# Adversarial search and game playing

**Adversarial search is search when there is an "enemy" or "opponent" changing the state of the problem every step in a direction you do not want**

## Announcements

- A0 due Friday
  - Remember: automatic 48 hour extension with 10% grade penalty
- A1 coming next week

Puzzles and games have long been considered a challenge for human intelligence:

- Chess in Persia and India ~4000 years ago
- Checkers in 3600-year-old Egyptian paintings
- Go in China over 3000 years ago

# Computer chess: beginnings

Alex Bernstein, 1957: First computer chess program on an IBM 704 (vacuum tubes, ~12k FLOPS)

1966 Hubert Dreyfus writes
"computers cannot play chess"…

Seymour Papert arranged for a match between Dreyfus and "Mac Hack".

Dreyfus lost (1967)

In 1968, David Levy, International Master in Chess, had a friendly game against John McCarthy, which he won.

McCarthy then bet that within 10 years, a computer could beat Levy. They had a bet of 500 pounds

Levy won the bet in 1978. The first computer program that beat him was “Deep Thought”, in 1989.

# Game Playing vs. the "Real World"

- Games are compact settings that mimic the uncertainty of interacting with the natural world

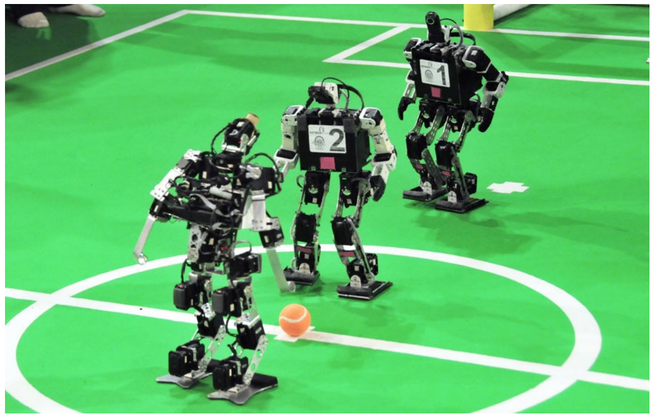
- Games are (kind of) like the real world:
  - I have a goal (to win) that I want to work/plan towards
  - I have to decide among many possible actions
  - Other agents can try to keep me from achieving my goal

- Game are much "simpler" than the real world:
  - **Chess:** I need to know the rules of chess and the state of the chess board
  - **Walking across campus to get to class:** So many things to consider…

## Specific Setting

This lecture considers:

- Two-player
- turn-taking
- fully observable
- zero-sum
- time-constrained games



# Specific Setting

Two-player, turn-taking, fully observable, zero-sum, time-constrained game

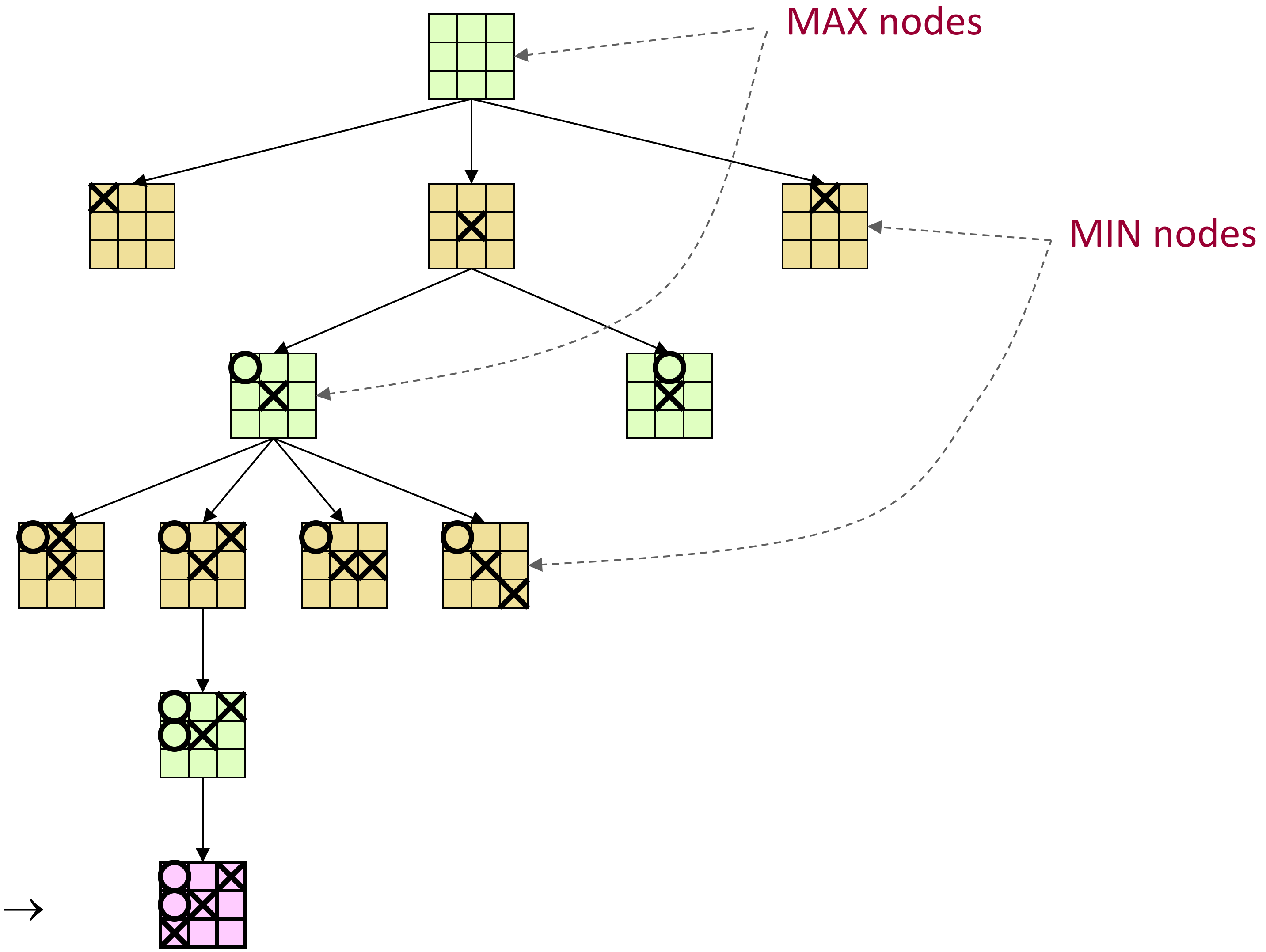
- State space?
- Initial state?
- Successor function?
- Goal state?
- Cost?

# Game Tree

MAX's play →

MIN's play →

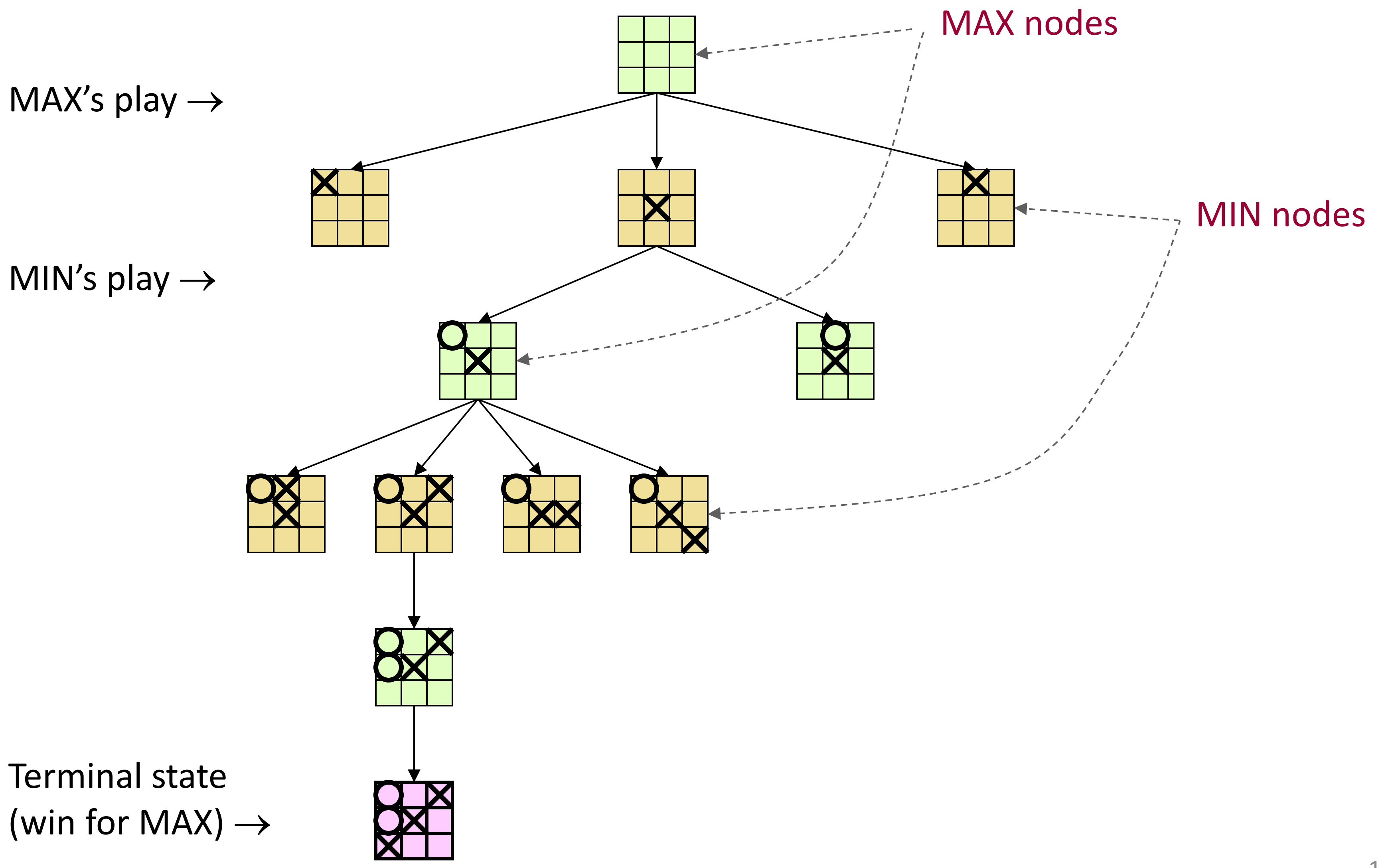
Terminal state
(win for MAX) →



## Formal Abstraction of Games

- State space
- Initial state
- Successor function: which actions can be executed in each state and the successor state for each action
- MAX’s and MIN’s actions alternate, with MAX playing first in the initial state
- Terminal test: tells if a state is terminal and, if yes, if it’s a win or a loss for MAX, or a draw

# Game Tree

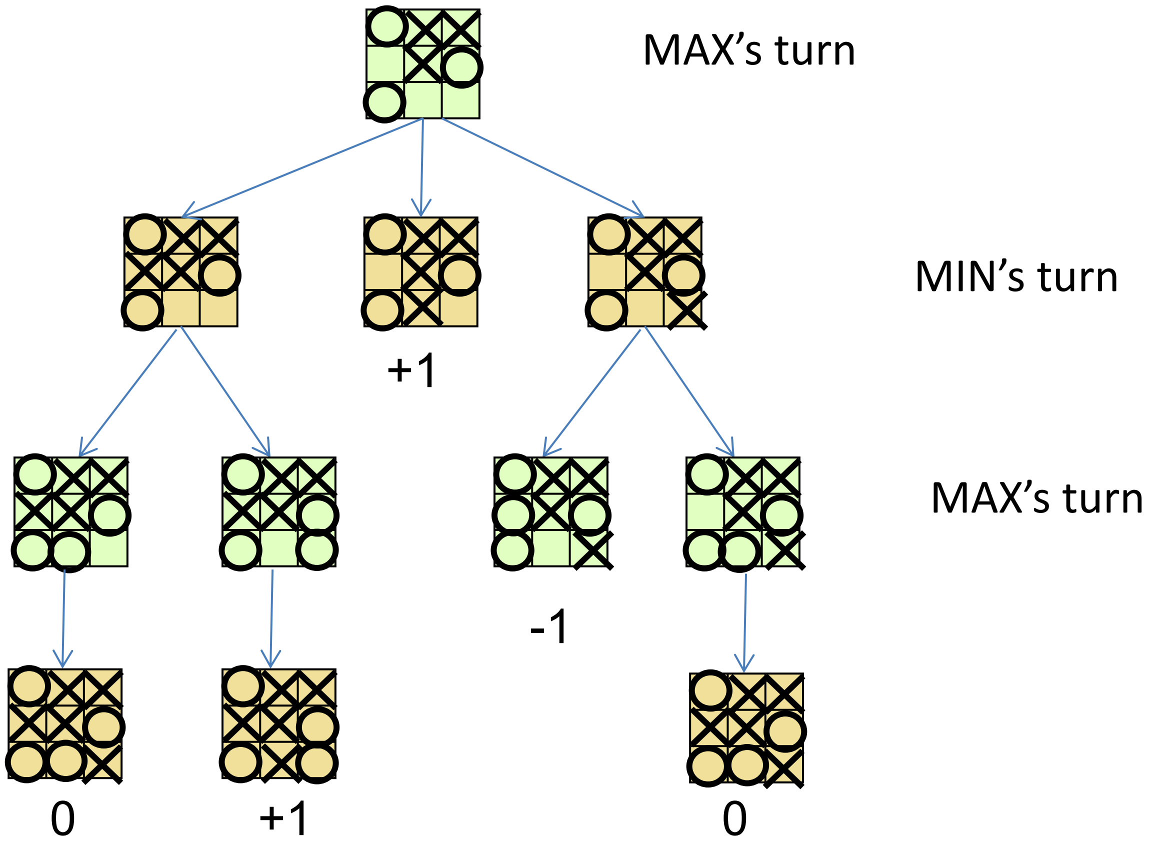

MAX nodes
MAX's play →
MIN nodes
MIN's play →
Terminal state
(win for MAX) →

## Choosing an Action: Basic Idea

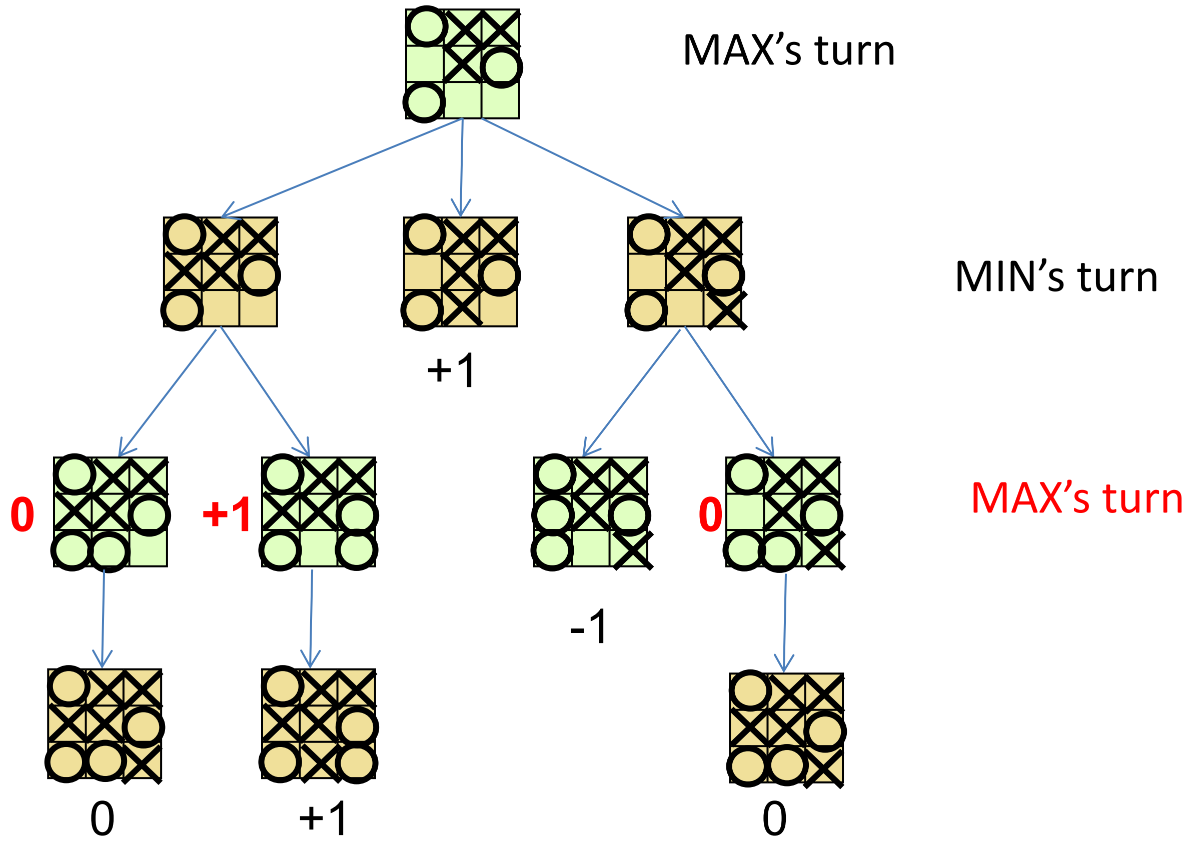
1. Using the current state as the initial state, build the entire game tree to the leaf nodes
2. Evaluate whether leaf nodes are wins (+1), losses (-1), or draws (0). Other payoffs are possible.
3. Back up the results from the leaves to the root and pick the best action assuming the “worst” from MIN (MIN plays optimally)

$\rightarrow$ **Minimax algorithm**

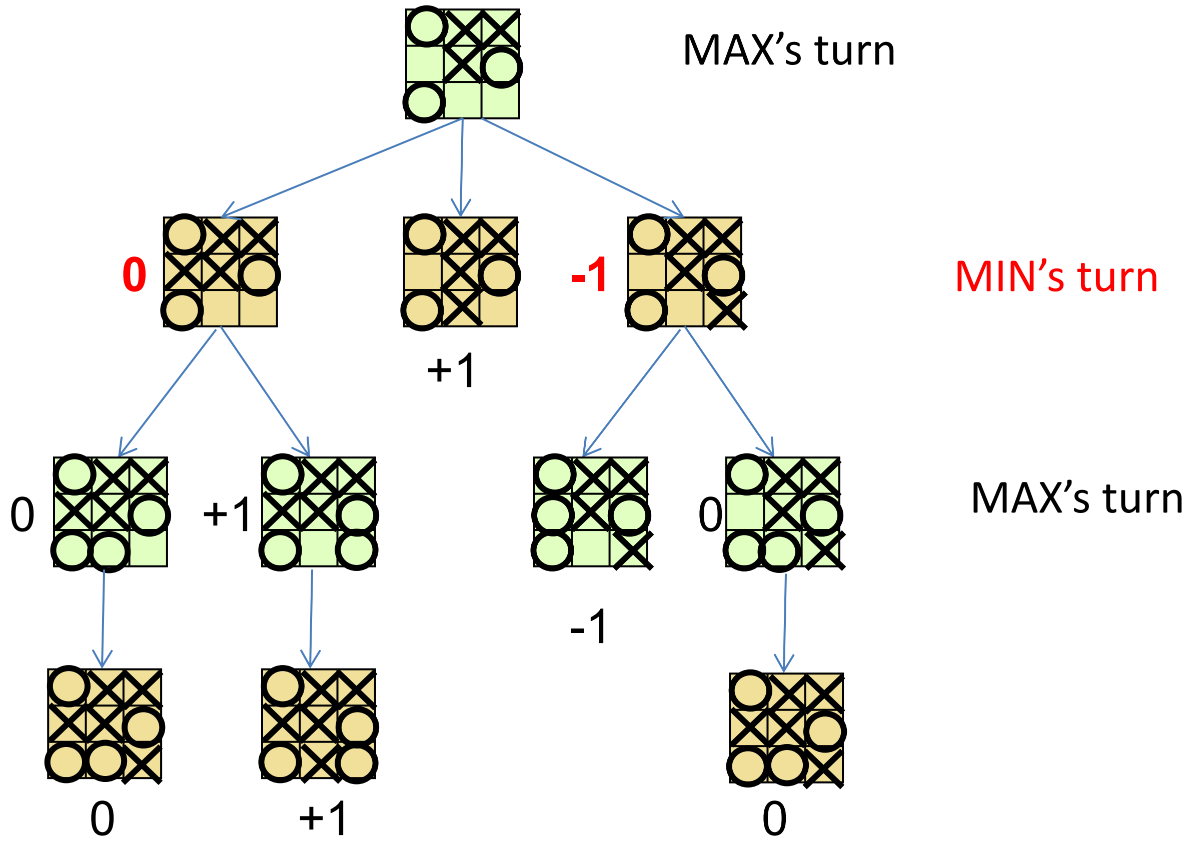
# Minimax Backup



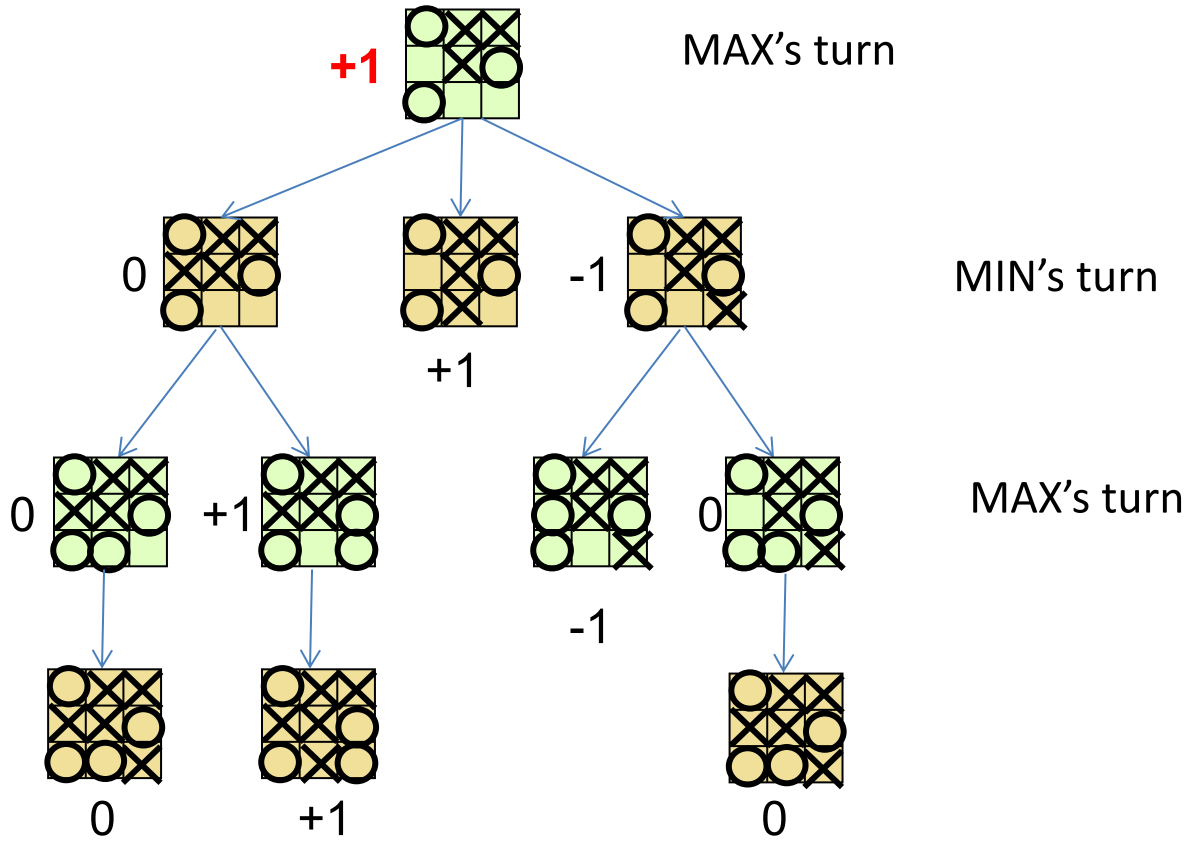
# Minimax Backup



# Minimax Backup



# Minimax Backup



## Minimax Algorithm

- Expand the game tree from current state (MAX’s turn)
- Evaluate if each leaf is a win (+1), lose (-1), draw (0)
- Backup the values from leaves to root tree as follows:
  - MAX node gets maximum of the utilities of its successors
  - MIN node gets minimum of the utilities of its successors
- Choose the move that has the largest backed-up value

# Recursive Minimax Algorithm

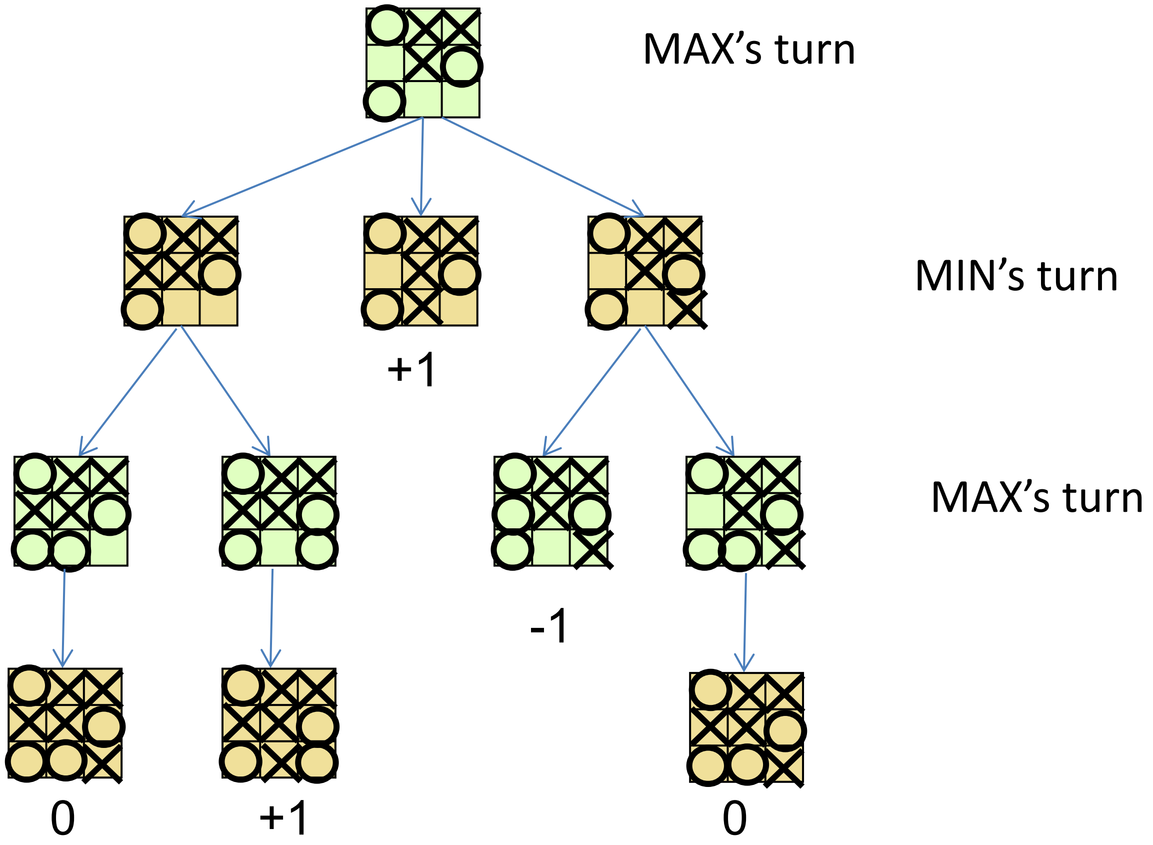
- MINIMAX-Decision(S)
  - Return action leading to state $S' \in SUCC(S)$ that maximizes MIN-Value(S')
- MAX-Value(S)
  - **If** Terminal?(S) return Result(S)
  - **Else** return $\max_{S' \in SUCC(S)}$ MIN-Value(S')
- MIN-Value(S)
  - **If** Terminal?(S) return Result(S)
  - **Else** return $\min_{S' \in SUCC(S)}$ MAX-Value(S')

- MINIMAX-Decision(S)
  - Return action leading to state $S' \in$ SUCC(S) that maximizes MIN-Value(S')
- MAX-Value(S)
  - **If** Terminal?(S) return Result(S)
  - **Else** return $\max_{S' \in SUCC(S)}$ MIN-Value(S')
- MIN-Value(S)
  - **If** Terminal?(S) return Result(S)
  - **Else** return $\min_{S' \in SUCC(S)}$ MAX-Value(S')

## Questions

- Does Minimax perform a depth- or breadth-first exploration of the game tree?
- Assuming tree has depth *d* with *b* moves at each point,
  - What's the time complexity?
  - What's the space complexity?

# Minimax Backup



## What’s the catch?

- The Minimax algorithm guarantees to make the optimal decision
- Is our AI already invincible?

In general, the branching factor and the depth of terminal states are large

Chess:

- Branching factor: ~35
- Number of total moves in a game: ~100

# Real-Time decisions

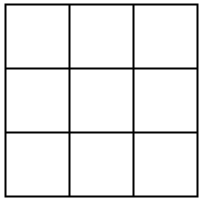
- In many games, the state space is *enormous*, and only a tiny fraction can be explored
- A heuristic:
  1. Using the current state as initial state, build game tree to **maximal depth h (the *horizon*) feasible within the time limit**
  2. Evaluate the states of the nodes at depth h
  3. Back up results from those nodes to the root and pick the best action, assuming the worst from MIN

# Evaluation Function

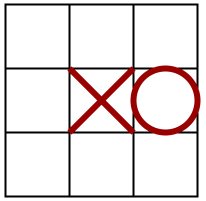
- Function e: state s $\rightarrow$ number e(s)
- e(s): heuristic estimating favorability of s for MAX
  - e(s) > 0 means that s is favorable to MAX (the larger the better)
  - e(s) < 0 means that s is favorable to MIN
  - e(s) = 0 means that s is neutral
- Other payoffs are possible, but still MAX wants to maximize and MIN to minimize

# Example: Tic-tac-Toe

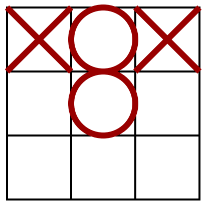
e(s) = (number of rows, columns,
and diagonals open for MAX)
– (number of rows, columns,
and diagonals open for MIN)



8–8 = 0



6–4 = 2



3–3 = 0

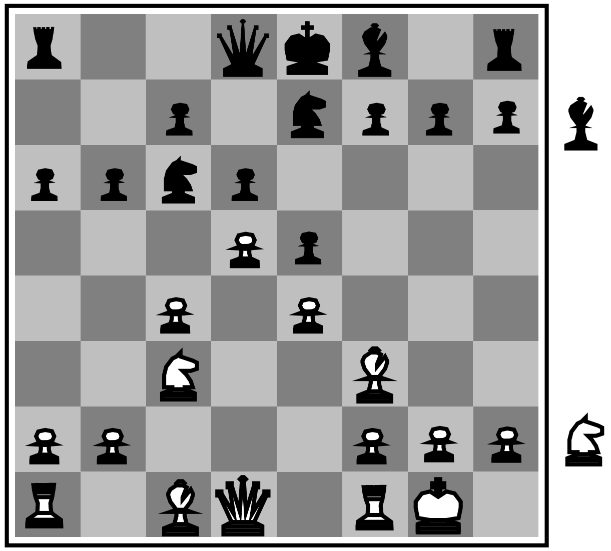
# Construction of a Static Evaluation Function

- Usually a weighted sum of “features”:

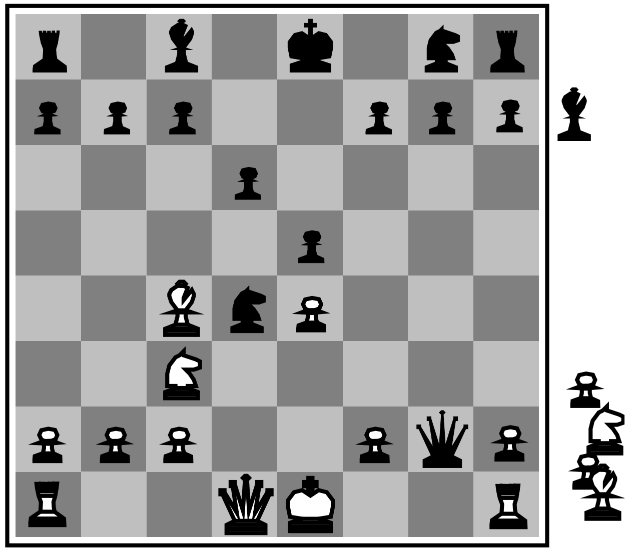
$$e(s) = \sum_{i=1}^{n} w_i f_i(s)$$

- Features may include
  - Number of pieces of each type
  - Number of possible moves
  - Number of squares controlled

# Example from chess


**Black to move**

White slightly better


**White to move**

Black winning

e(s) = 1(number of white pawns − number of black pawns) +
+ 3(number of white knights − number of black knights) +
+…

# Minimax Algorithm

- Expand the game tree from current state (MAX’s turn)

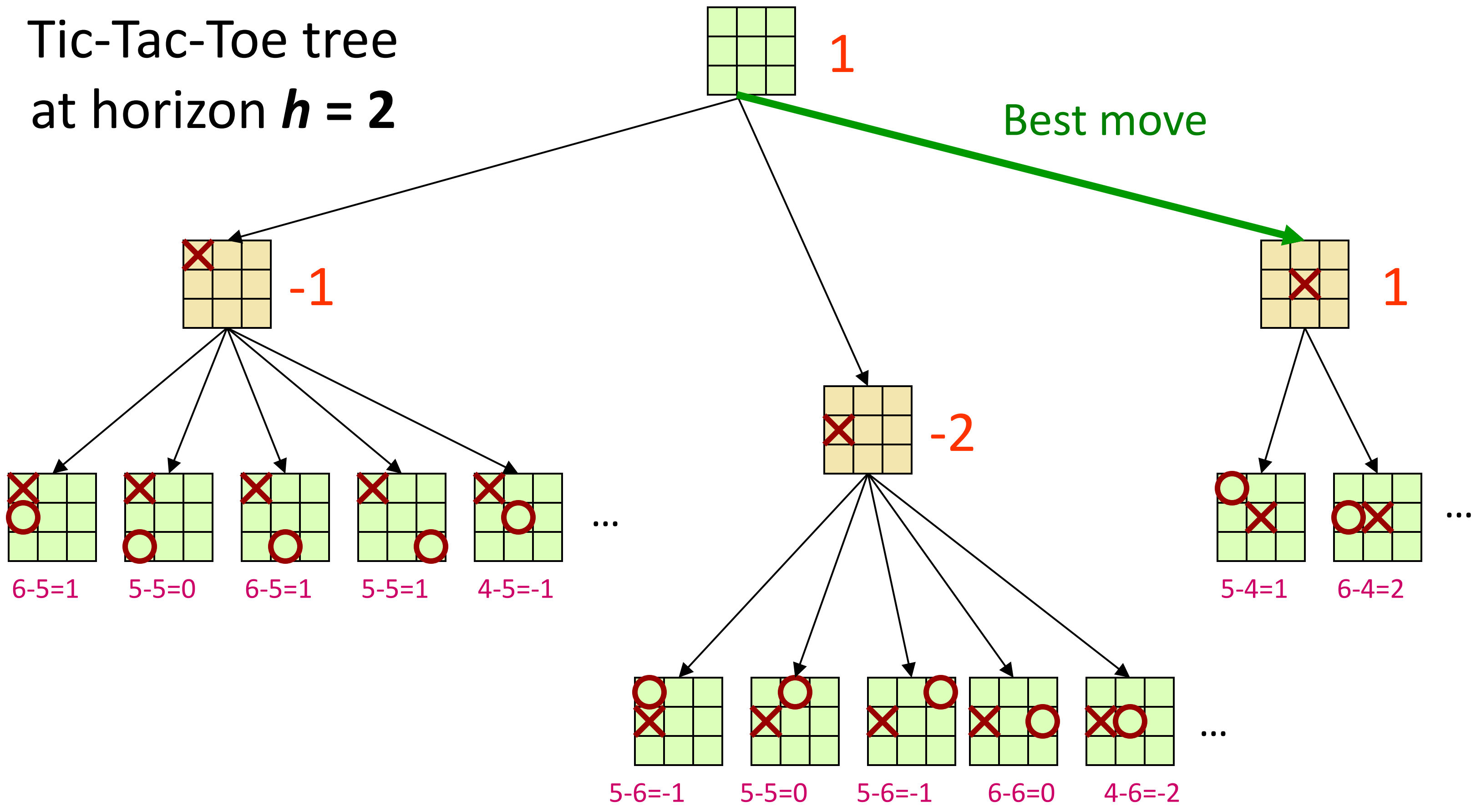
- Evaluate if each leaf is a win (+1), lose (-1), draw (0)
- Backup the values from leaves to root tree as follows:
  - MAX node gets maximum of the utilities of its successors
  - MIN node gets minimum of the utilities of its successors
- Choose the move that has the largest backed-up value

# Minimax Algorithm

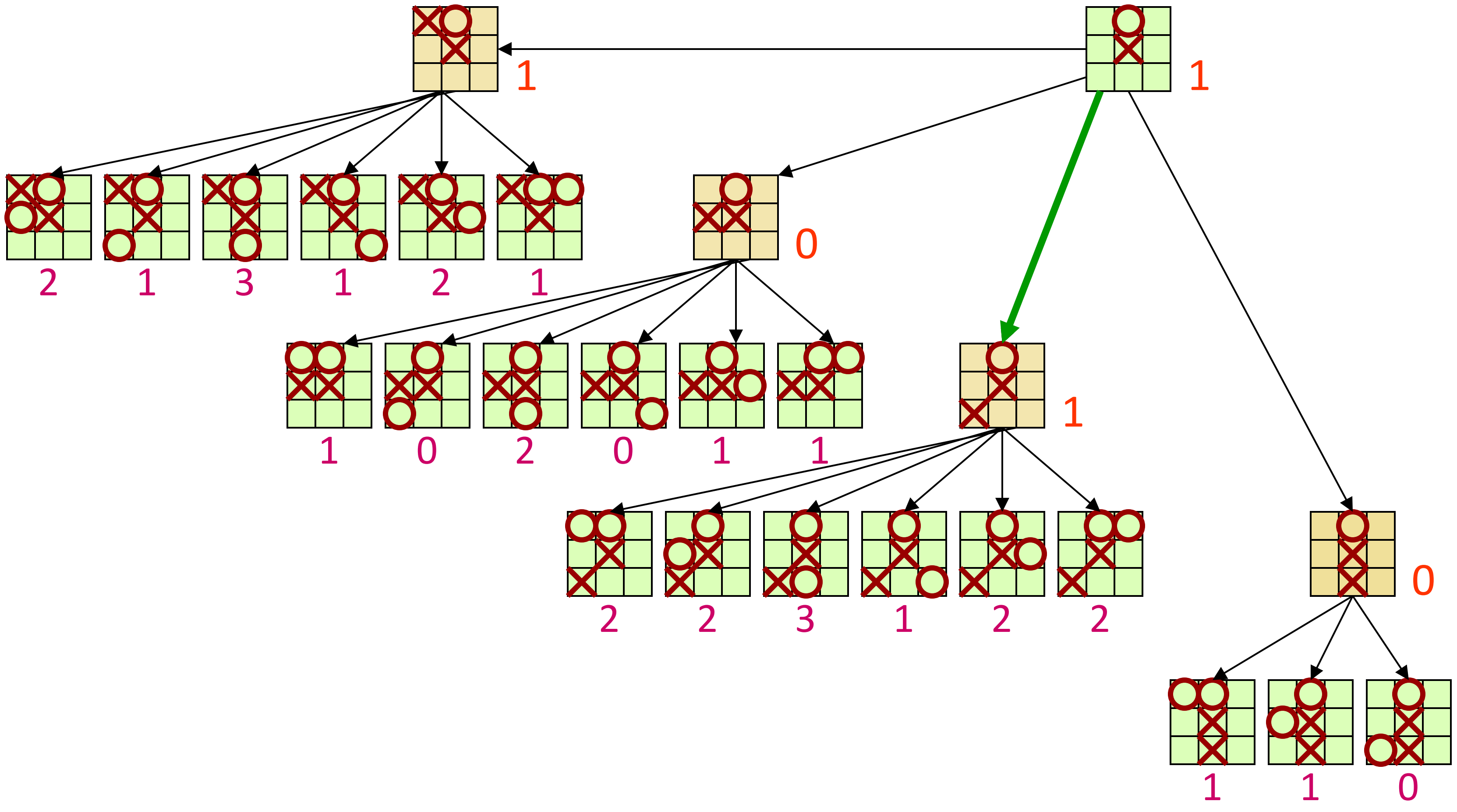
- Expand the game tree from current state (MAX's turn) **to depth h**
- Evaluate ~~if~~ **heuristic function at** each leaf ~~is a win (+1), lose (-1), draw (0)~~
- Back-up the values from leaves to root tree as follows:
  - MAX node gets maximum of the utilities of its successors
  - MIN node gets minimum of the utilities of its successors
- Choose the move that has the largest backed-up value

# Backing up Values

Tic-Tac-Toe tree
at horizon ***h* = 2**



# Continuation



## Chess Horizon

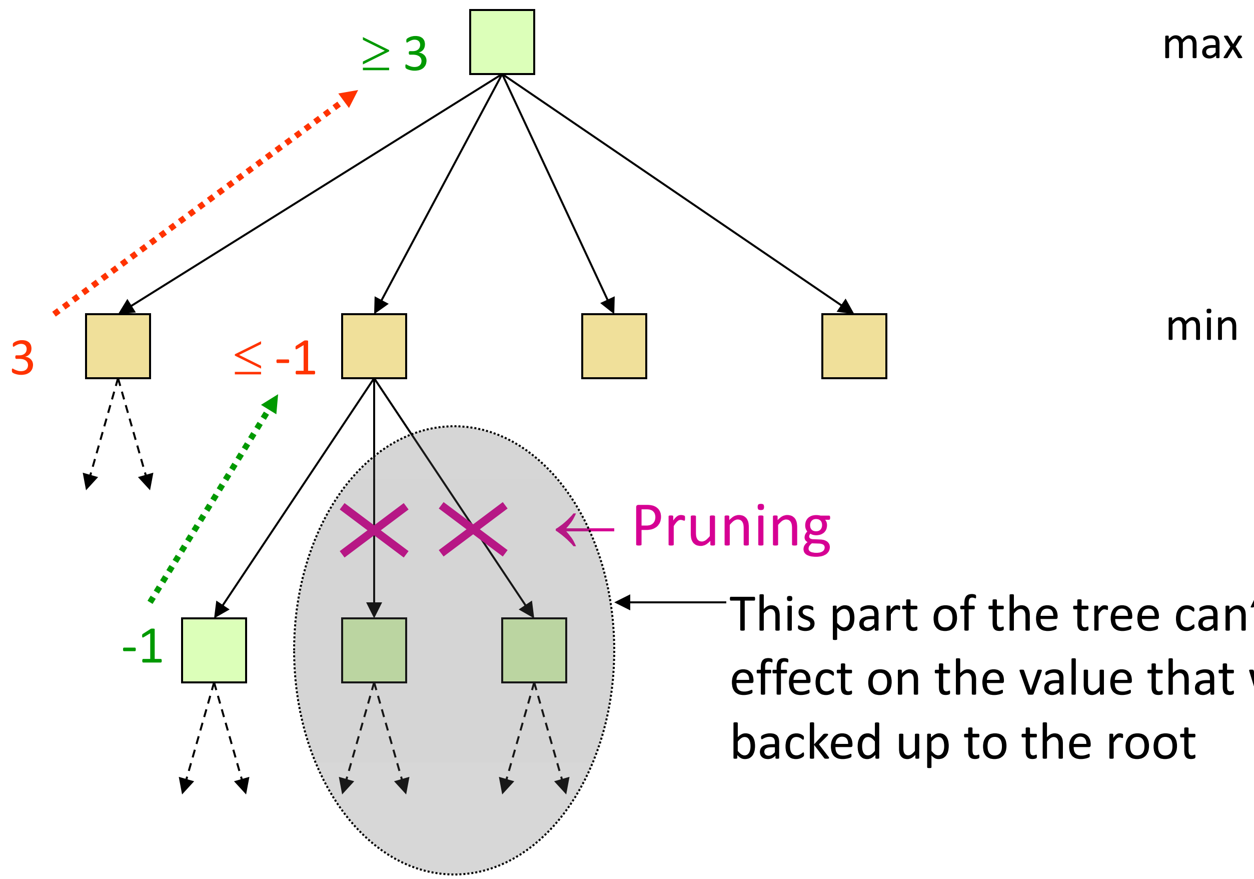
- Time complexity? $O(b^h)$
- Space complexity? $O(bh)$

- For chess, b=35:

| | h | $b^h$ |
|---|---|---|
| | 3 | 42875 |
| | 5 | $5 \times 10^7$ |
| Good $\rightarrow$ | 10 | $3 \times 10^{15}$ |
| Master $\rightarrow$ | 15 | $1 \times 10^{23}$ |

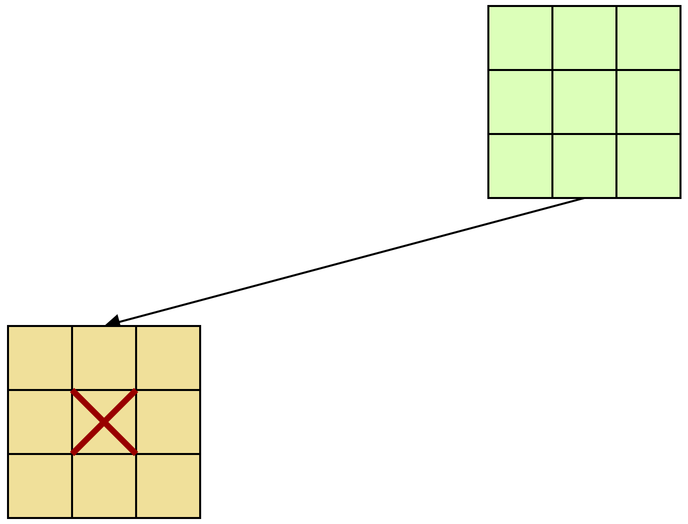
# Can we do better?

- Yes!

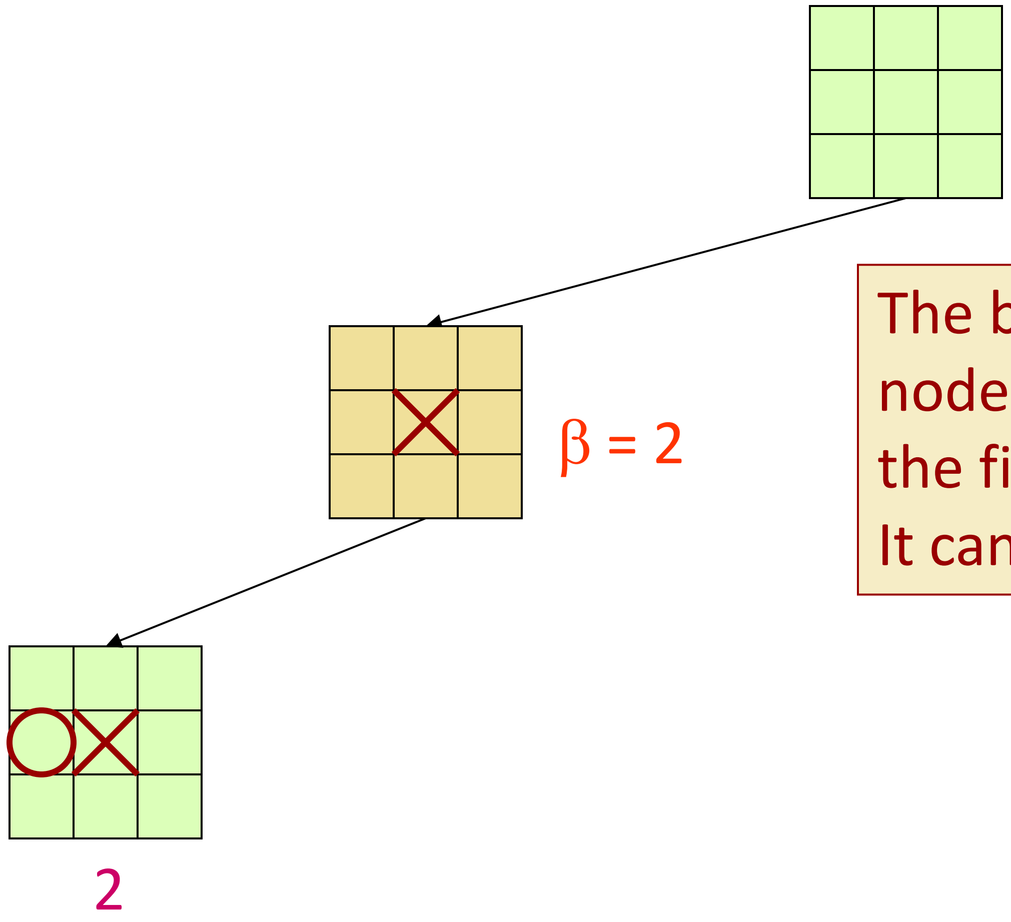


This part of the tree can't have any effect on the value that will be backed up to the root

# Example

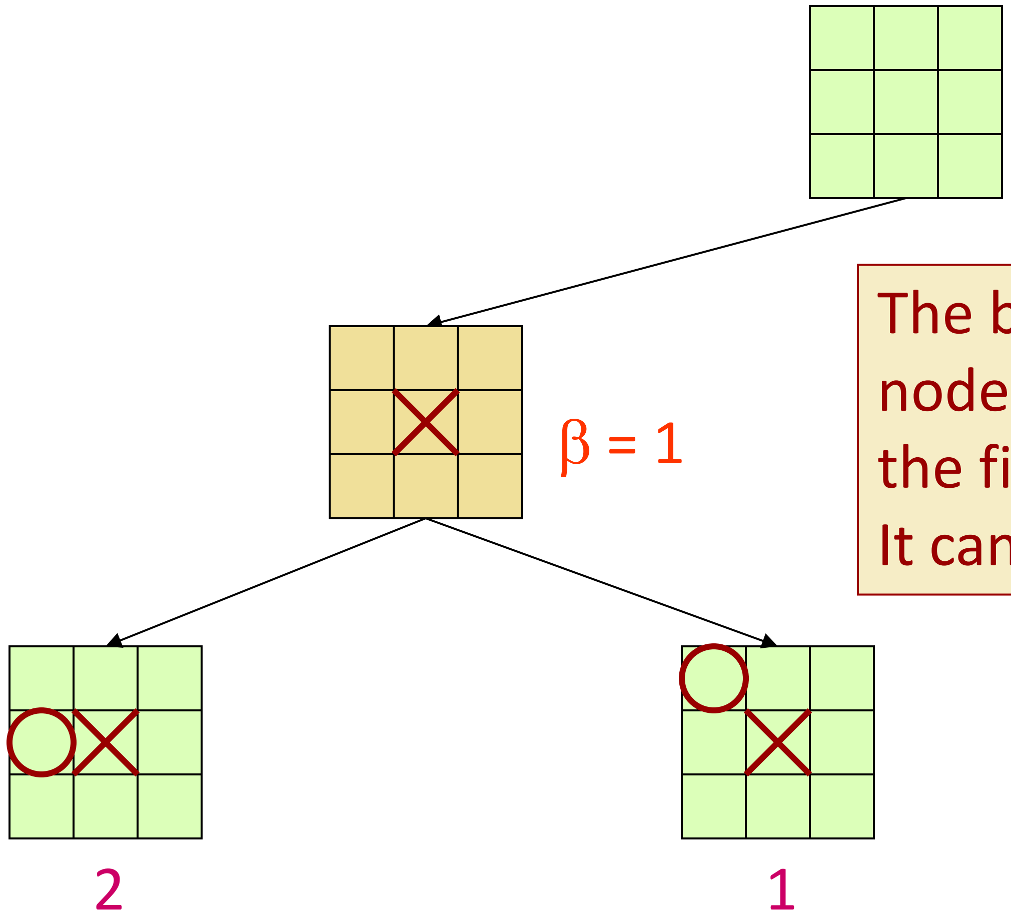


# Example



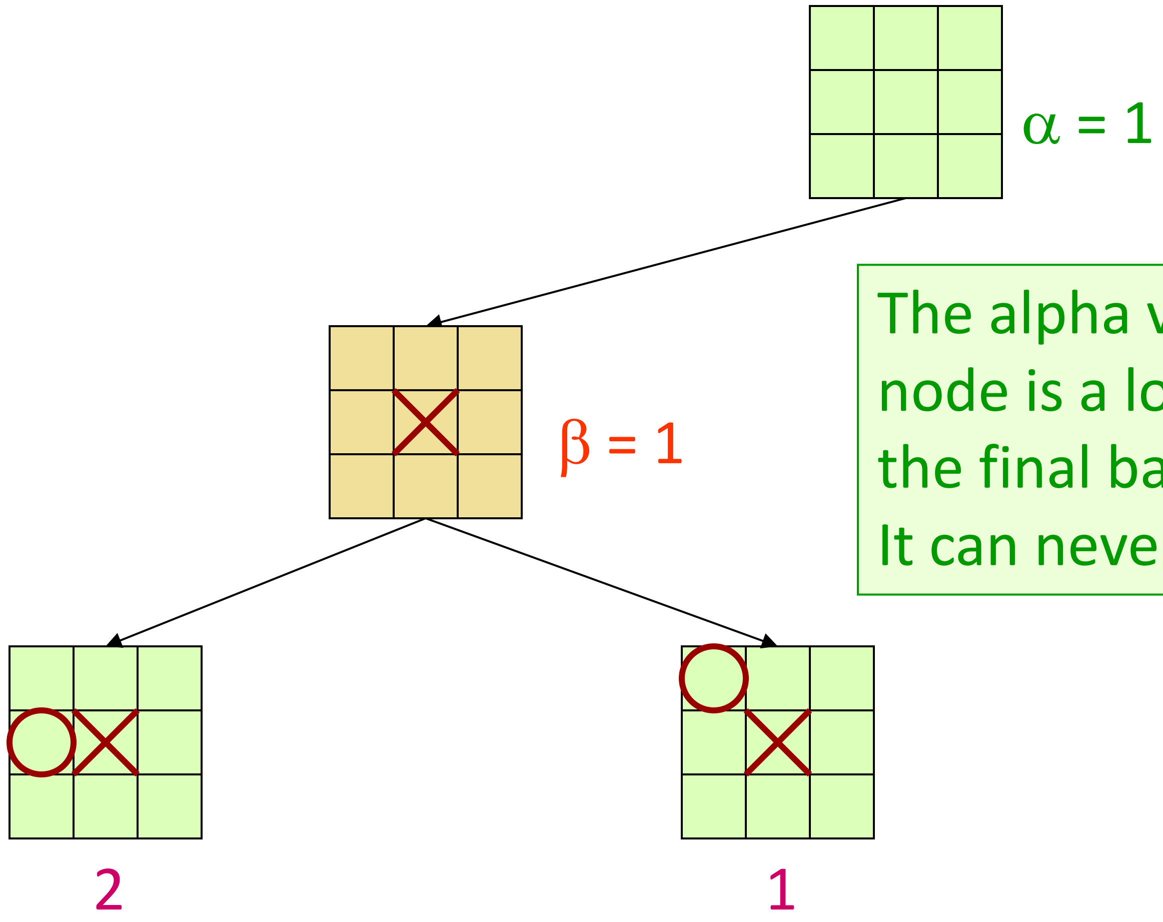
The beta value of a MIN node is an upper bound on the final backed-up value. It can never increase

# Example



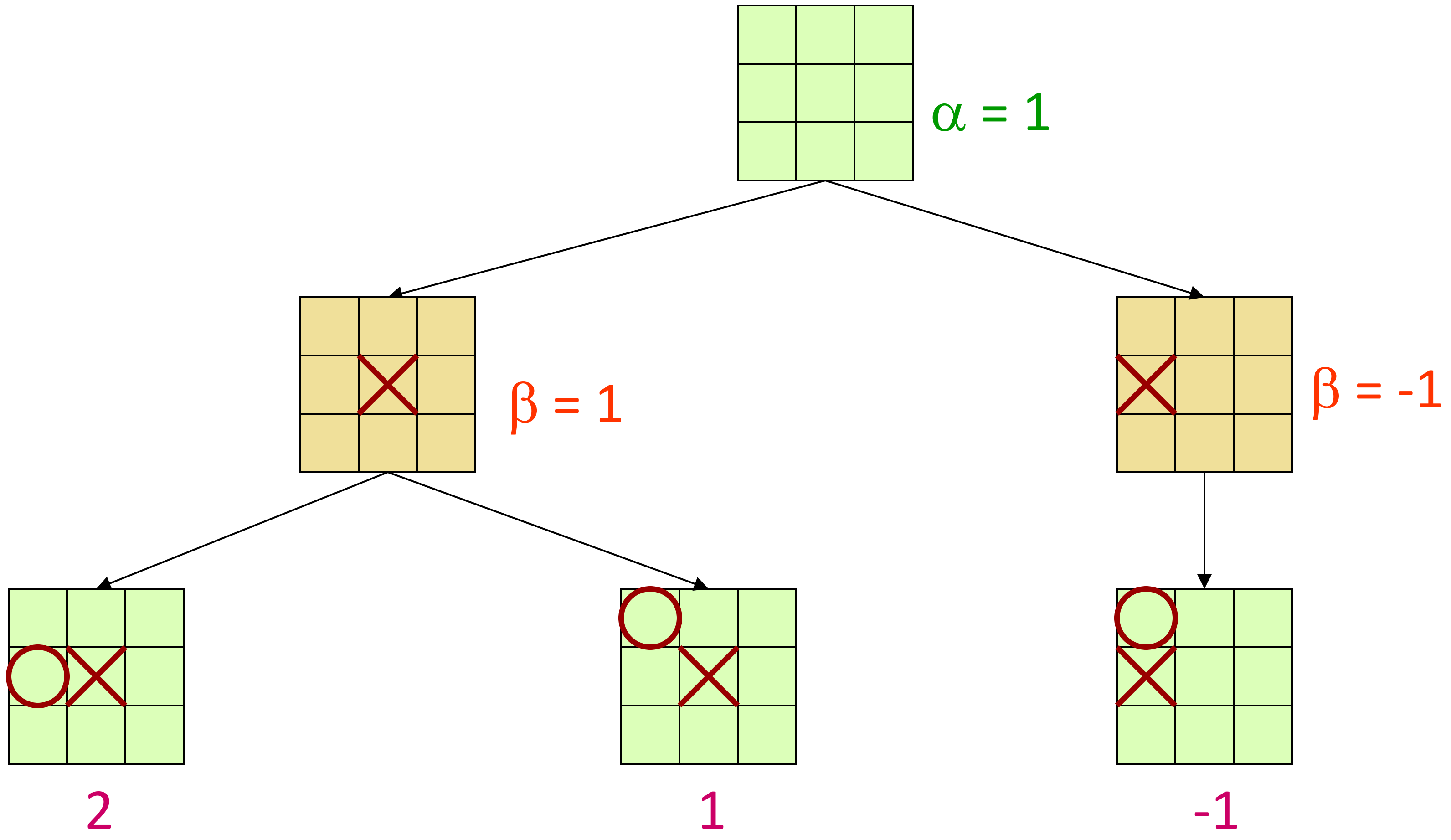
The beta value of a MIN node is an upper bound on the final backed-up value. It can never increase

# Example

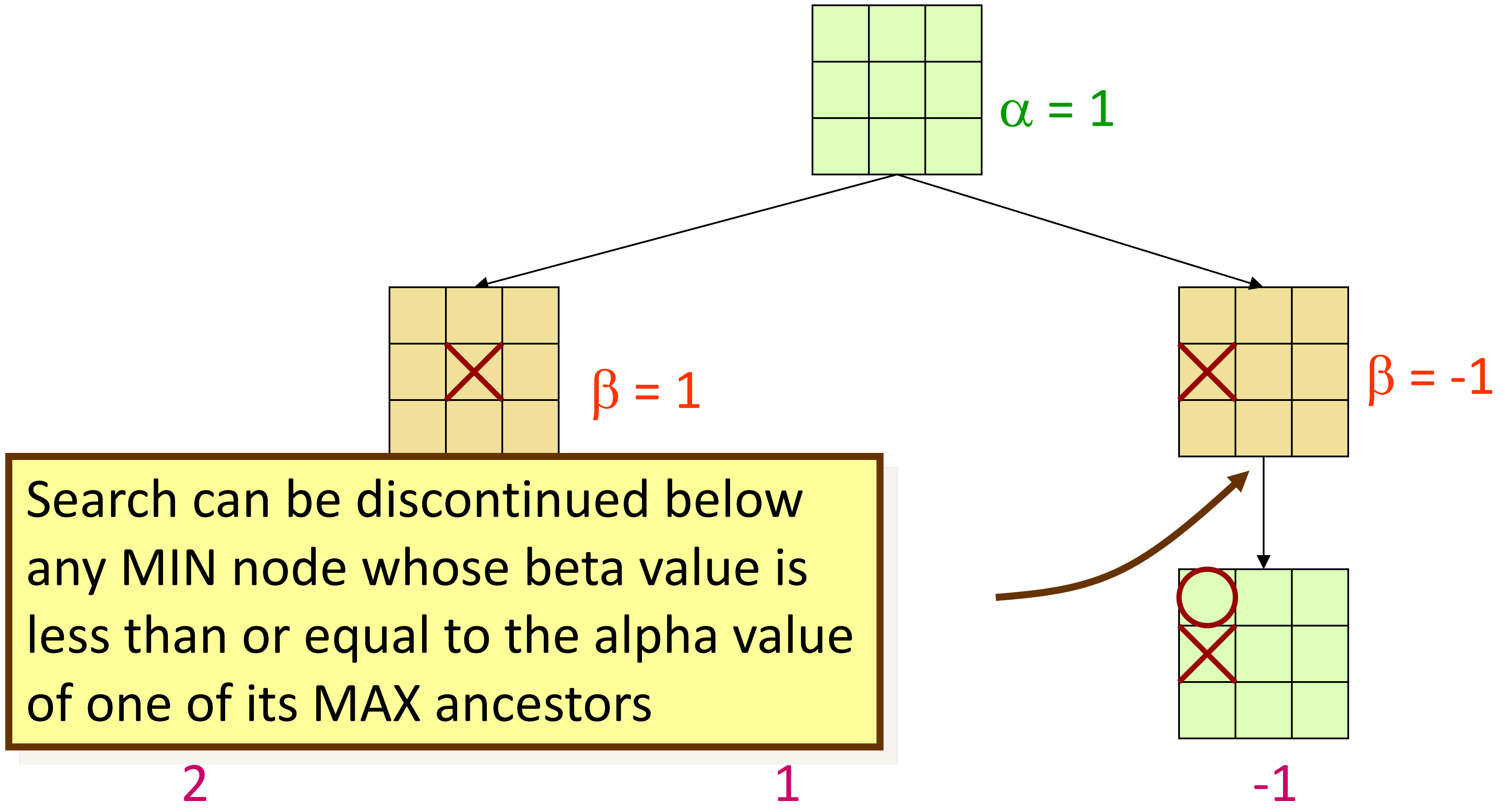


The alpha value of a MAX node is a lower bound on the final backed-up value. It can never decrease

# Example



# Example



Search can be discontinued below any MIN node whose beta value is less than or equal to the alpha value of one of its MAX ancestors

## Alpha-Beta Pruning

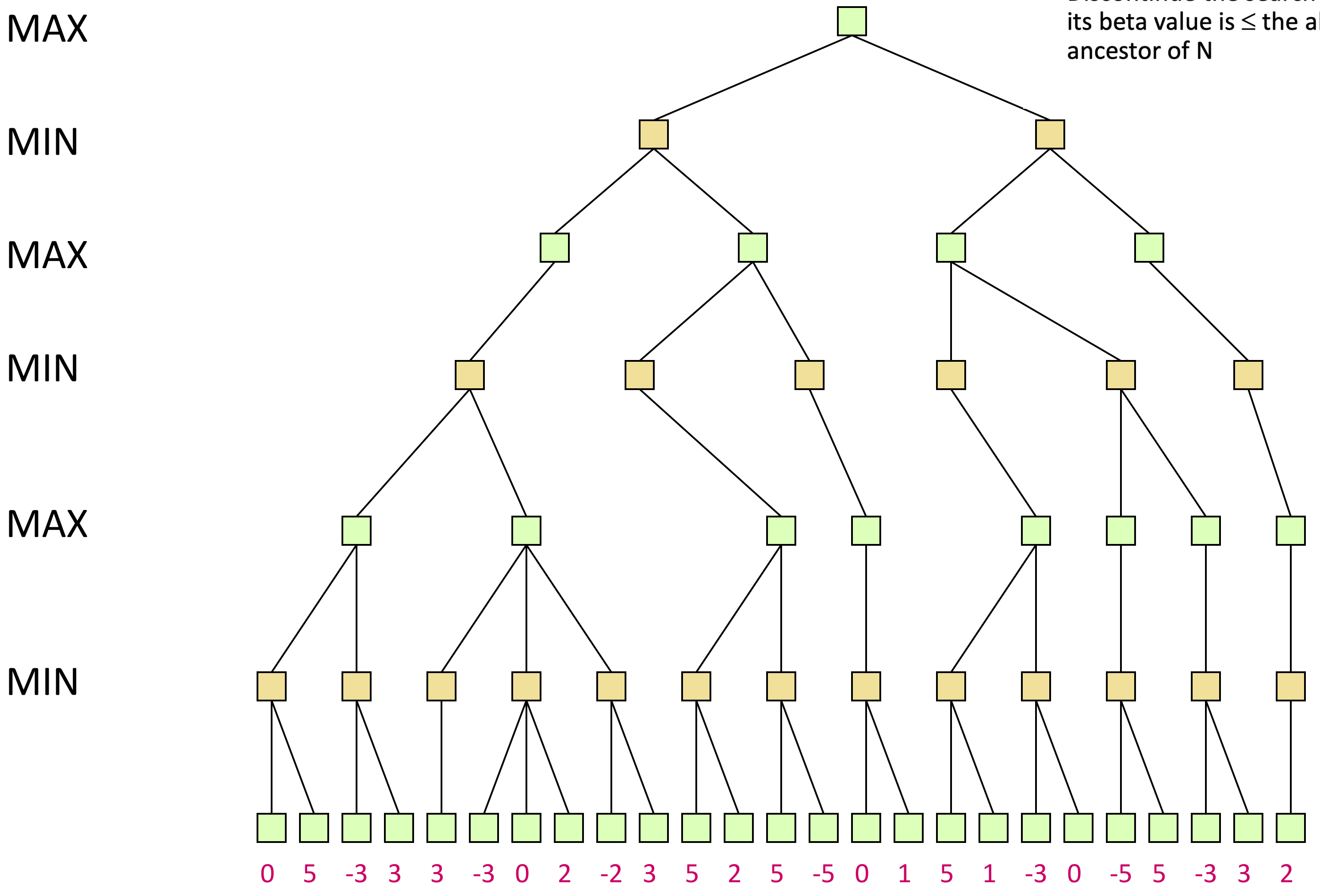
- Explore the game tree to depth h in depth-first manner
- Back up alpha and beta values whenever possible
- Prune branches that can’t lead to changing the final decision

# Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is $\geq$ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is $\leq$ the alpha value of a MAX ancestor of N

# Example

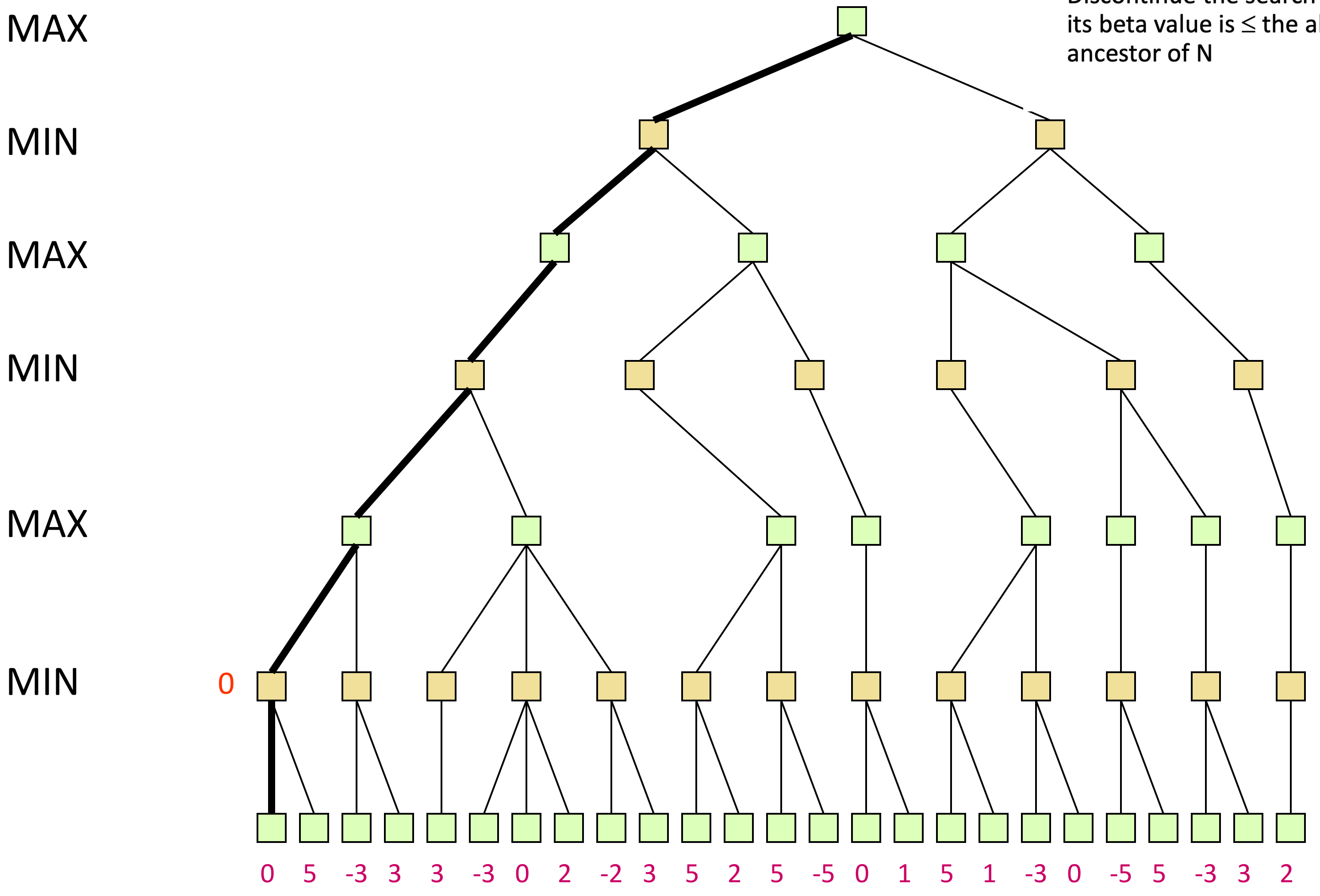
- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



Alpha-Beta Algorithm

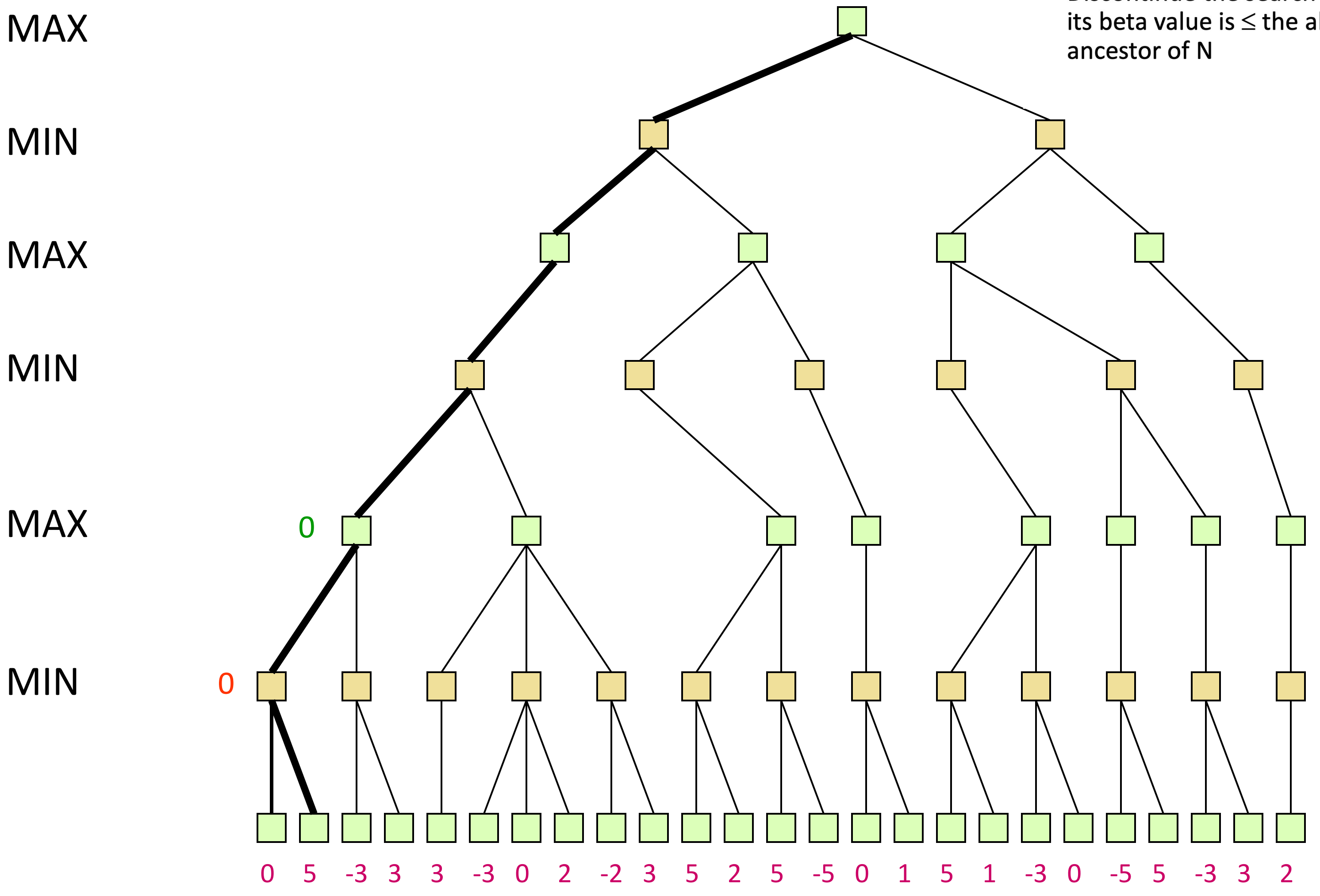
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



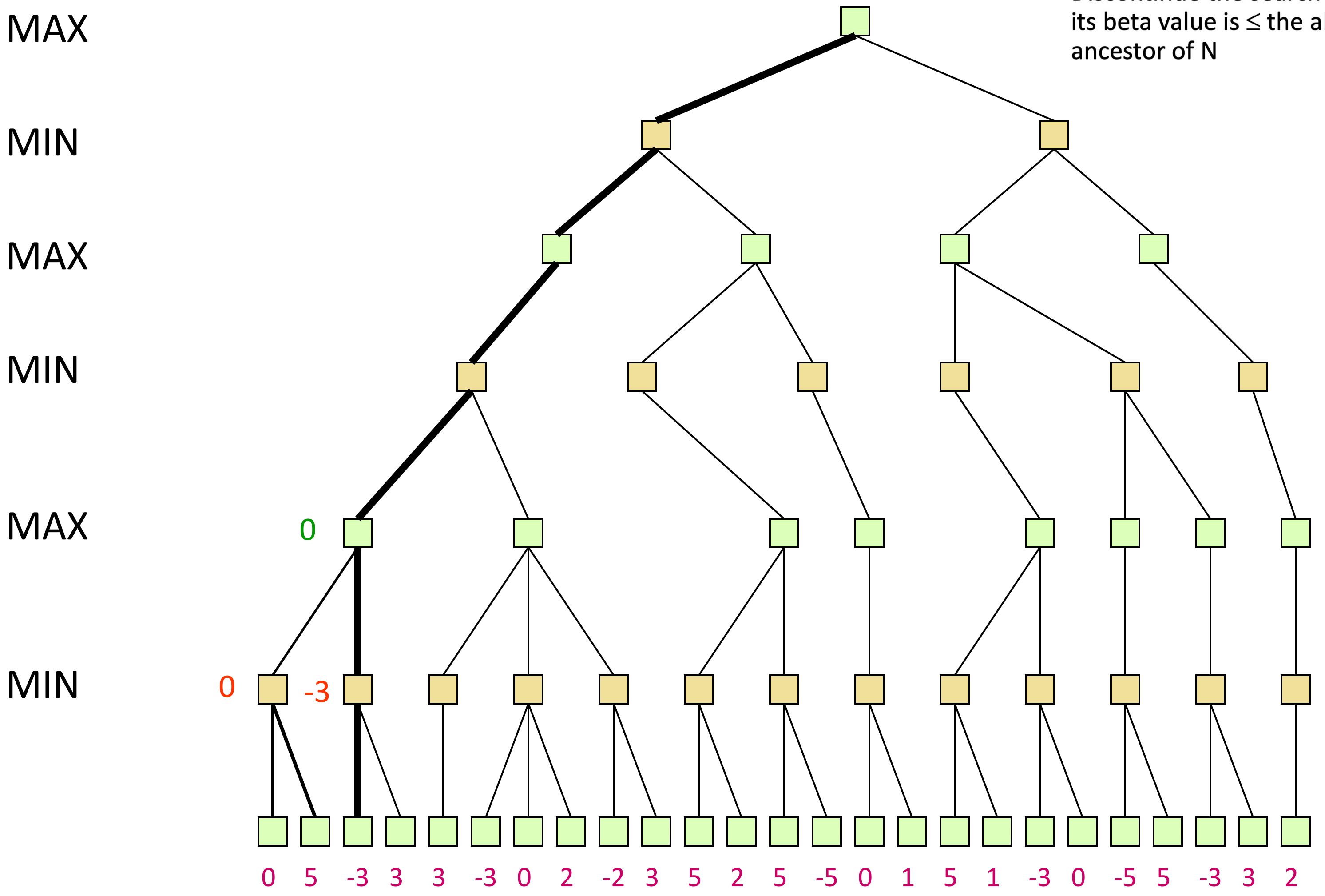
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



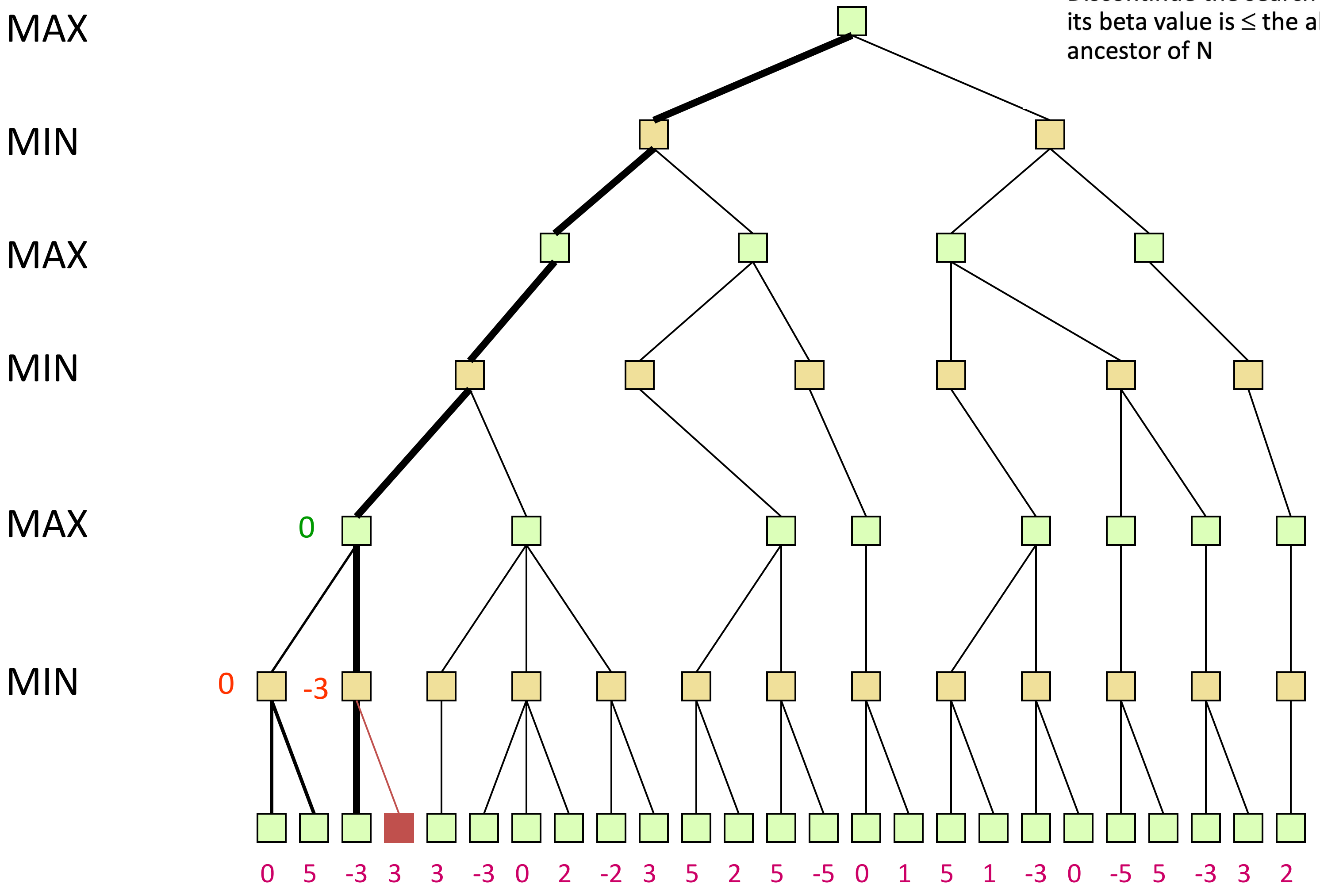
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is $\geq$ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is $\leq$ the alpha value of a MAX ancestor of N



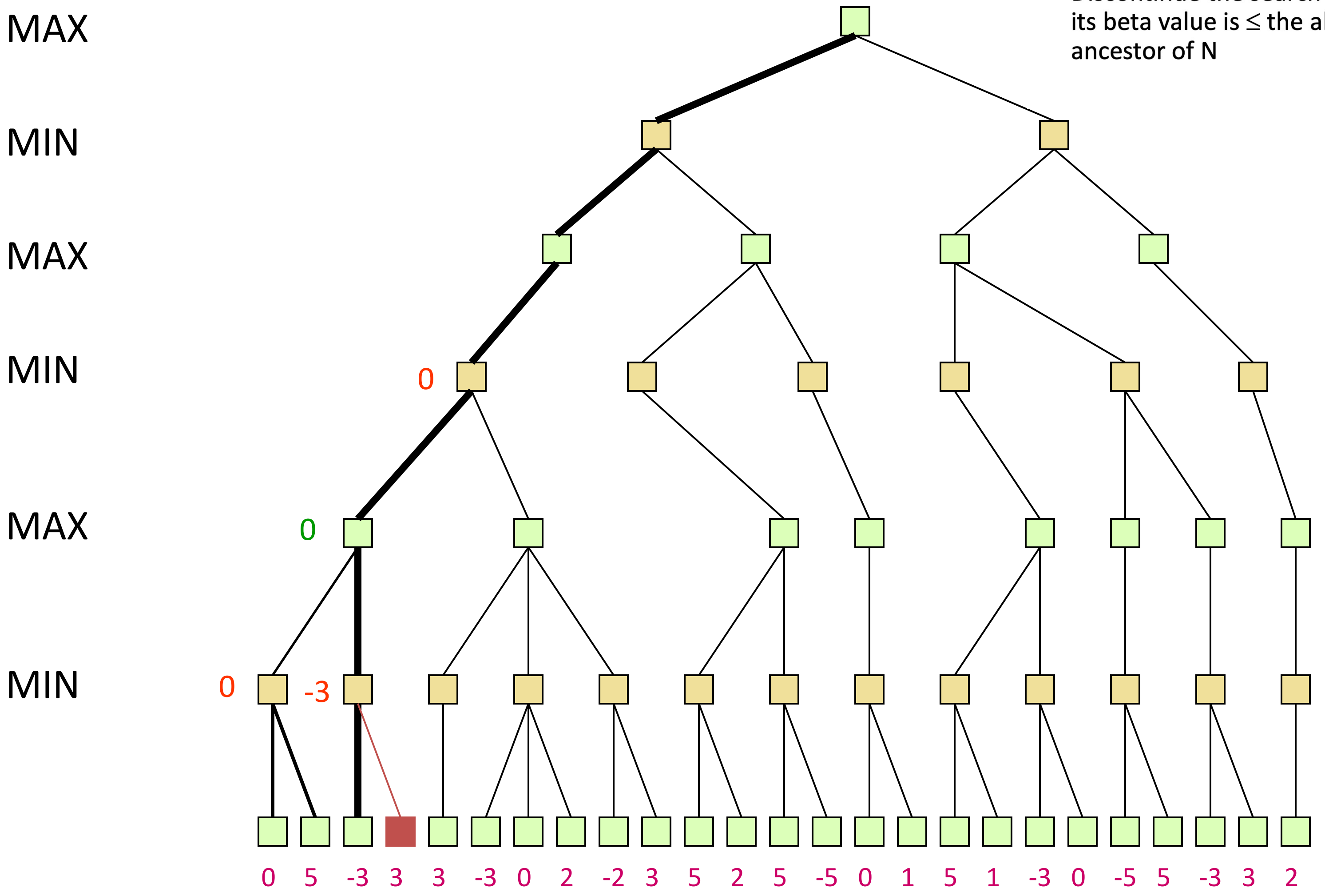
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



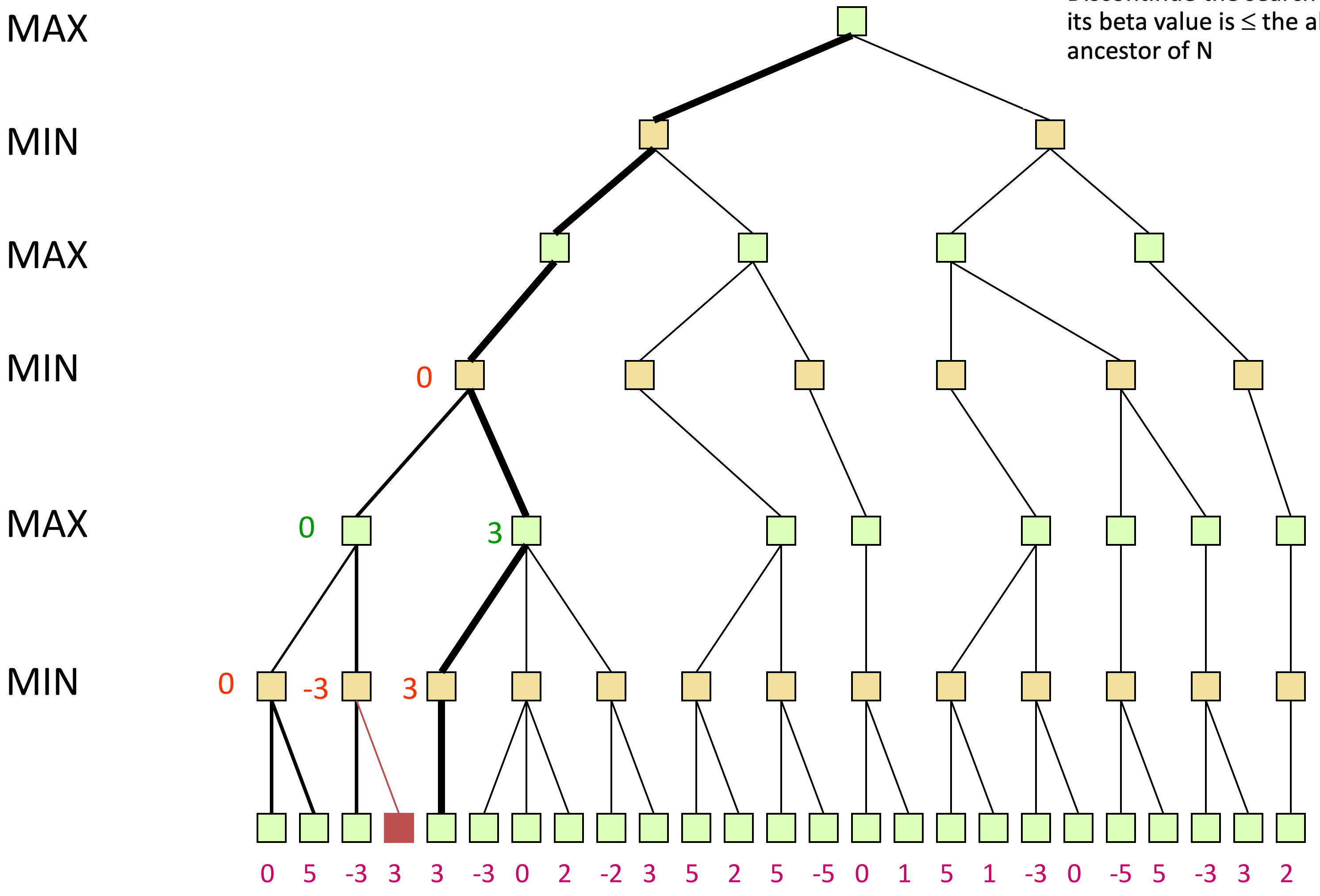
- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N

# Example



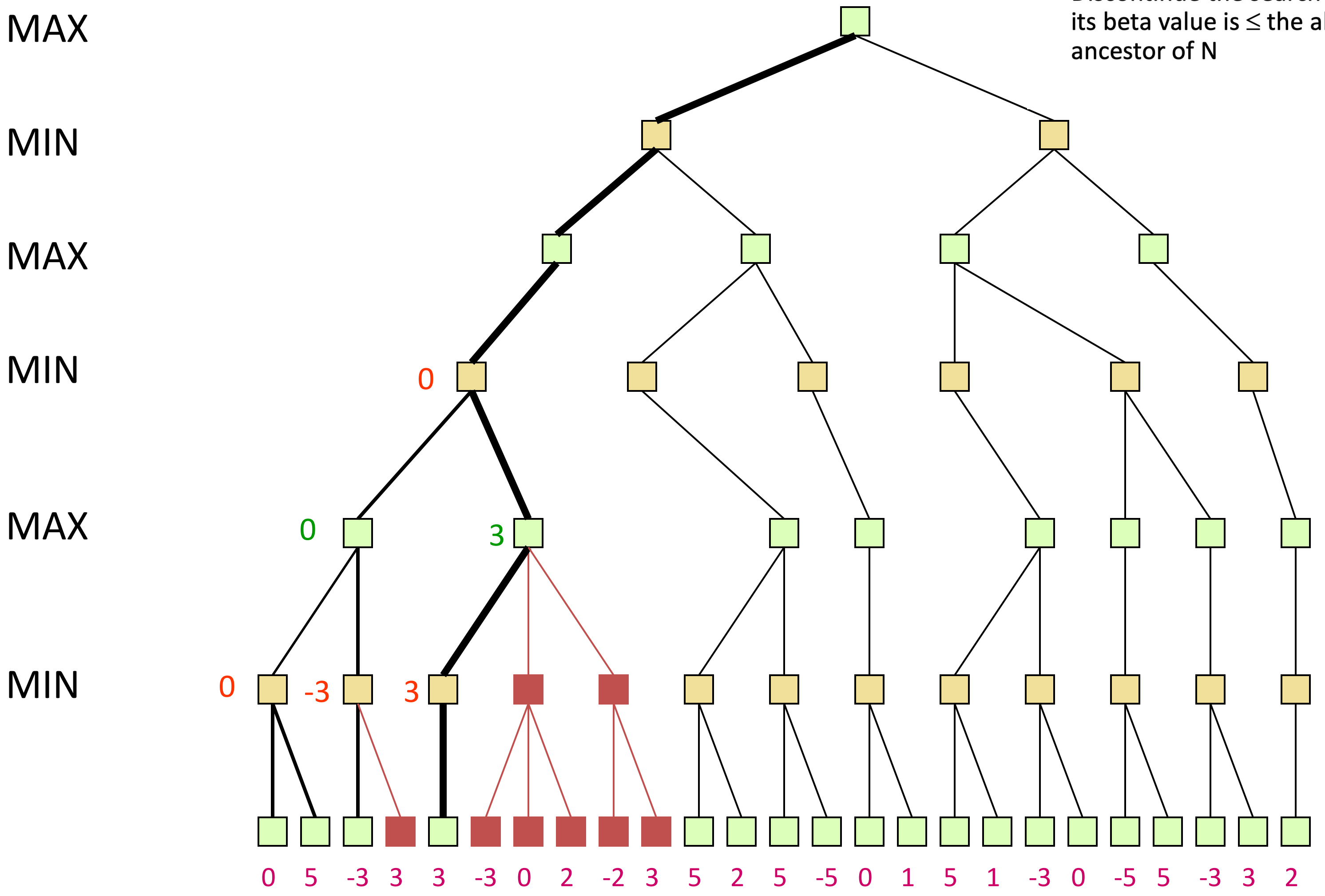
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



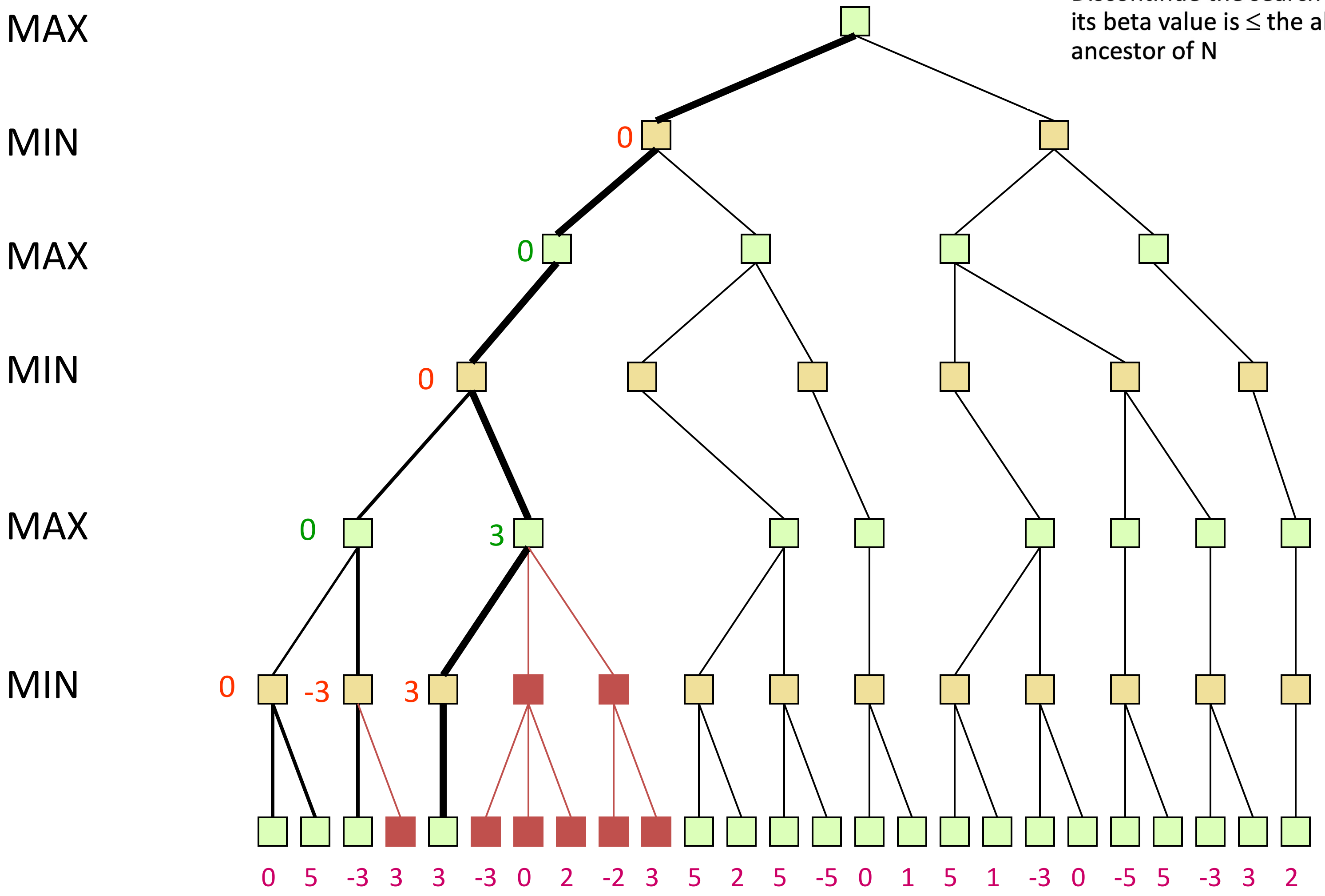
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



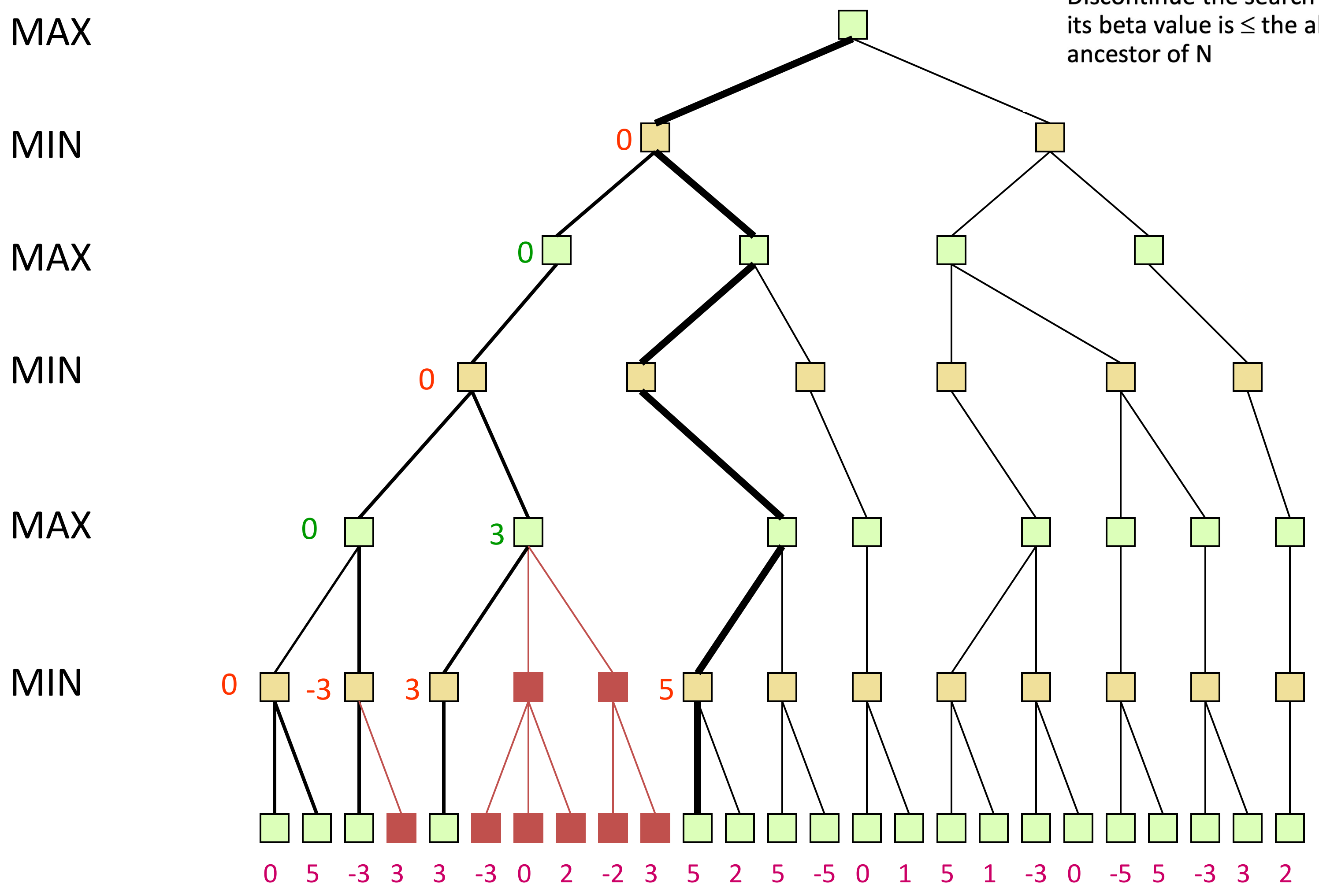
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



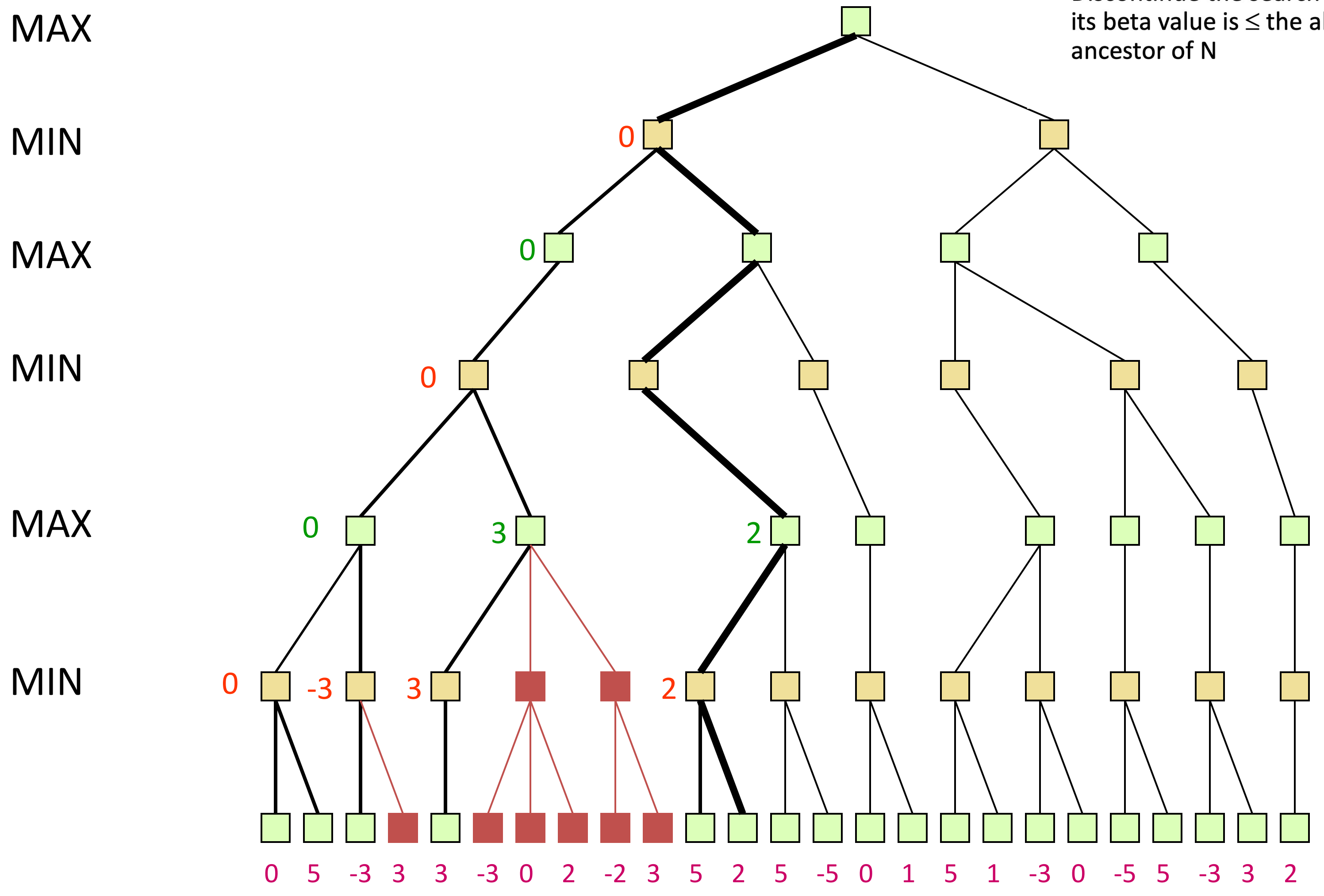
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



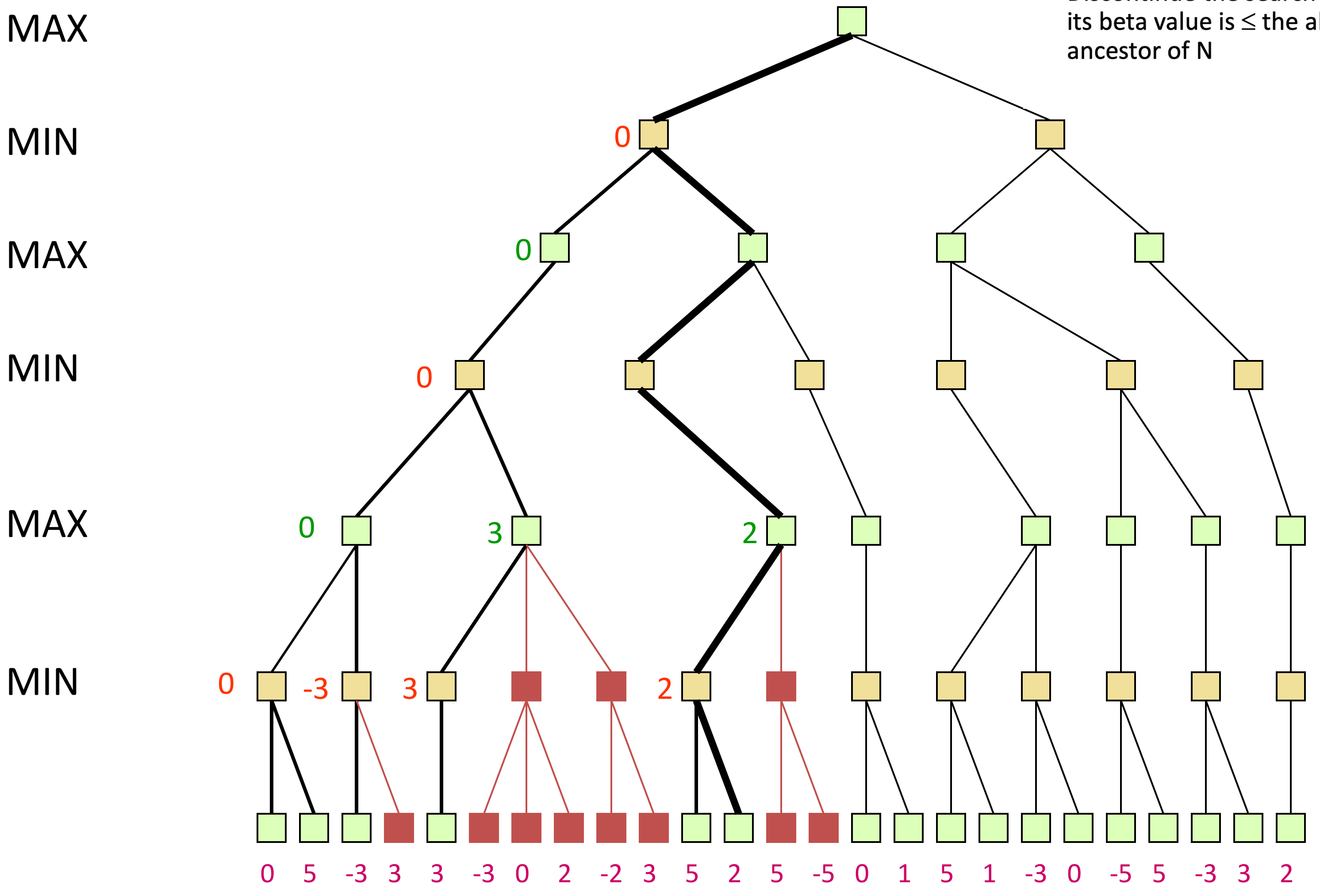
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



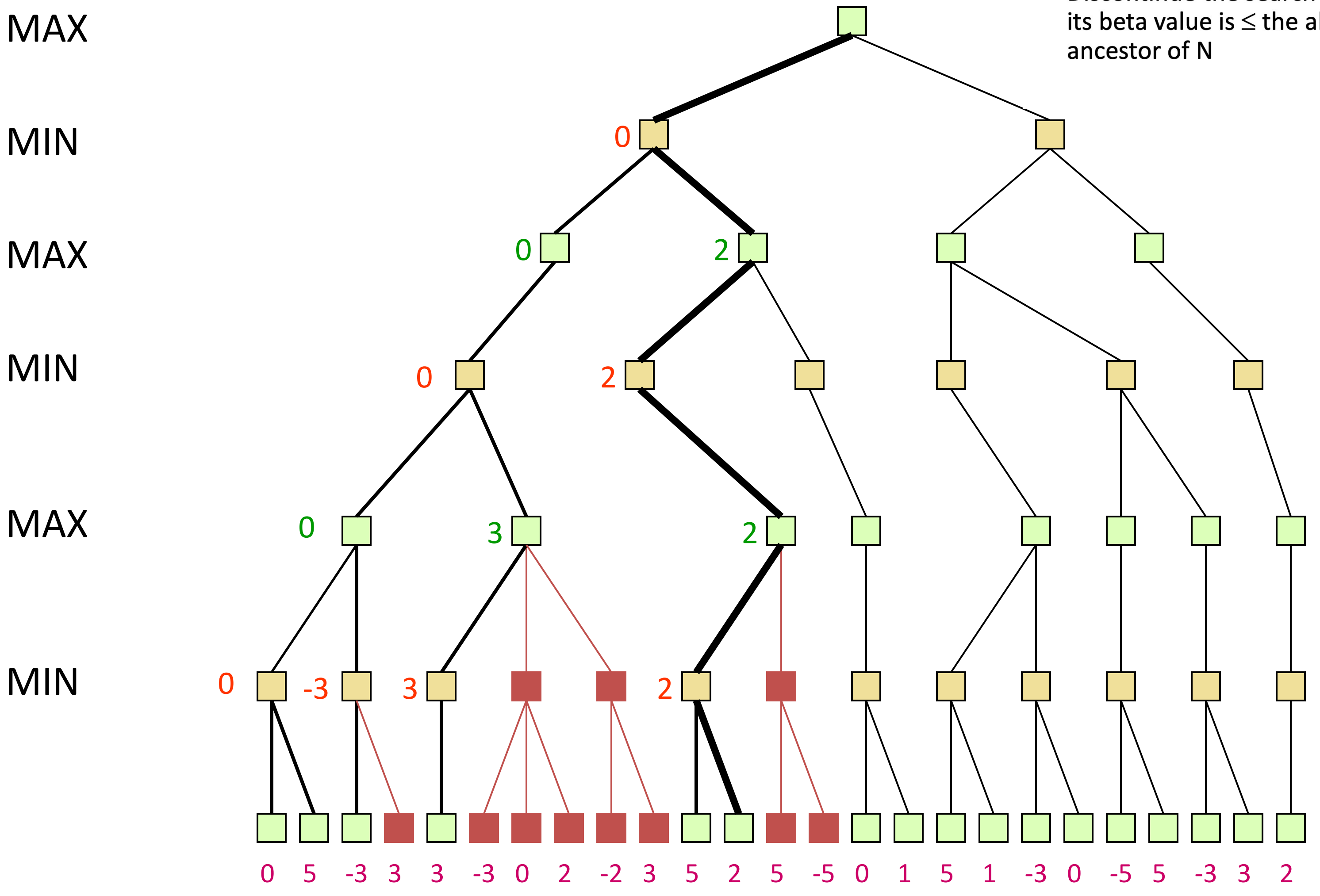
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



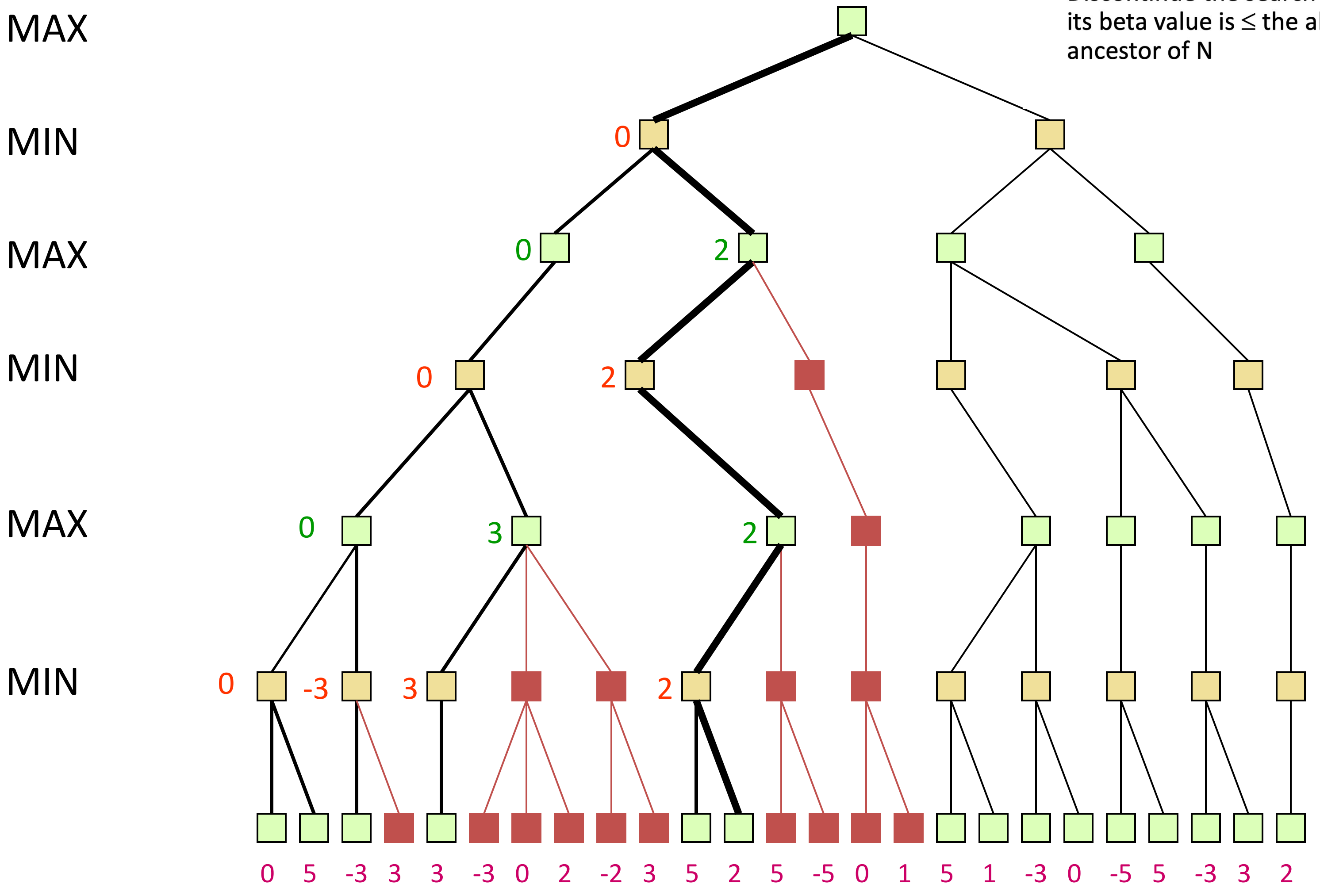
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



# Example

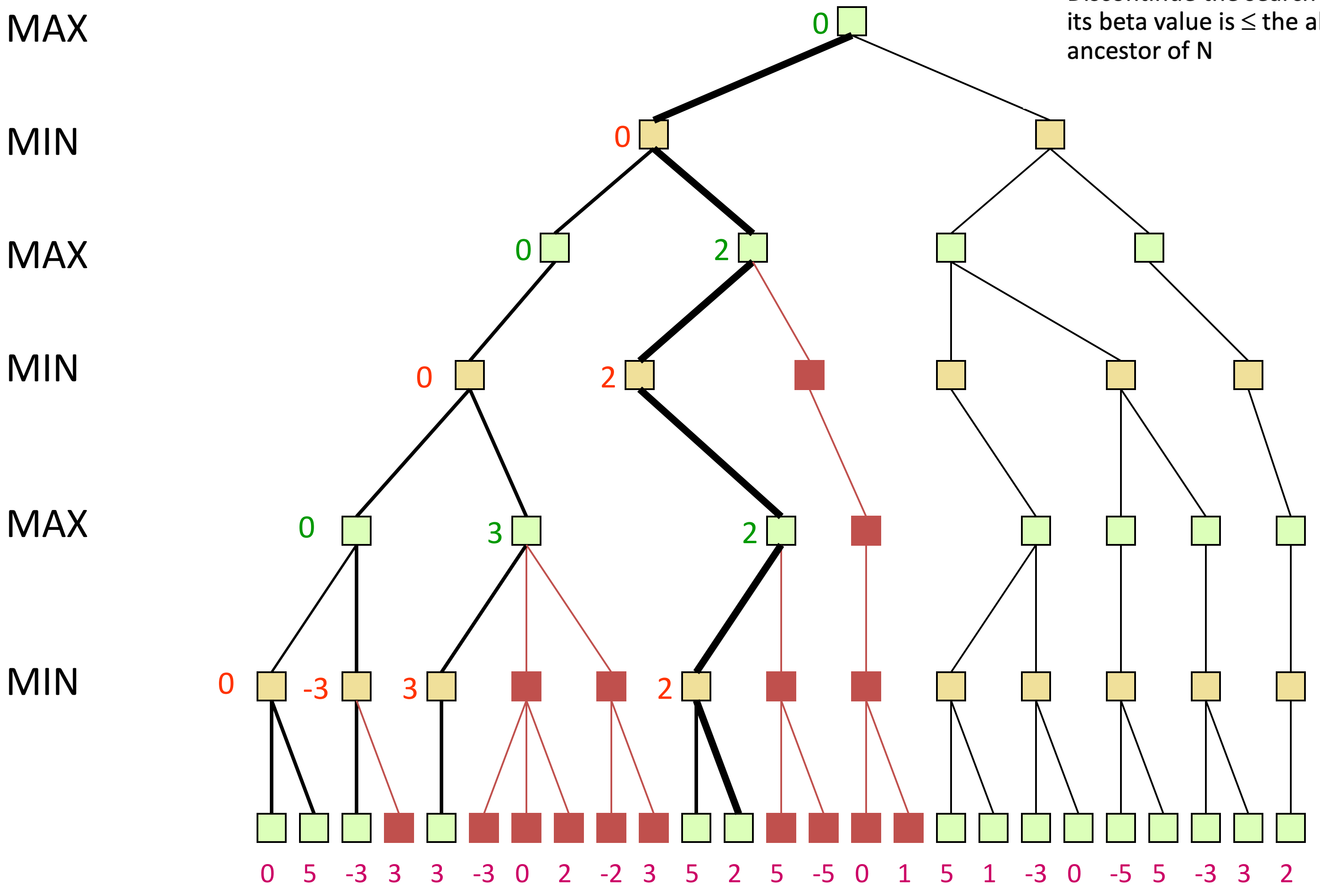
- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



## Alpha-Beta Algorithm

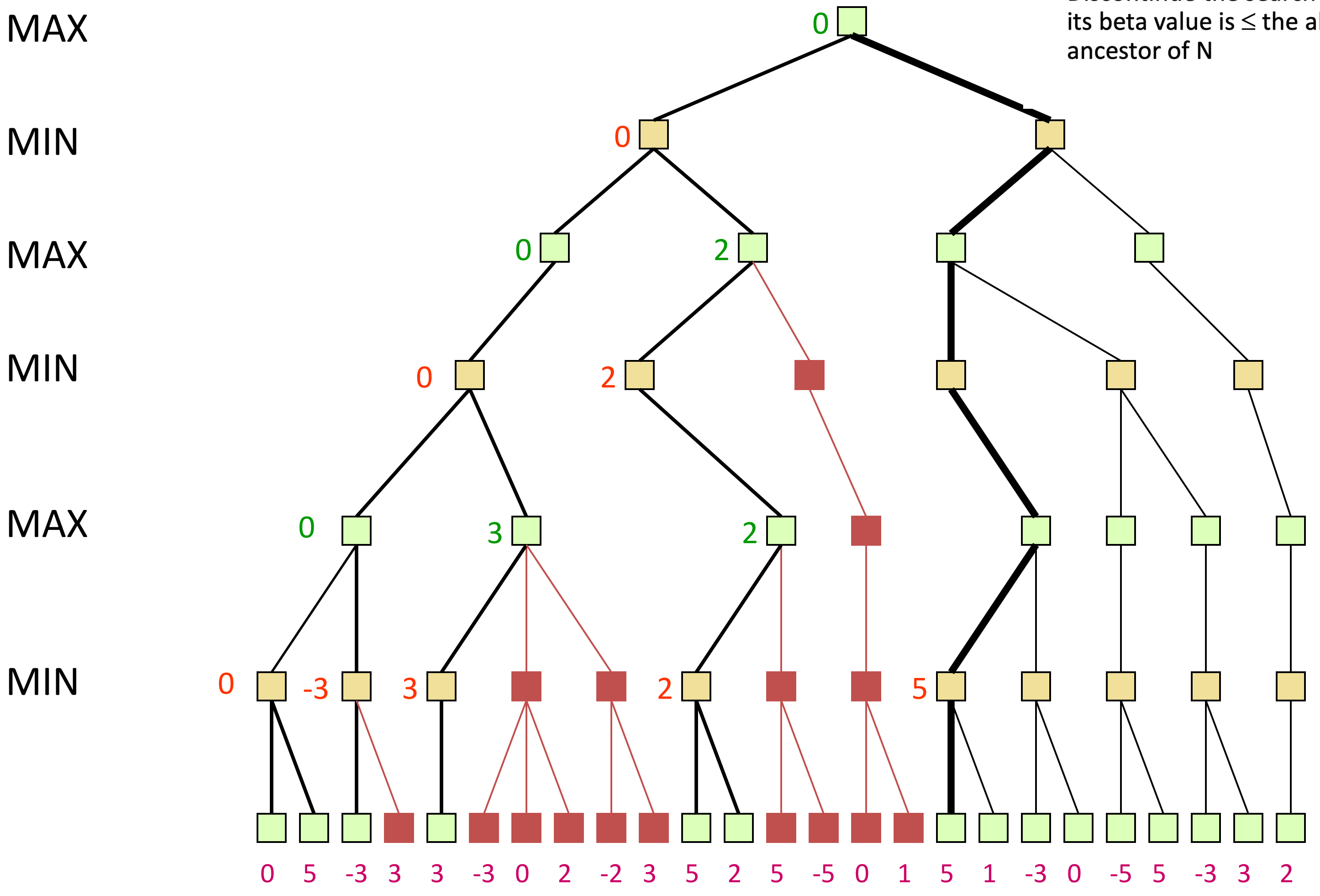
- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N

# Example



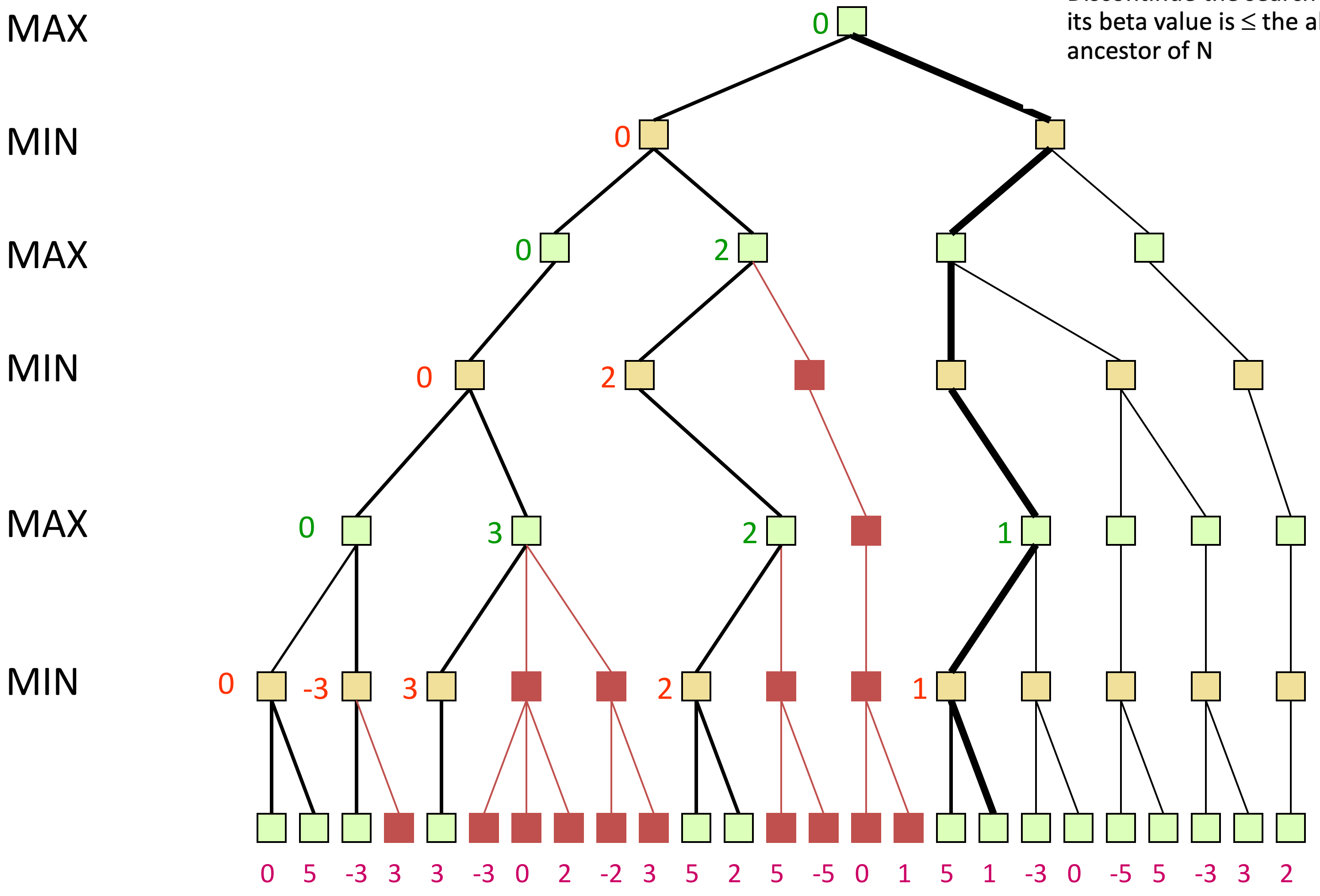
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is $\geq$ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is $\leq$ the alpha value of a MAX ancestor of N



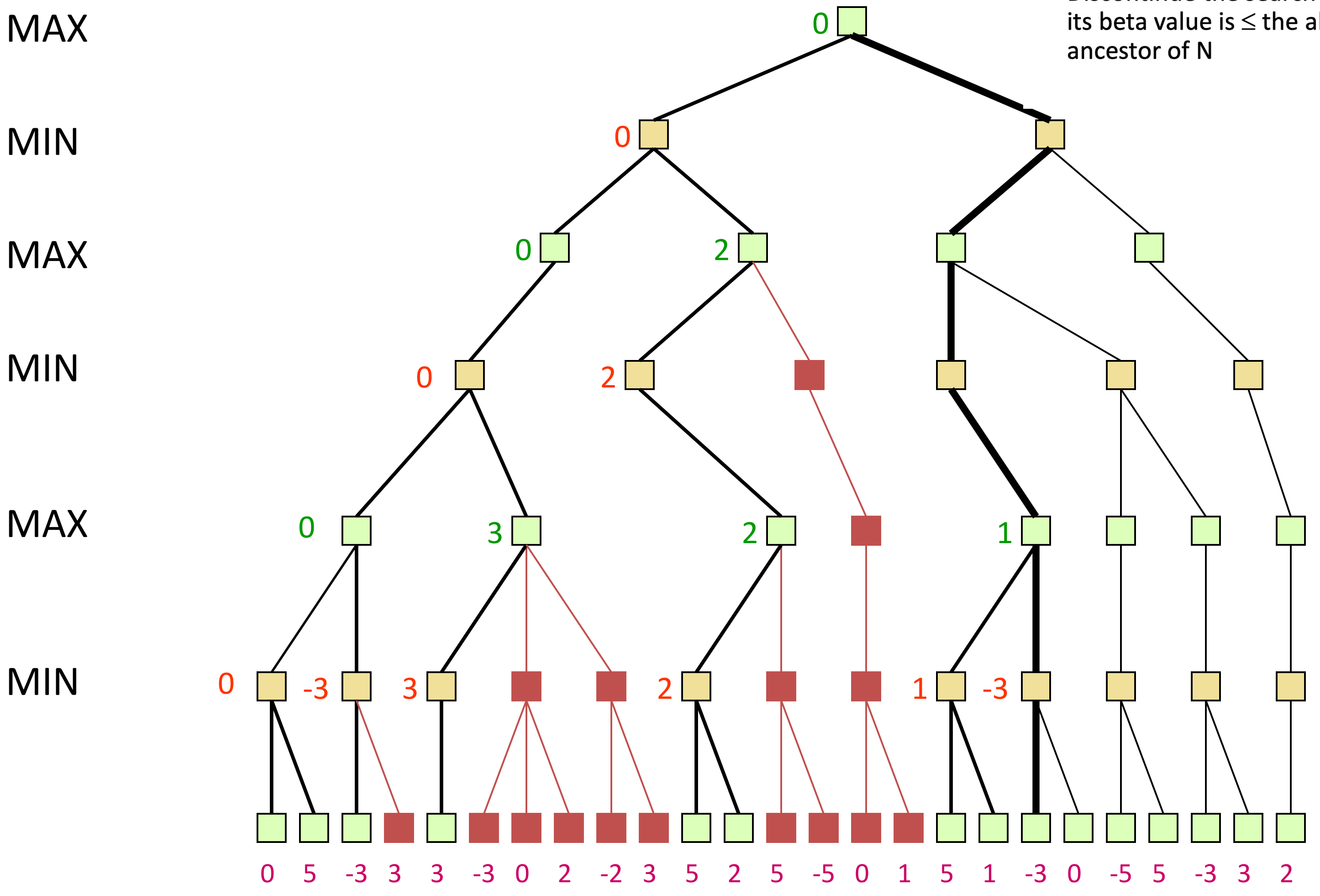
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



# Example

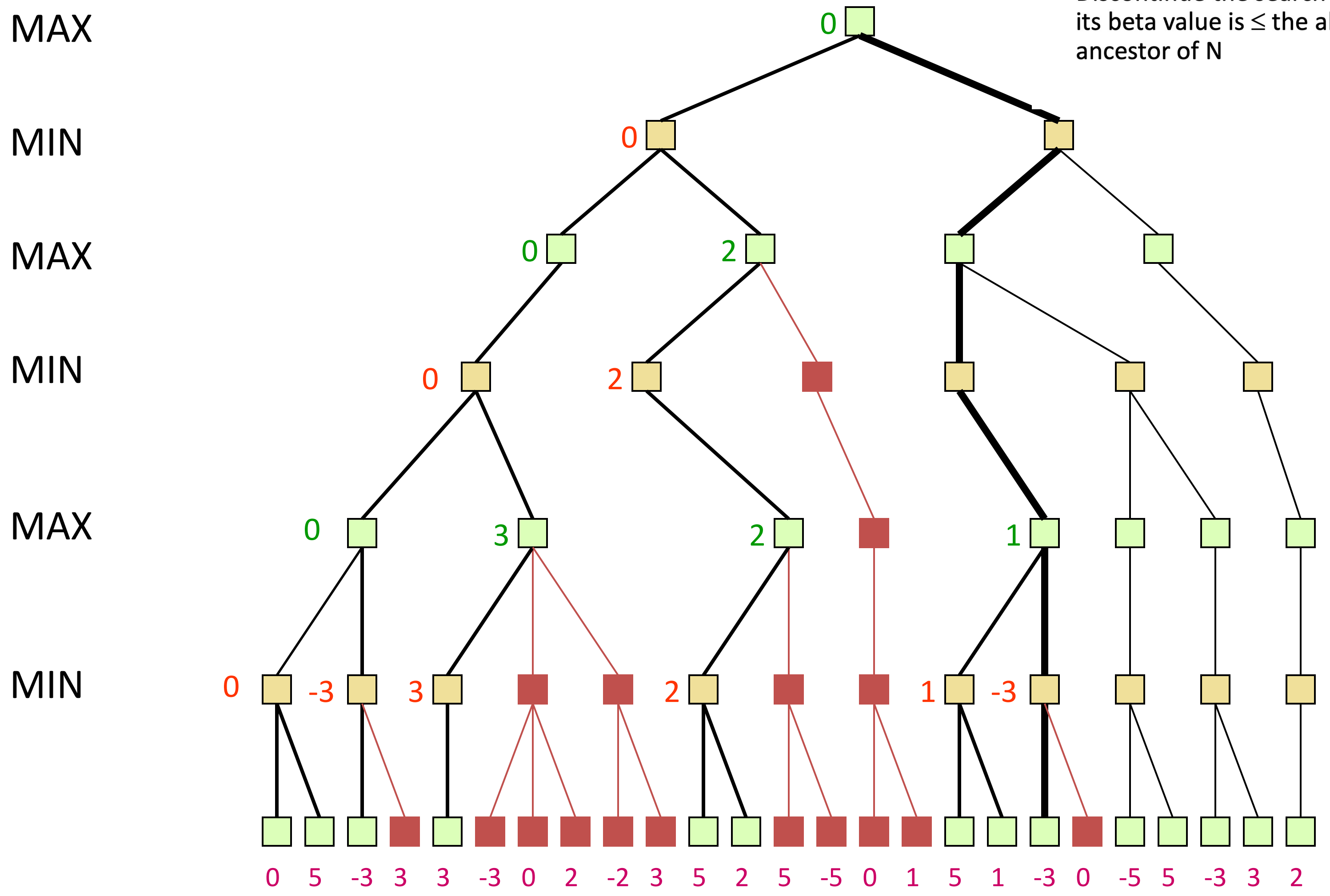
- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



# Example

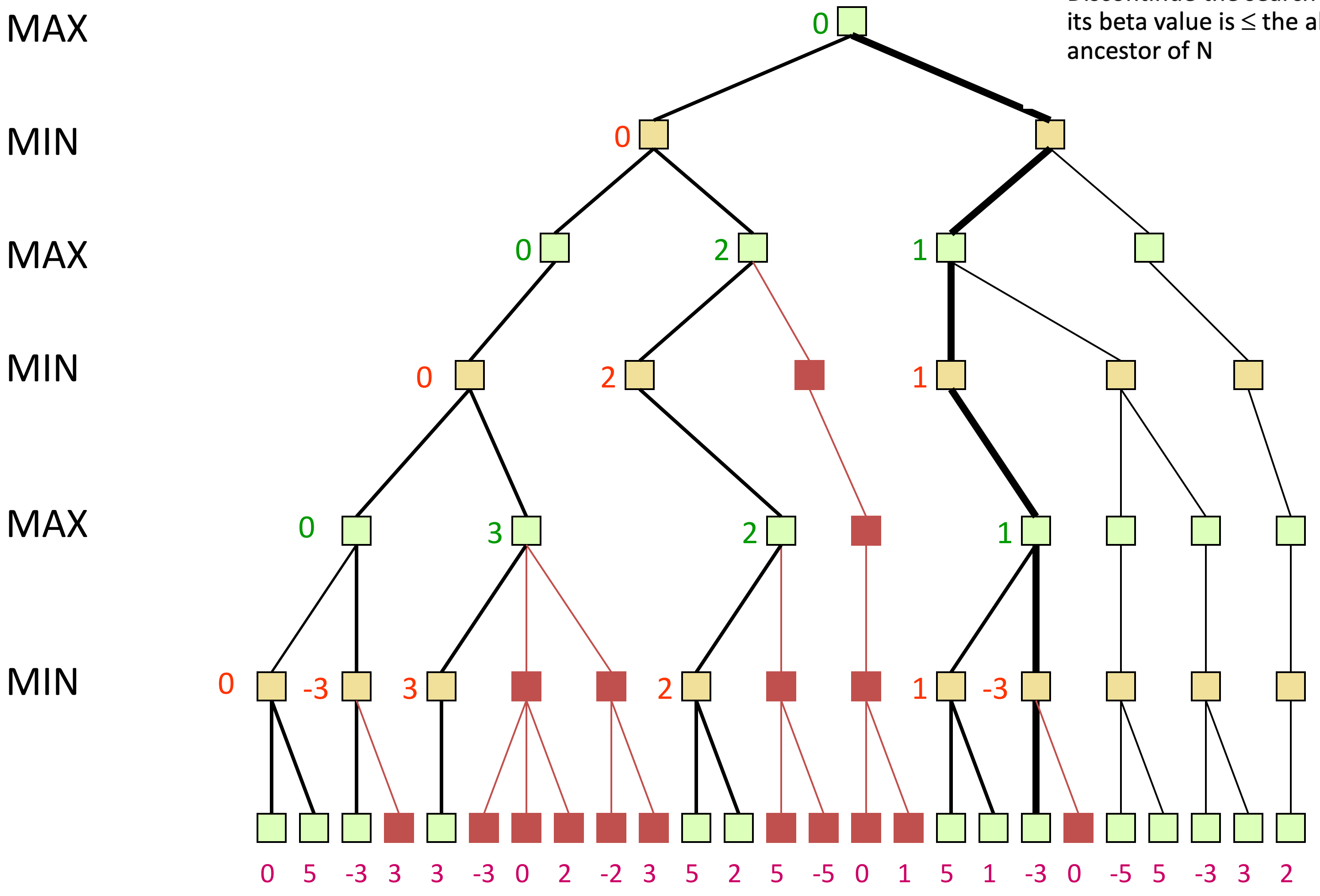
## Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



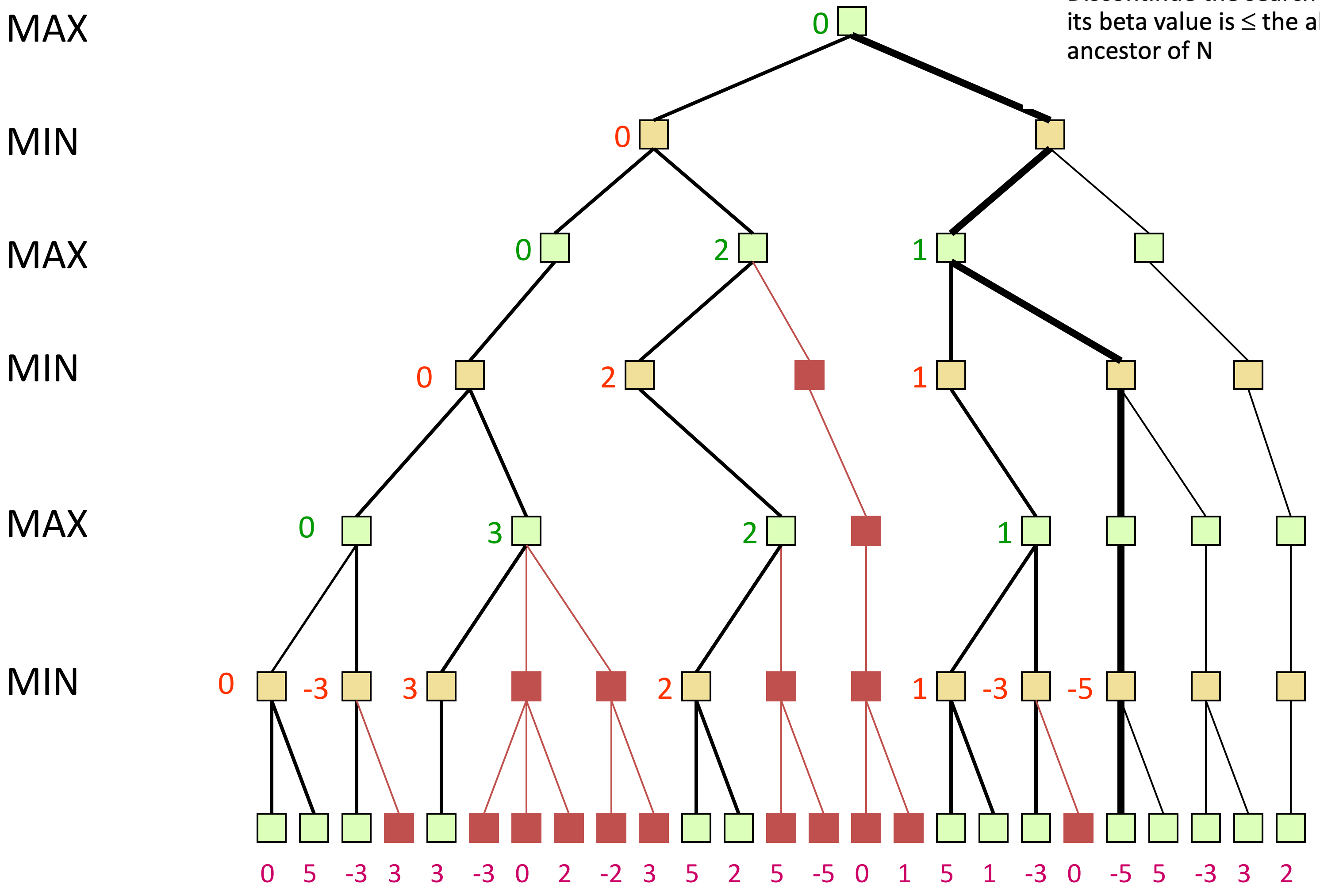
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



# Example

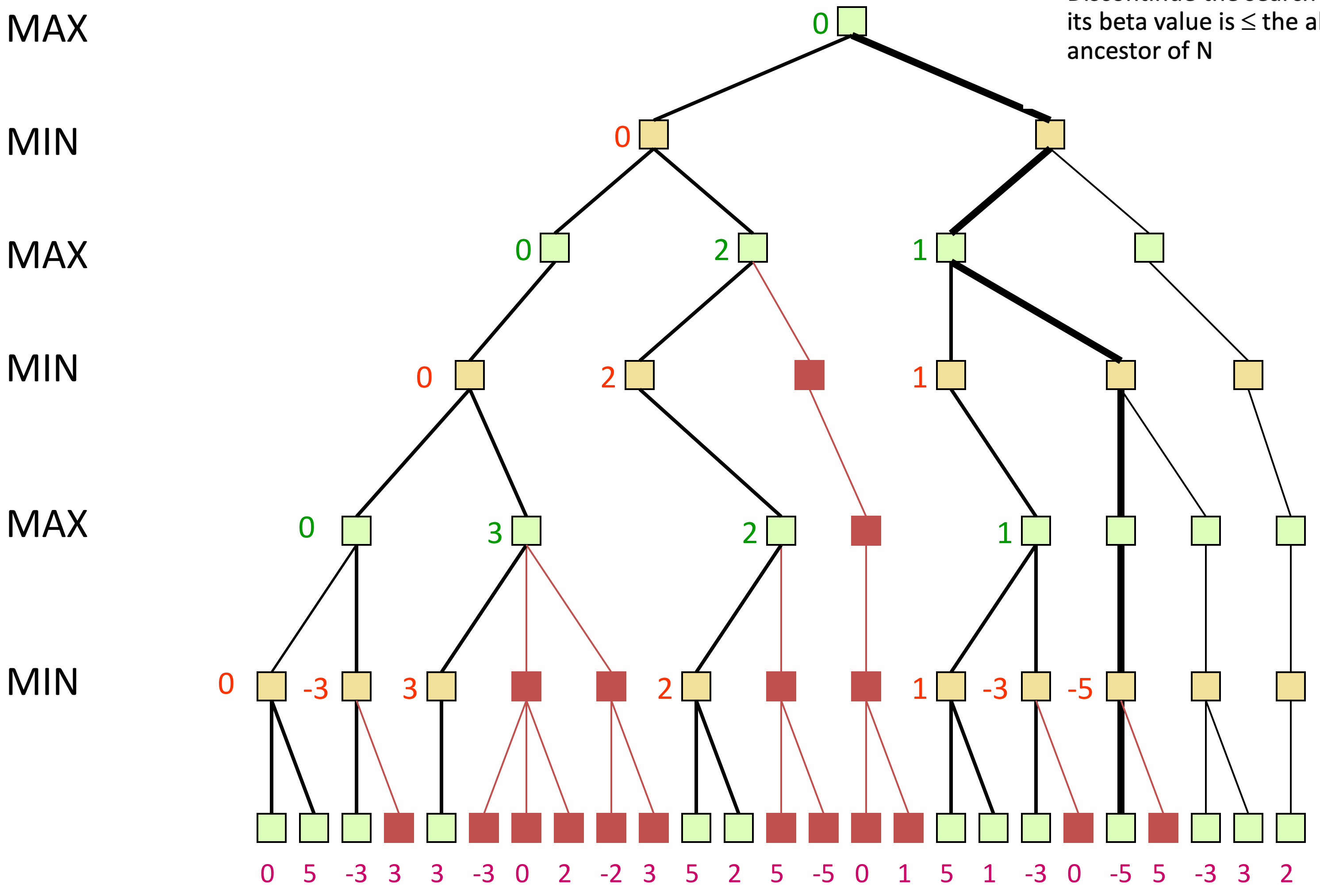
- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



# Example

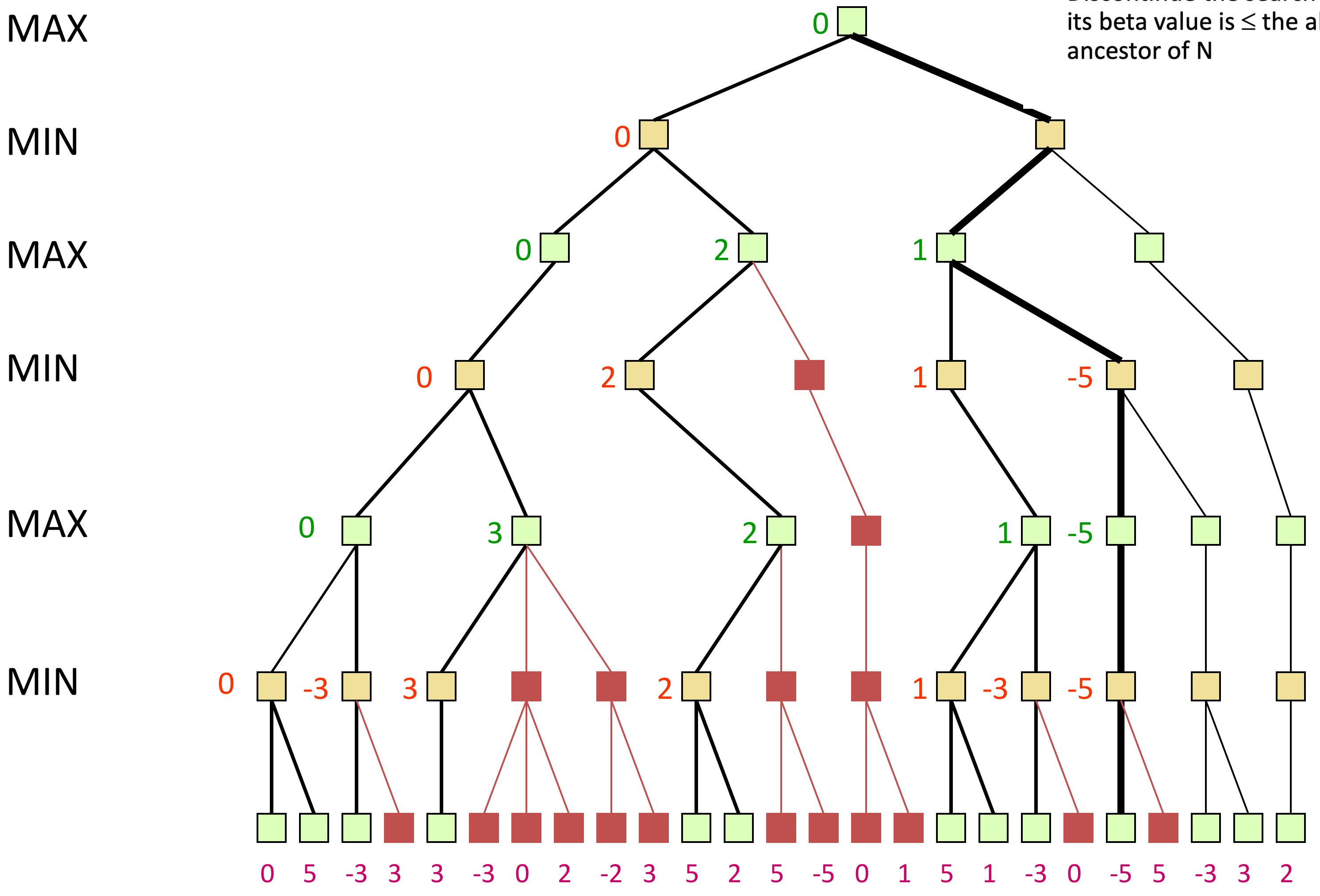
## Alpha-Beta Algorithm

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



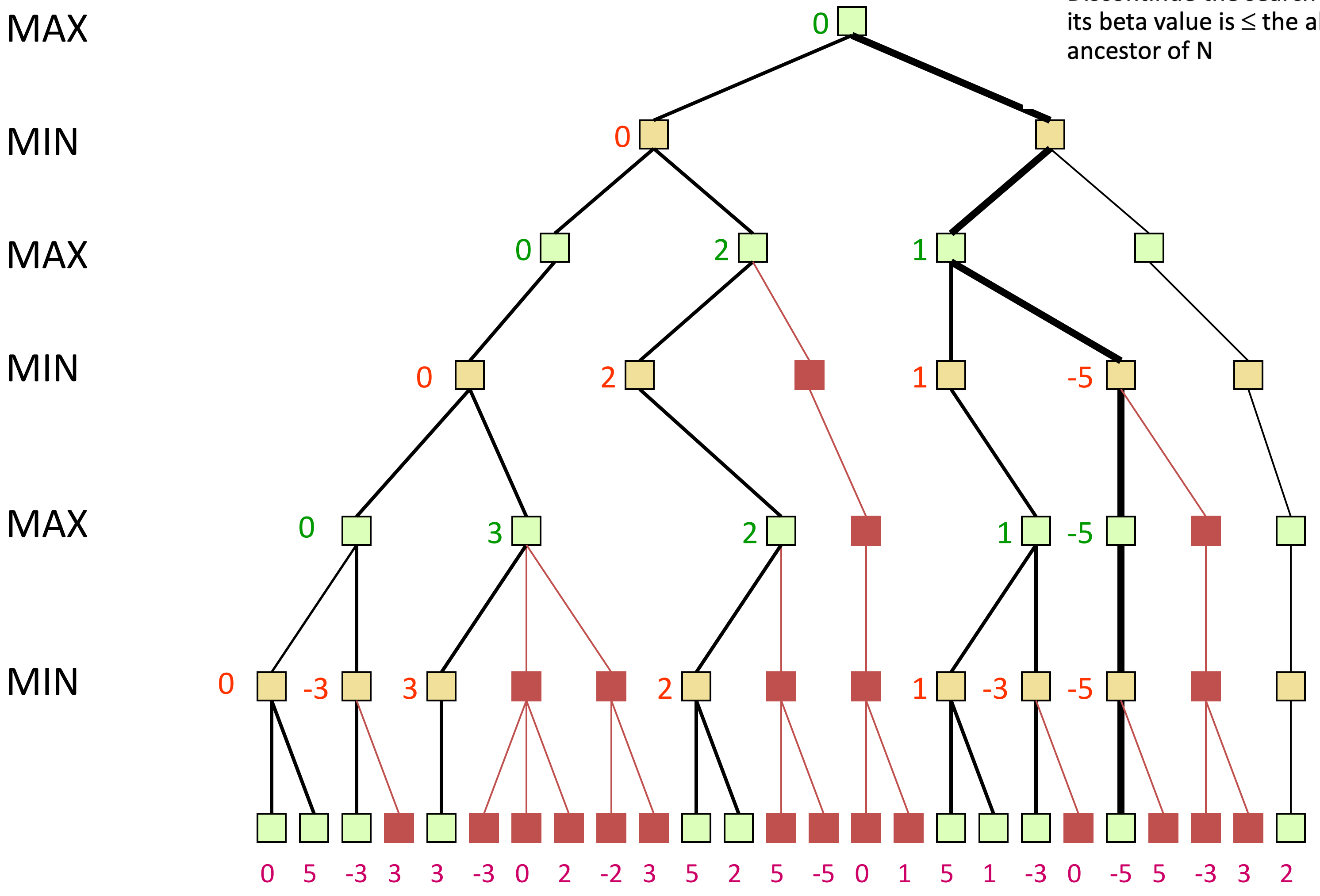
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



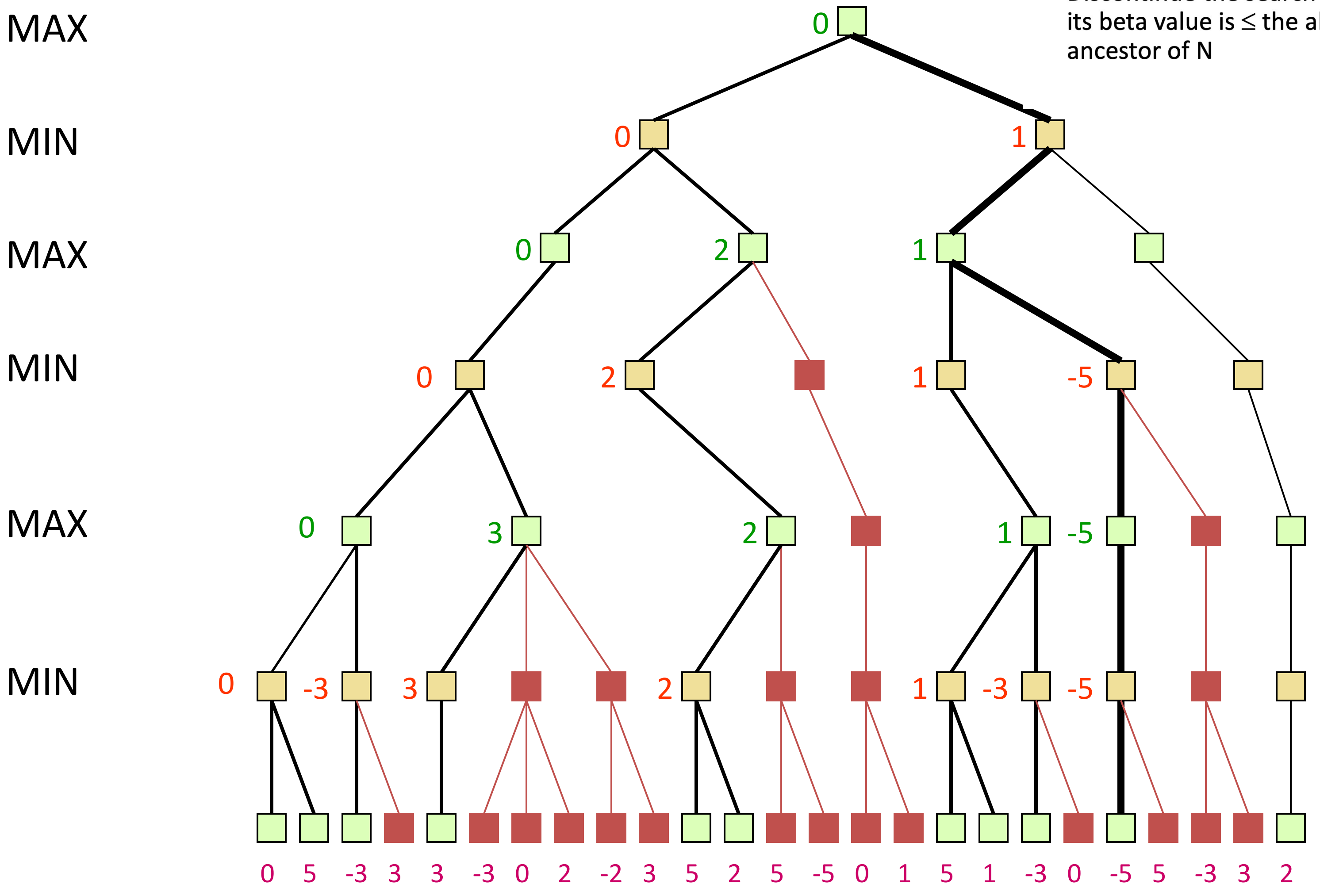
- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N

# Example



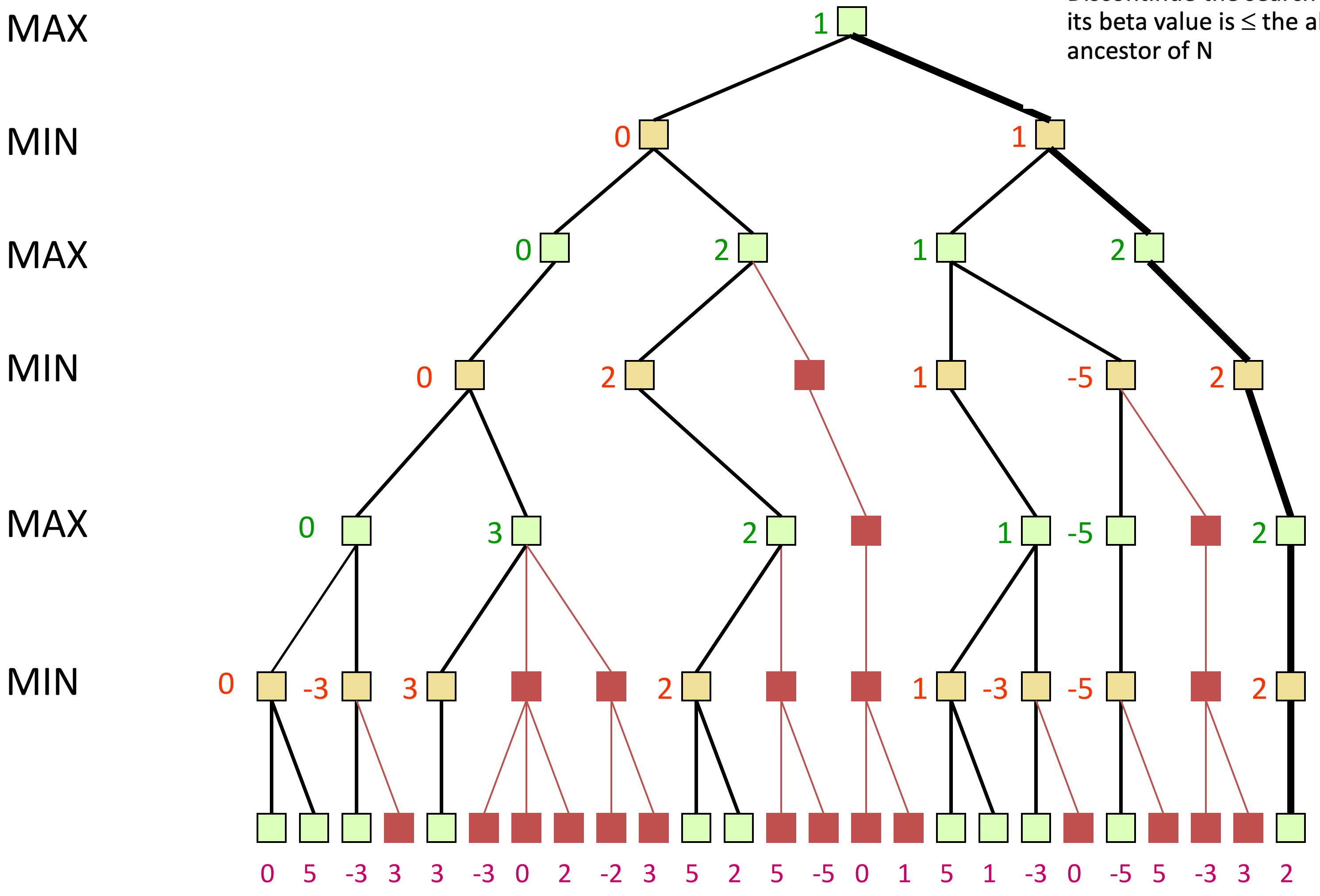
# Example

- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is $\geq$ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is $\leq$ the alpha value of a MAX ancestor of N



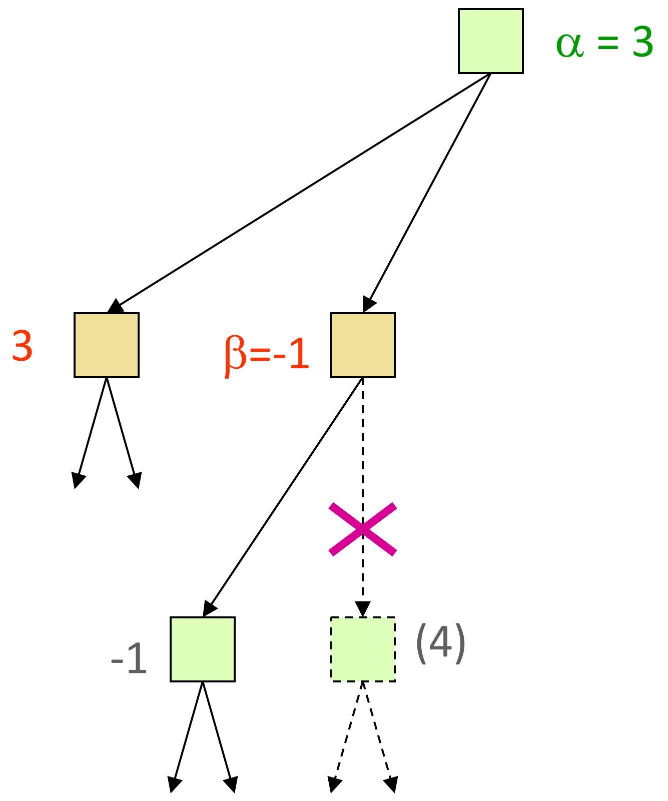
# Example

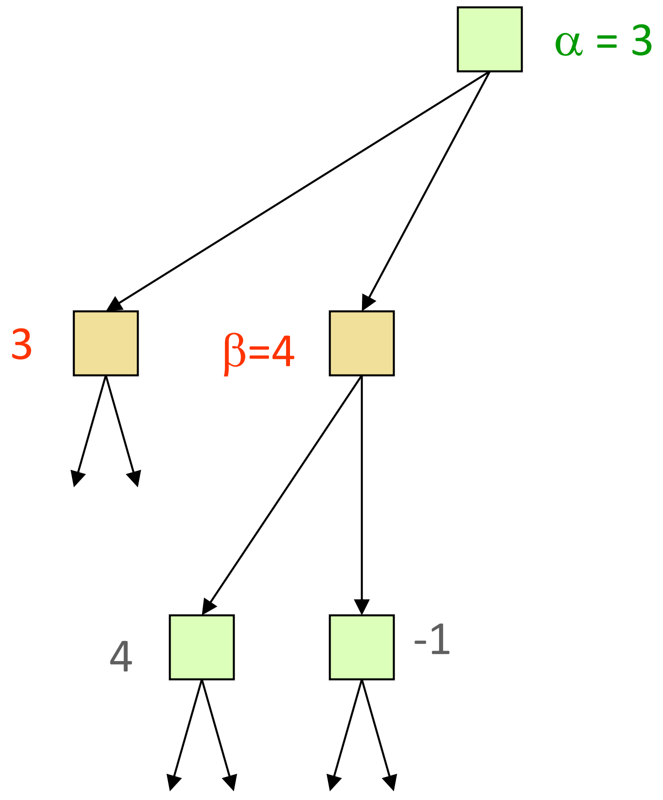
- Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha value is ≥ the beta value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta value is ≤ the alpha value of a MAX ancestor of N



# How much do we gain?

- Consider these two cases:





# How much do we gain?

- Assume a game tree of uniform branching factor b
- Minimax examines $O(b^h)$ nodes, as does alpha-beta in worst-case
- The gain for alpha-beta is maximum when:
  - children of a MAX node are ordered in decreasing backed up values
  - children of a MIN node are ordered in increasing backed up values
- Then alpha-beta examines $O(b^{h/2})$ nodes [Knuth&Moore1975]
- But this requires an oracle
- If nodes are ordered at random, then the average number of nodes examined by alpha-beta is $\sim O(b^{3h/4})$

# Alpha-Beta Implementation

- Alpha-Beta-Decision(S)
  - Return action leading to state $S' \in SUCC(S)$ that maximizes MIN-Value($S'$,$-\infty$,$+\infty$)
- MAX-Value($S$,$\alpha$,$\beta$)

```
– **If** Terminal?(S) return Result(S)
– For all $S' \in SUCC(S)$
–     $\alpha \leftarrow \max(\alpha, \text{MIN-Value}(S',\alpha,\beta))$
–     ??????????????
– Return $\alpha$
```

- MIN-Value($S$,$\alpha$,$\beta$)

```
– **If** Terminal?(S) return Result(S)
– For all $S' \in SUCC(S)$
–     $\beta \leftarrow \min(\beta, \text{MAX-Value}(S',\alpha,\beta))$
–     If $\alpha \geq \beta$, then return $\beta$
– Return $\beta$
```

## Examples & Current Directions

# Checkers: Tinsley vs. Chinook

| | |
|---|---|
| Name: | Marion Tinsley |
| Profession: | Teach mathematics |
| Hobby: | Checkers |
| Record: | Over 42 years loses only 3 games of checkers |

World champion for over 40 years

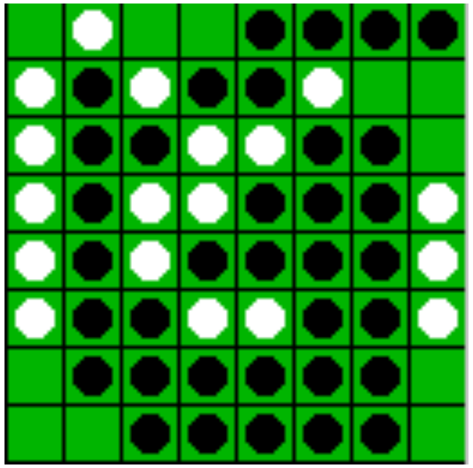
Checkers was solved in April 2007: from the standard starting position, both players can guarantee a draw with perfect play. This took $10^{14}$ calculations over 18 years. Checkers has a search space of size $5 \times 10^{20}$

Mr. Tinsley suffered his 4th and 5th losses against Chinook (1990)

# Chinook

- First computer to become official world champion of Checkers!

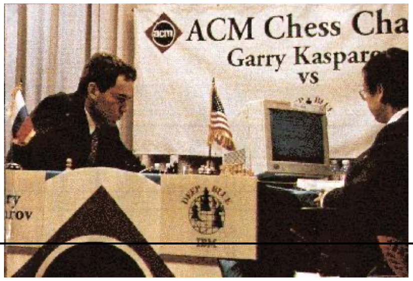
## Othello (Reversi): *Murakami vs. Logistello*



Takeshi Murakami
World Othello Champion

1997: The Logistello software crushed Murakami by 6 games to 0

# Chess: Kasparov vs. Deep Blue



| Kasparov | | Deep Blue |
|---|---|---|
| 5’10” | Height | 6’ 5” |
| 176 lbs | Weight | 2,400 lbs |
| 34 years | Age | 4 years |
| 50 billion neurons | Computers | 32 RISC processors + 256 VLSI chess engines |
| 2 pos/sec | Speed | 200,000,000 pos/sec |
| Extensive | Knowledge | Primitive |
| Electrical/chemical | Power Source | Electrical |
| Enormous | Ego | None |

1997: Deep Blue wins by 3 wins, 1 loss, and 2 draws

Jonathan Schaeffer

# Chess: Kasparov vs. Deep Junior

Deep Junior

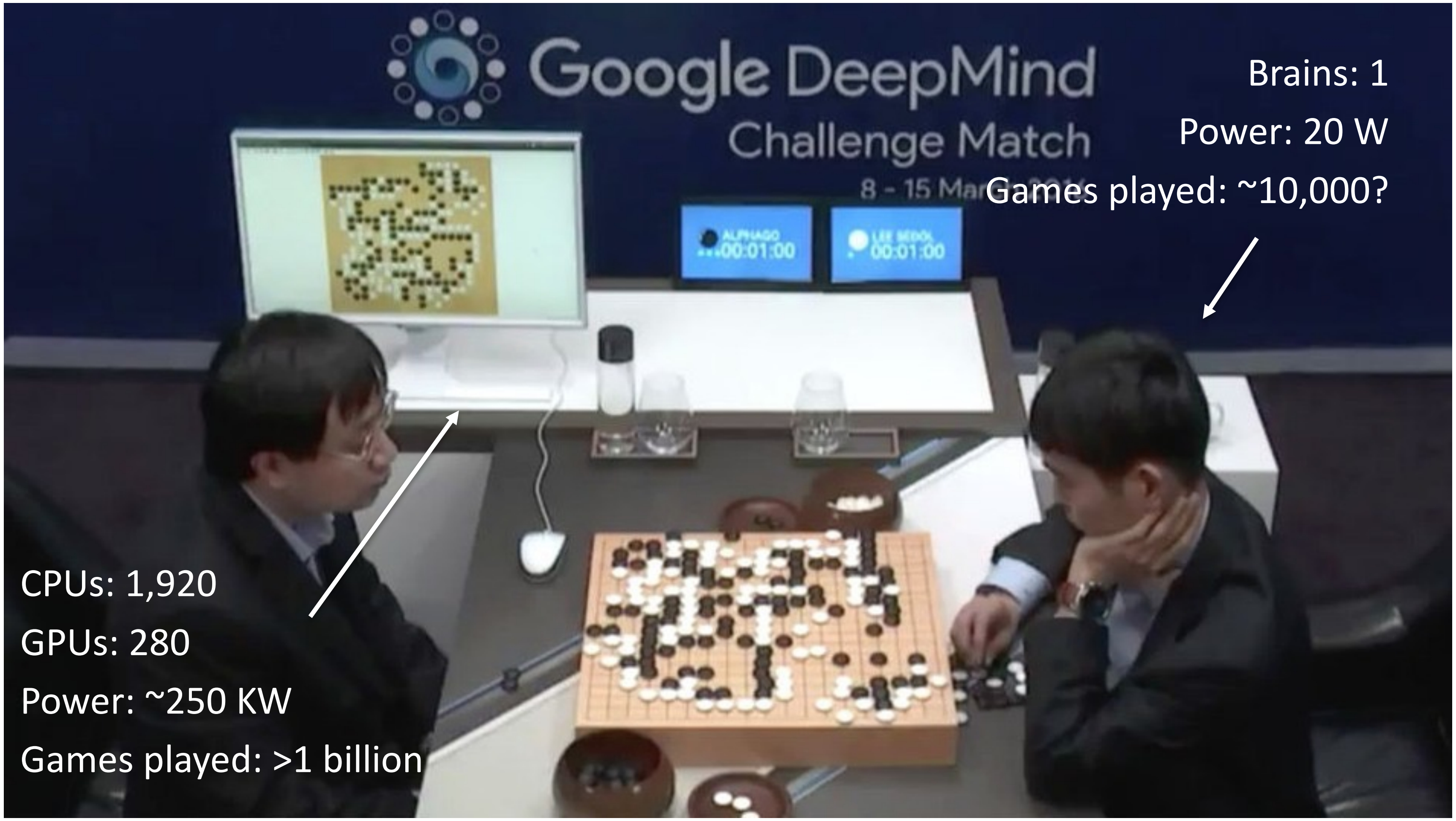
8 CPU, 8 GB RAM, Win 2000

2,000,000 pos/sec

Available at $100

August 2, 2003: Match ends in a 3/3 tie!

# Milestone: Go (2016)



Google's AlphaGo beats Lee Sodol (9-dan player), March 2016

# Secrets

- Alpha-beta pruning + iterative deepening + transposition tables (hash tables containing minimax values) + huge databases + Monte Carlo Tree Search + Deep Learning + … ...

- E.g. Chinook searched all checkers configurations with 8 pieces or less and created an endgame database of 444 billion board configurations

- The methods are general, but their implementation is dramatically improved by many specifically tuned-up enhancements (e.g., the evaluation functions)

## Perspective on Games: Con and Pro

Chess is the Drosophila of artificial intelligence. However, computer chess has developed much as genetics might have if the geneticists had concentrated their efforts starting in 1910 on breeding racing Drosophila. We would have some science, but mainly we would have very fast fruit flies.

John McCarthy

Saying Deep Blue doesn't really think about chess is like saying an airplane doesn't really fly because it doesn't flap its wings.

Drew McDermott

# Next time

- **Uncertainty and Probability**