Adversarial Search and Game Playing

(Respect your opponent to make good decisions)

Russell and Norvig:

Chapter 6 (4th edition)

Slides adopted from Jean-Claude Latombe at Stanford University (used with permission)

Games

Games like Chess or Go are compact settings that mimic the uncertainty of interacting with the natural world.

For centuries humans have used them to exert their intelligence.

Recently, there has been great success in building game programs that challenge human supremacy.

Specific Setting

- two-player
- turn-taking:
  at each step only one player chooses a move
- deterministic
- fully observable
- zero-sum:
  the rewards of all players add up to zero
  = if one player wins, the other player looses
- time-constrained game:
  only a limited amount of time to make a move,
  typically not enough to "solve" the game

Search Problem Formulation

Initial state: Game setup at start
Player(s): Which player moves in state
Action(s): Legal moves in a state
Result(s,a): Transition model
Terminal-Test(s): True when game over
Evaluate(s, p): Estimate of how good s is for player p

Simplification

For two-player zero-sum games:

- we name the players **MAX** and **MIN**
- **MAX** wants to maximize his score
- **MIN** wants to minimize **MAX**'s score

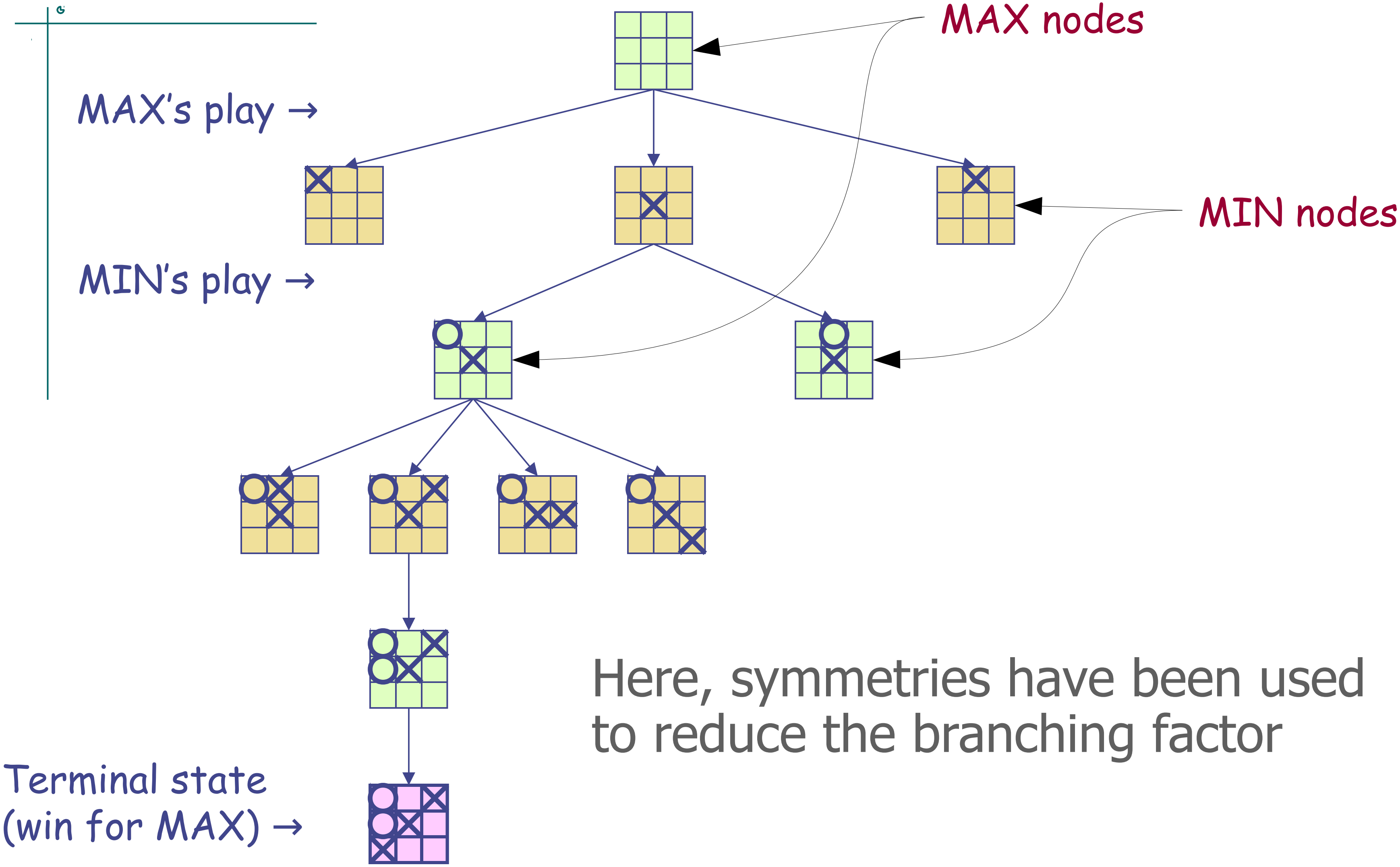
→ we only consider the score of **MAX**

Time Limit

At each turn, the choice of which action to perform must be made within a **specified time limit**

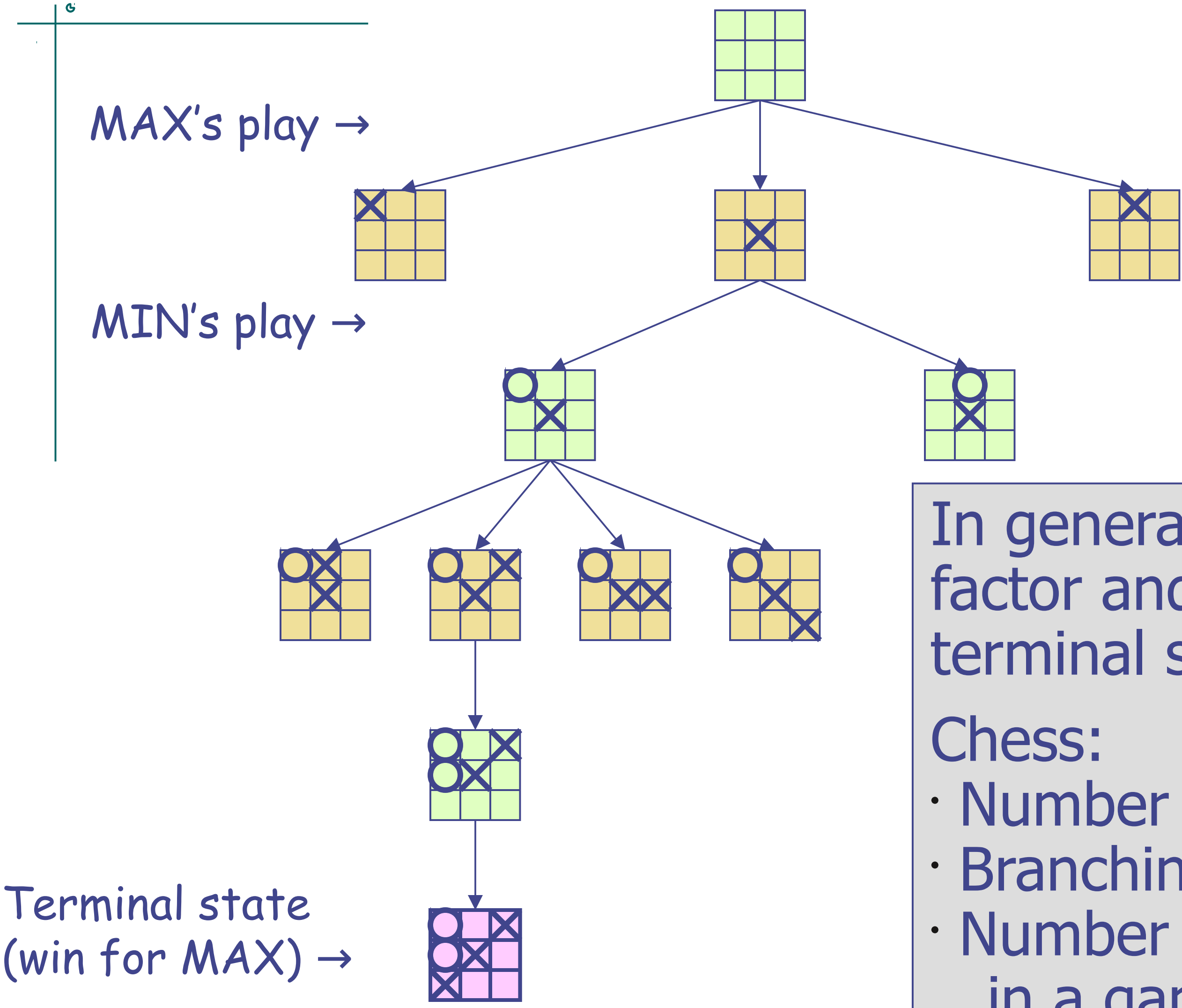
The state space is enormous:
only **a tiny fraction** of this space
can be explored within the time limit

Game Tree



Here, symmetries have been used to reduce the branching factor

Game Tree



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

Chess:

- Number of states: ~10^{40}
- Branching factor: ~35
- Number of total moves in a game: ~100

Minimax Algorithm

- Expand the game tree uniformly from the current state (where it is MAX's turn to play) to depth h ("horizon")
- Compute evaluation function at every leaf
- Back-up the values from the leaves to the root:
  - MAX node → maximum evaluation of its successors
  - MIN node → minimum evaluation of its successors (assume the worst from MIN)
- Select the move toward a MIN node that has the largest backed-up value

Evaluation Function

- Function **e**: State -> Number
- e(s) estimates how favorable s is for MAX

  e(s) > 0 means that s is favorable to MAX
  (the larger the better)

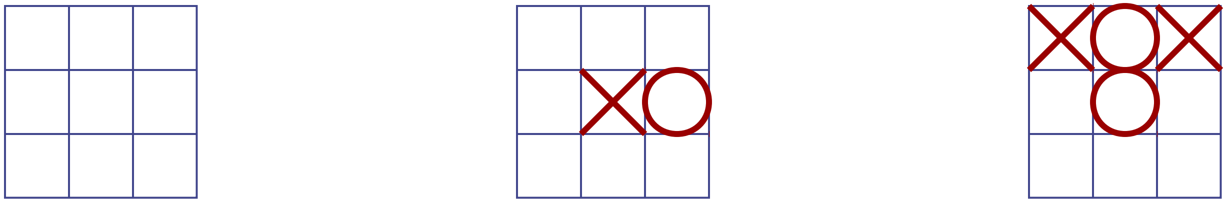
  e(s) < 0 means that s is favorable to MIN

  e(s) = 0 means that s is neutral

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

Creating an Evaluation Function

Usually a weighted sum of “features”:

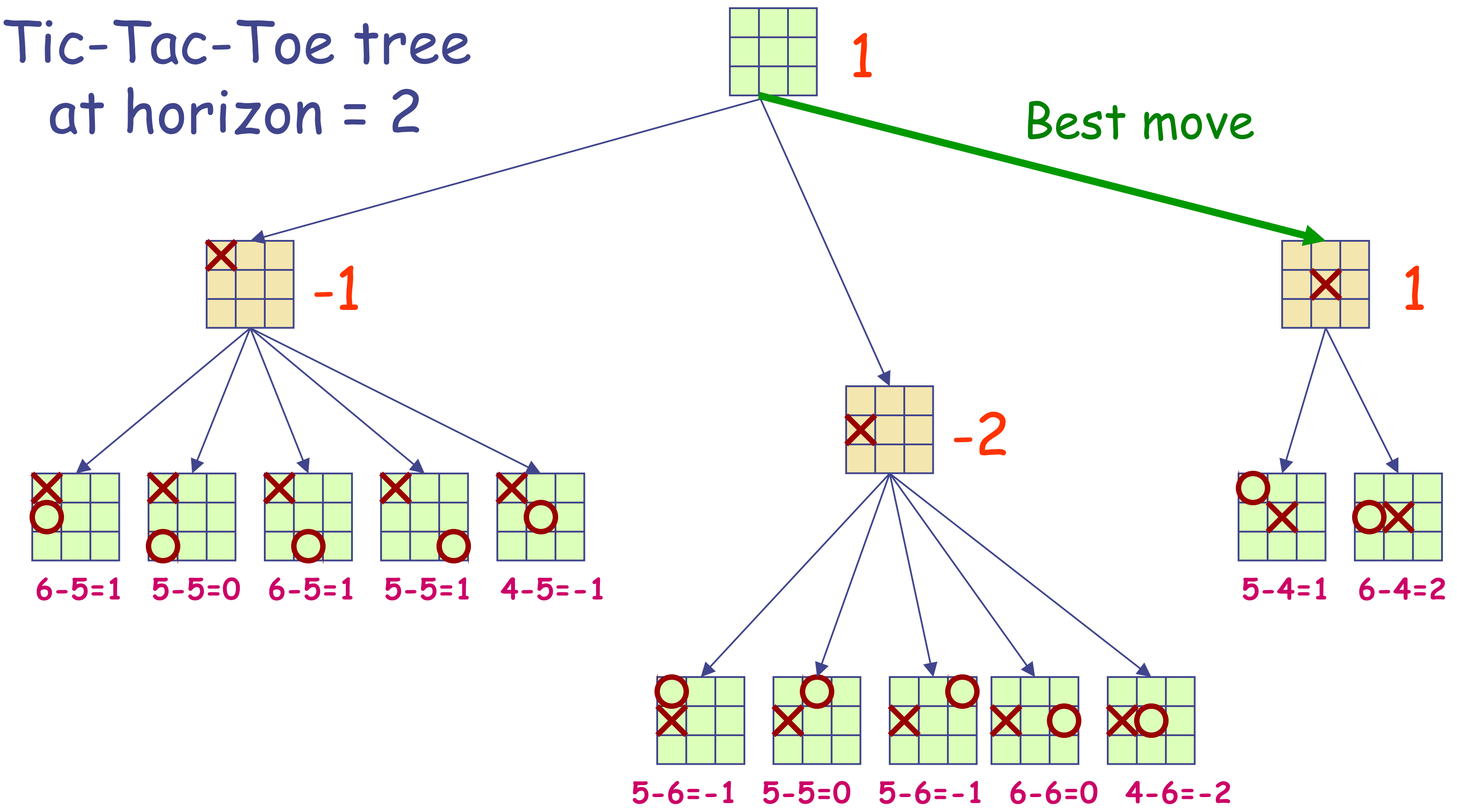
$$e(s) = \sum_{i=1}^{n} w_i * f_i(s)$$

Features may include:

- Number of pieces of each type
- Number of possible moves
- Number of squares controlled

Backing up Values

Tic-Tac-Toe tree
at horizon = 2



Why using backed-up values?

At each non-leaf node N, the backed-up value is the value of the **best state that MAX can reach** at depth h if MIN plays well (by the same criterion as MAX applies to itself)

If e is to be trusted in the first place, then the backed-up value is a better estimate of how favorable STATE(N) is than e(STATE(N))

Game Playing (for MAX)

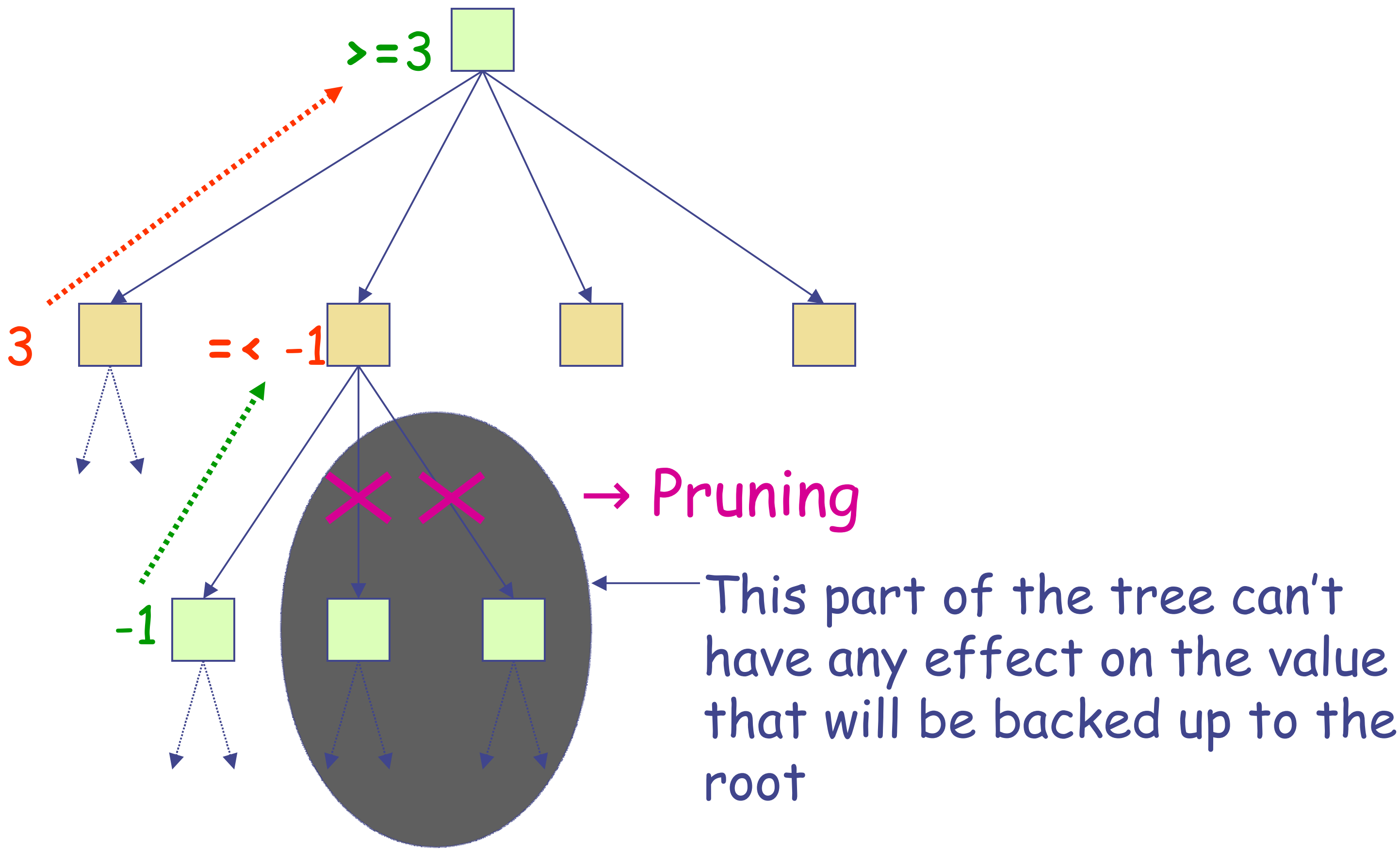
- Repeat until a terminal state is reached
  - Select move using Minimax
  - Execute move
  - Observe MIN's move

Note that at each cycle the large game tree built to horizon h is used to select only one move

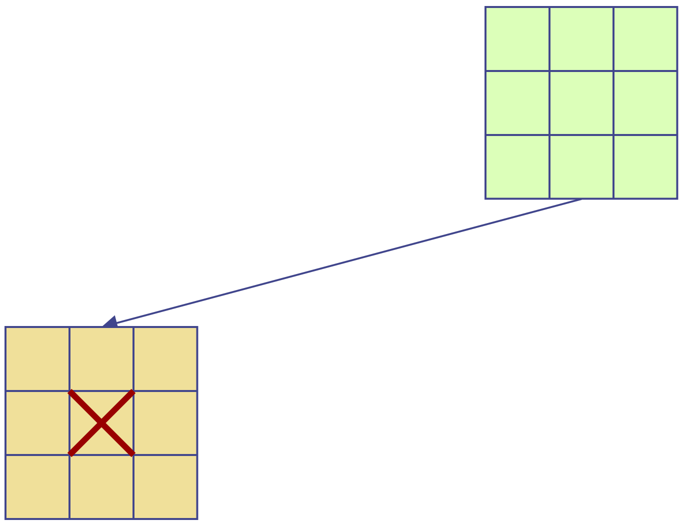
All is repeated again at the next cycle (a sub-tree of depth h-2 can be re-used)

Can we do better?

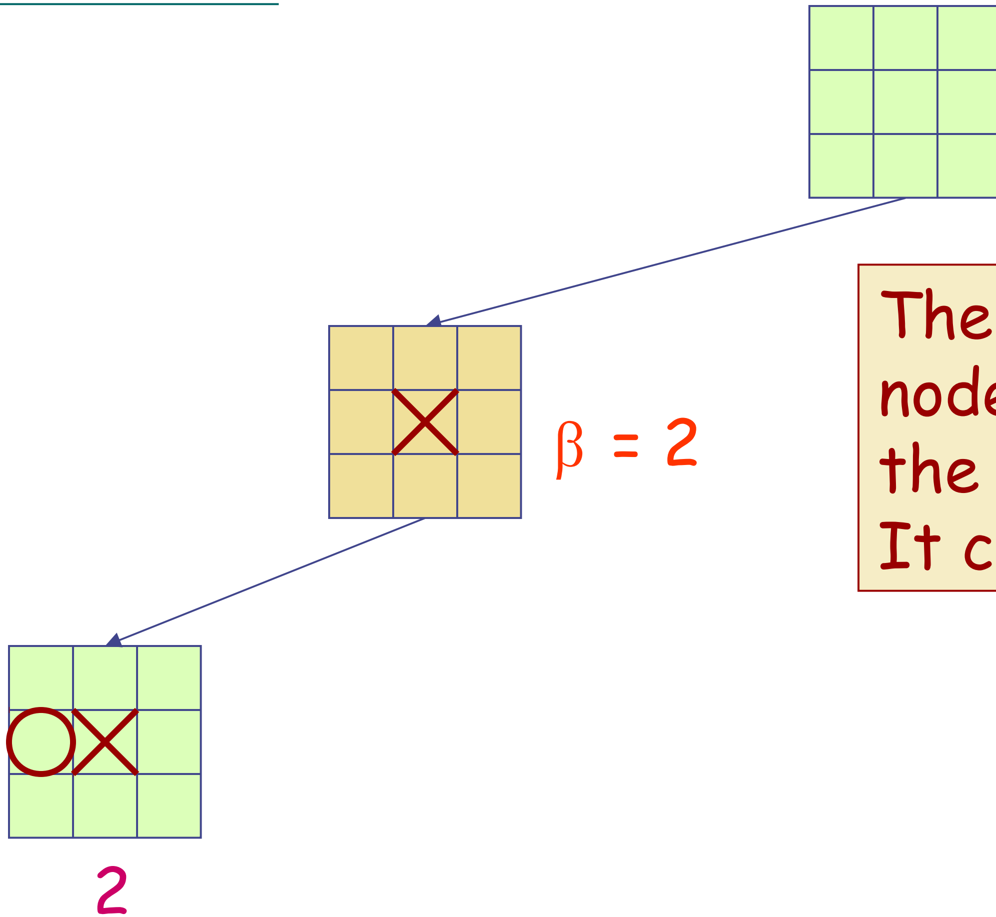
Yes ! Much better !



Example

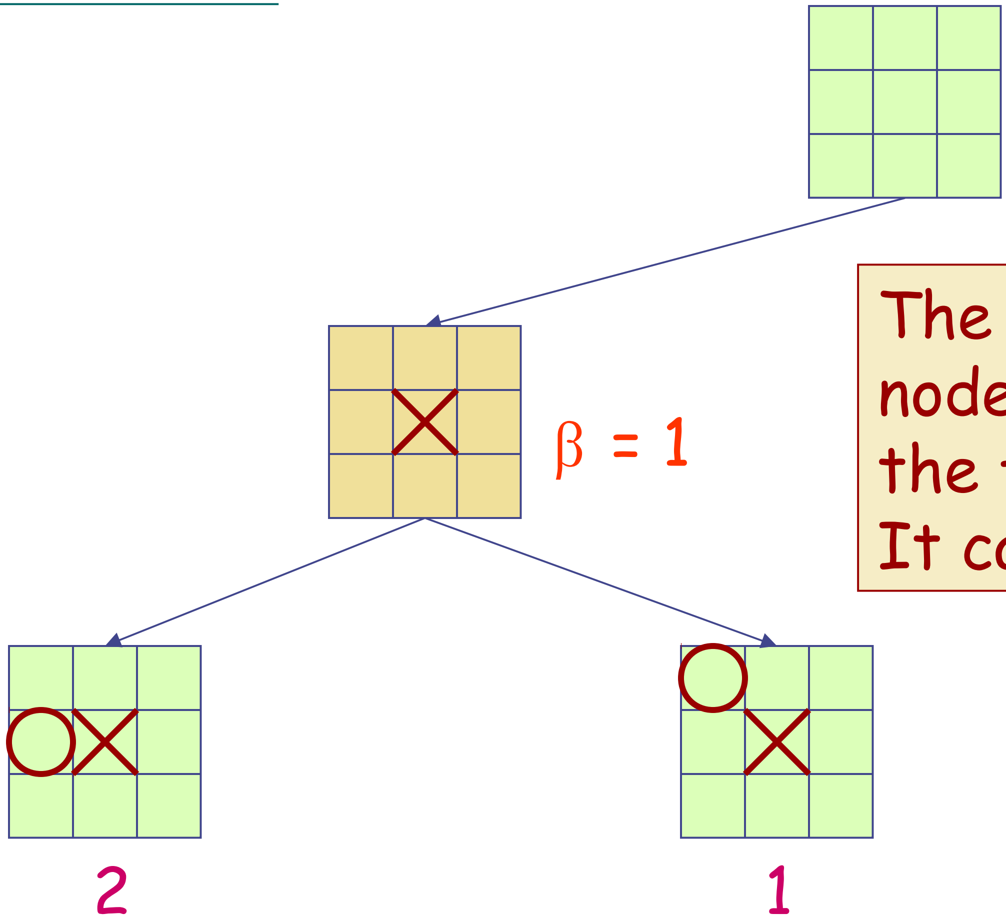


Example



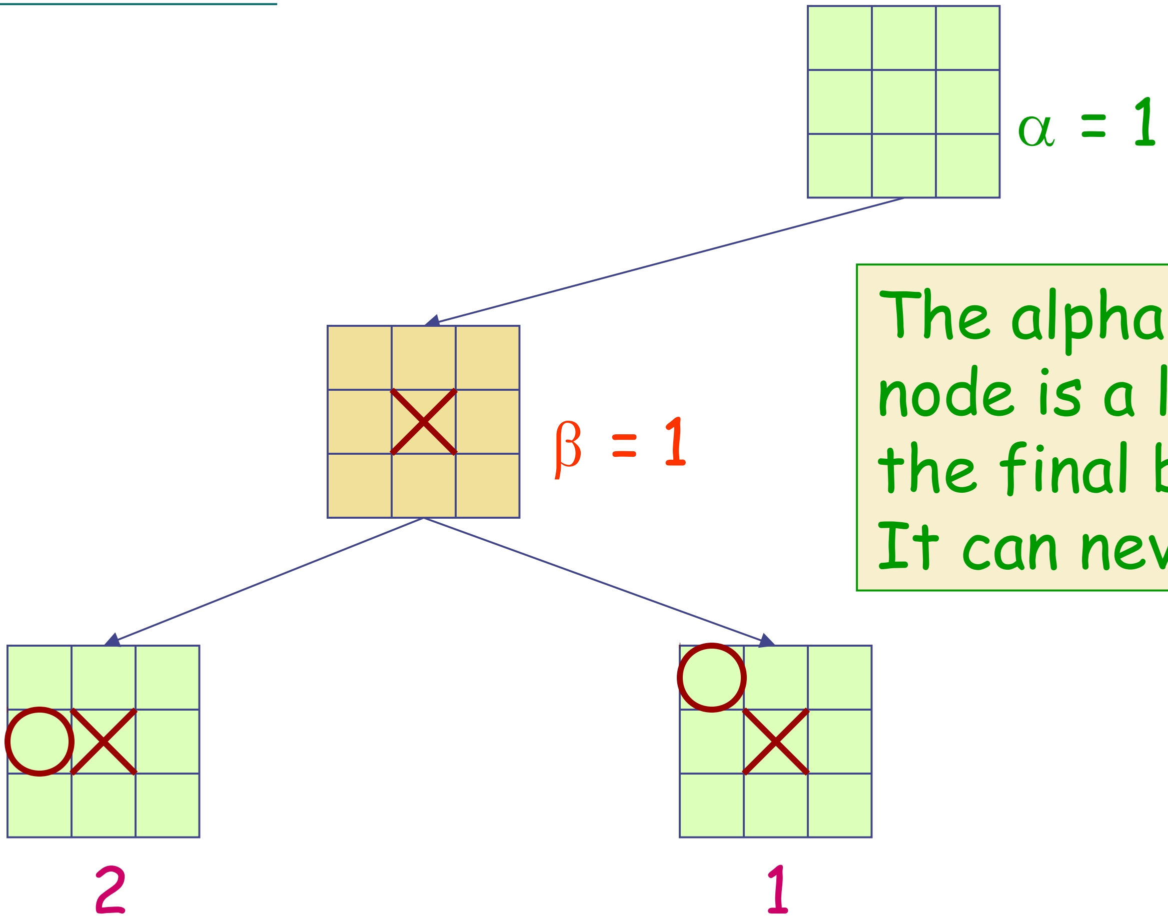
The beta value of a MIN node is an upper bound on the final backed-up value. It can never increase

Example



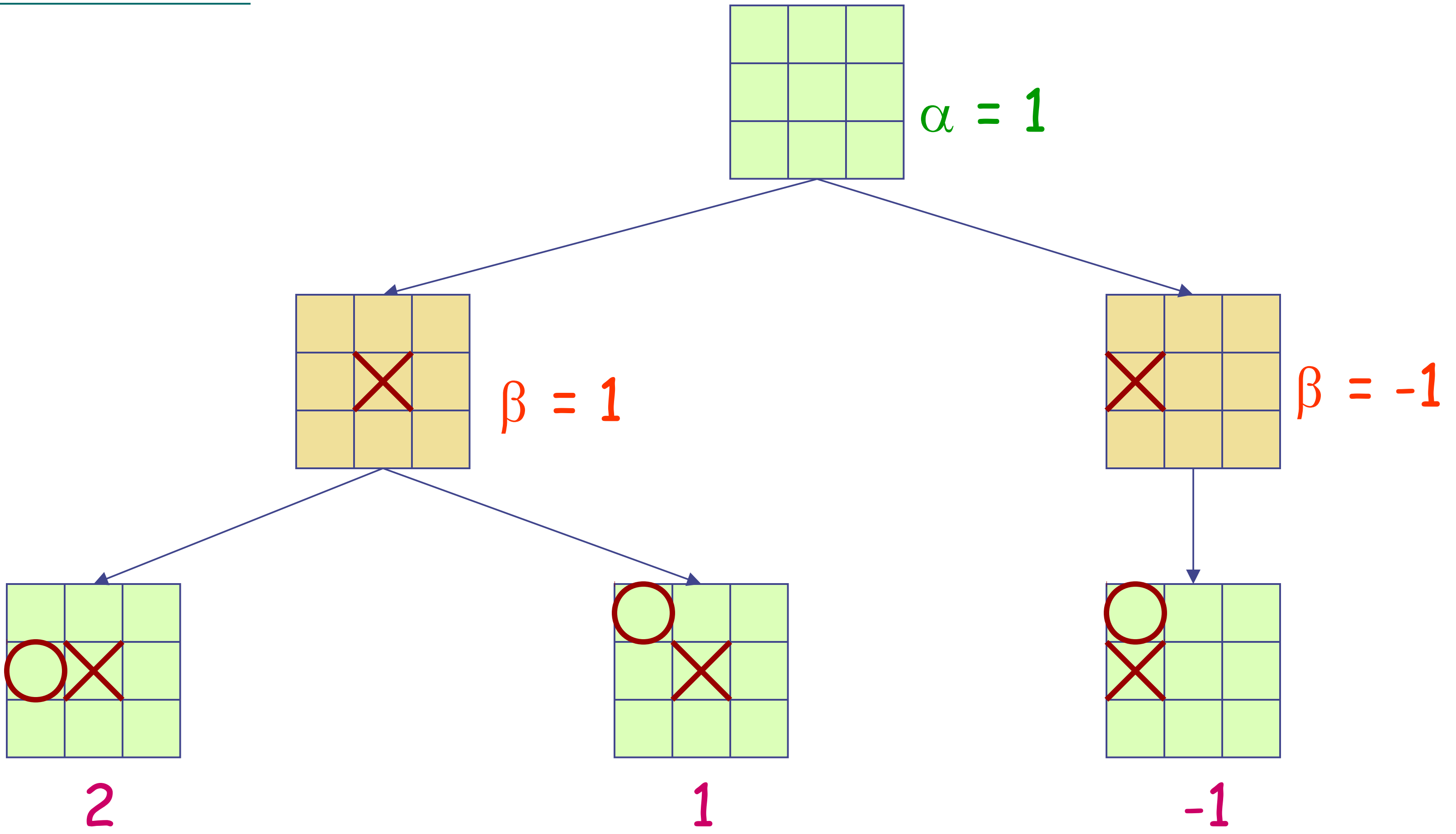
The beta value of a MIN node is an upper bound on the final backed-up value. It can never increase

Example

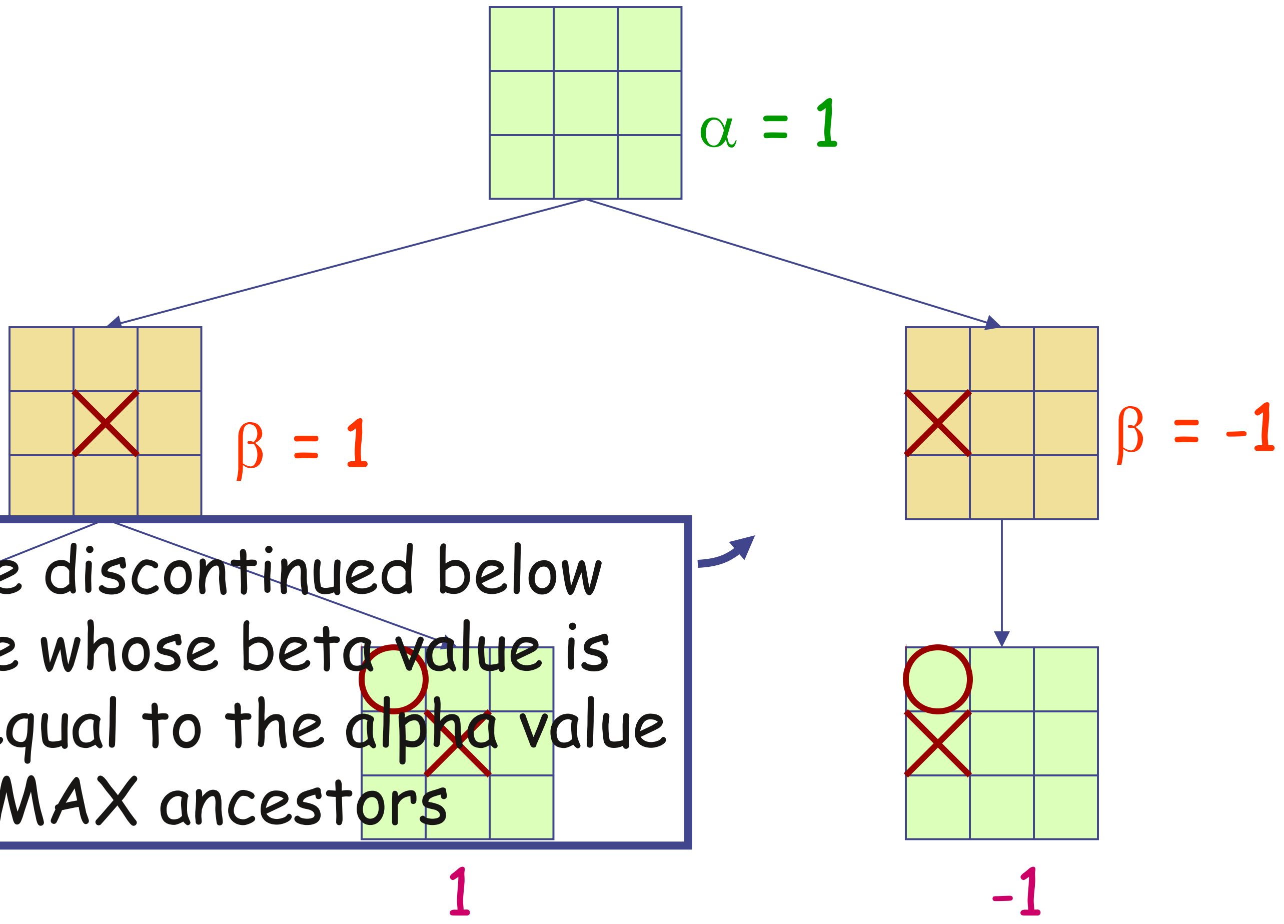


The alpha value of a MAX node is a lower bound on the final backed-up value. It can never decrease

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

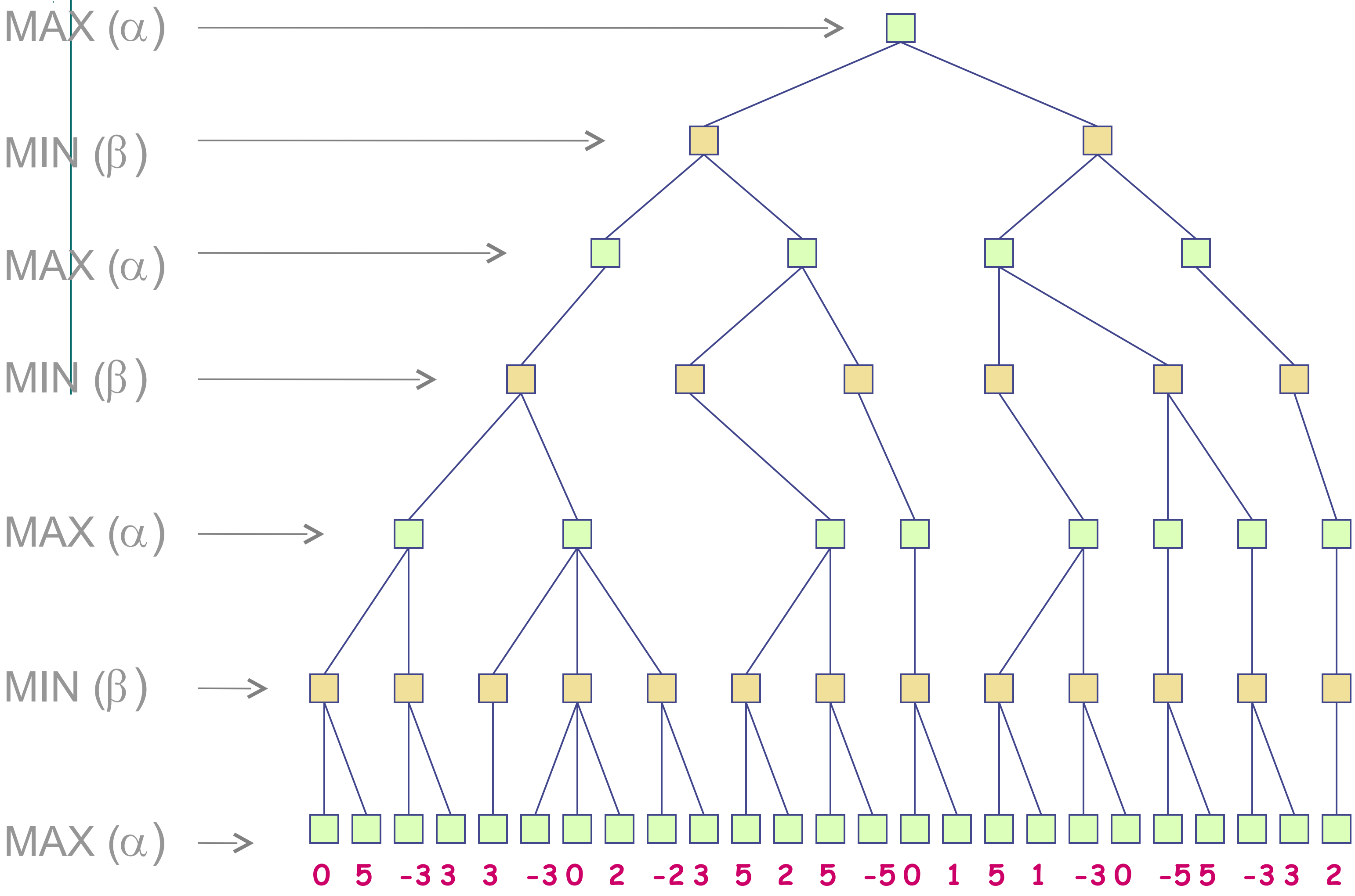
Alpha-Beta Algorithm

Update the alpha/beta value of the parent of a node N when the search below N has been completed or discontinued

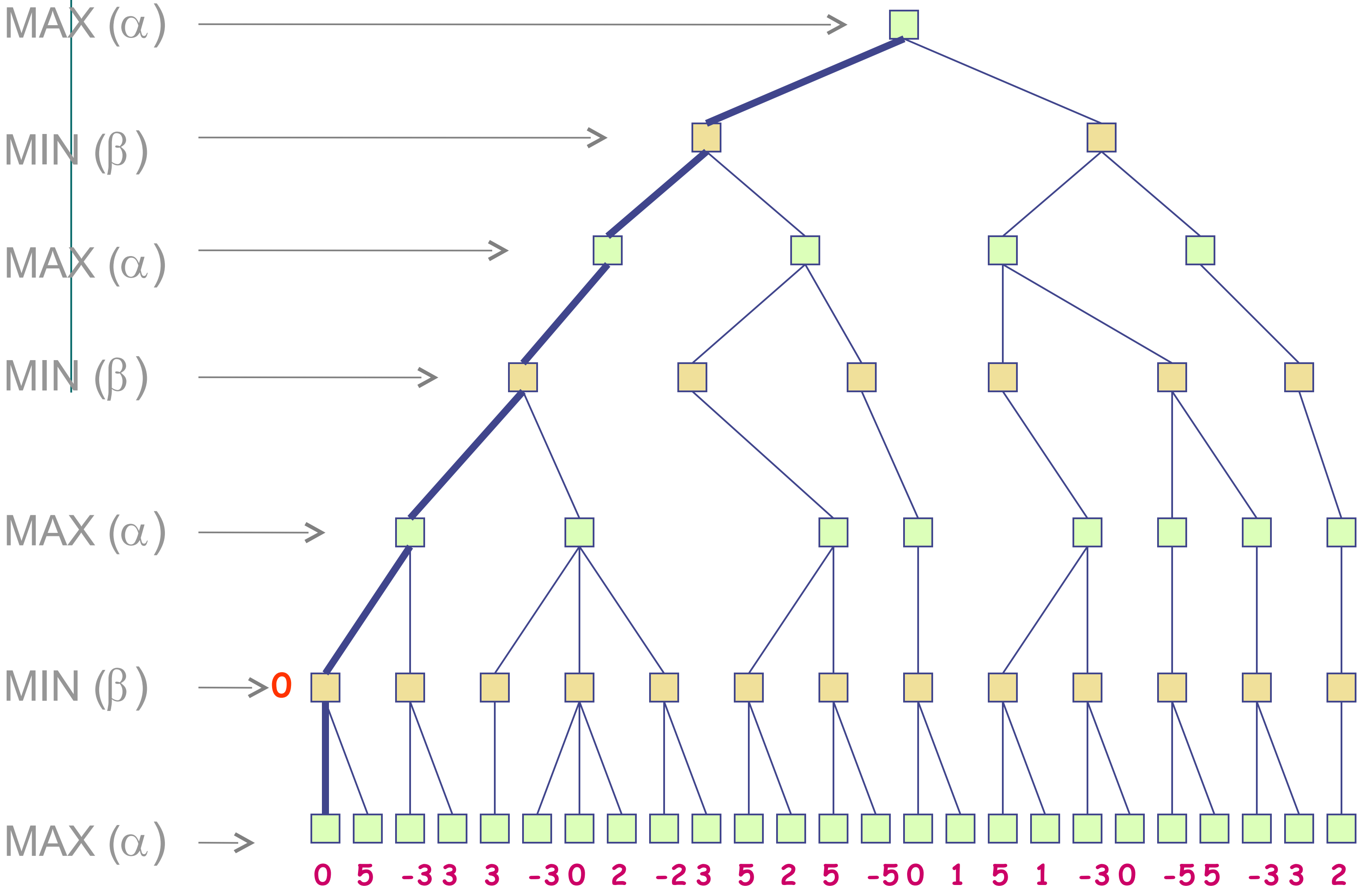
- Discontinue the search below a **MAX** node N if its alpha value is >= the beta value of a MIN ancestor of N

- Discontinue the search below a **MIN** node N if its beta value is <= the alpha value of a MAX ancestor of N

Example

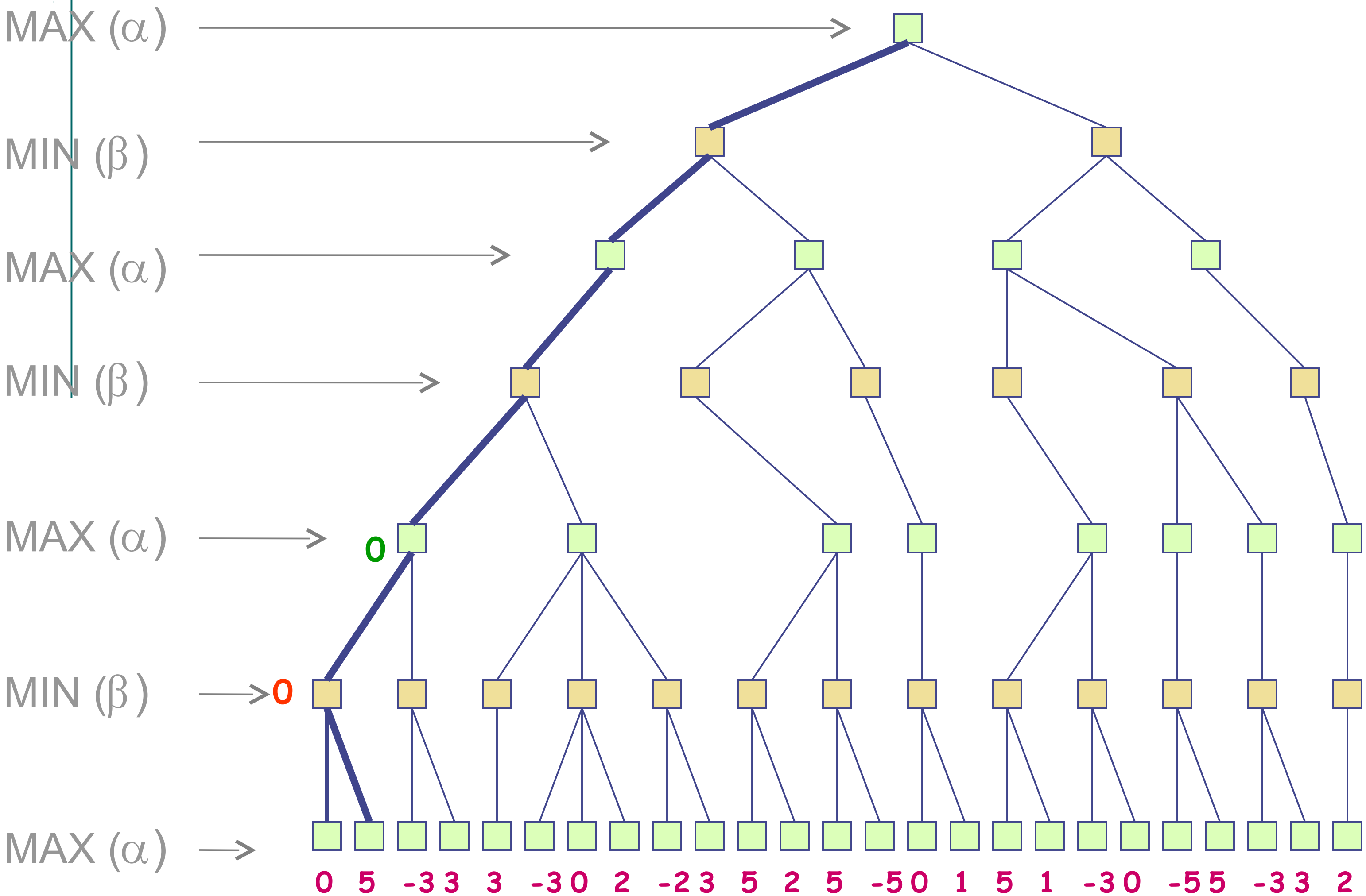

MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example

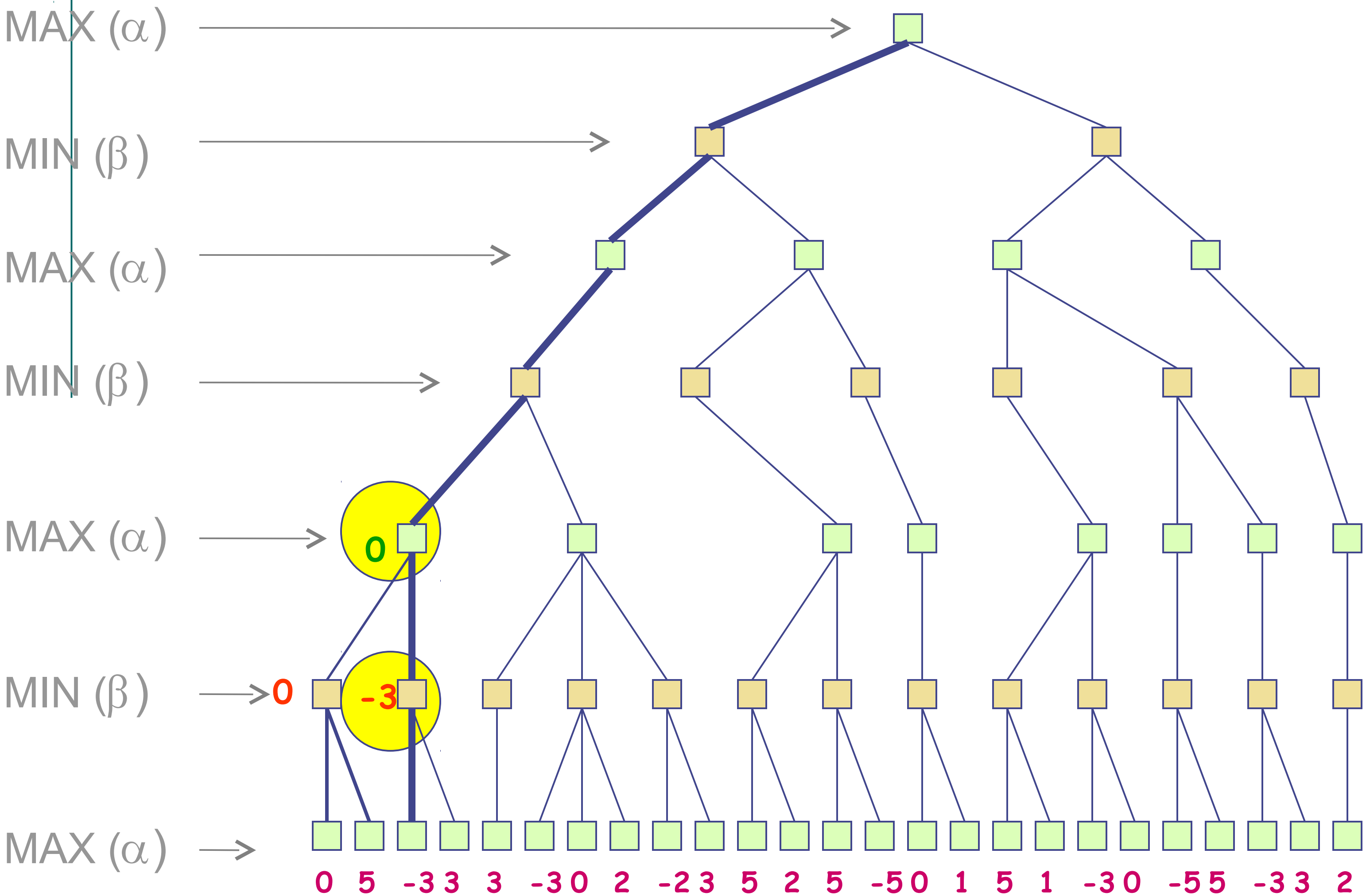


MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
0
MAX (α)
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example

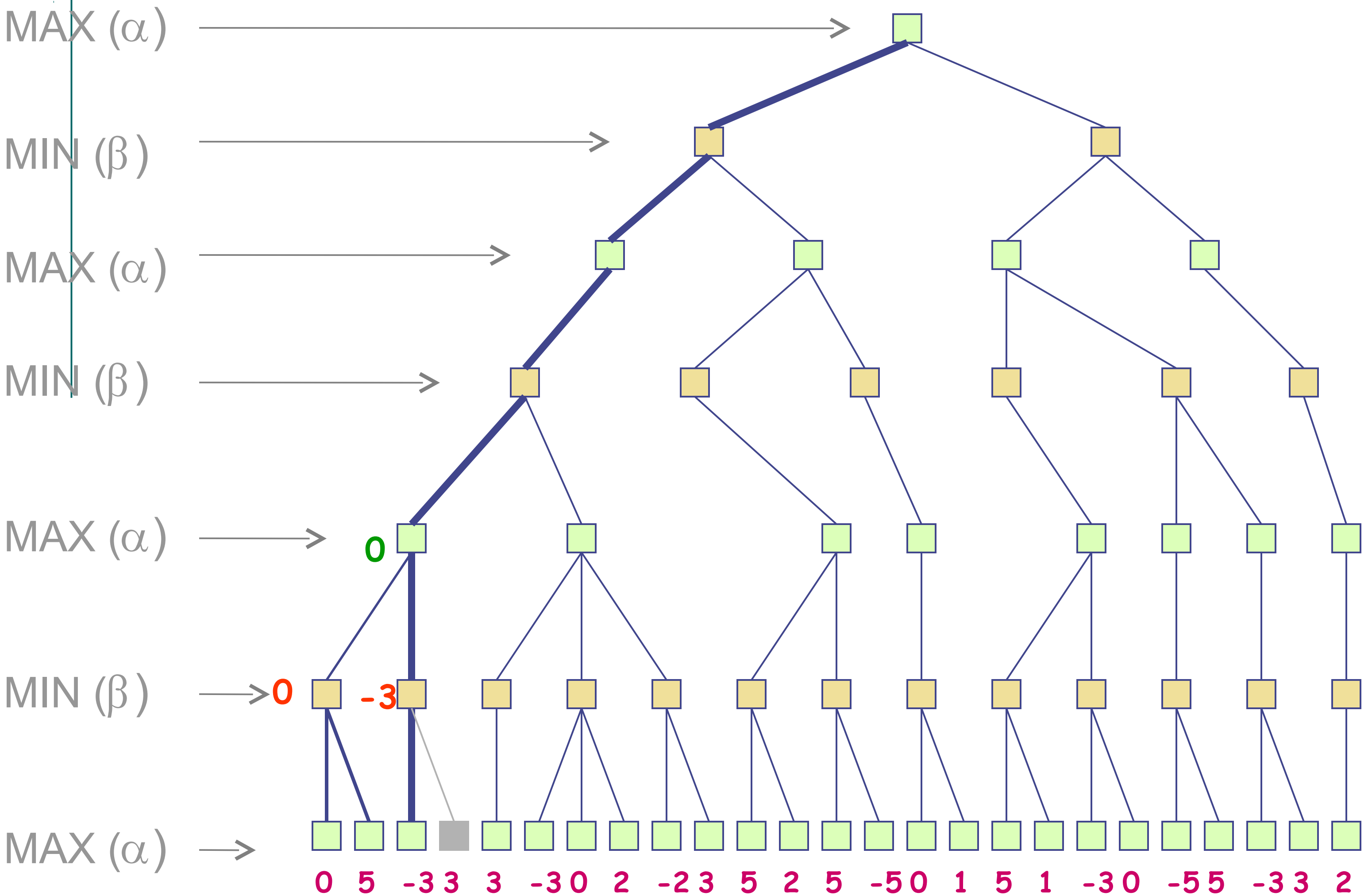

MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



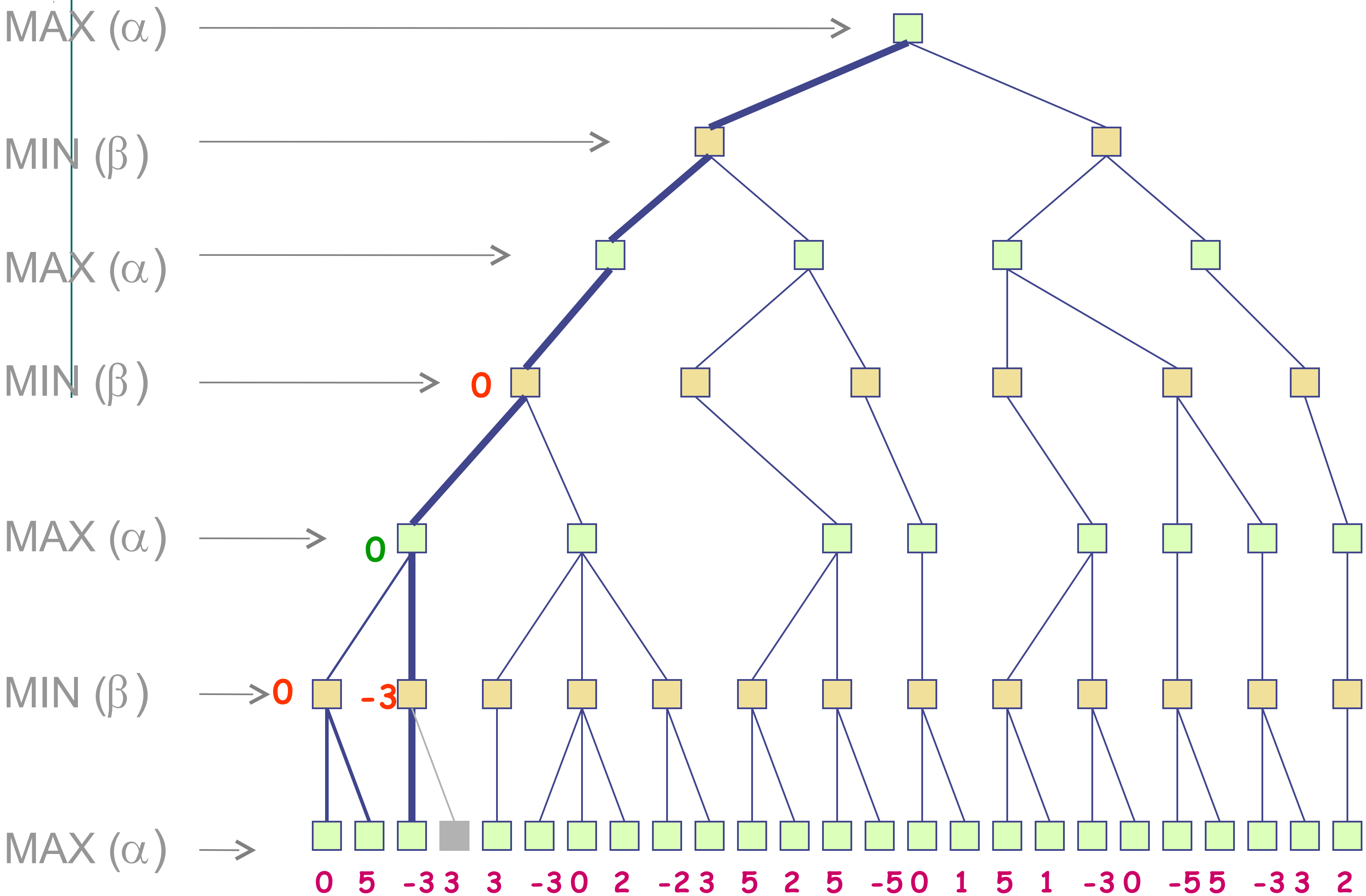
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



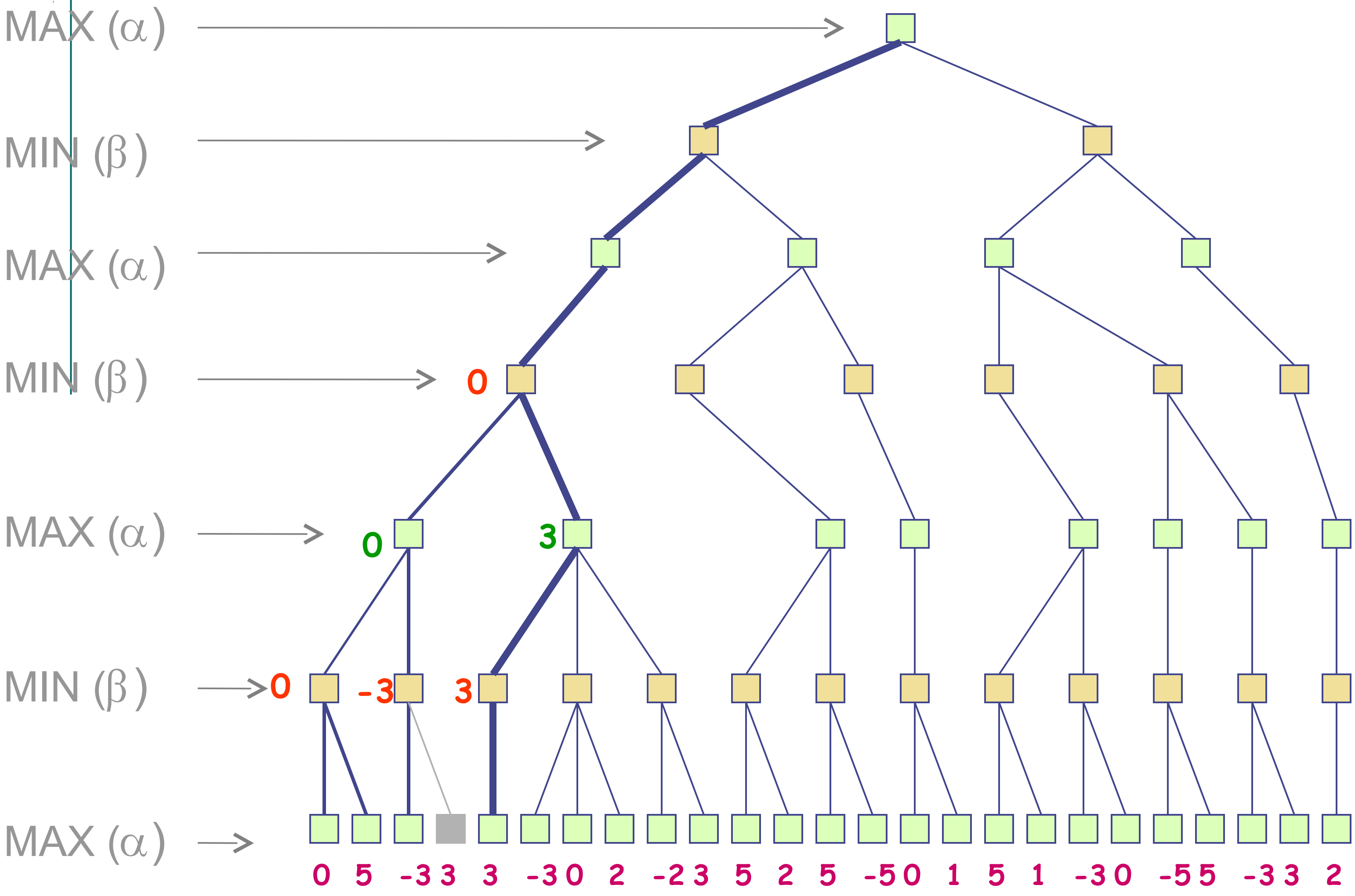
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example

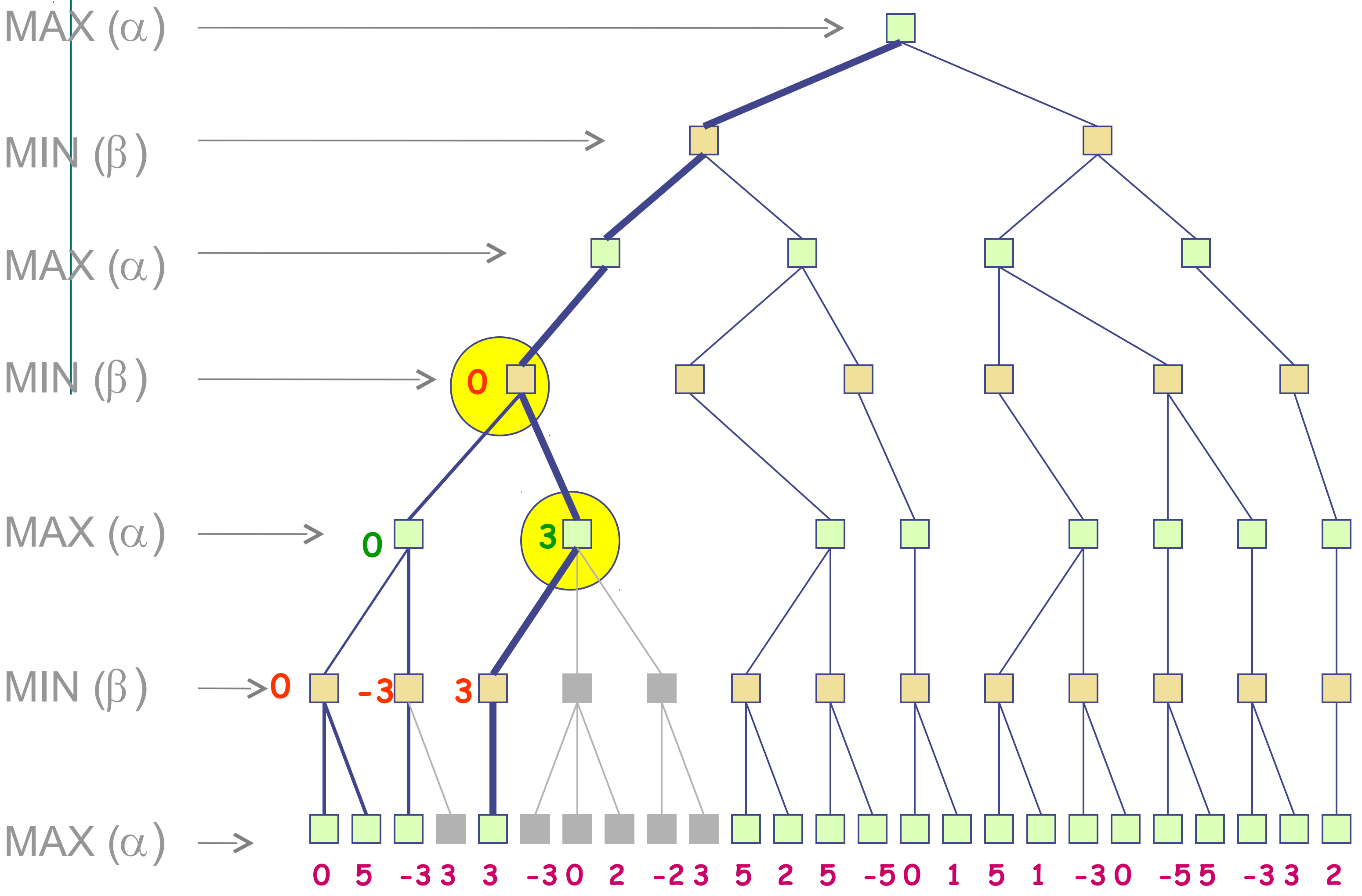


MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example

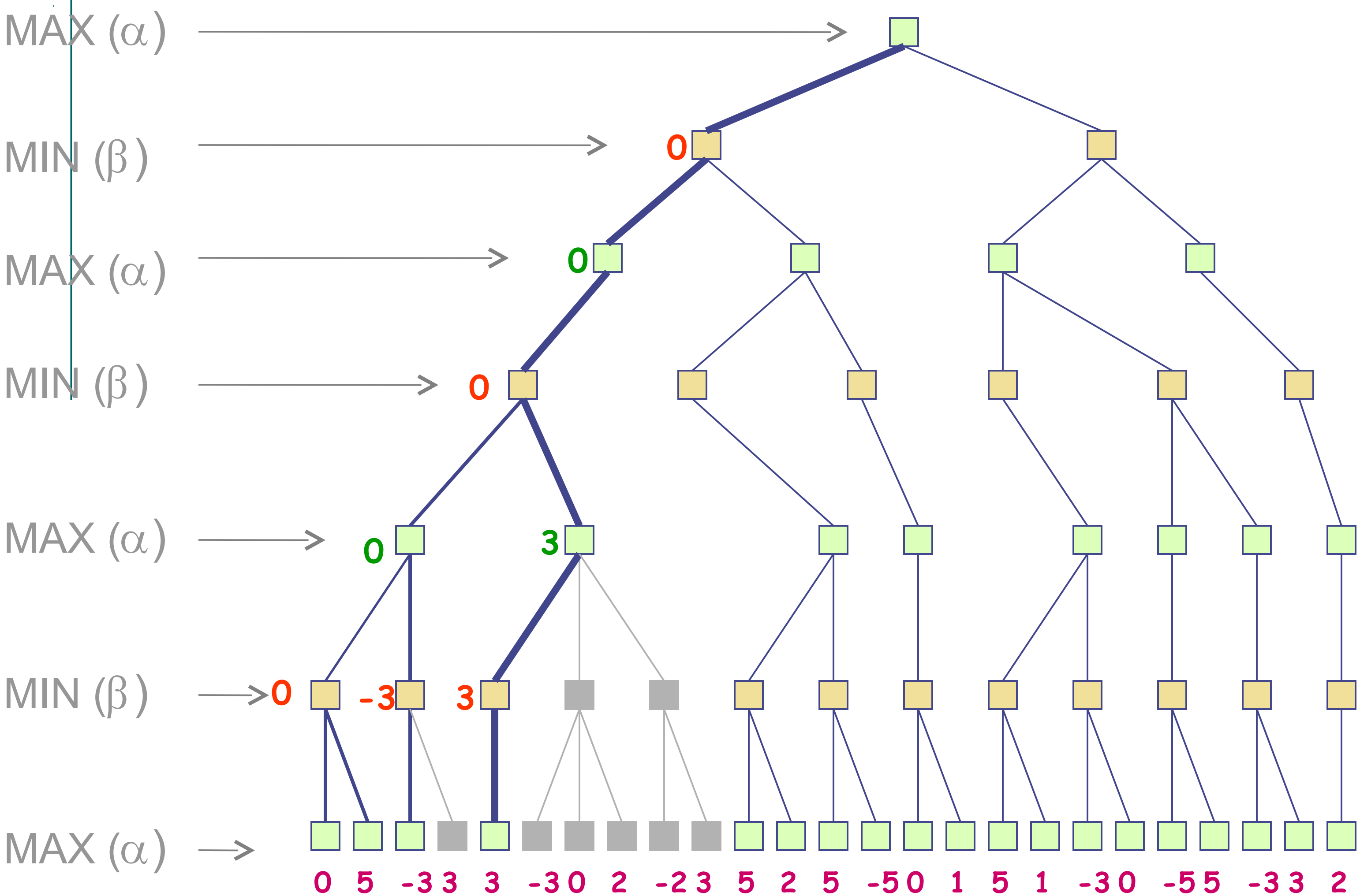

MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
3
0
-3
3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example

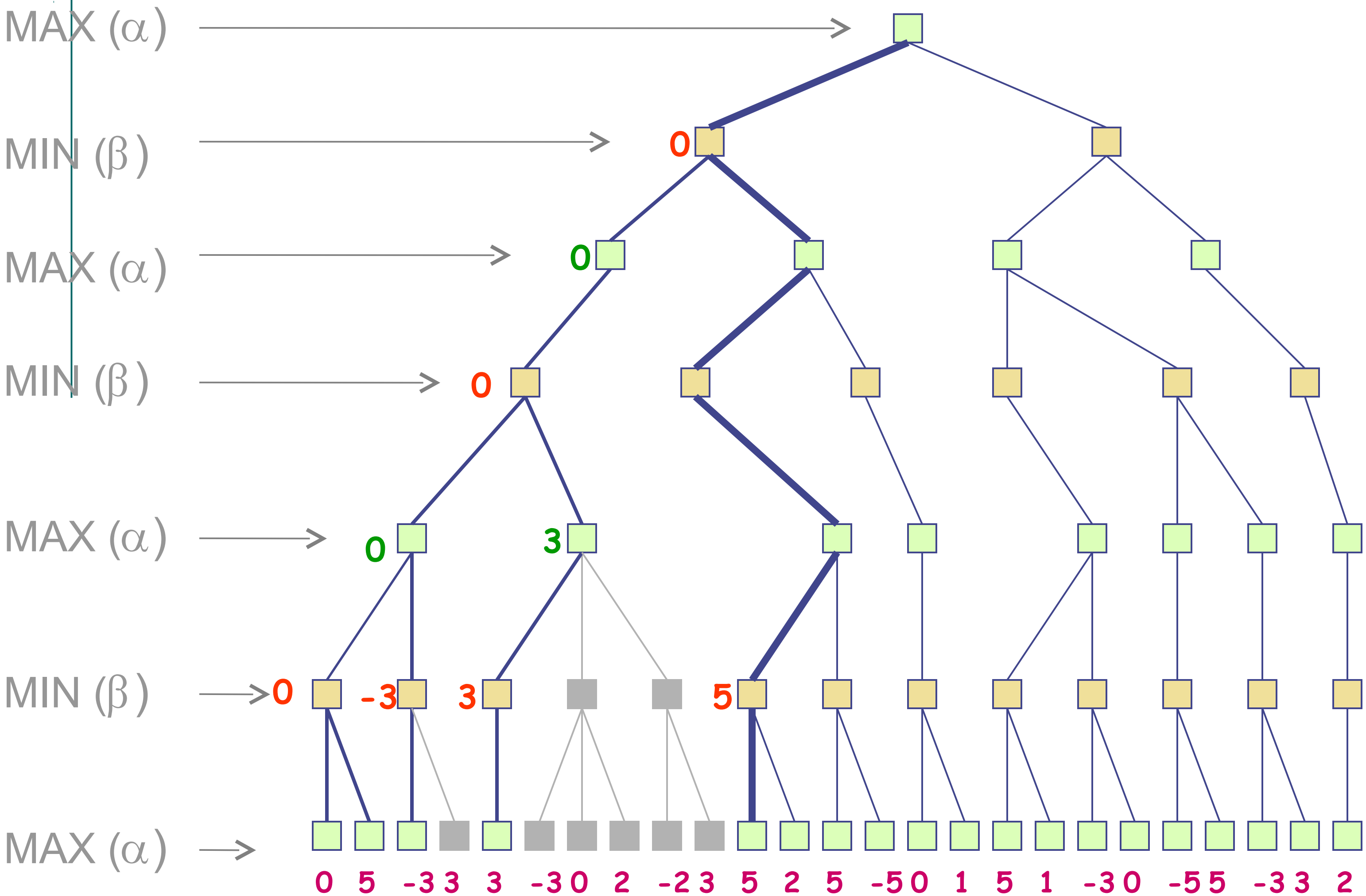


MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
3
0
0
-3
3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example

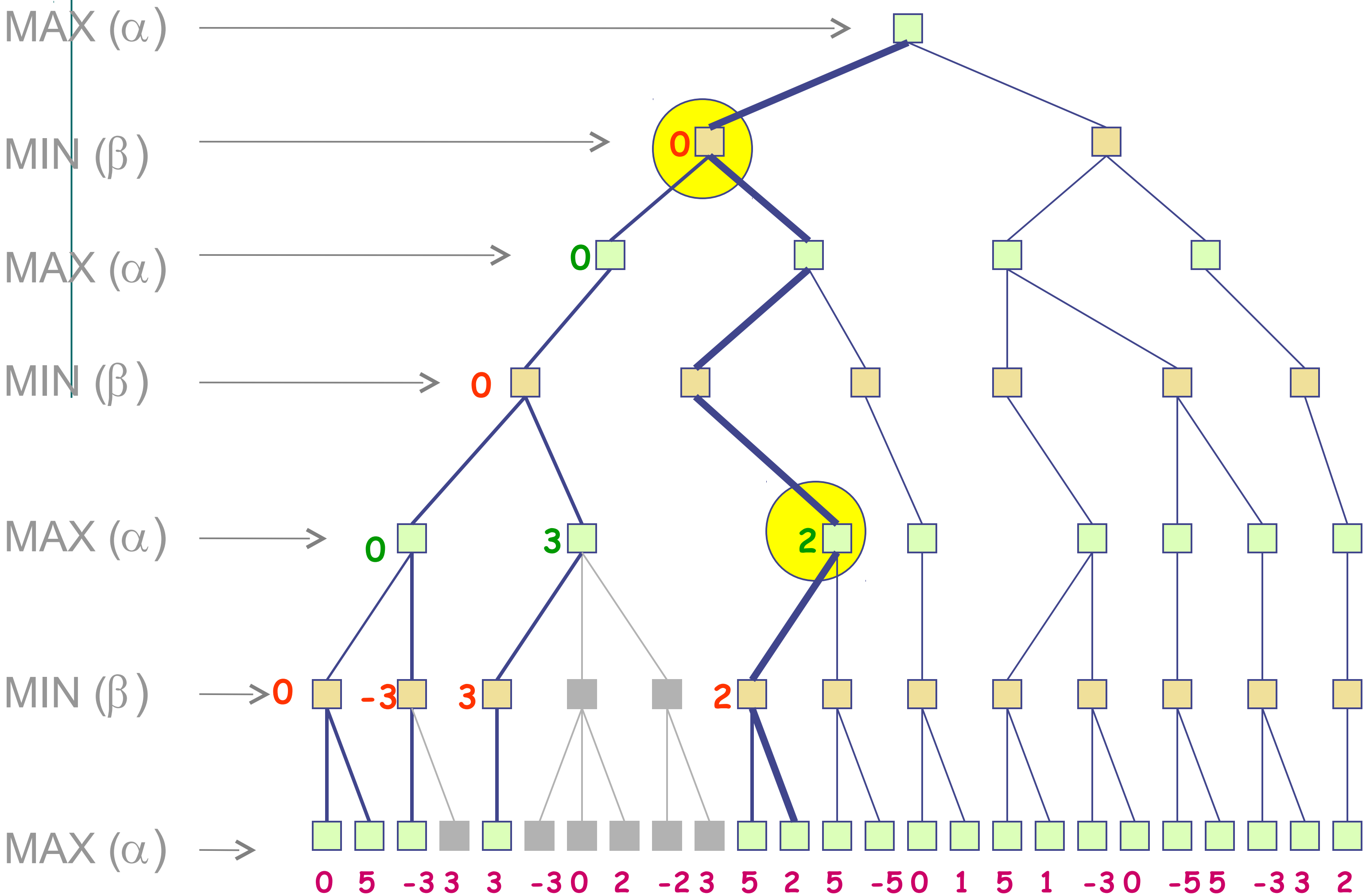

MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0
0
3
0
-3
3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



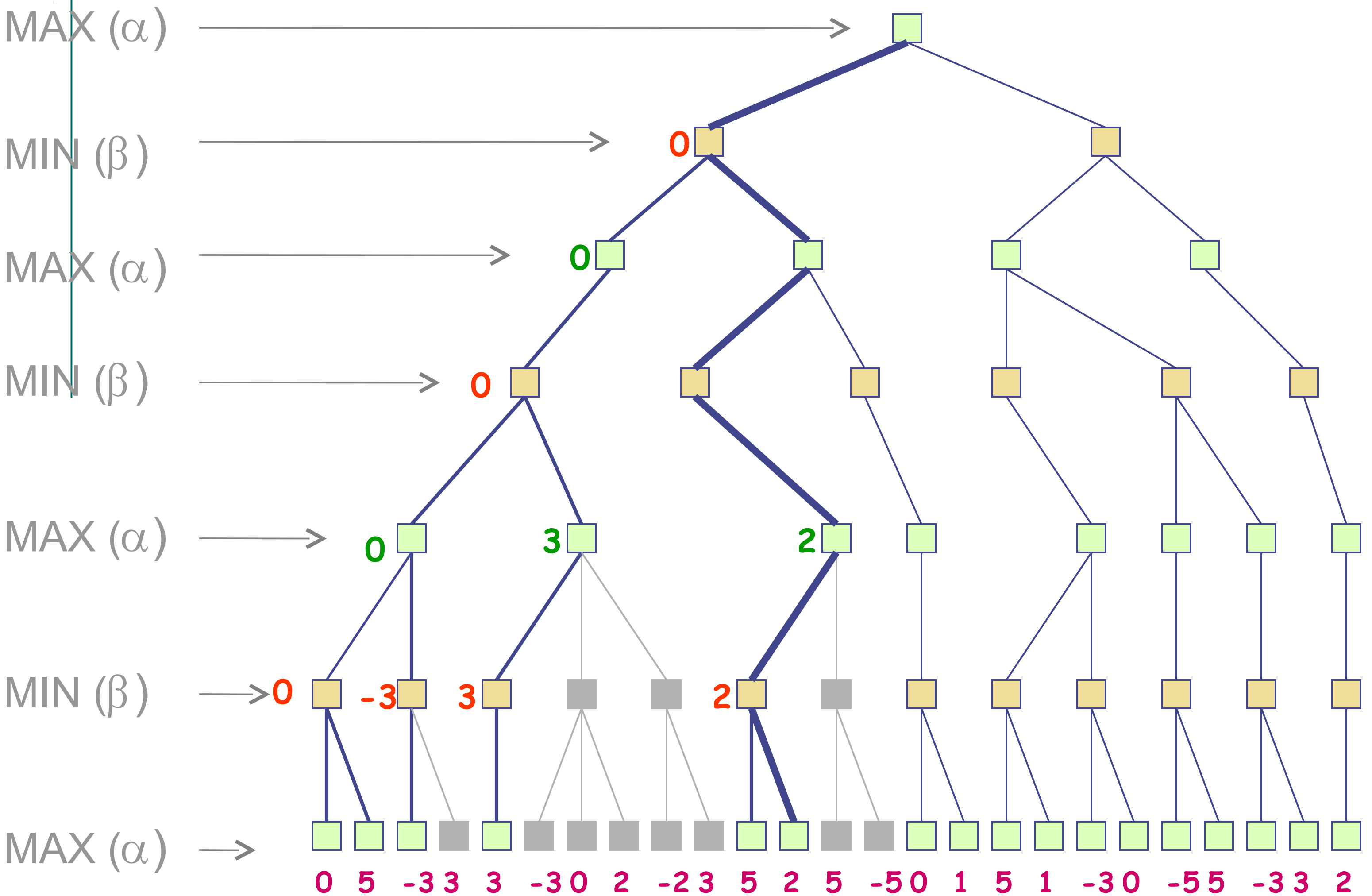
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0
0
3
0
-3
3
5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



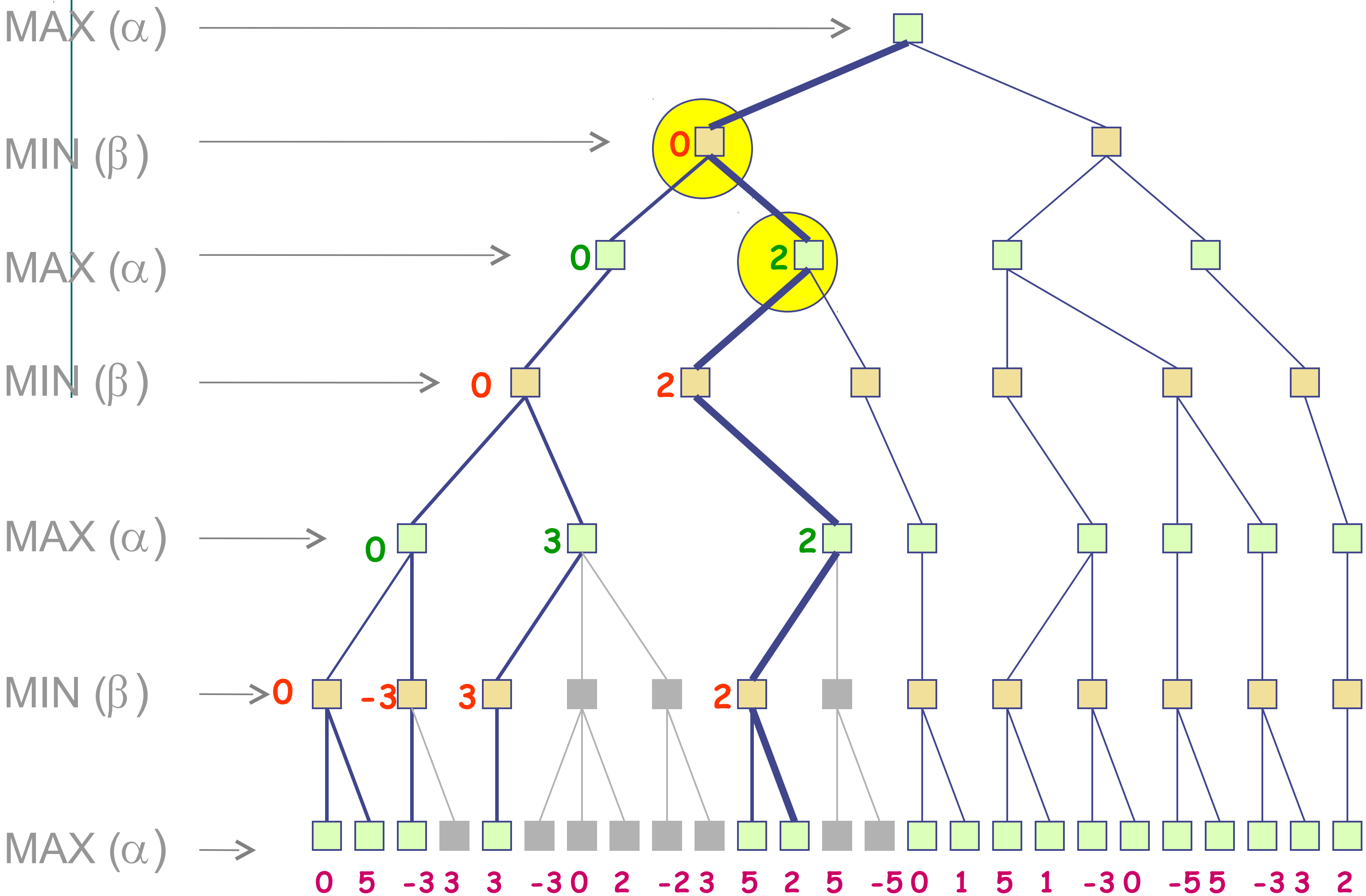
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0
0 3 2
0 -3 3 2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



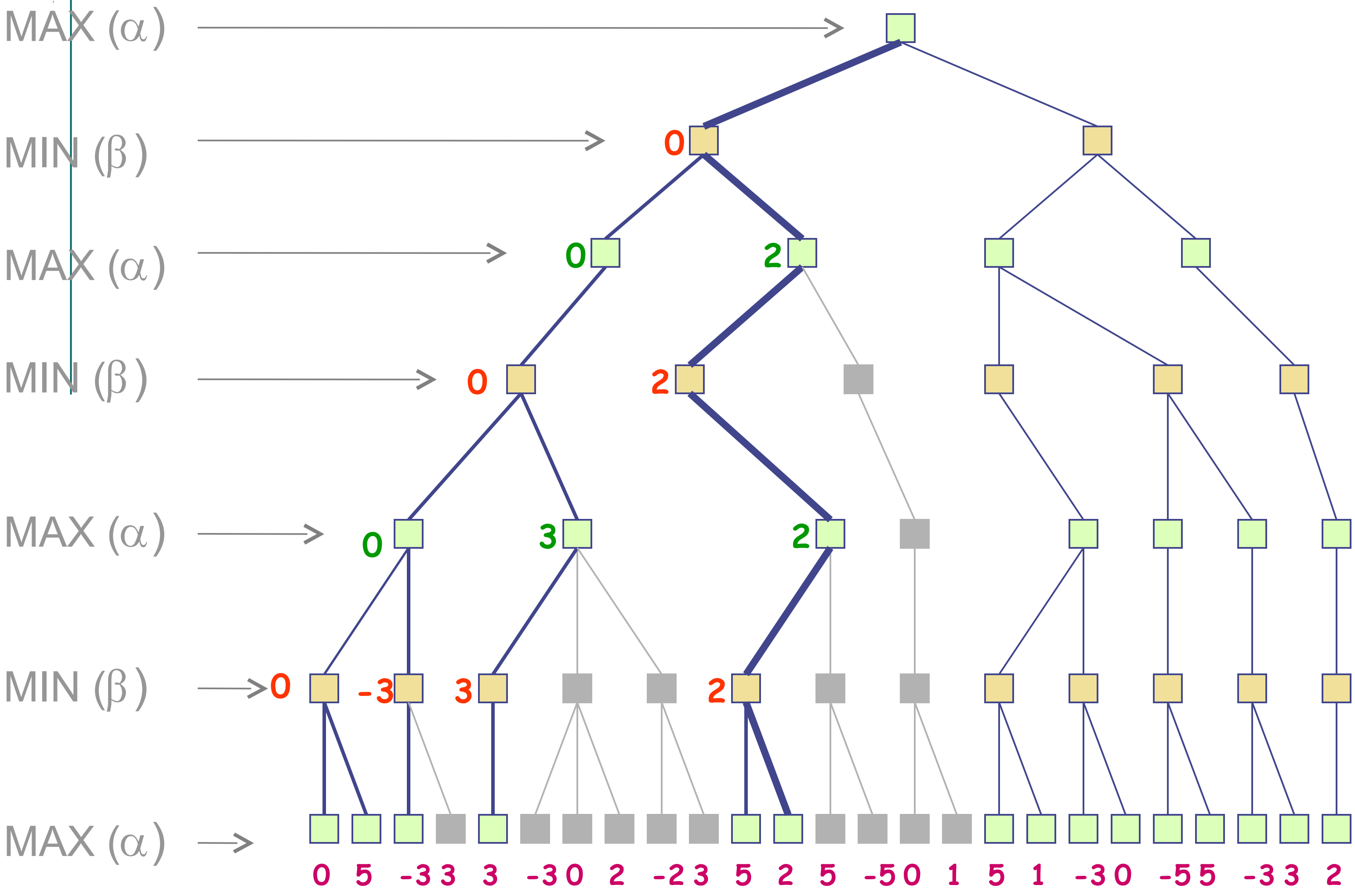
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0
0
0
3
2
0
-3
3
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



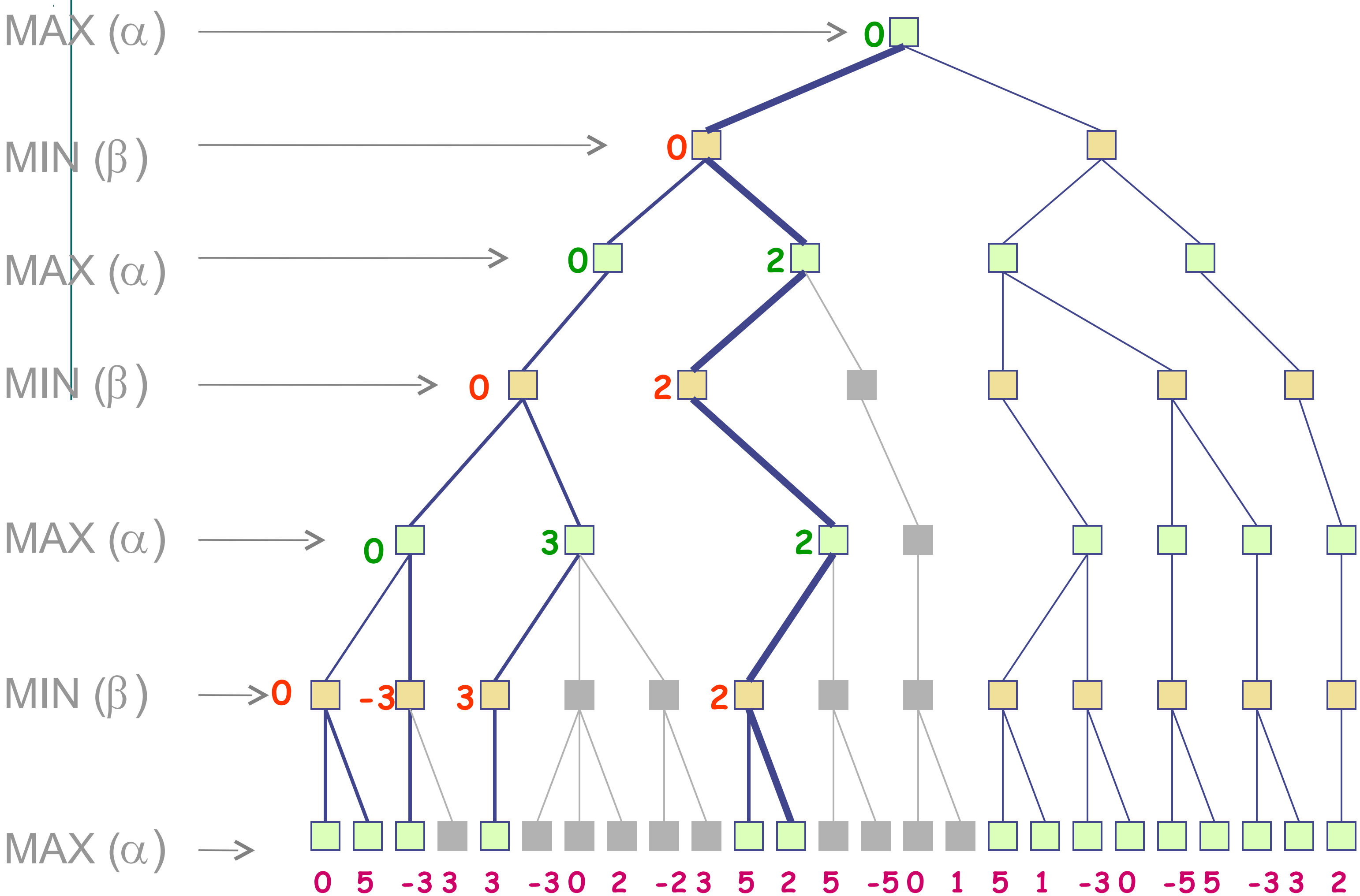
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0 2
0 2
0 3 2
0 -3 3 2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example

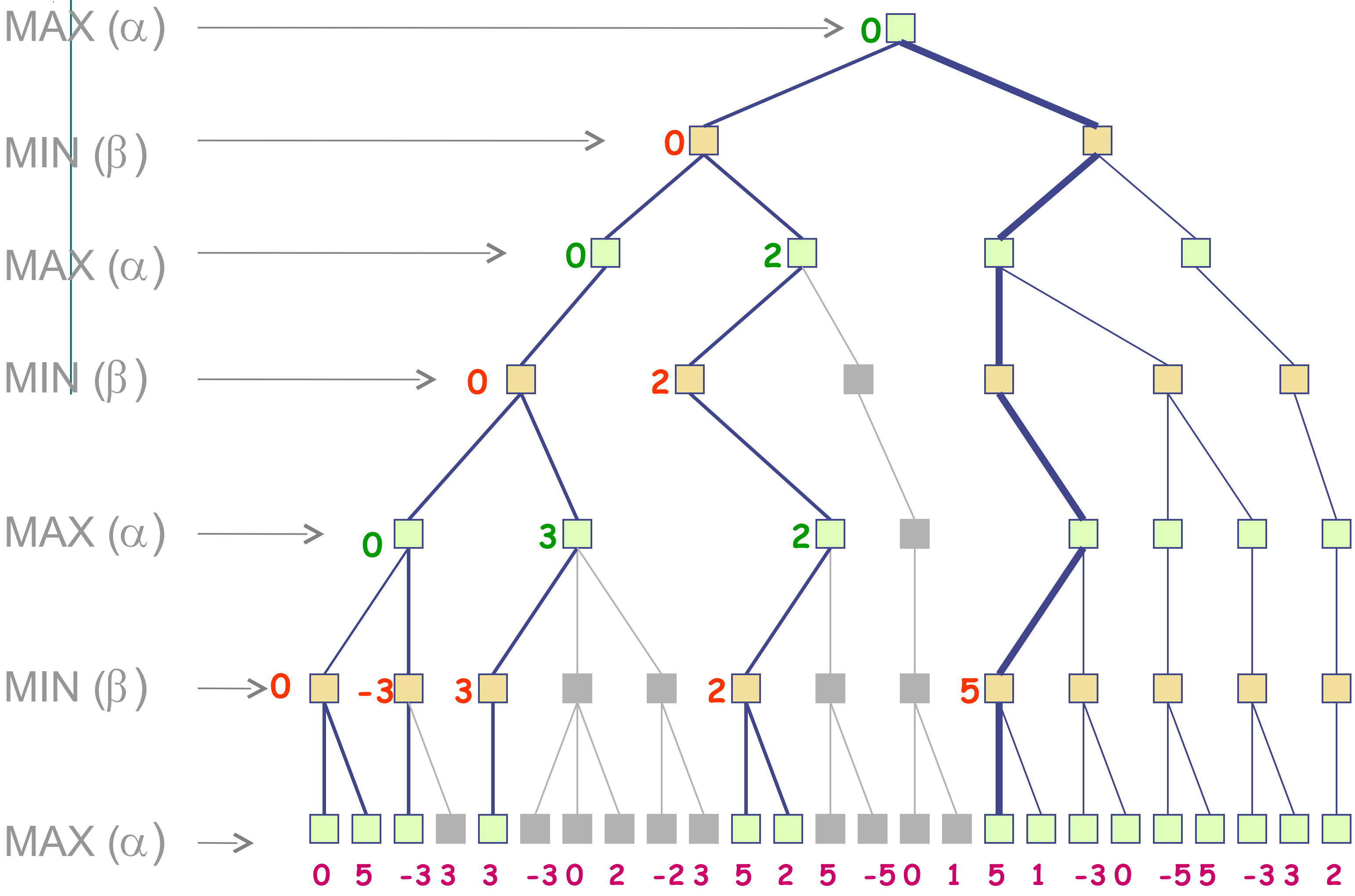


MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
2
0
2
0
3
2
0
-3
3
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example

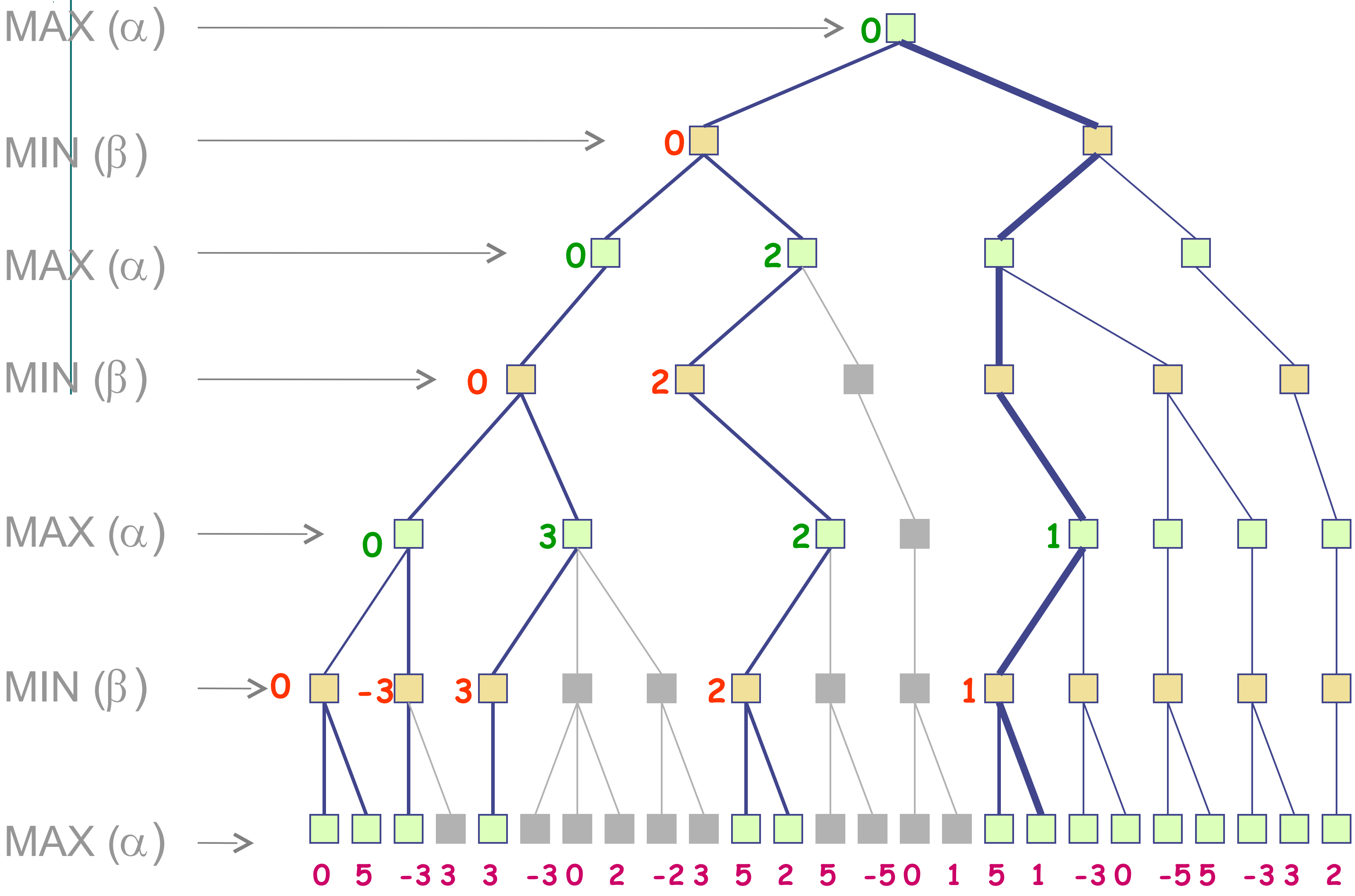

MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0
2
0
2
0
3
2
0
-3
3
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example

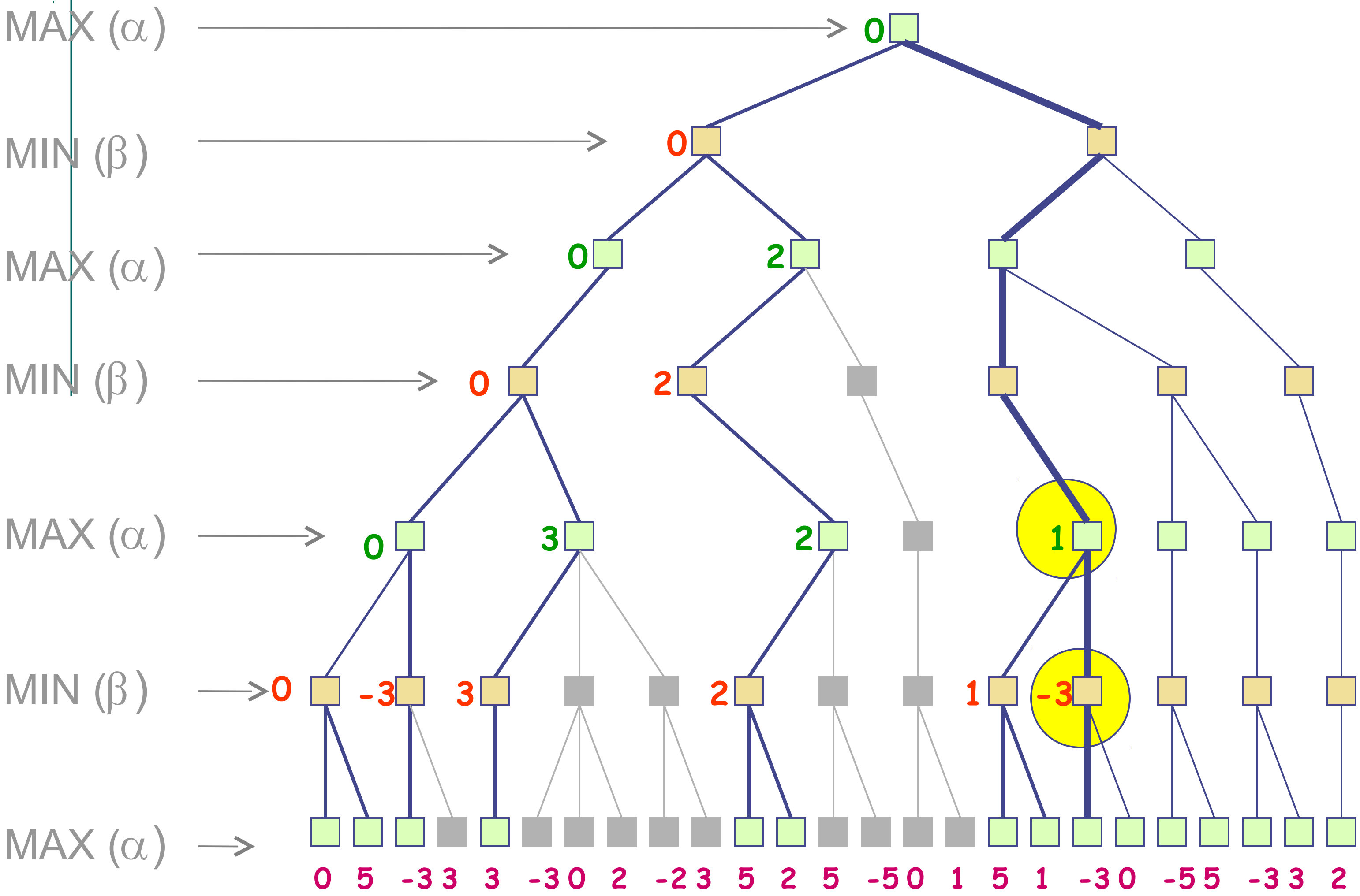


MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0
2
0
2
0
3
2
0
-3
3
2
5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example

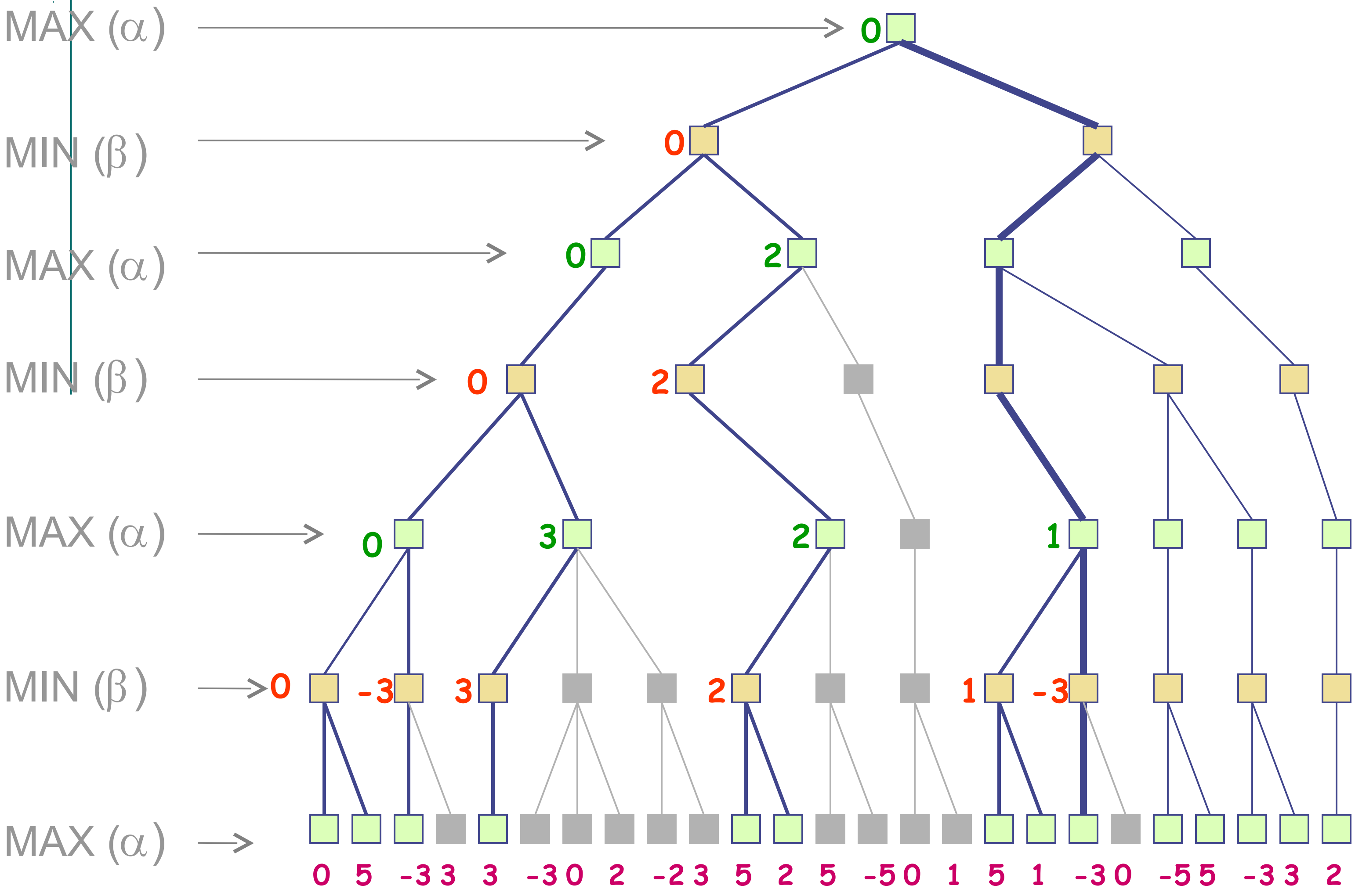

MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0 2
0 2
0 3 2 1
0 -3 3 2 1
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



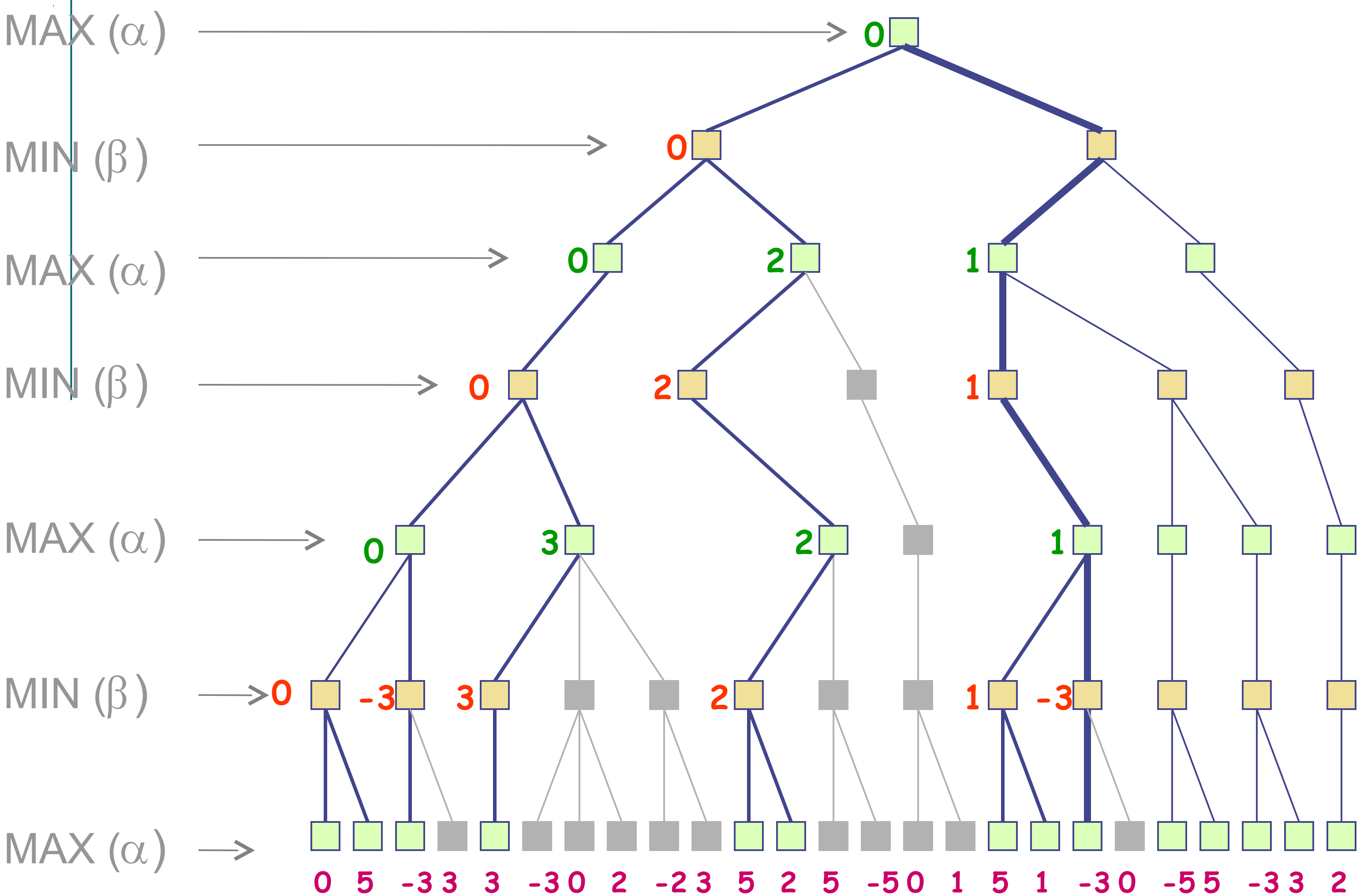
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0 2
0 2
0 3 2 1
0 -3 3 2 1 -3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



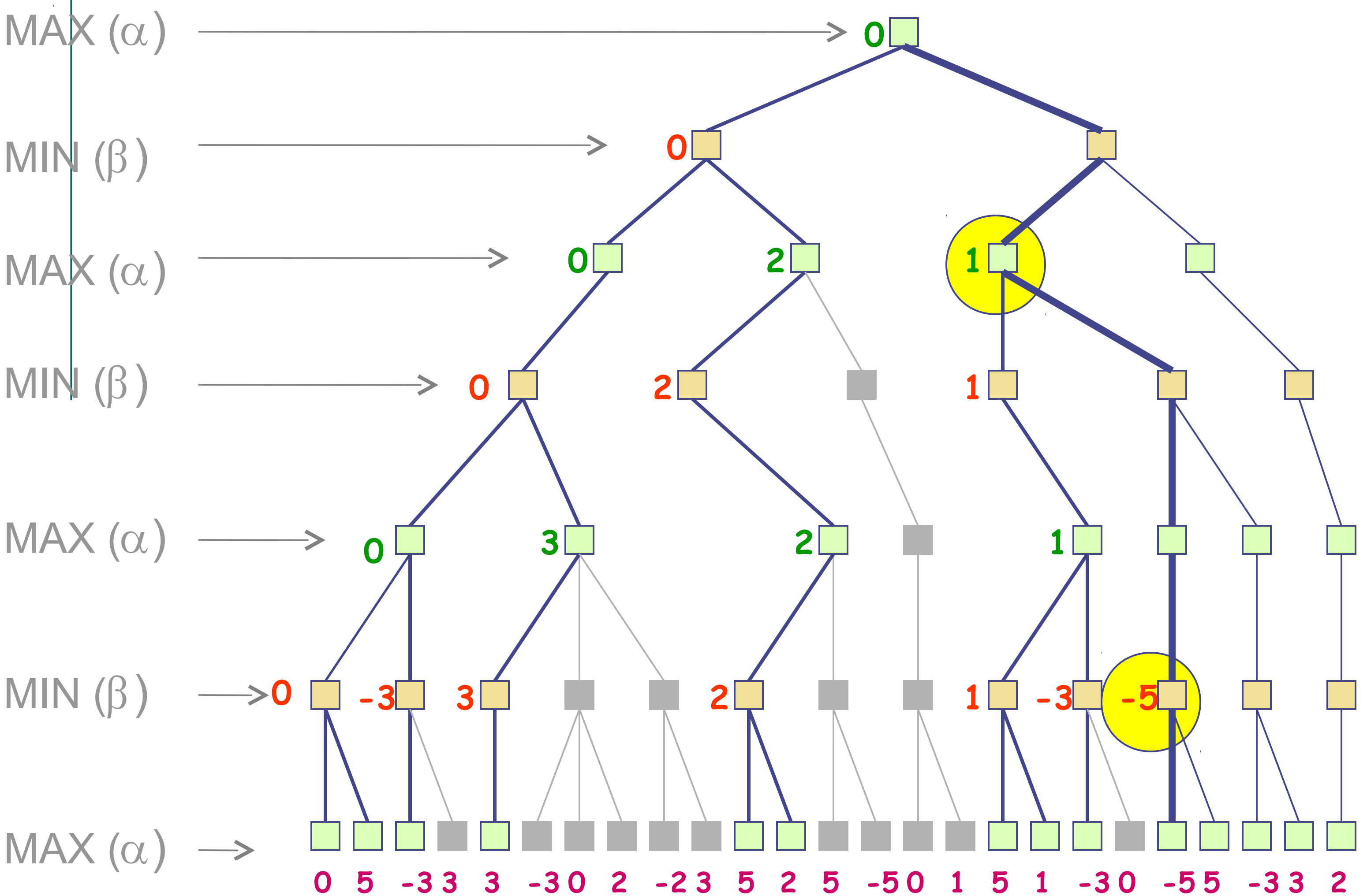
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0 2
0 2
0 3 2 1
0 -3 3 2 1 -3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



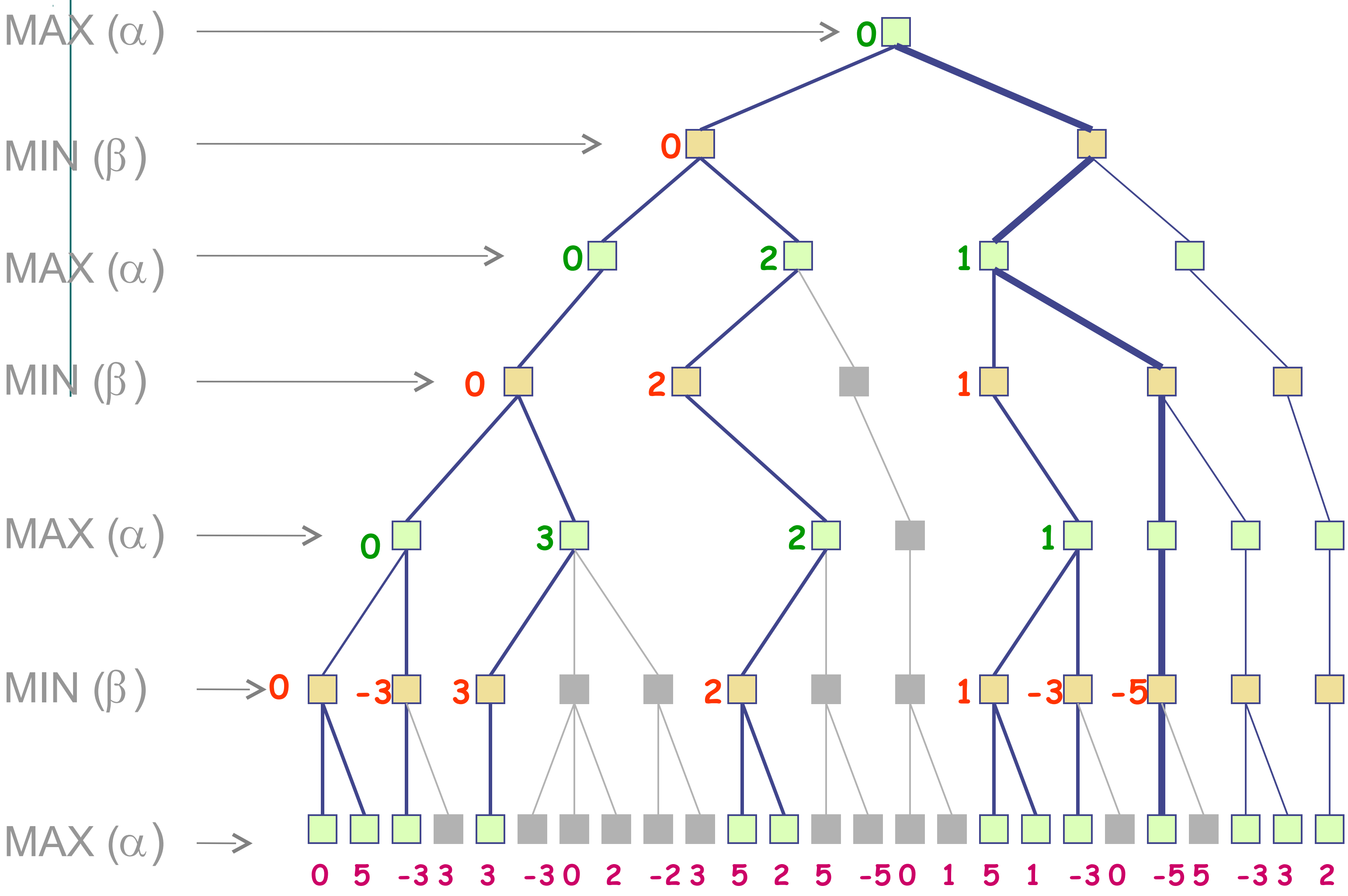
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0
2
1
0
2
1
0
3
2
1
0
-3
3
2
1
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



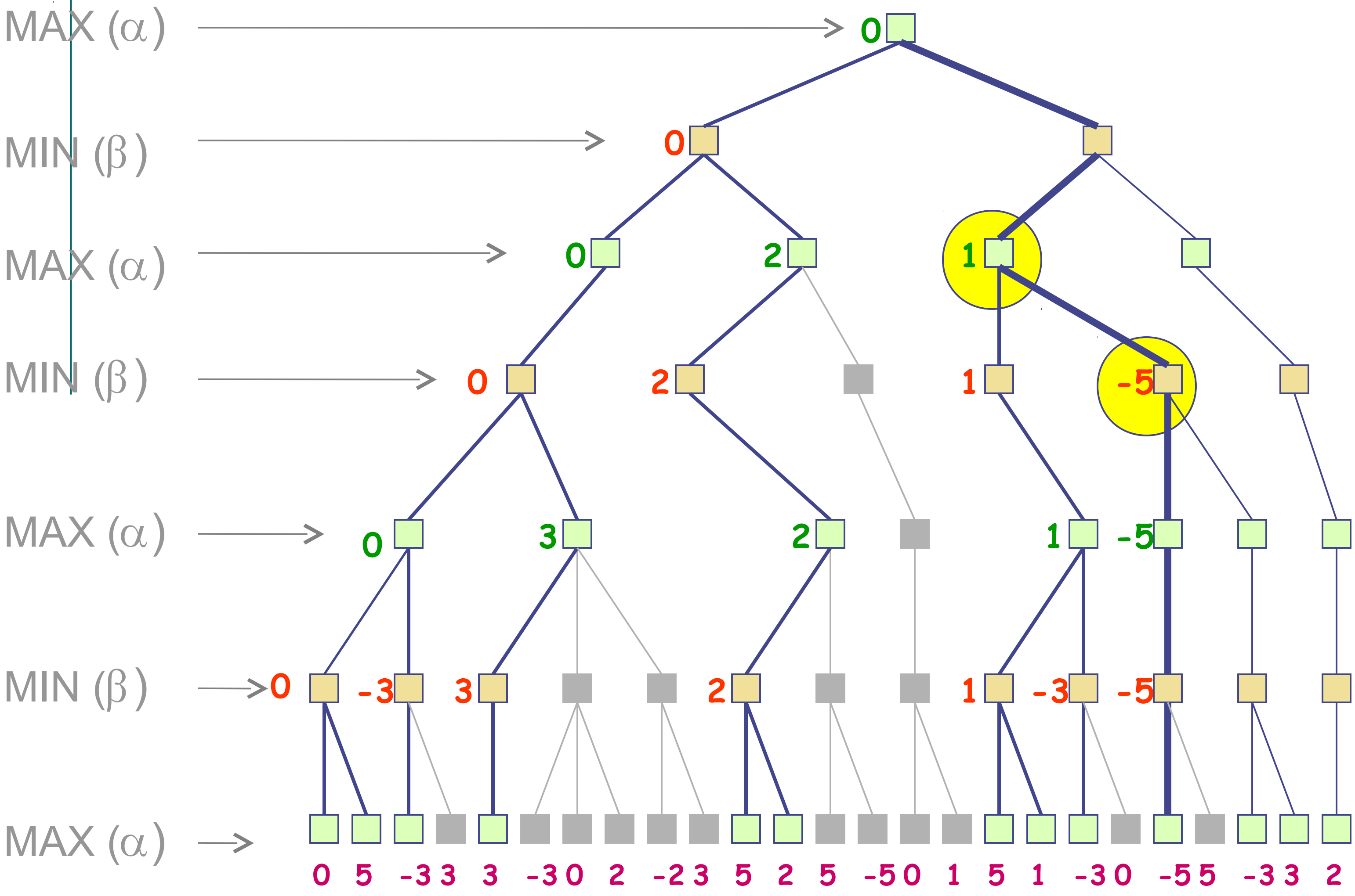
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0 2 1
0 2 1
0 3 2 1
0 -3 3 2 1 -3 -5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



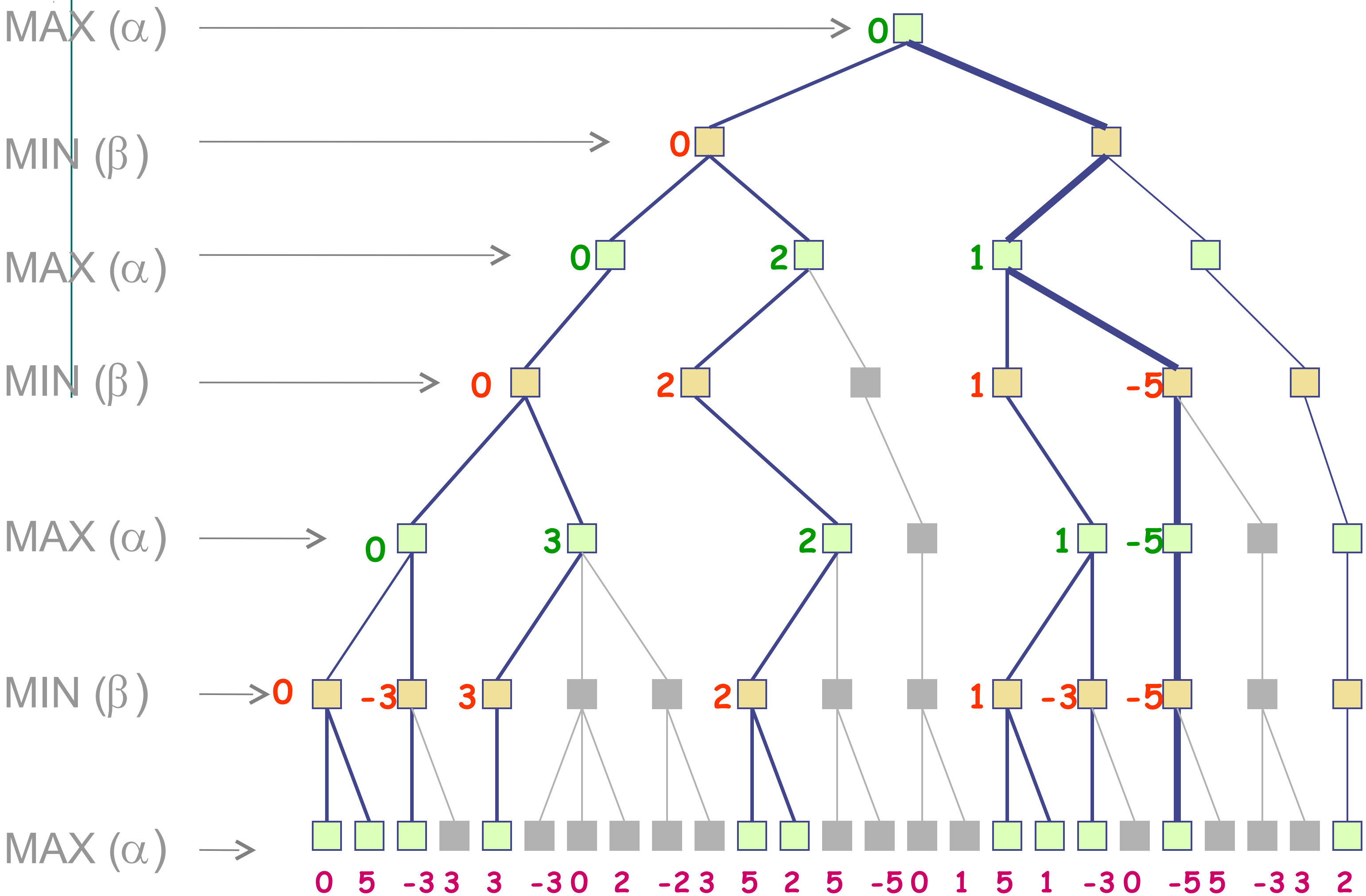
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0 2 1
0 2 1
0 3 2 1
0 -3 3 2 1 -3 -5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



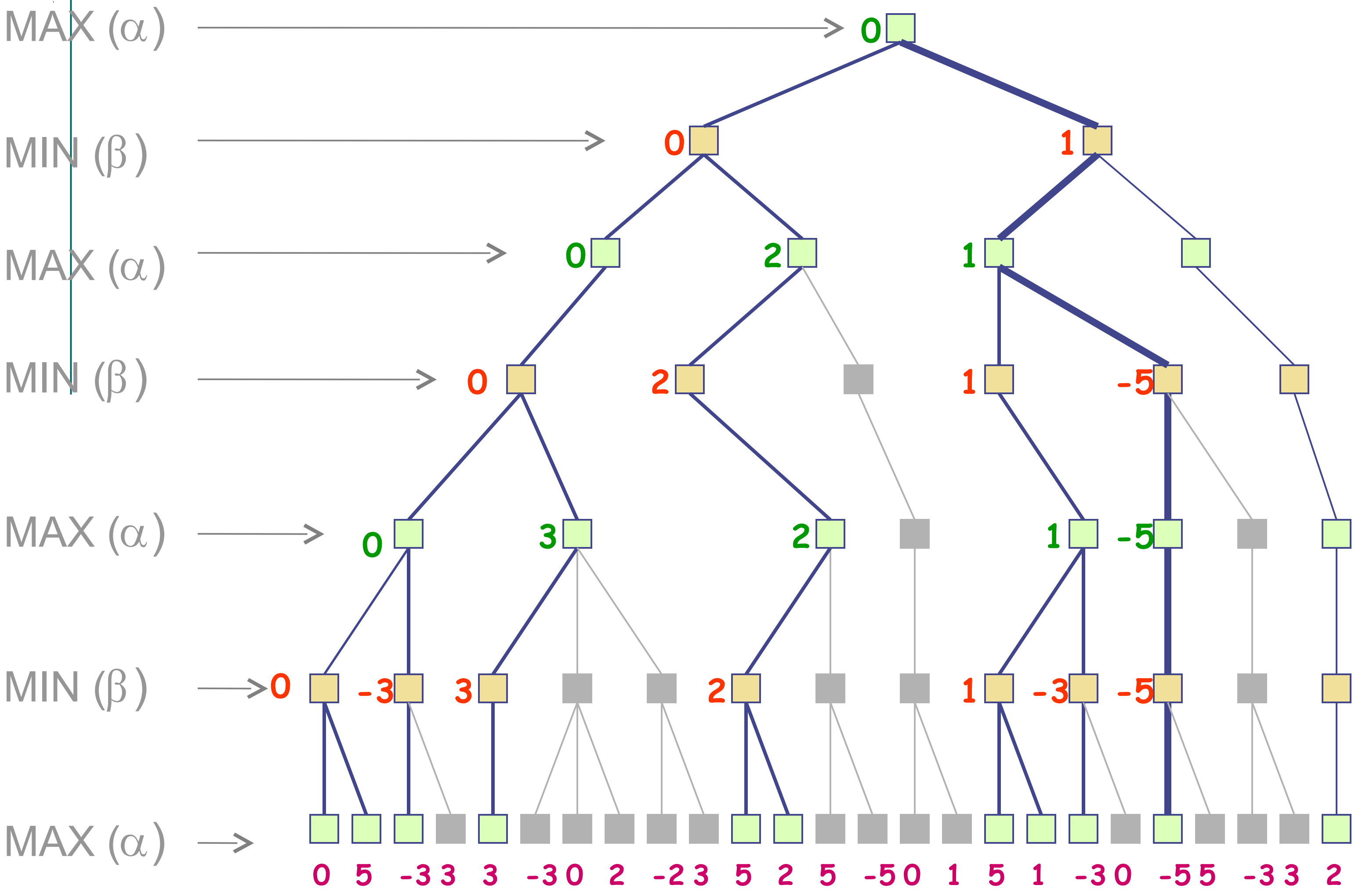
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0 2 1
0 2 1 -5
0 3 2 1 -5
0 -3 3 2 1 -3 -5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



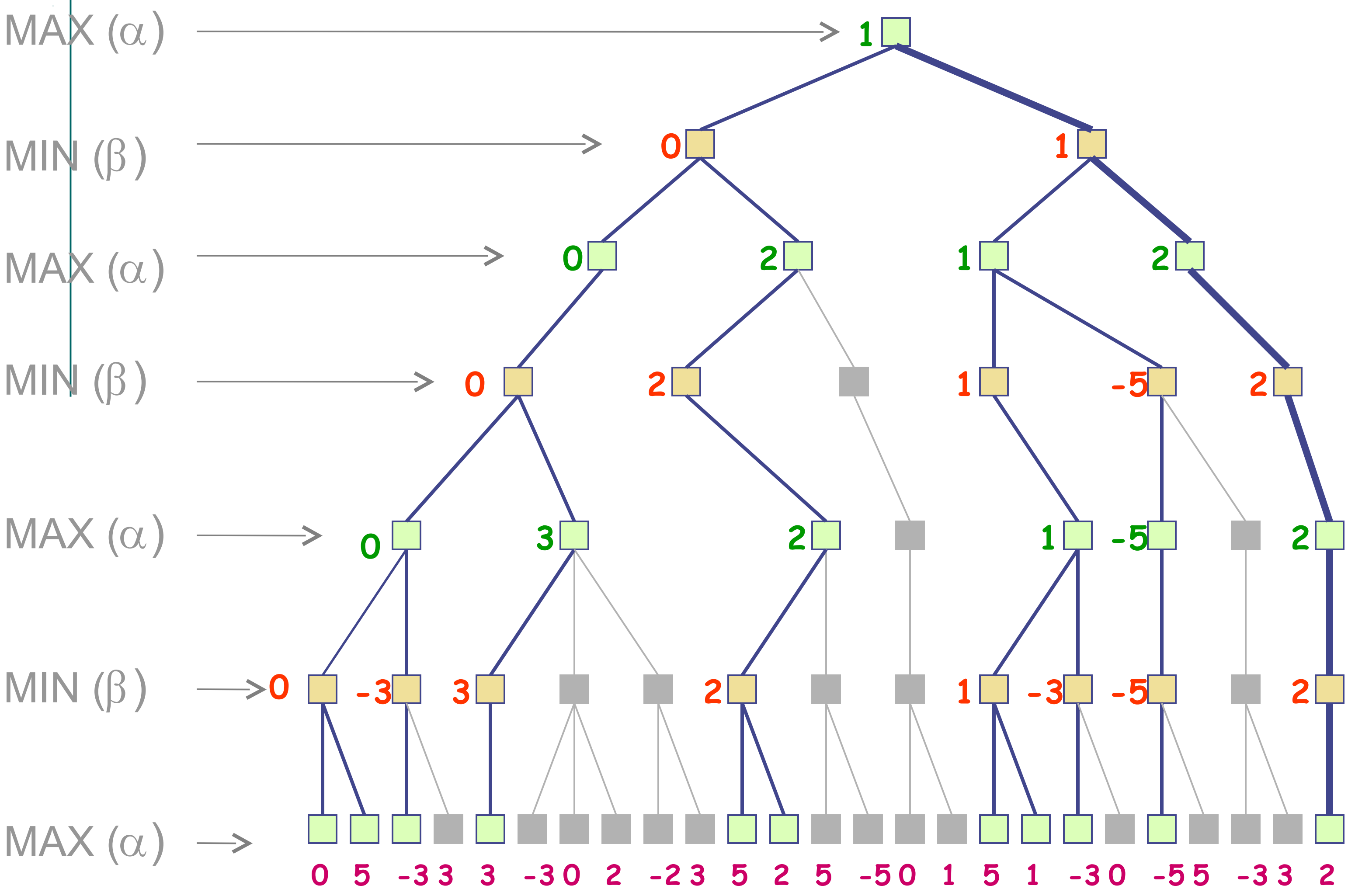
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0
0 2 1
0 2 1 -5
0 3 2 1 -5
0 -3 3 2 1 -3 -5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



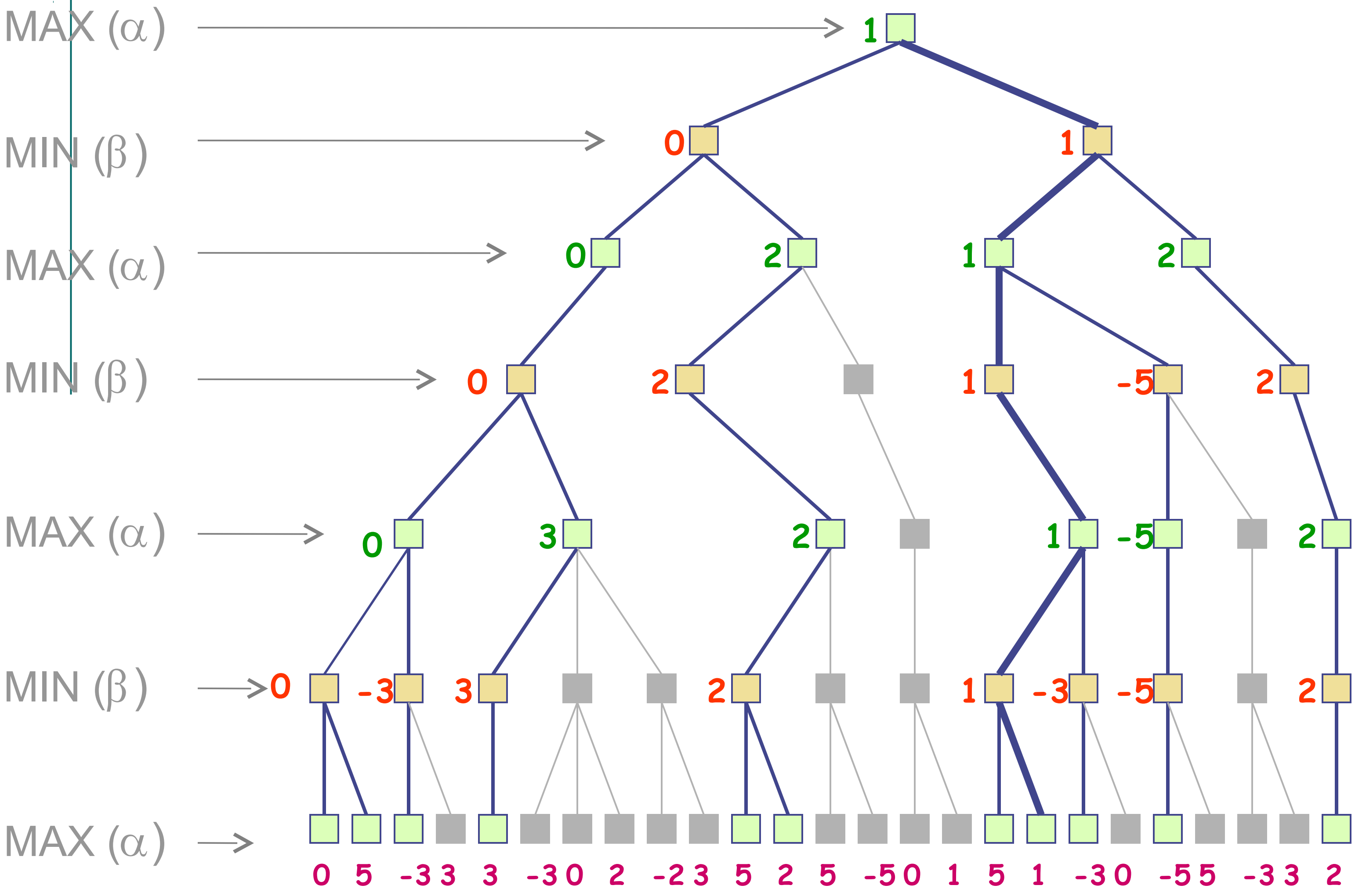
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
0
0 1
0 2 1
0 2 1 -5
0 3 2 1 -5
0 -3 3 2 1 -3 -5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example



MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
1
0 1
0 2 1 2
0 2 1 -5 2
0 3 2 1 -5 2
0 -3 3 2 1 -3 -5 2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

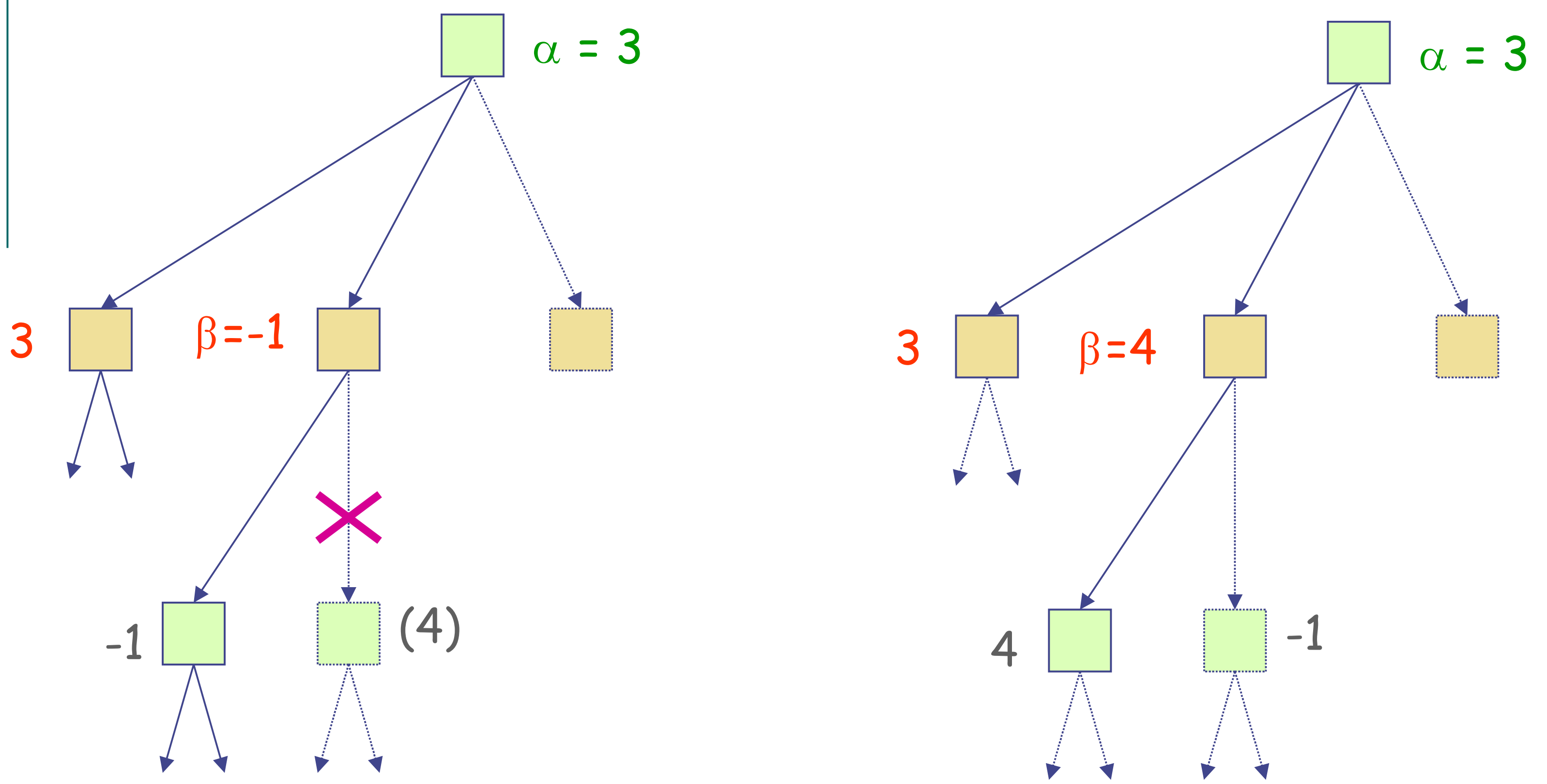
Example



MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
MIN (β)
MAX (α)
1
0
1
0
2
1
2
0
2
1
-5
2
0
3
2
1
-5
2
0
-3
3
2
1
-3
-5
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

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, so does alpha-beta in the worst-case

How much do we gain?

- The gain for alpha-beta is maximum when:
  - The MIN children of a MAX node are ordered in decreasing backed up values
  - The MAX children of a MIN node are ordered in increasing backed up values
- Then alpha-beta examines $O(b^{h/2})$ nodes [Knuth and Moore, 1975]

How much do we gain?

- **But** this requires an oracle (if we knew how to order nodes perfectly, we would not need to search the game tree)

- If nodes are ordered at random, then the average number of nodes examined by alpha-beta is $O(b^{3h/4})$

  → Good move ordering is essential for efficient alpha-beta pruning!

Heuristic Ordering of Nodes

- Use iterative deepening
- Order the nodes below the root according to the values backed-up at the previous iteration

Other Improvements

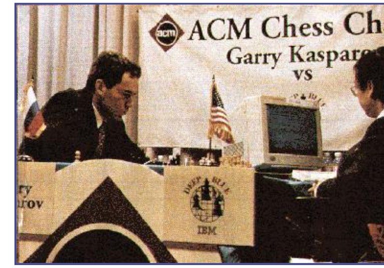
- Other heuristics to increase cut-offs:
  - Killer move heuristics / refutation table
  - Null-move heuristics
  - Null-window search / Negascout
- Use transposition tables to deal with repeated states
- To mitigate the horizon effect:
  Quiescence search/ Singular extension
- Forward pruning
- End-game databases
- Opening books

Computers For The Win!

Chinook (1994)

First computer to become official world champion of Checkers!

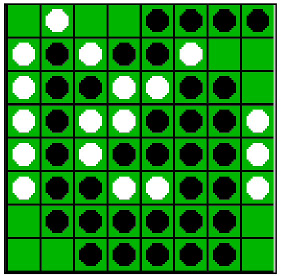
Chess:
Kasparov vs. Deep Blue



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

Othello:
Murakami vs. Logistello



Takeshi Murakami
World Othello Champion

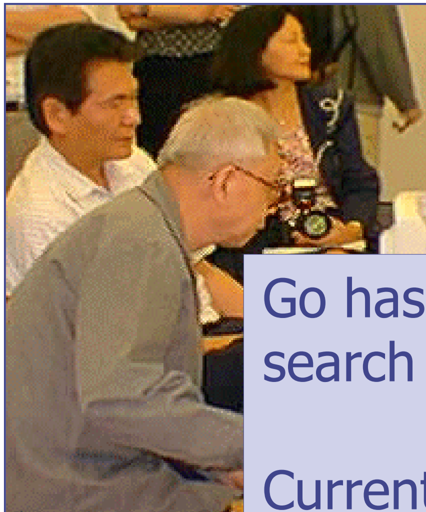
1997: The Logistello software crushed Murakami by 6 games to 0

Go: Goemate vs. ??

Name: Chen Zhixing
Profession: Retired
Computer skills:
    self-taught programmer
Author of Goemate (arguably the best Go program available today)

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

Go: Goemate vs. ??



Name: Chen Zhixing
Profession: Retired
Computer skills:

est

Go has too high a branching factor for existing search techniques

Current and future software must rely on huge databases and pattern-recognition techniques

handicap and still easily beat the program, thereby winning $15,000

Secrets

Many game programs are based on alpha-beta + iterative deepening + extended/singular search + transposition tables + huge databases + ...

For instance, Chinook searched all checkers configurations with 8 pieces or less and created an endgame database of 444 billion board configurations

Secrets

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

Other Types of Games

- Multi-player games, with alliances or not
- Games with randomness in successor function (e.g., rolling a dice)
  → Expectiminimax algorithm (adds chance nodes to the Minimax tree)
- Games with partially observable states (e.g., card games)
  → Search of belief state spaces