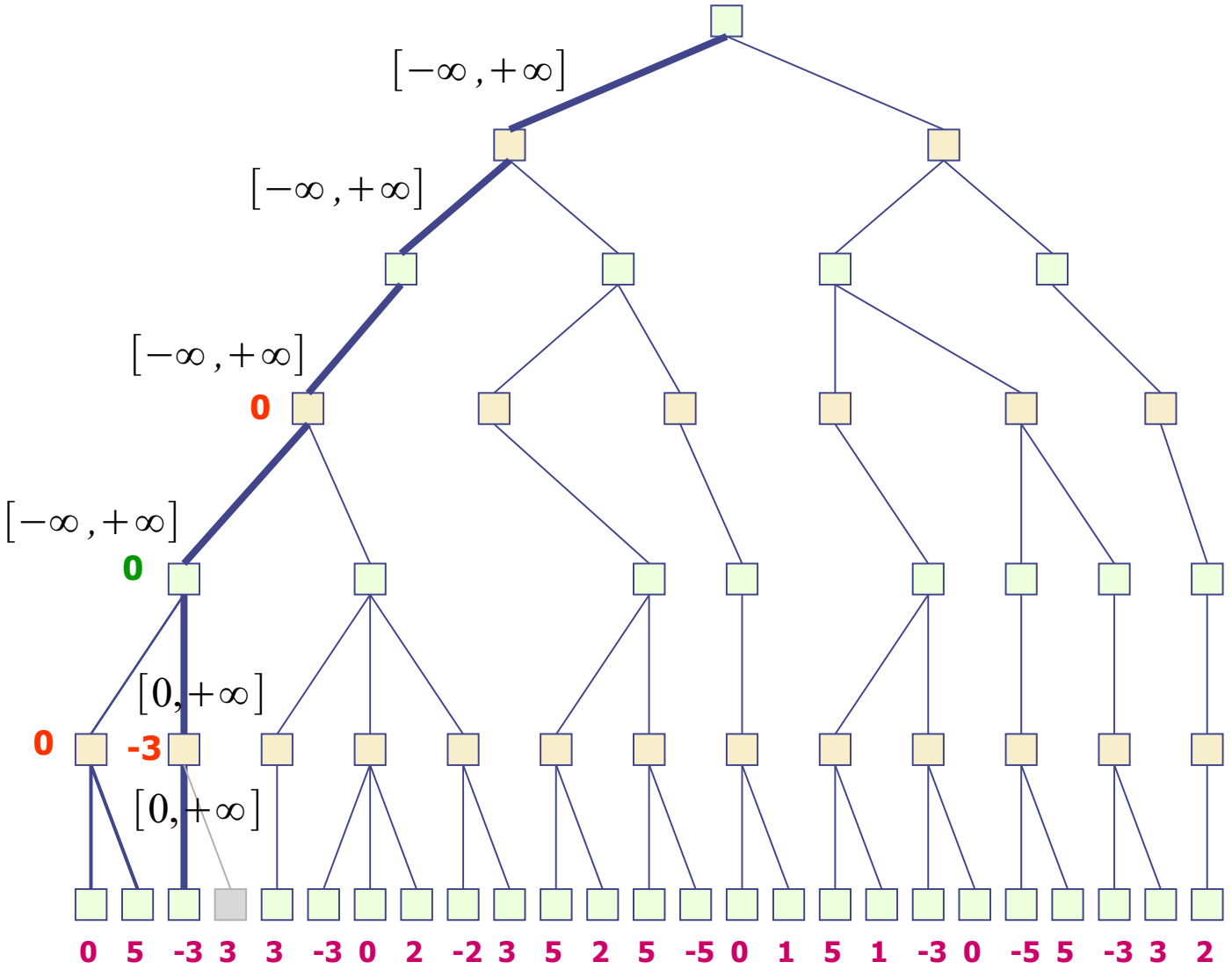
Alpha-Beta Example



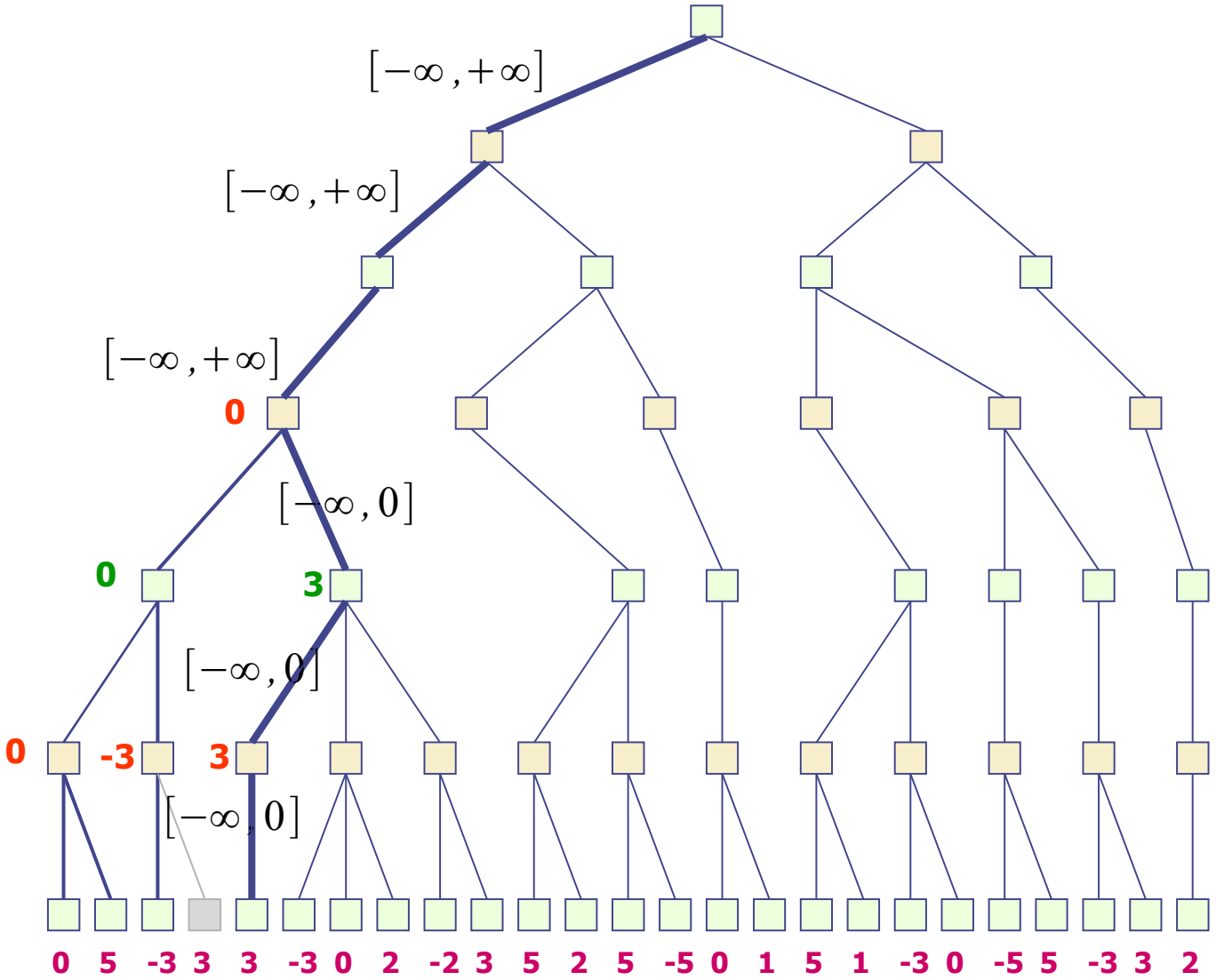
Alpha-Beta Example



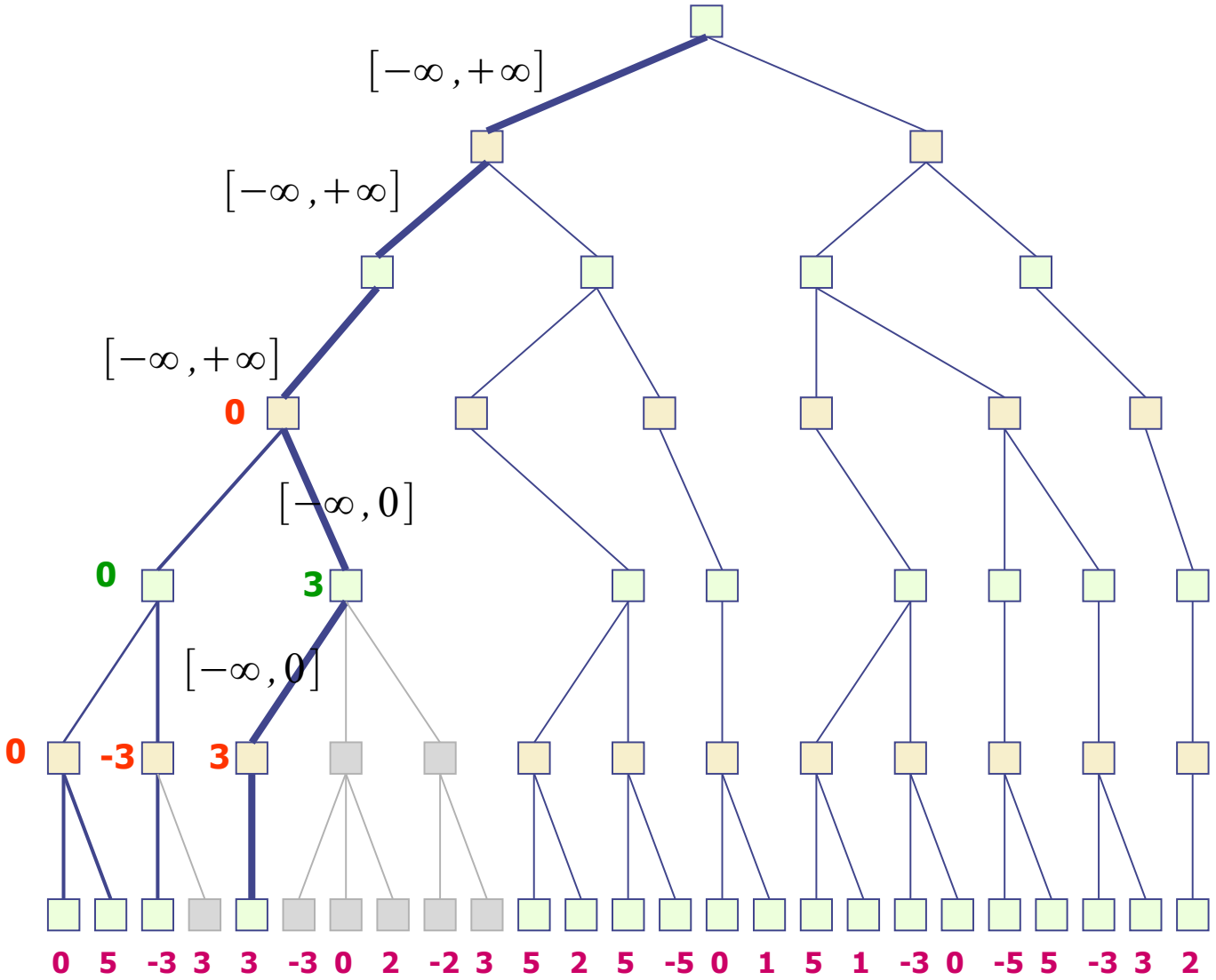
Alpha-Beta Example



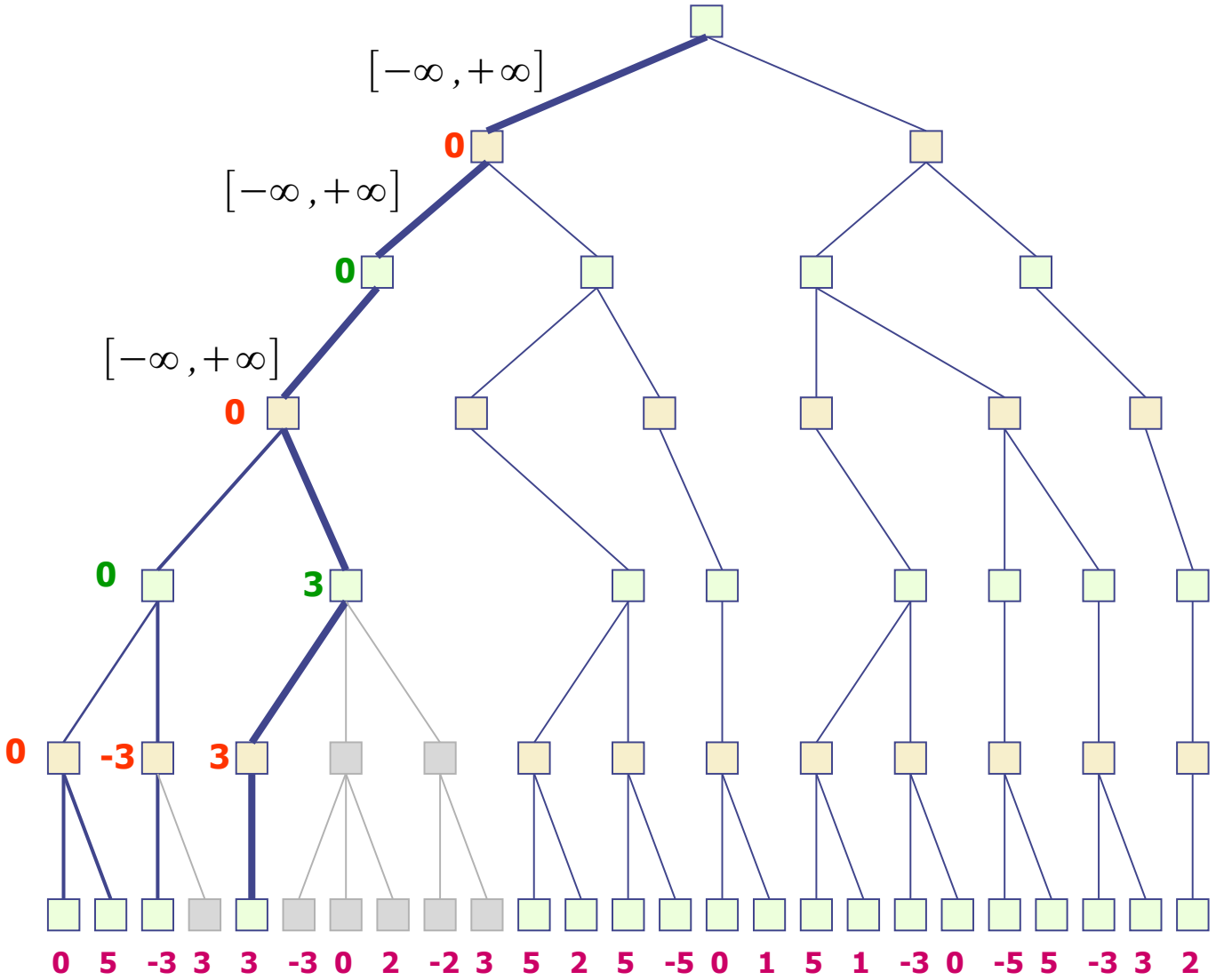
Alpha-Beta Example



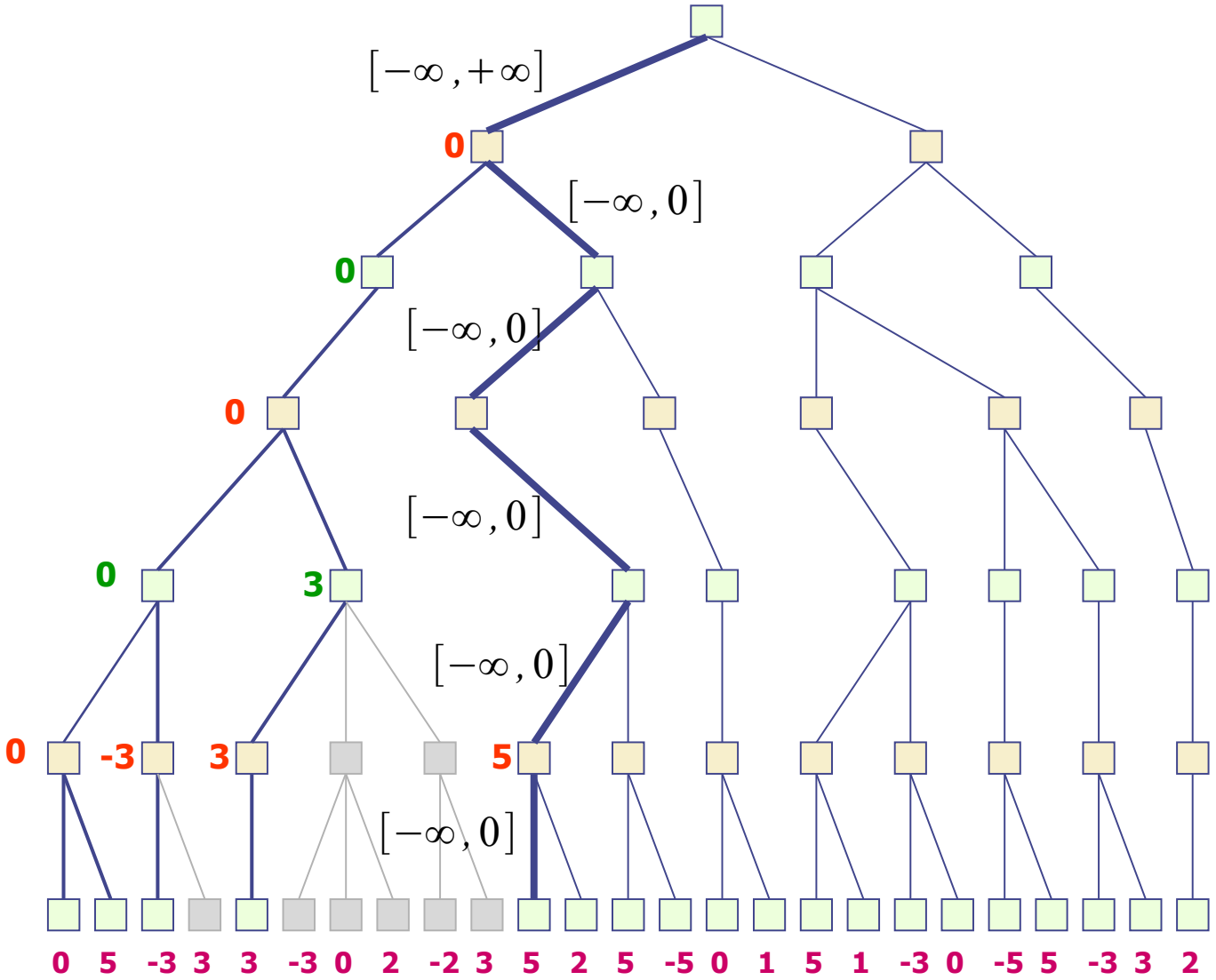
Alpha-Beta Example



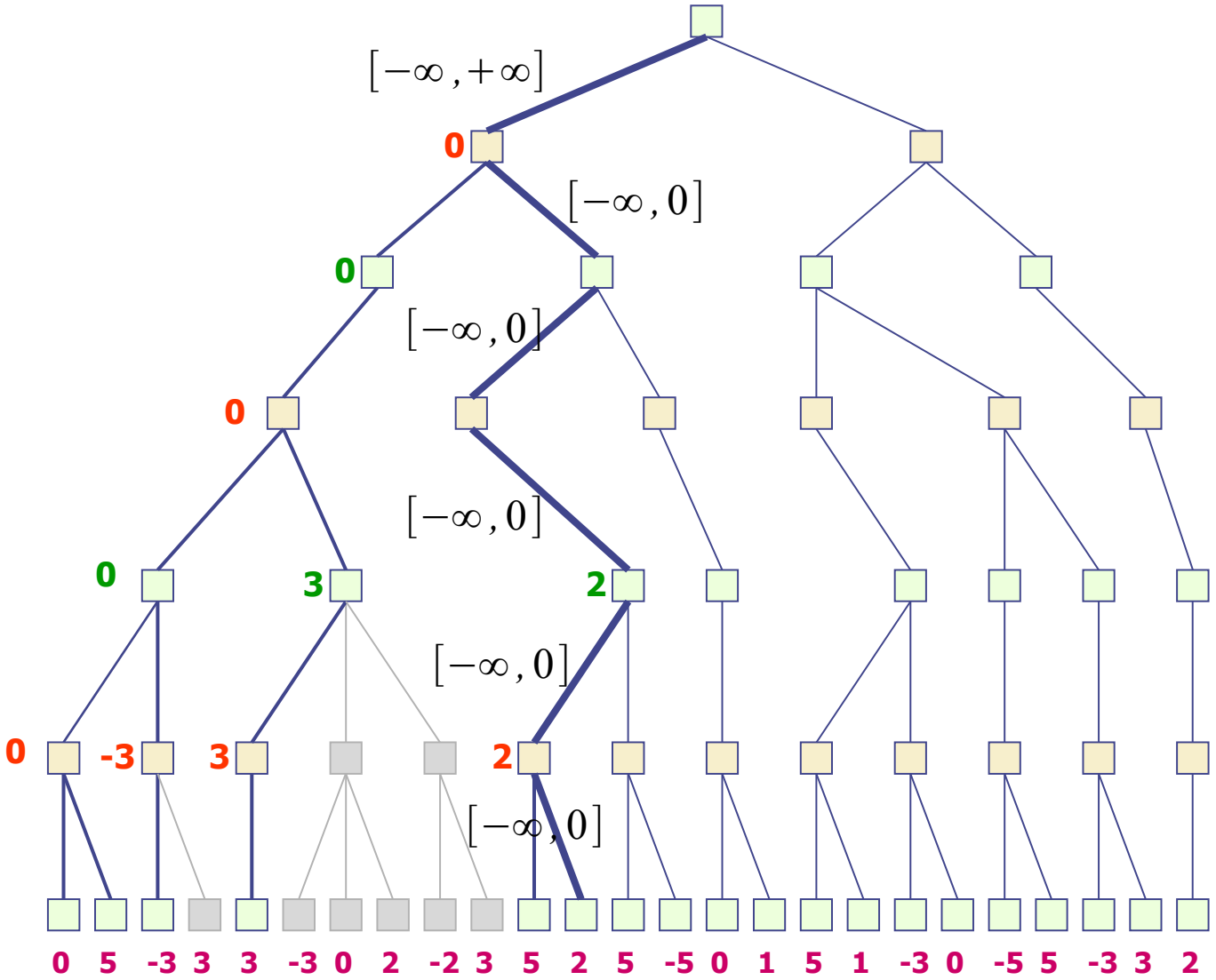
Alpha-Beta Example



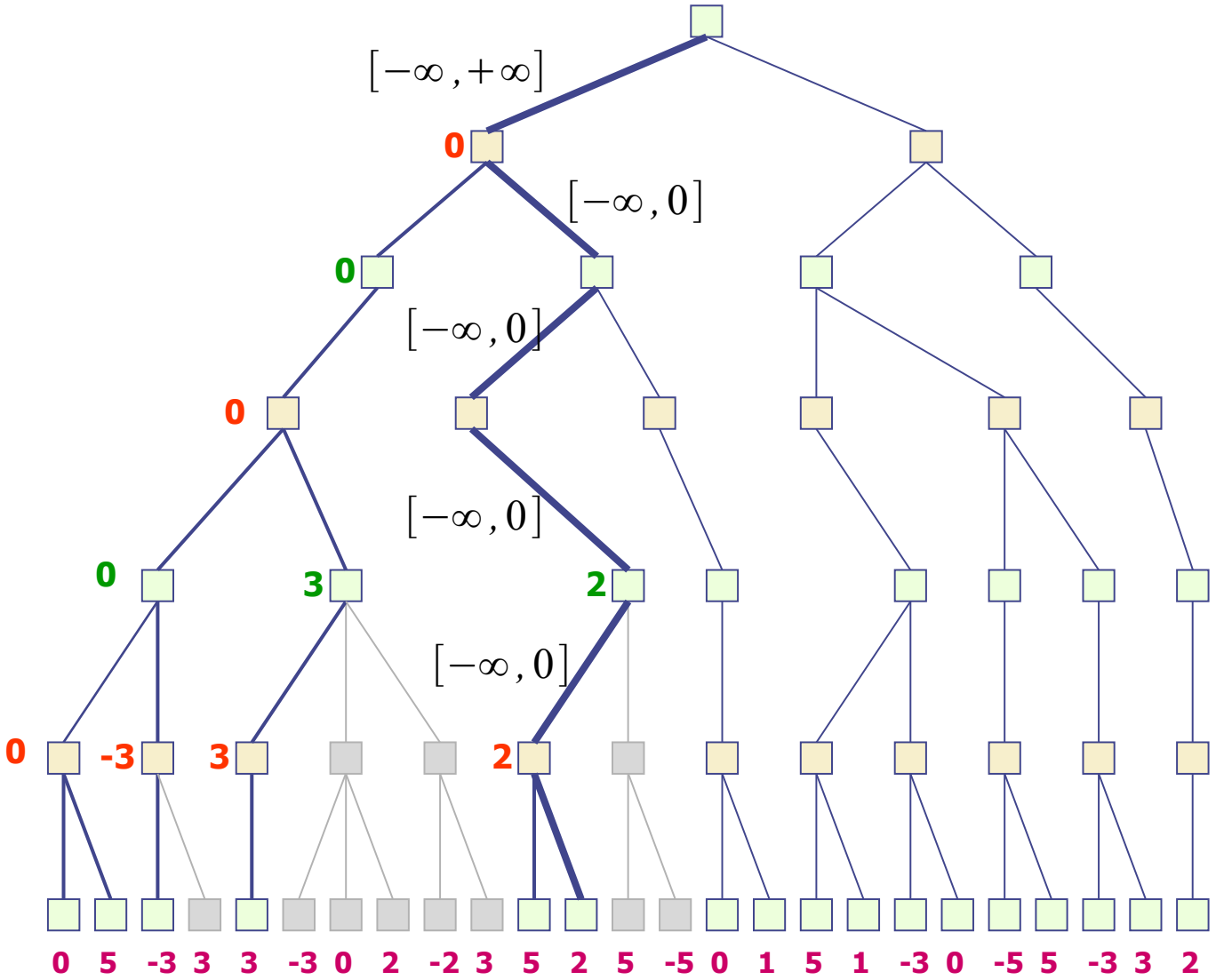
Alpha-Beta Example



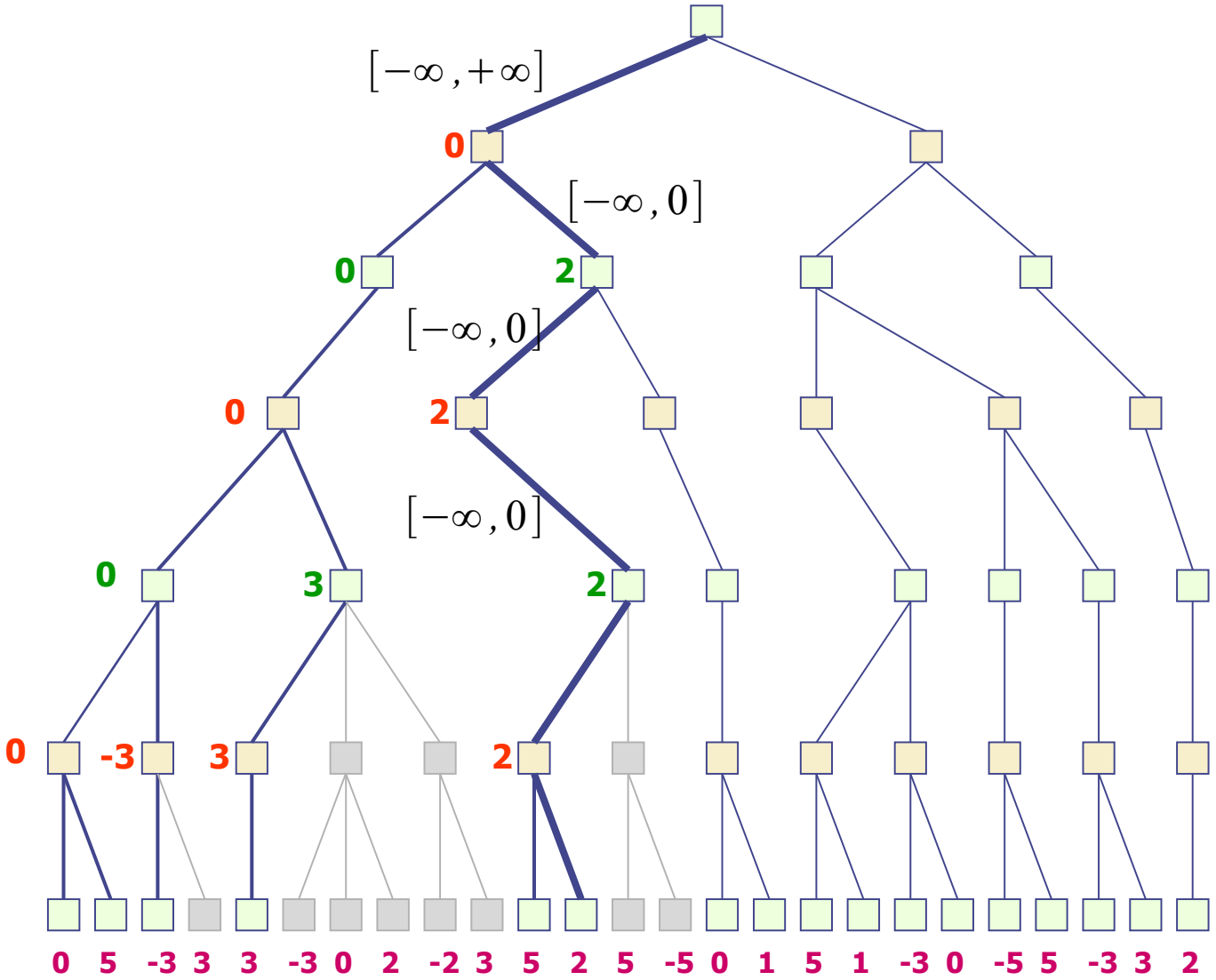
Alpha-Beta Example



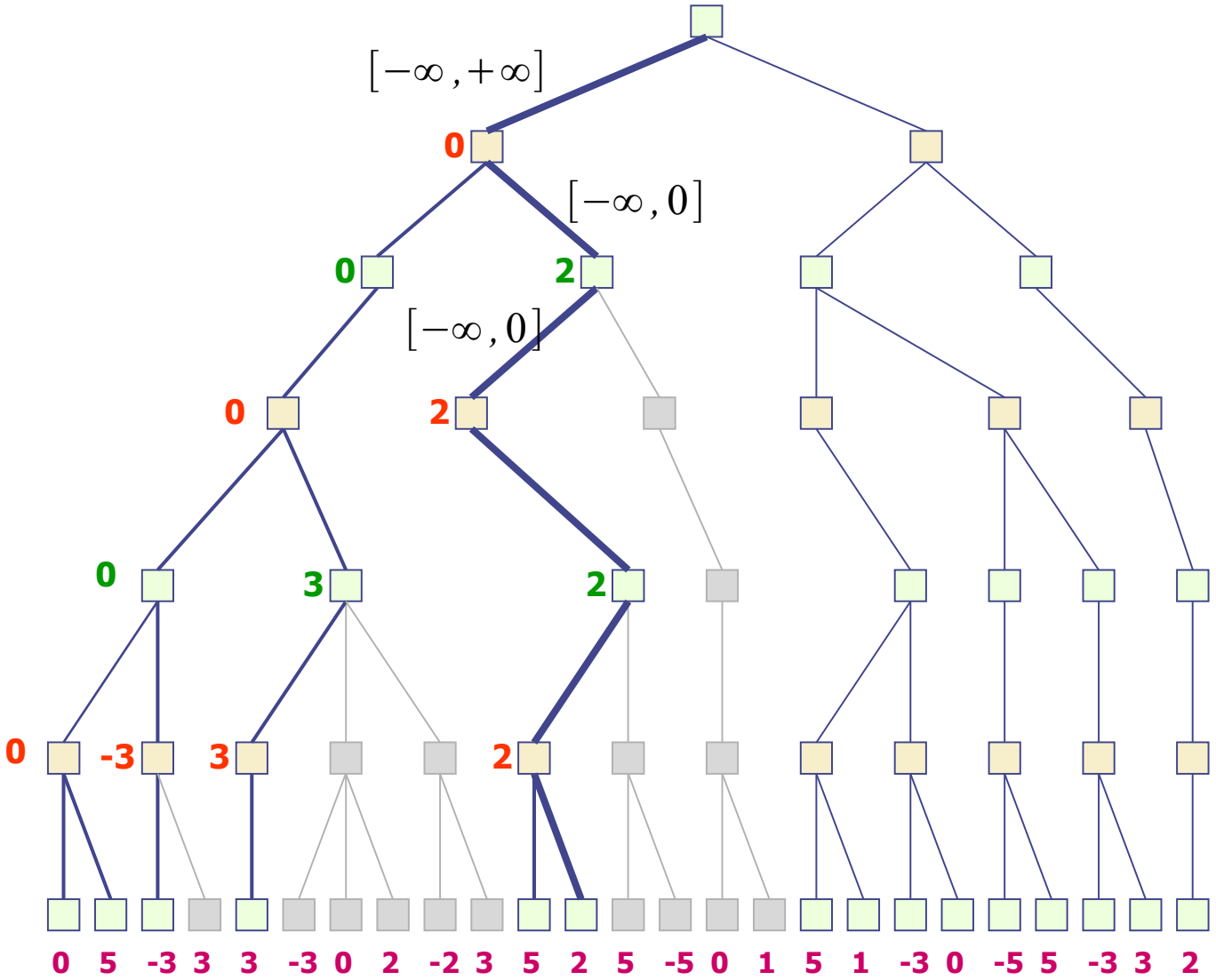
Alpha-Beta Example



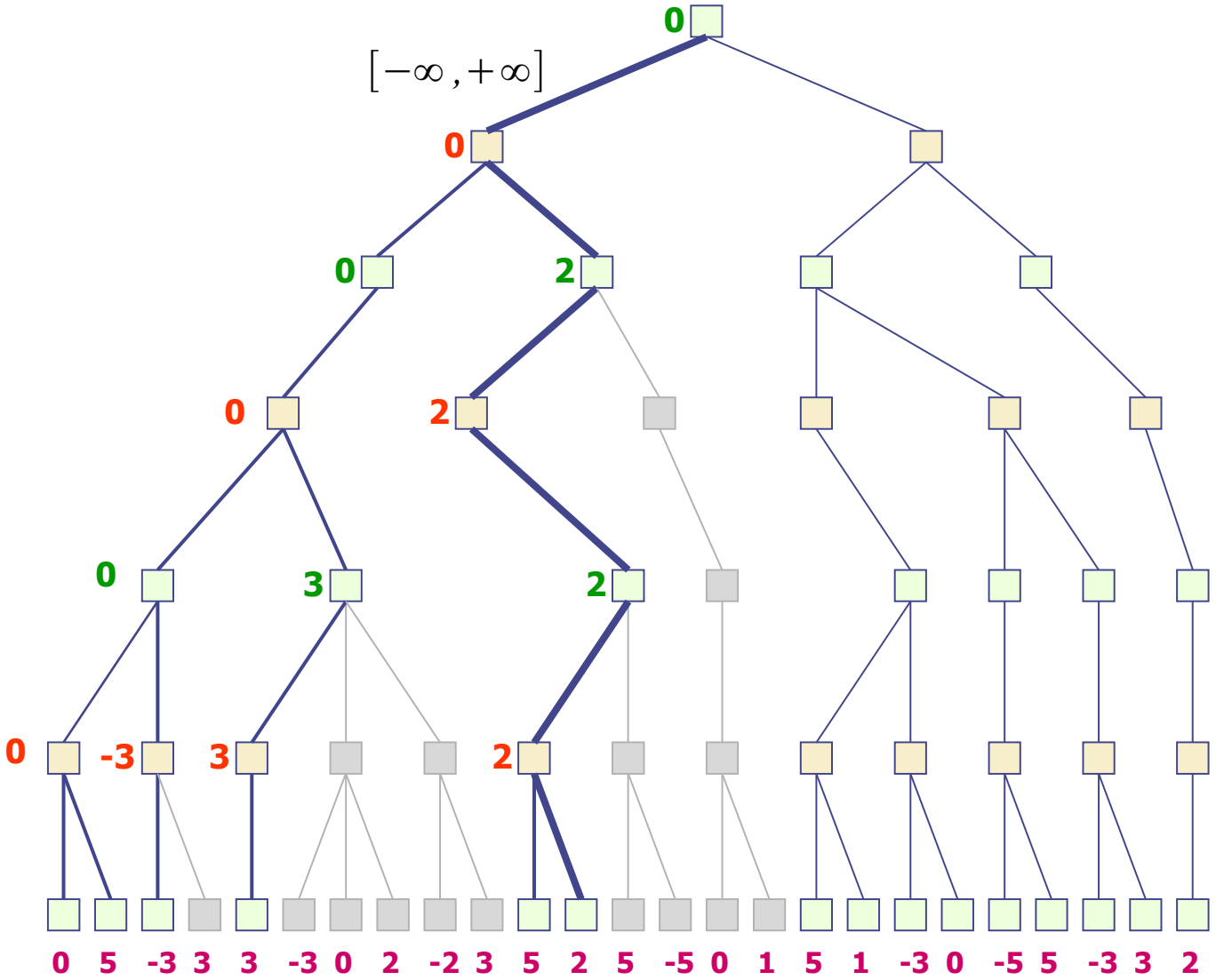
Alpha-Beta Example



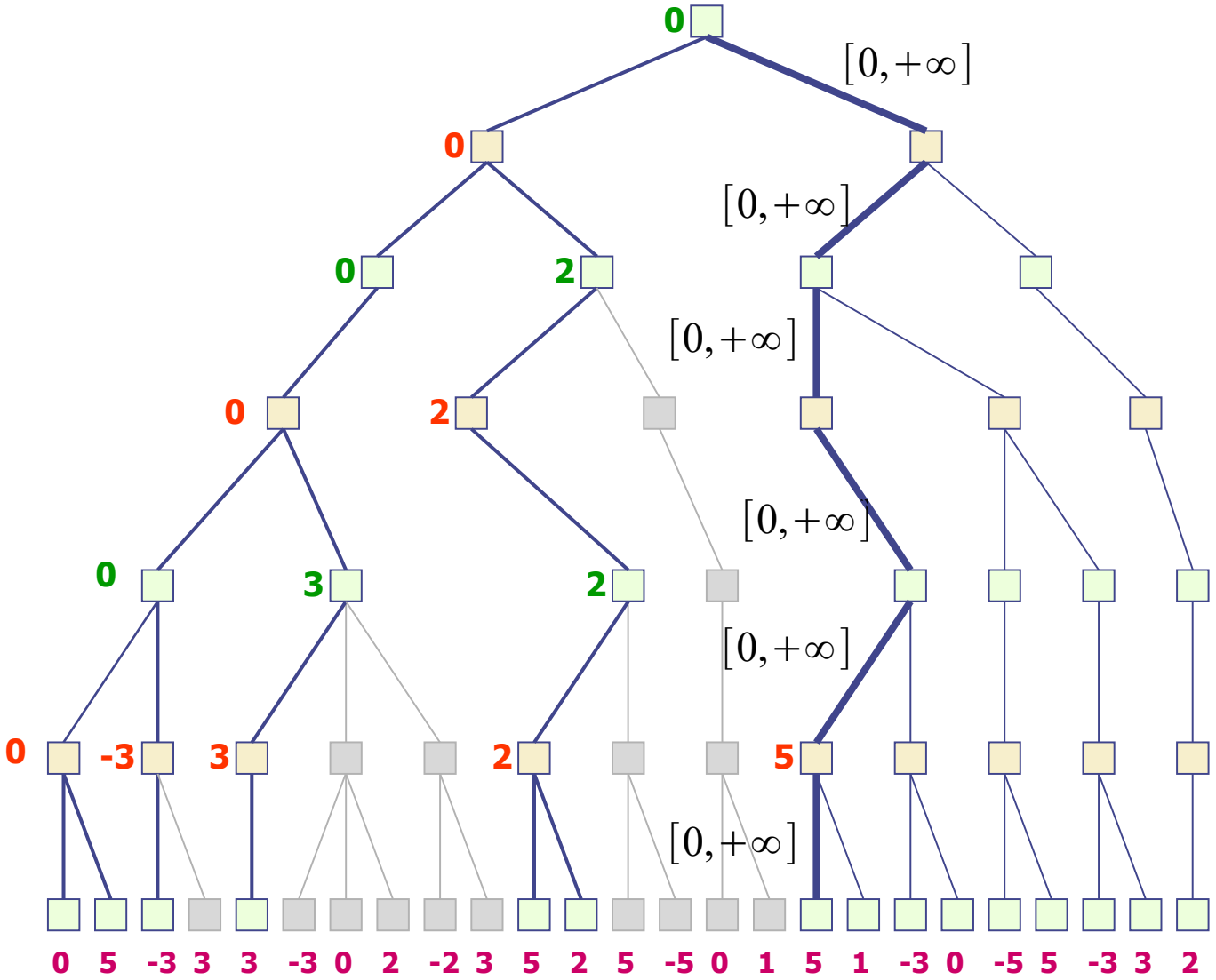
Alpha-Beta Example



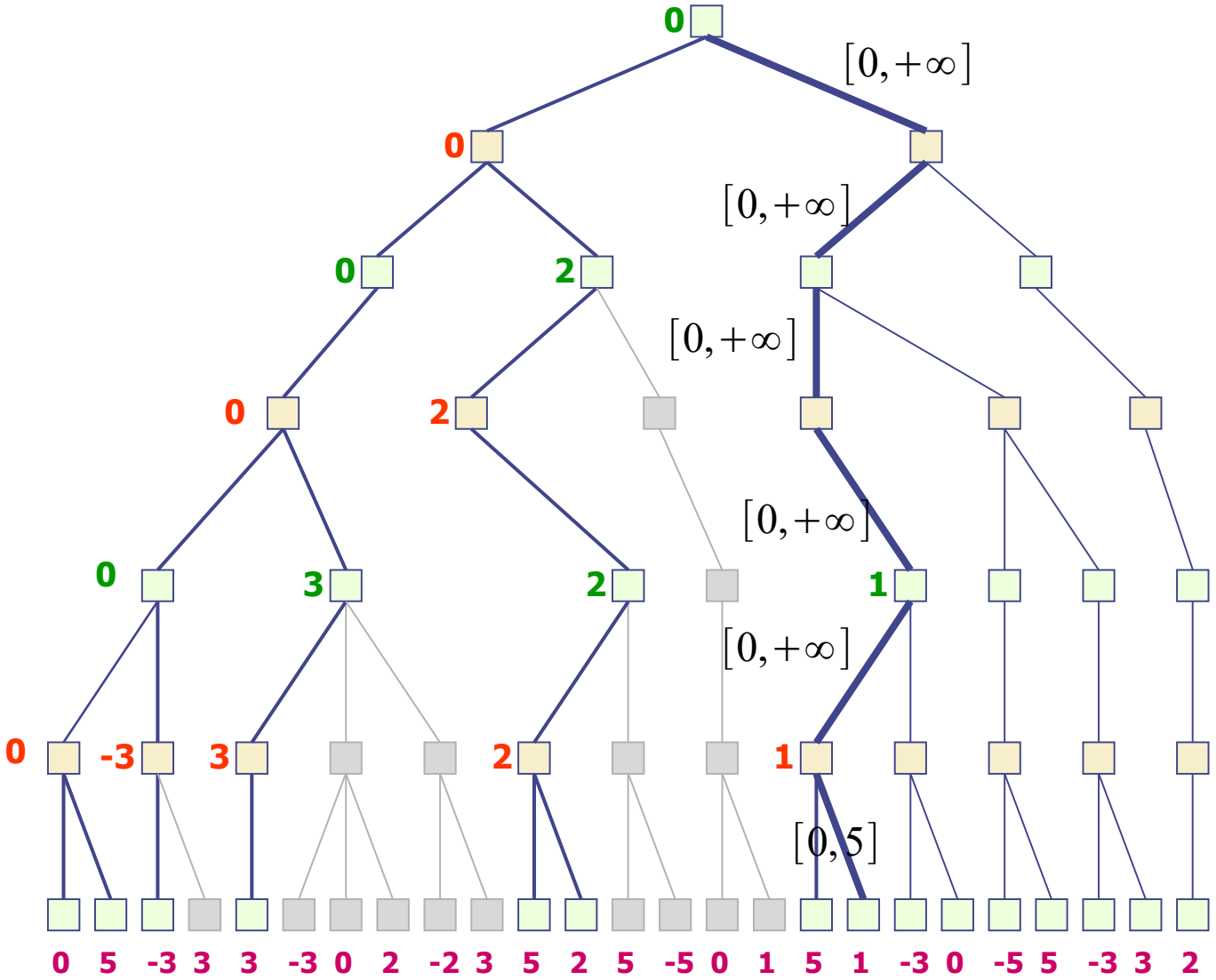
Alpha-Beta Example



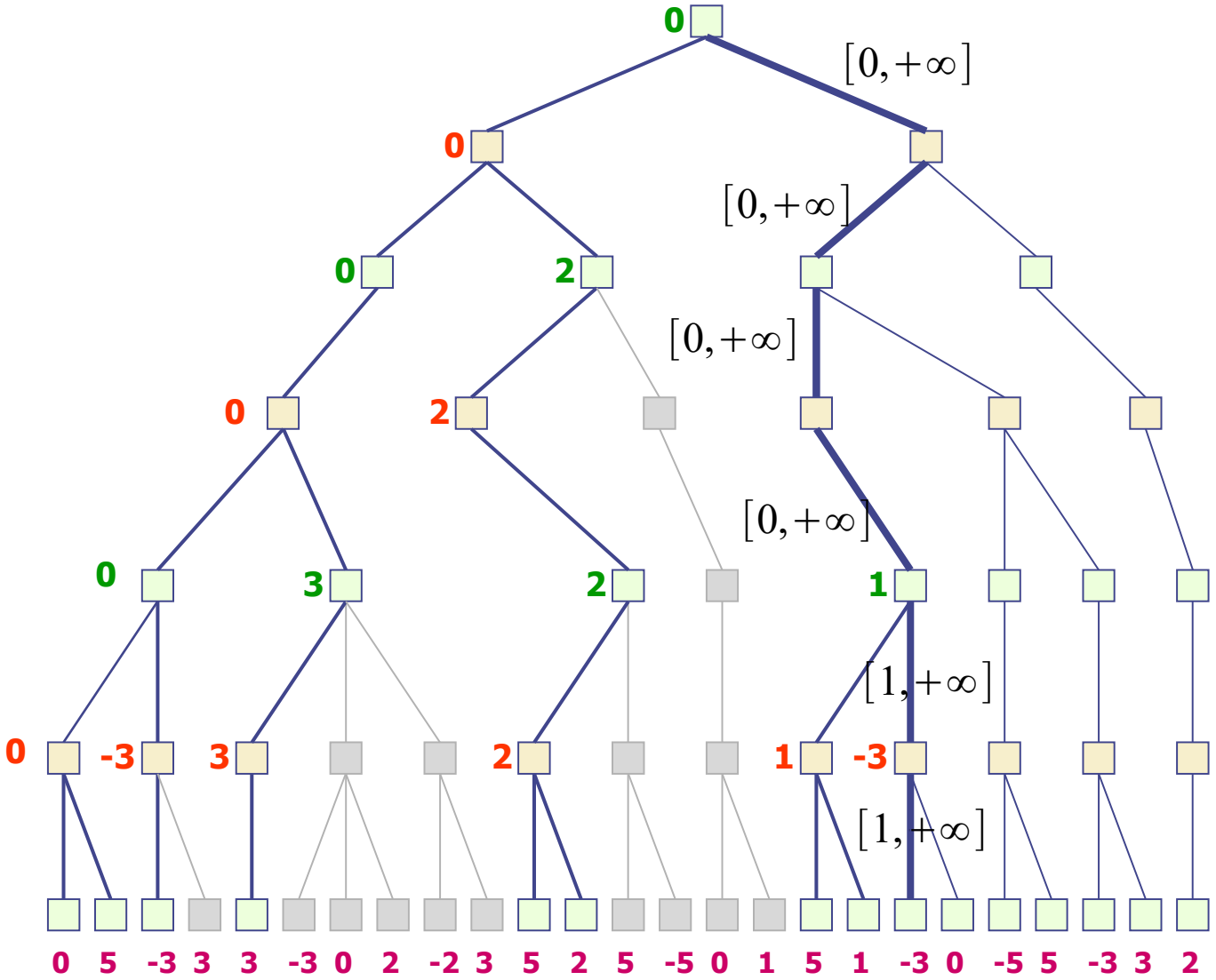
Alpha-Beta Example



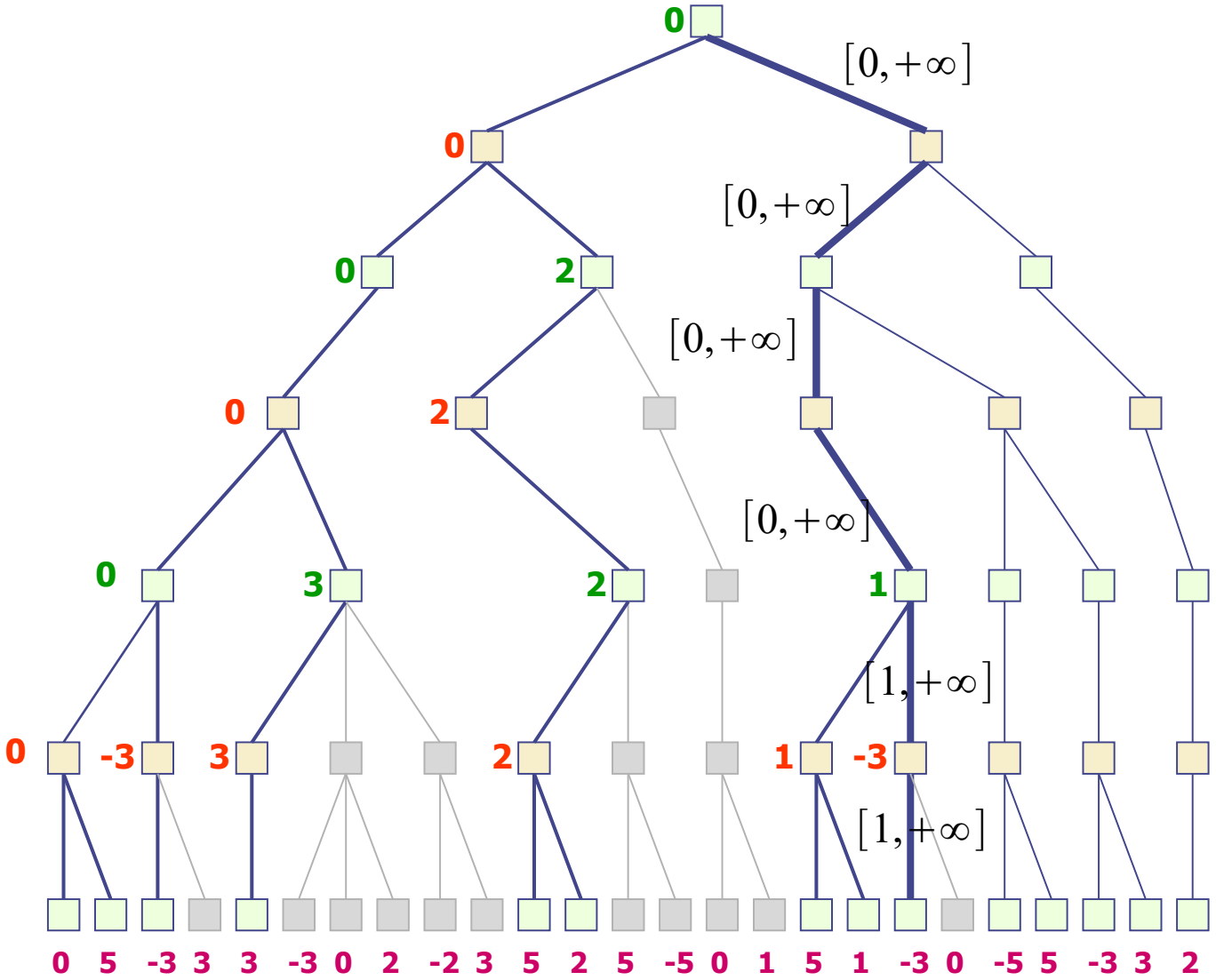
Alpha-Beta Example



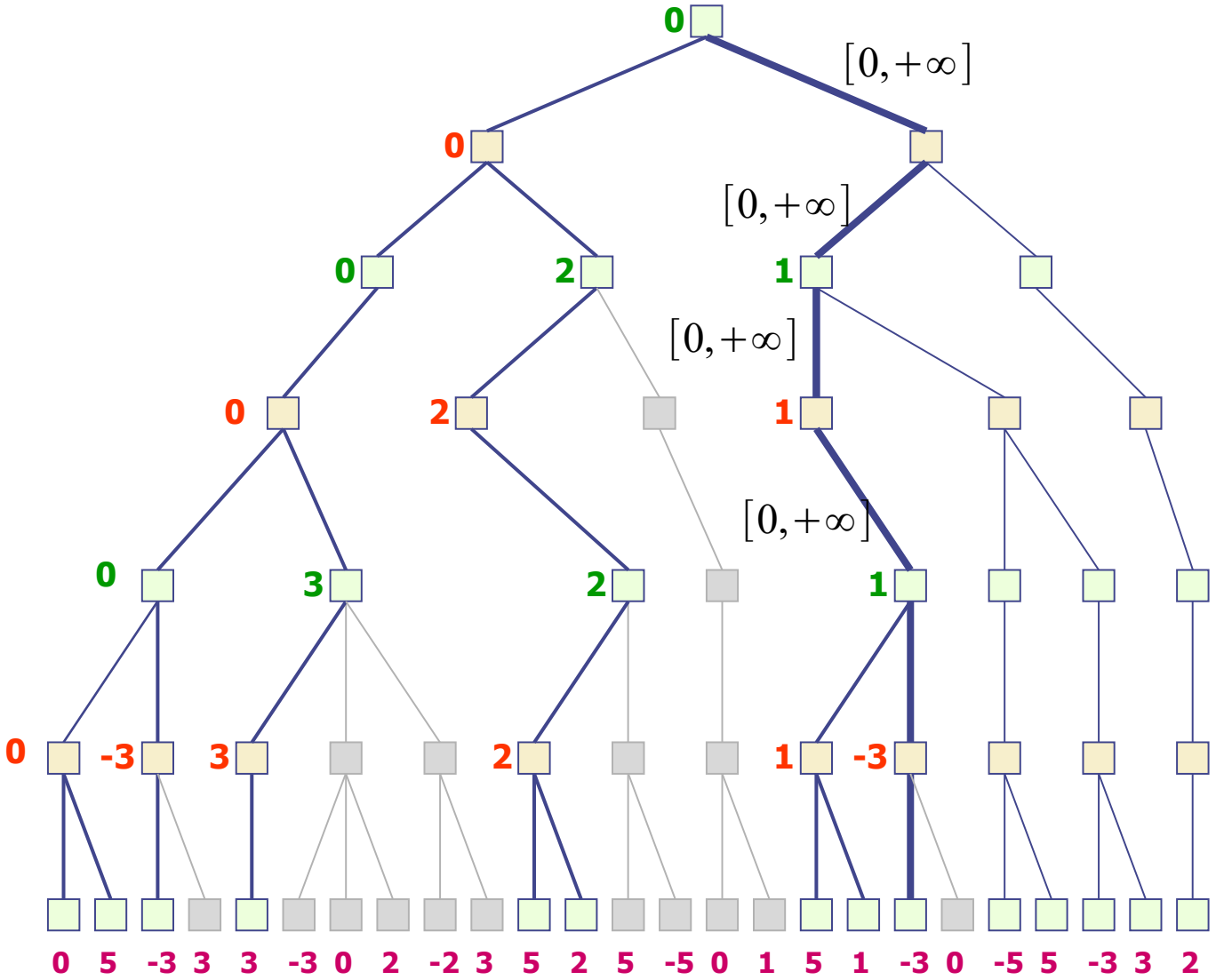
Alpha-Beta Example



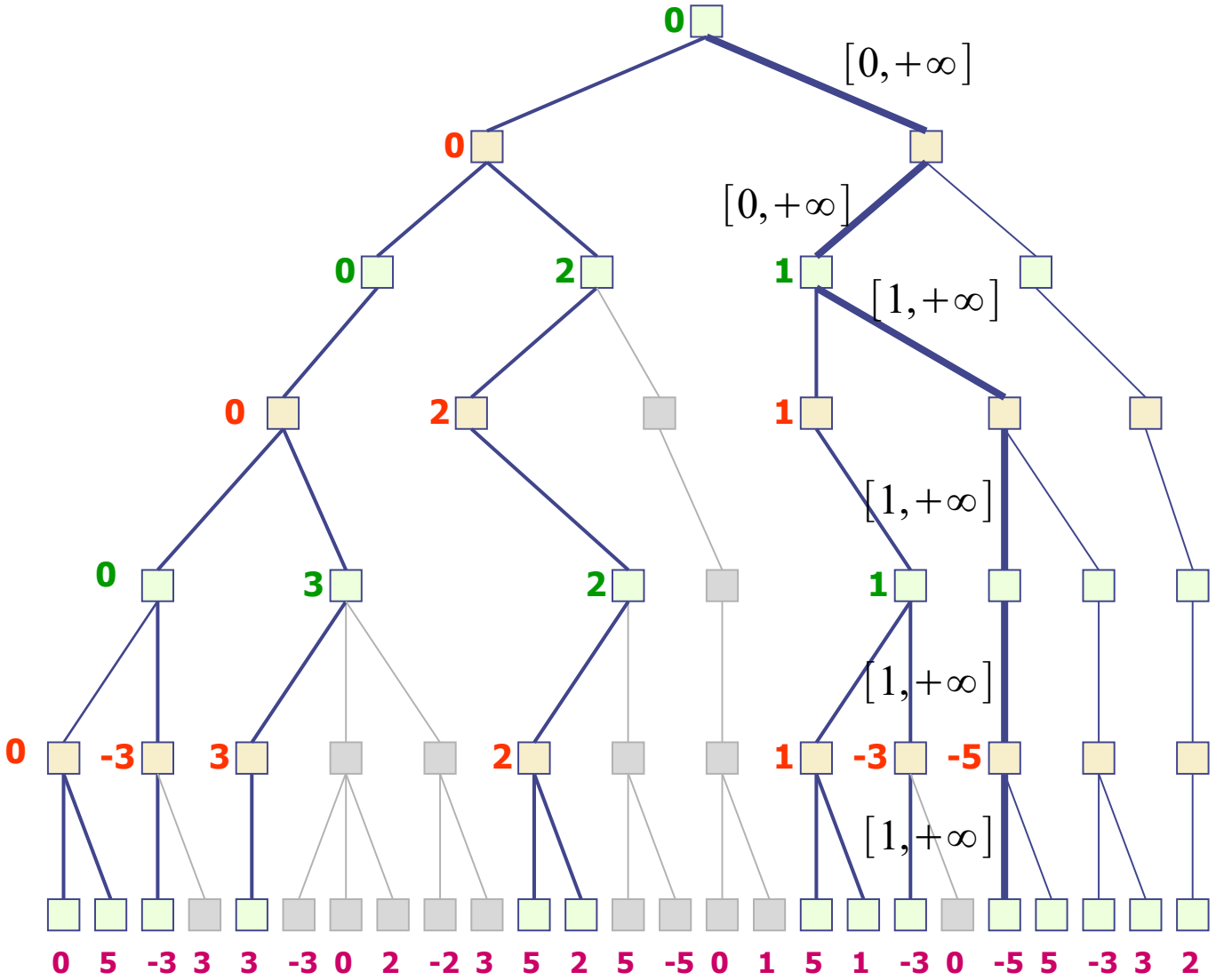
Alpha-Beta Example



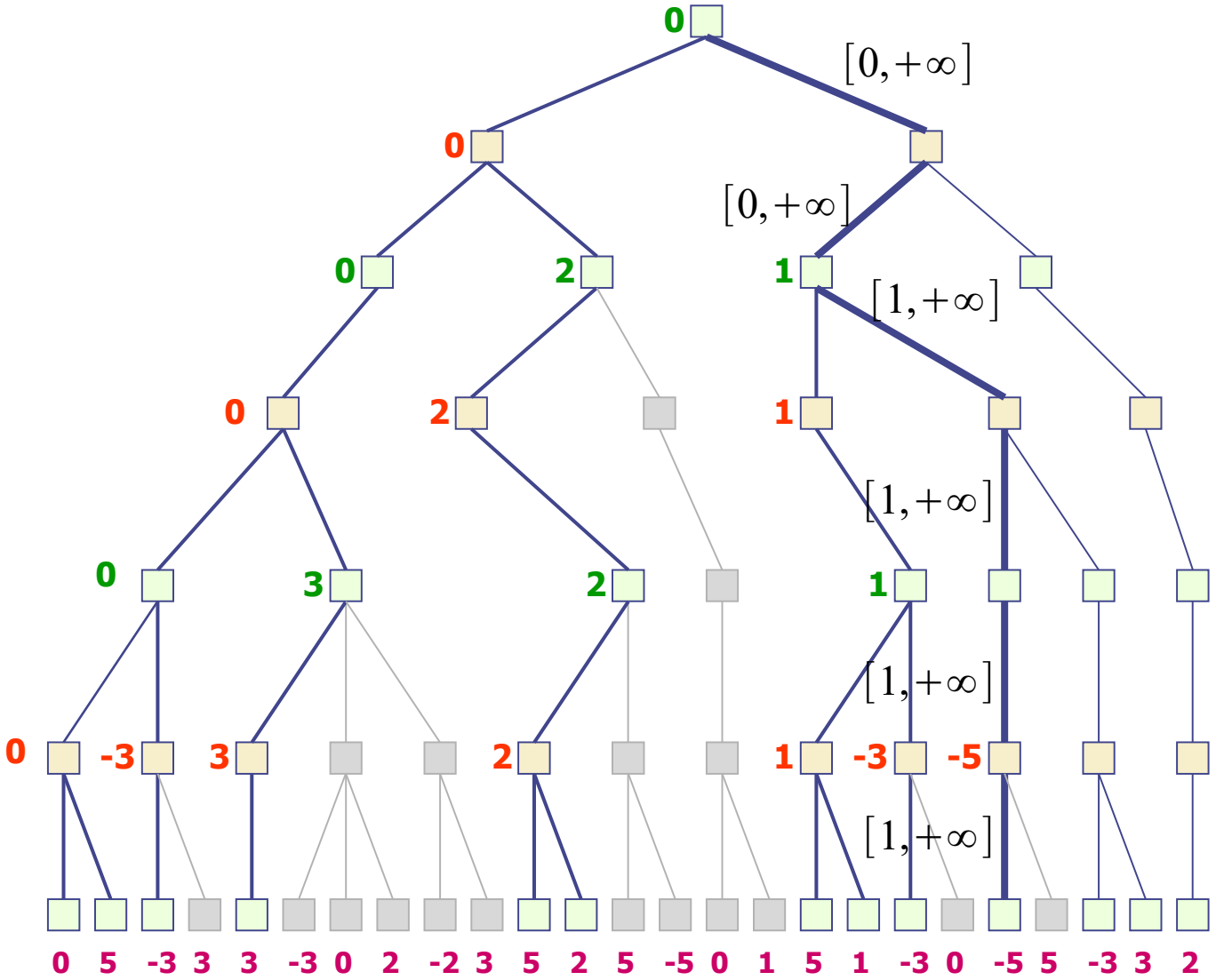
Alpha-Beta Example



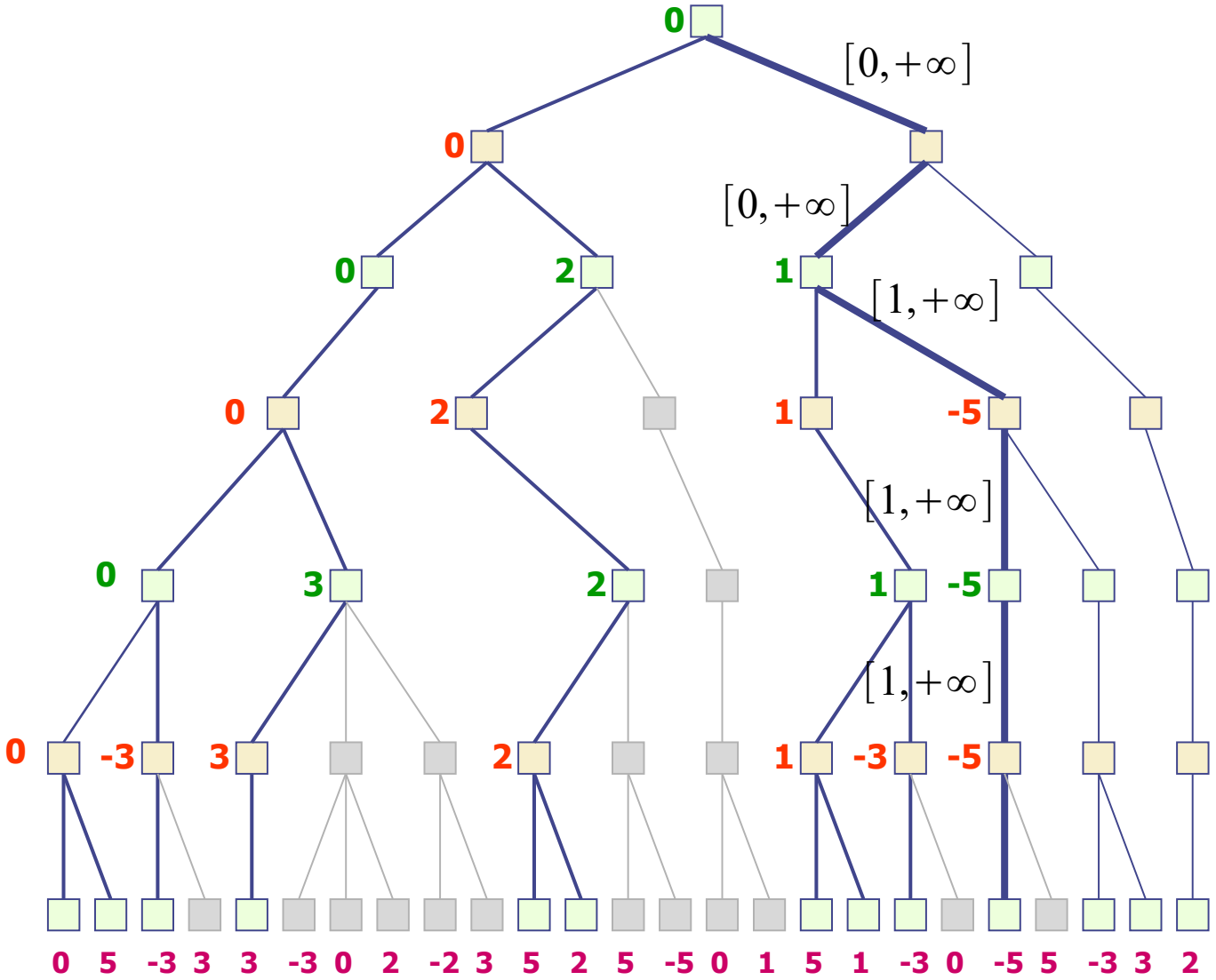
Alpha-Beta Example



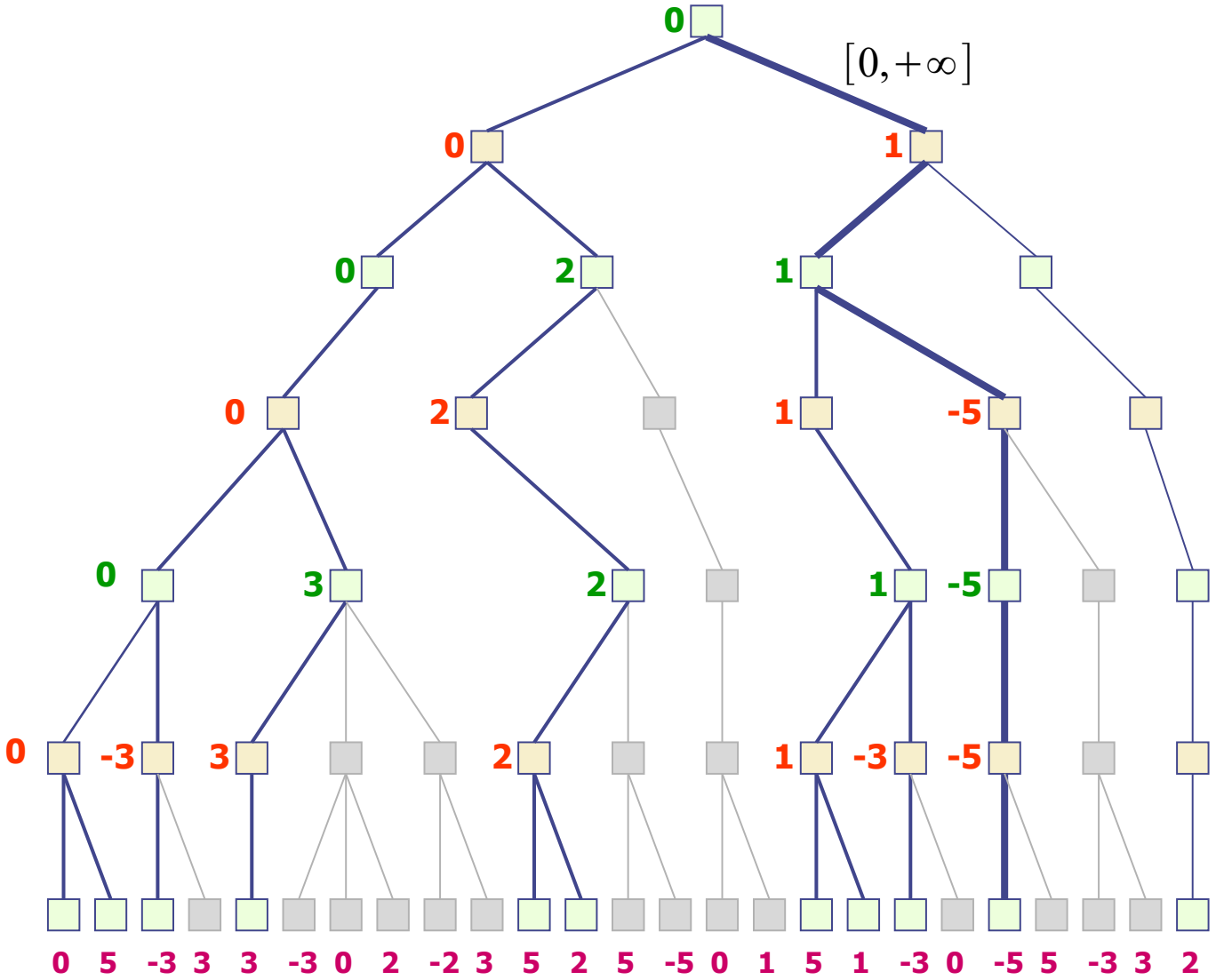
Alpha-Beta Example



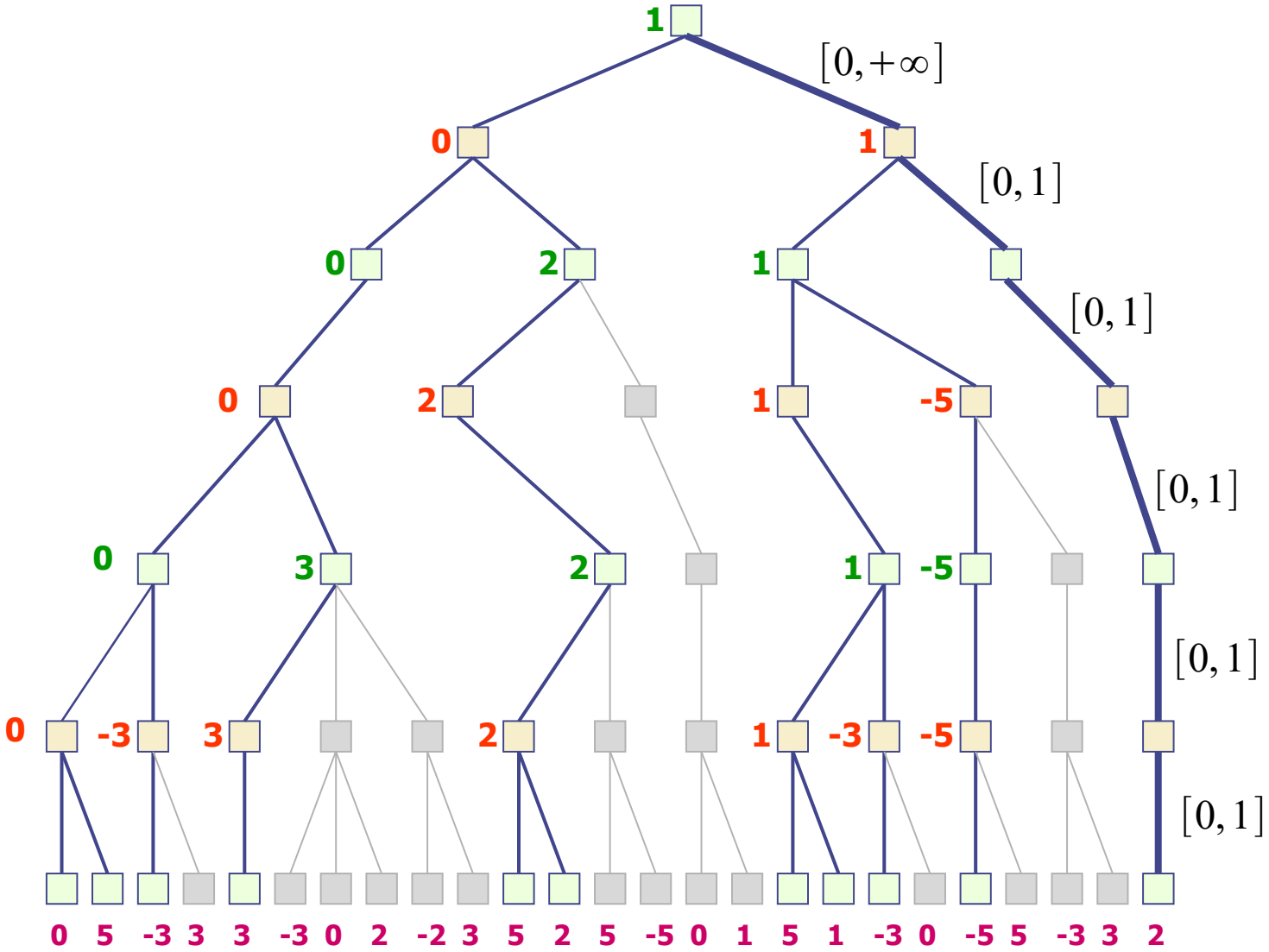
Alpha-Beta Example



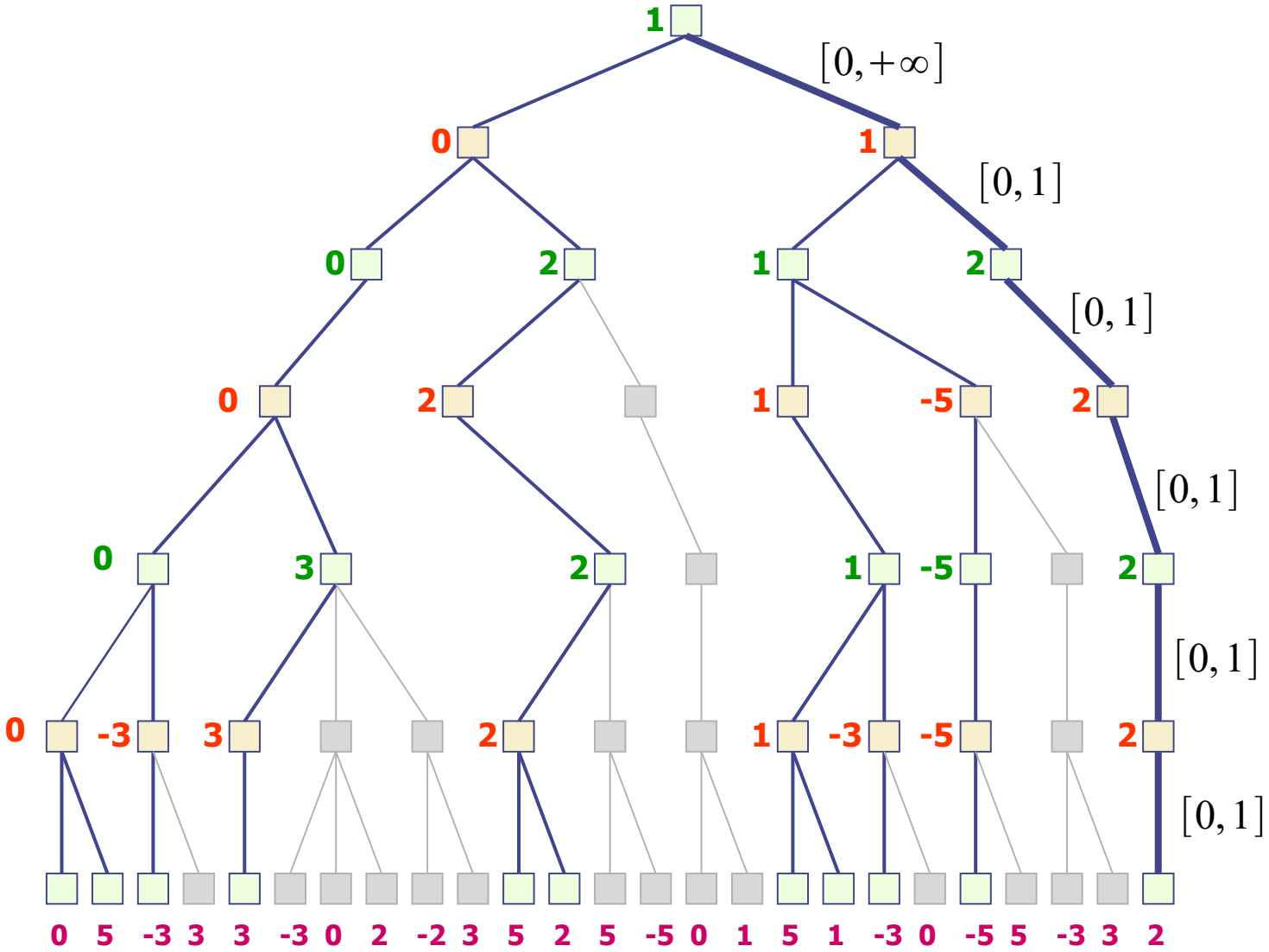
Alpha-Beta Example



Alpha-Beta Example

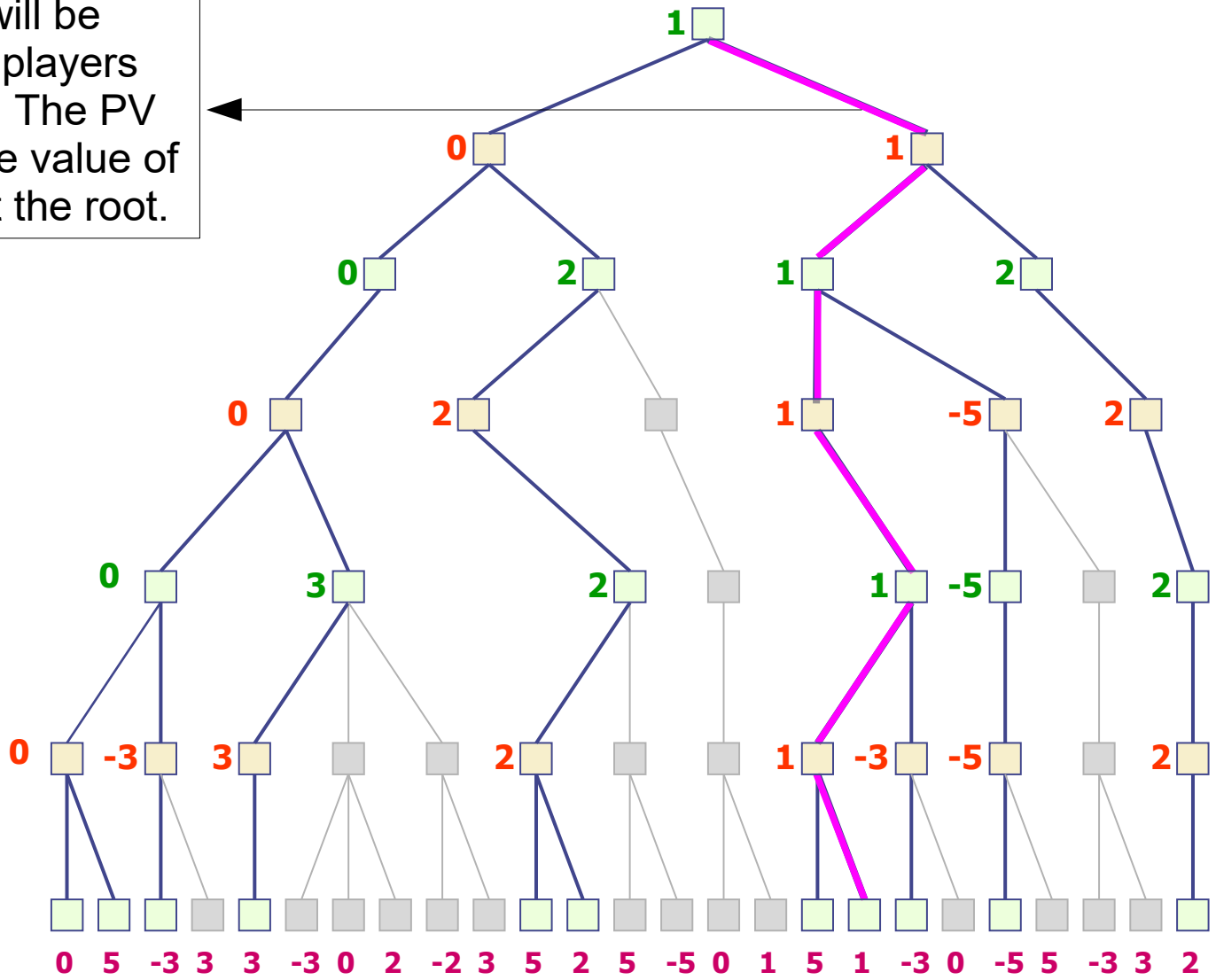


Alpha-Beta Example



Alpha-Beta Example

Principal Variation
The line that will be played if both players play optimally. The PV determines the value of the position at the root.



Properties of Alpha-Beta Pruning

- Pruning does not affect final results
- Entire subtrees can be pruned.
- Effectiveness depends on ordering of branches
  - Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity is $O(b^{m/2})$
  - this corresponds to a branching factor of $\sqrt{b}$

  → Alpha-beta pruning can look twice as deep as minimax in the same amount of time
- However, perfect ordering not possible
  - perfect ordering implies perfect play w/o search
  - random orders have a complexity of $O(b^{3m/4})$
  - crude move orders are often possible and get you within a constant factor of $O(b^{m/2})$
    - e.g., in chess: captures and pawn promotions first, forward moves before backward moves

More Information

- Animated explanations and examples of Alpha-Beta at work (in German)
  - http://www-i1.informatik.rwth-aachen.de/~algorithmus/algo19.php