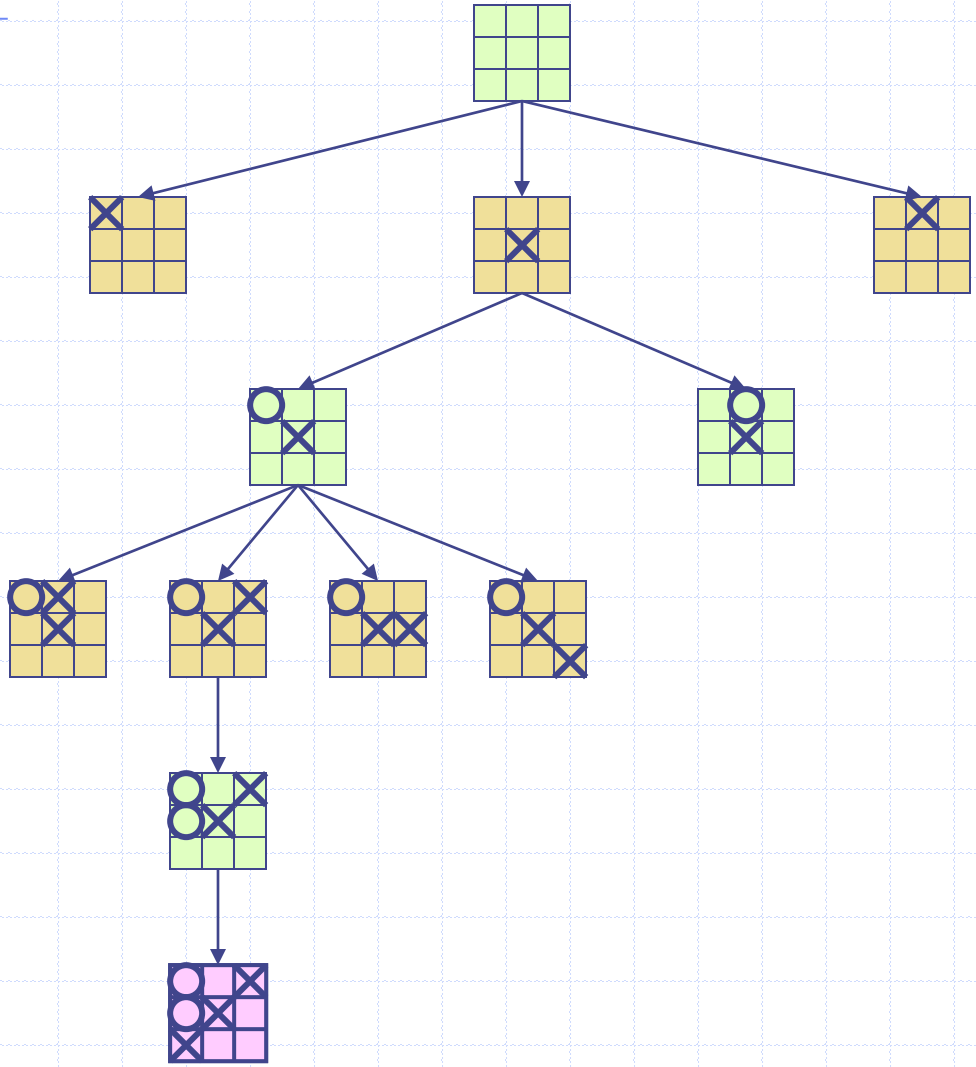
Adversarial Search and Game Playing

Russell and Norvig: Chapter 6

Perfect Two-Player Game

- Two players MAX and MIN take turn (with MAX playing first)
- State space
- Initial state
- Successor function
- Terminal test
- Score function, that tells whether a terminal state is a win (for MAX), a loss, or a draw
- Perfect knowledge of states, no uncertainty in successor function

Partial Tree for Tic-Tac-Toe



But in general the search tree is too big to make it possible to reach the terminal states!

But in general the search tree is too big to make it possible to reach the terminal states!

Examples:

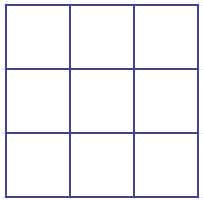
- Checkers: $\sim 10^{40}$ nodes
- Chess: $\sim 10^{120}$ nodes

Evaluation Function of a State

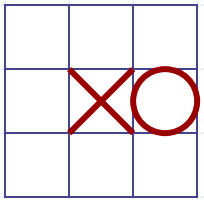
- e(s) = $+\infty$ if s is a win for MAX
- e(s) = $-\infty$ if s is a win for MIN
- e(s) = a measure of how "favorable" is s for MAX
  - > 0 if s is considered favorable to MAX
  - < 0 otherwise

Example: Tic-Tac-Toe

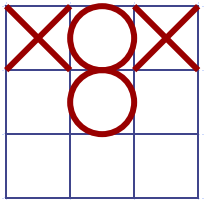
e(s) = number of rows, columns, and diagonals open for MAX
- number of rows, columns, and diagonals open for MIN



8-8 = 0



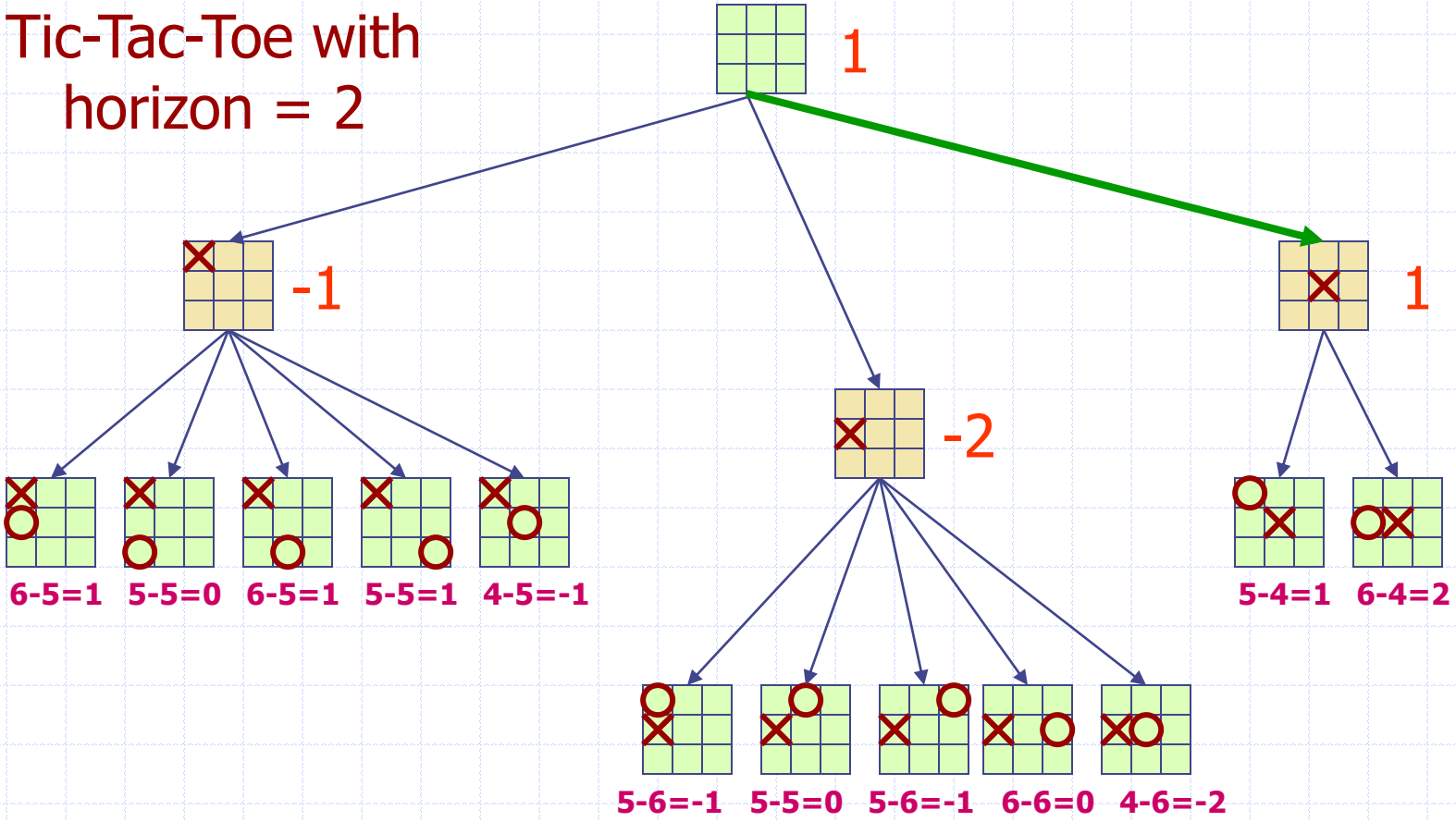
6-4 = 2



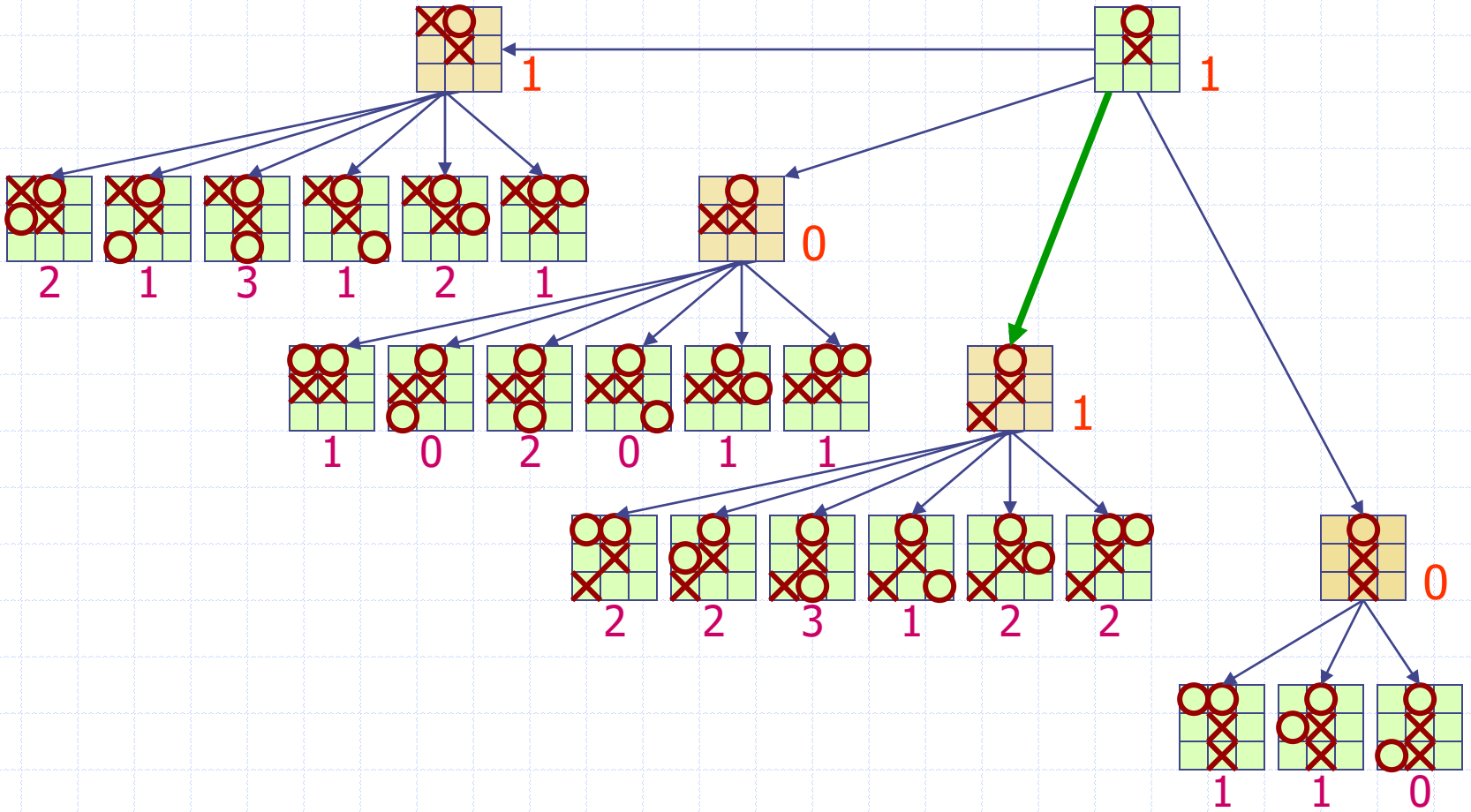
3-3 = 0

Example

Tic-Tac-Toe with horizon = 2



Example



Minimax procedure

1. Expand the game tree uniformly from the current state (where it is MAX's turn to play) to depth h
2. Compute the evaluation function at every leaf of the tree
3. Back-up the values from the leaves to the root of the tree as follows:
    1. A MAX node gets the maximum of the evaluation of its successors
    2. A MIN node gets the minimum of the evaluation of its successors
4. Select the move toward the MIN node that has the maximal backed-up value

Minimax procedure

1. Expand the game tree uniformly from the current state (where it is MAX's turn to play) to depth h
2. Compute the evaluation function at every leaf of the tree
3. Back-up the values from the leaves to the root of the tree a
   1. A MAX r
      succesc
   2. A MIN n
      succesc
4. Select the he maximal backed-up value

Horizon of the procedure

Needed to limit the size of the tree or to return a decision within allowed time

Game Playing (for MAX)

Repeat until *win*, *lose*, or *draw*

1. Select move using Minimax procedure
2. Execute move
3. Observe MIN's move

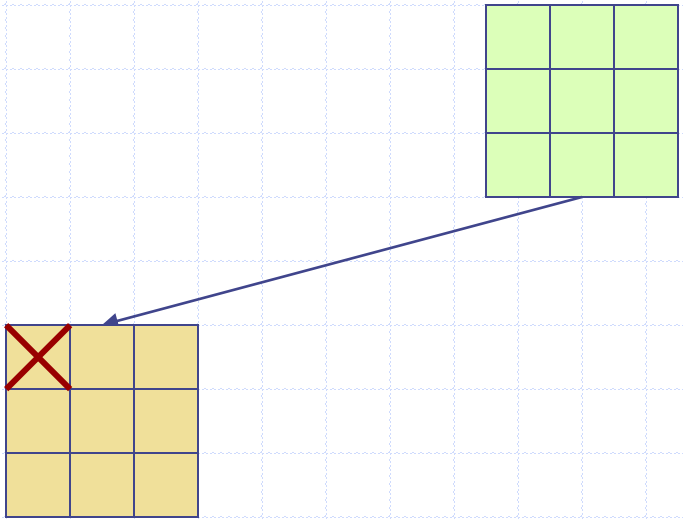
Issues

- Choice of the horizon
- Size of memory needed
- Number of nodes examined

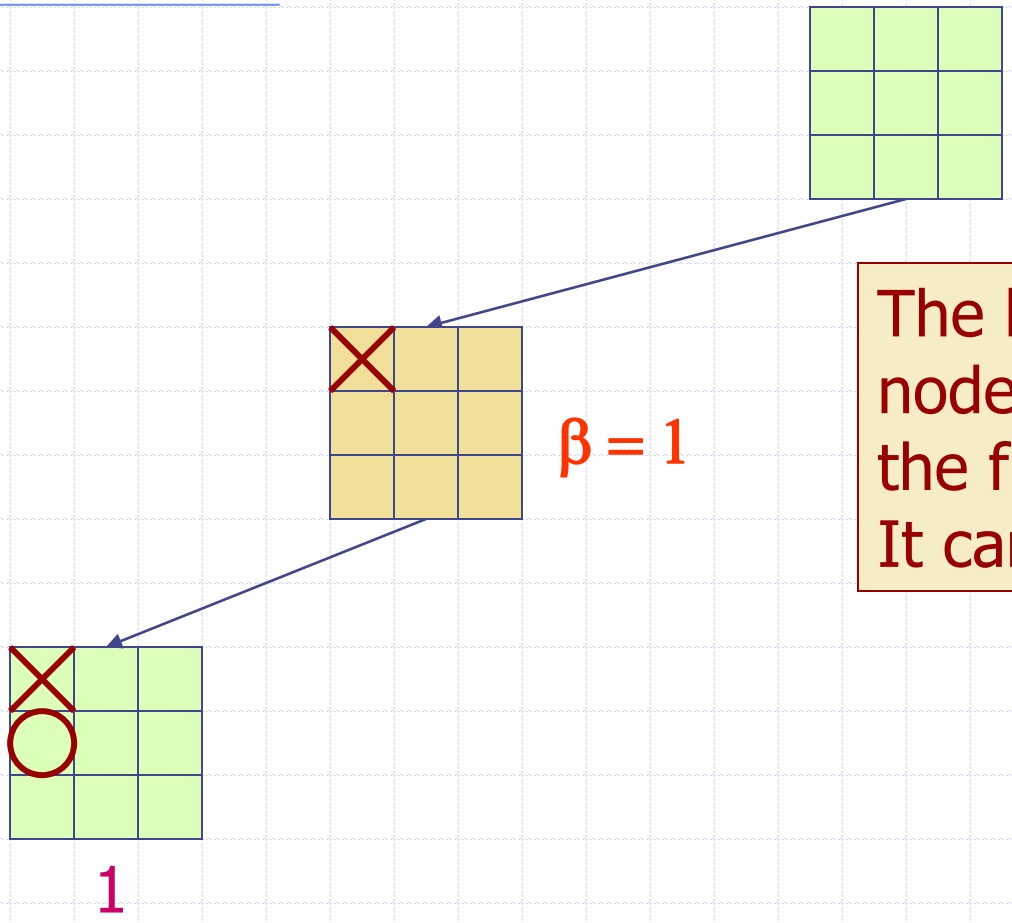
Alpha-Beta Procedure

- Generate the game tree to depth h in depth-first manner
- Back-up estimates (alpha and beta values) of the evaluation functions whenever possible
- Prune branches that cannot lead to changing the final decision

Example

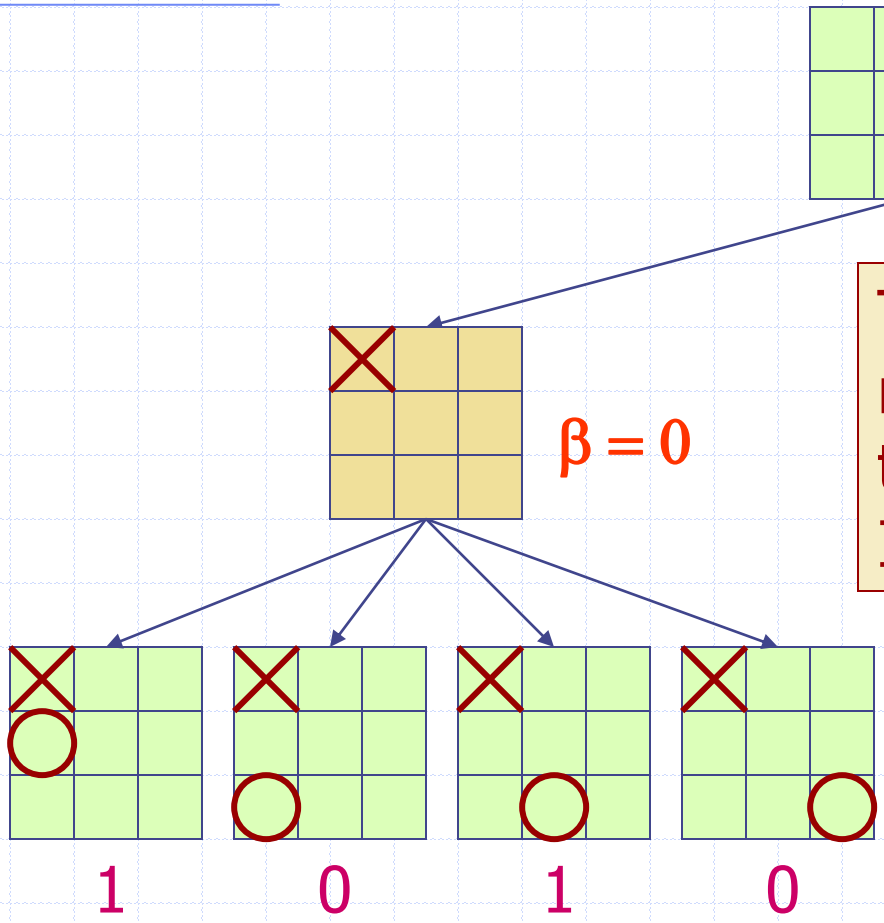


Example



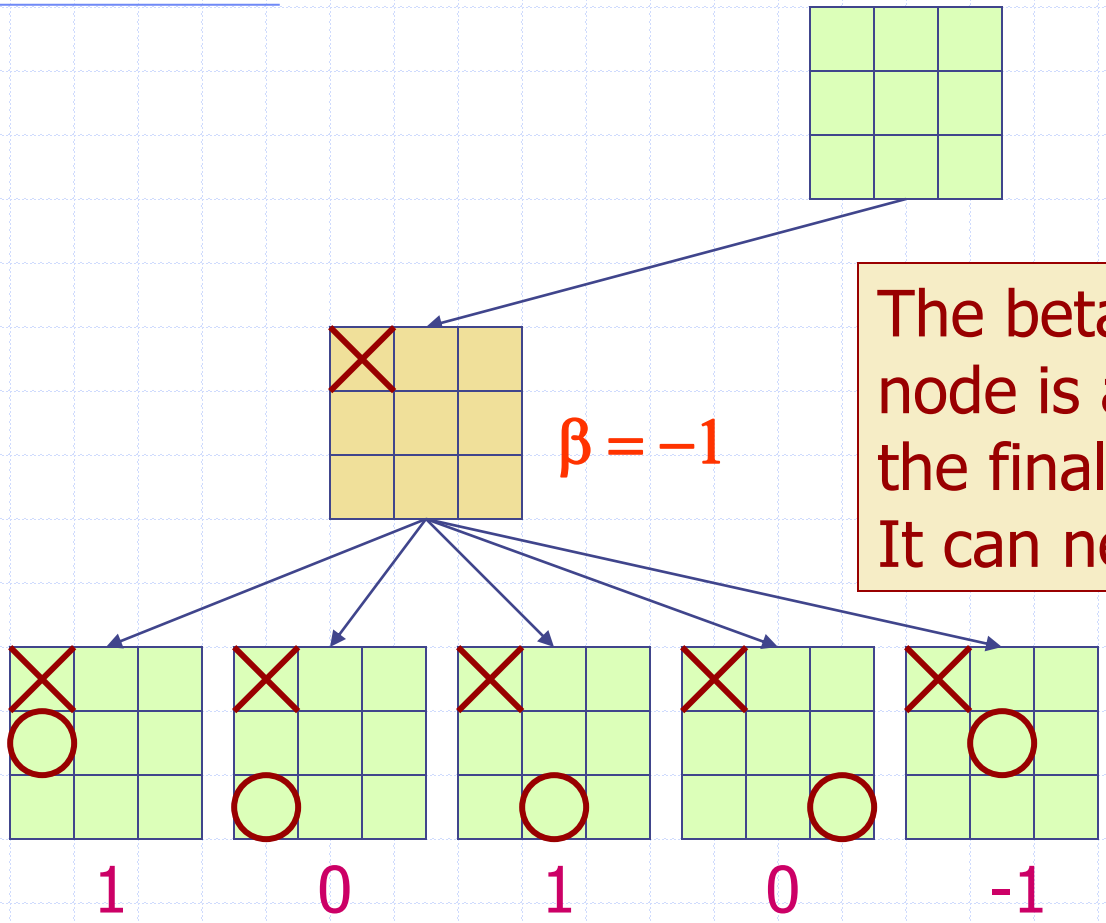
The beta value of a MIN node is a higher bound on the final backed-up value. It can never increase

Example



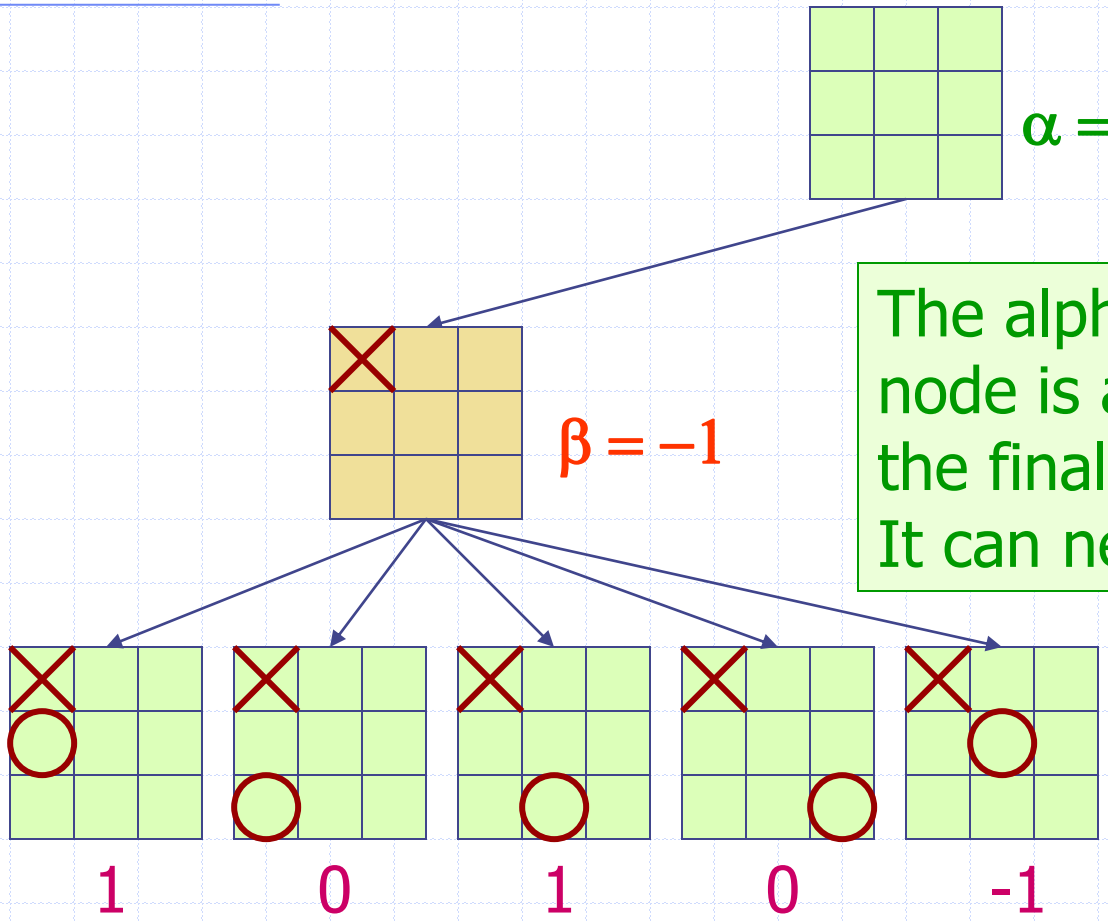
The beta value of a MIN node is a higher bound on the final backed-up value. It can never increase

Example



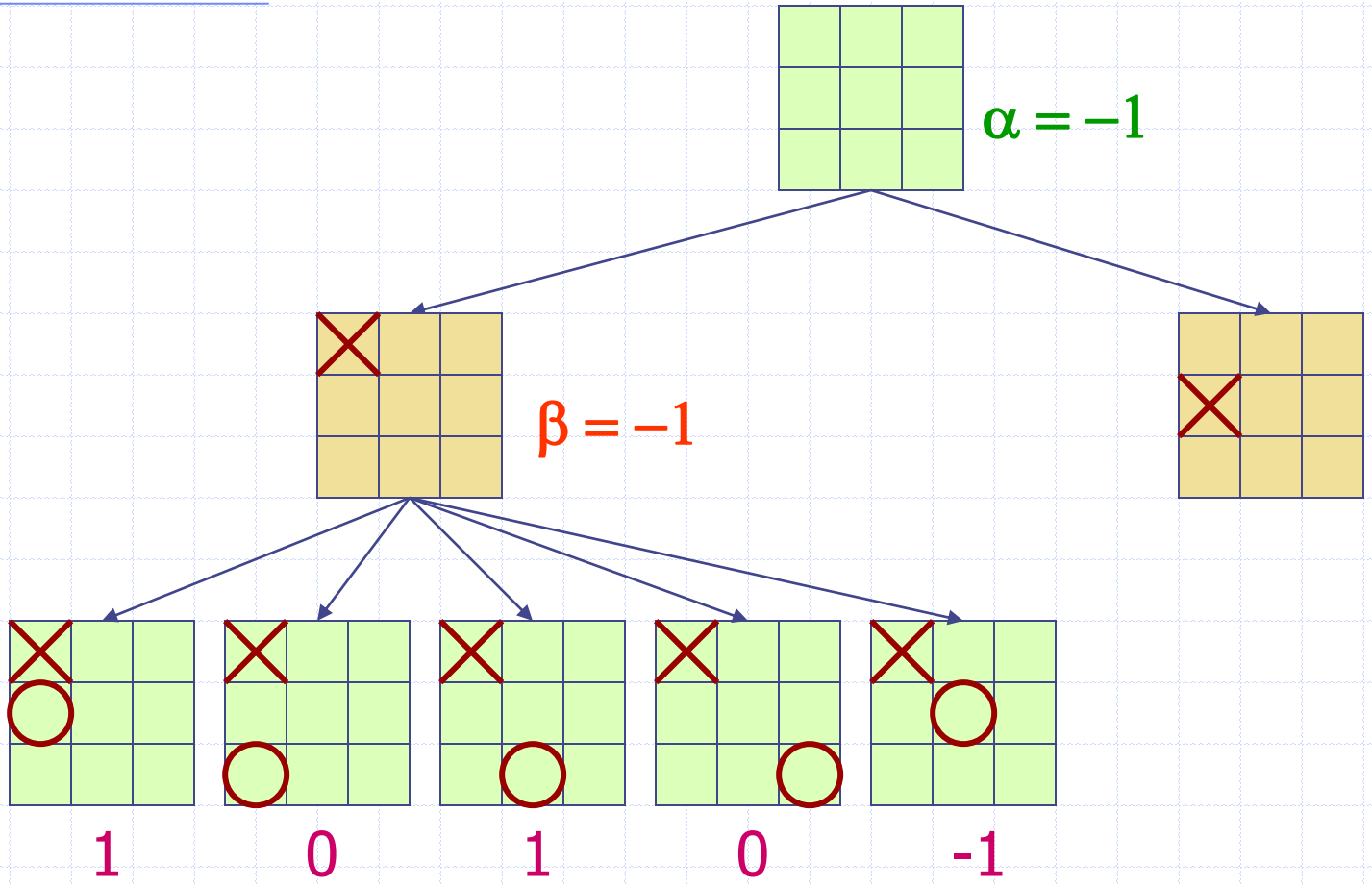
The beta value of a MIN node is a higher bound on the final backed-up value. It can never increase

Example

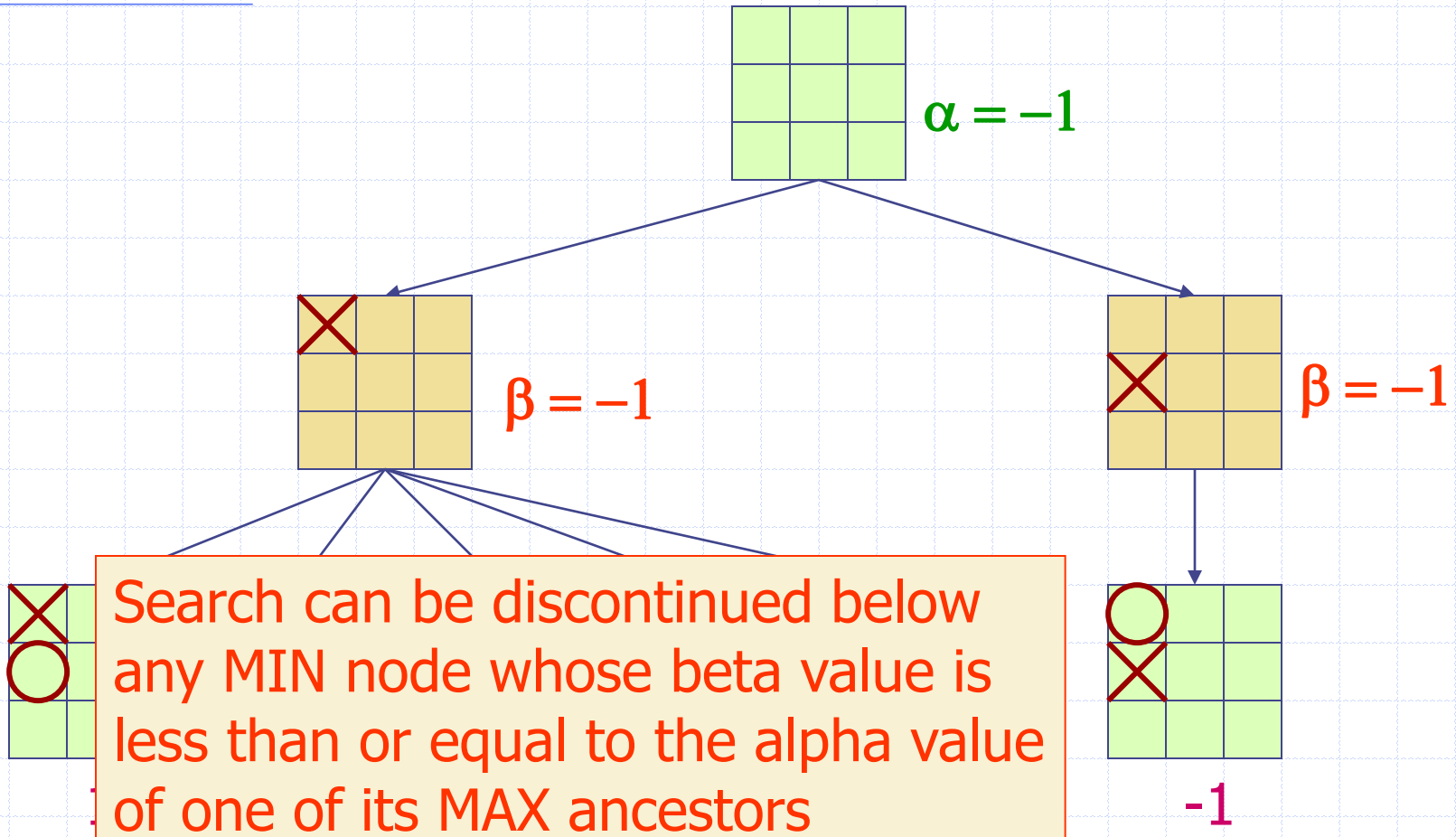


The alpha value of a MAX node is a lower bound on the final backed-up value. It can never decrease

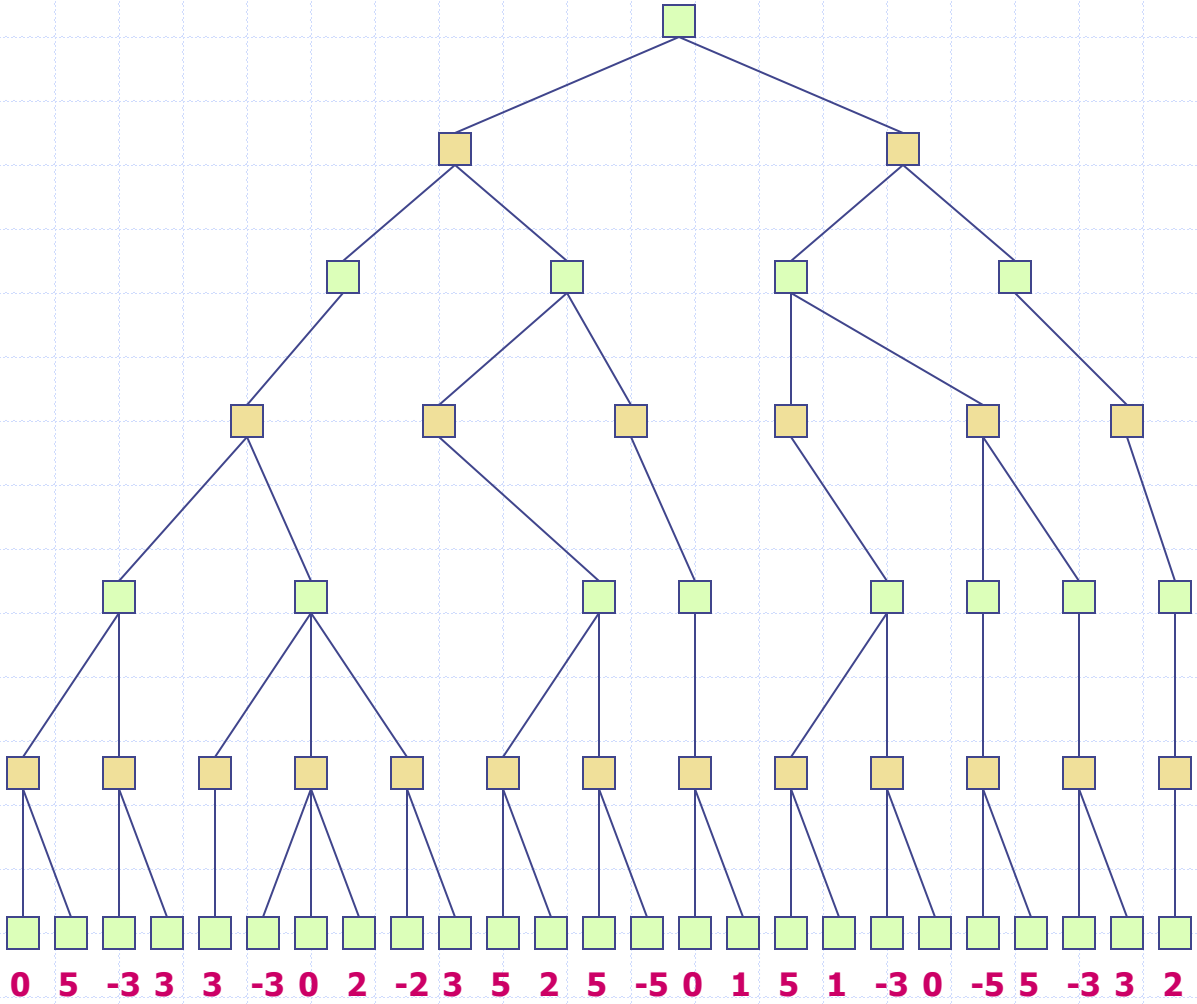
Example



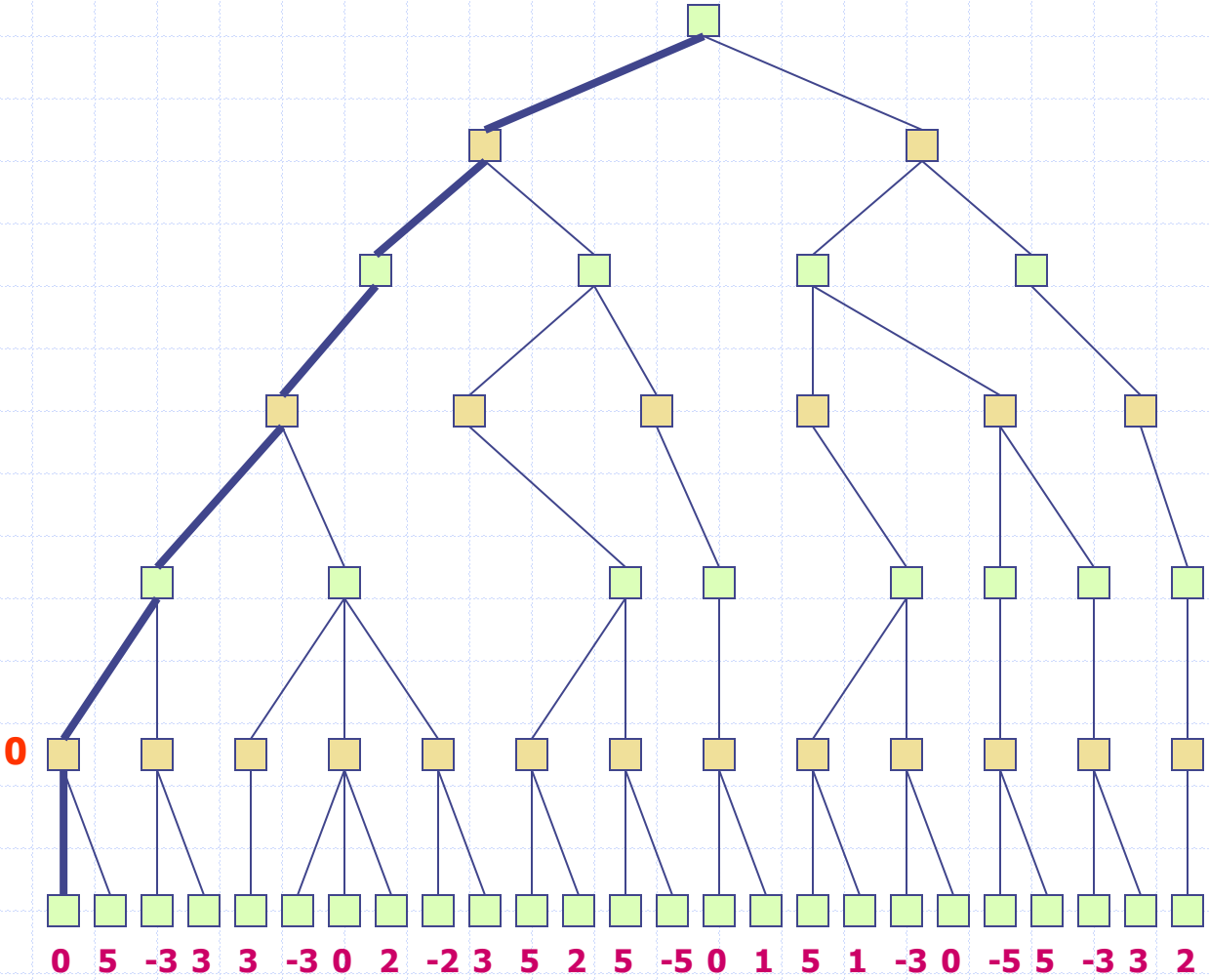
Example


α = –1
β = –1
β = –1
-1
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

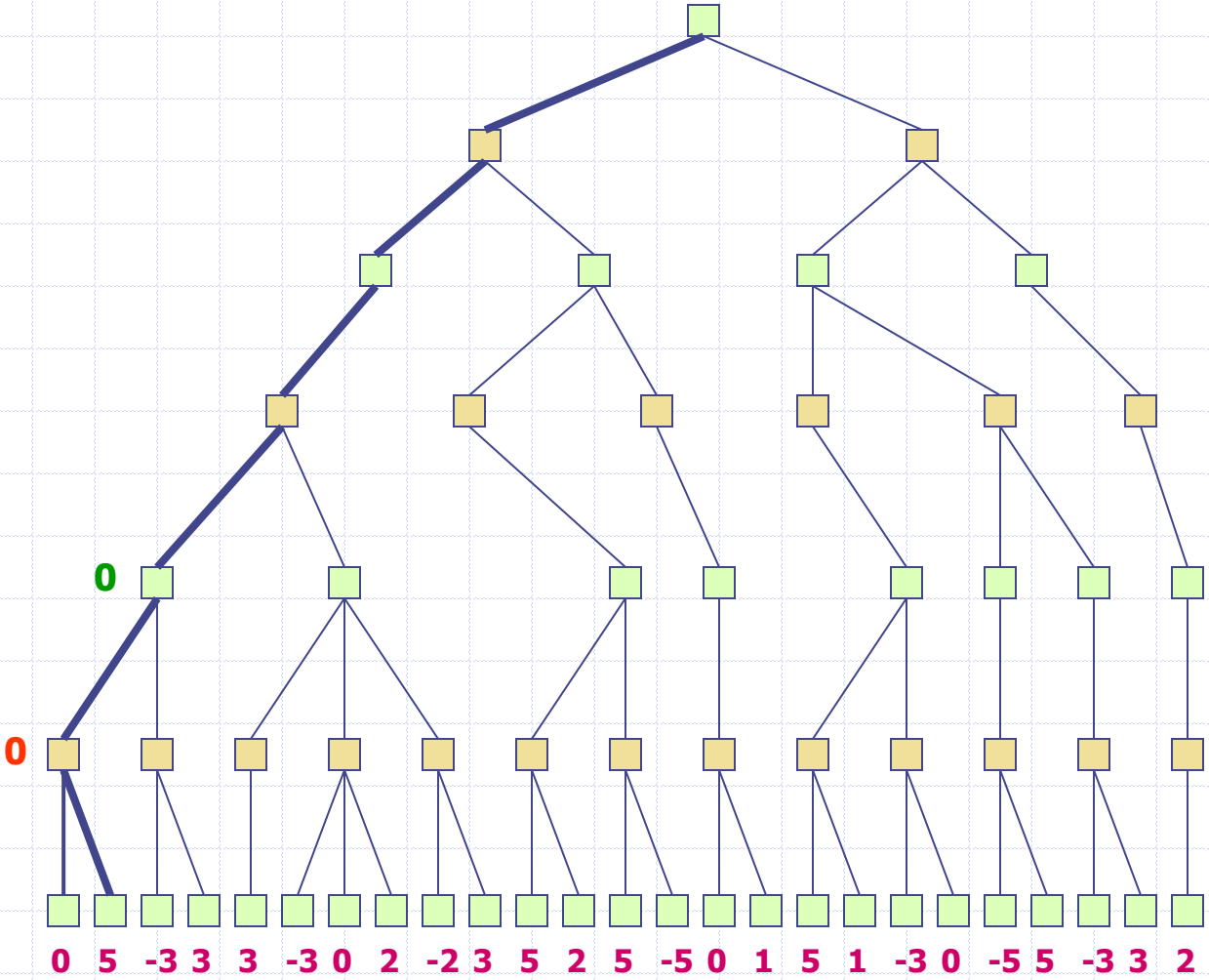
Alpha-Beta Example



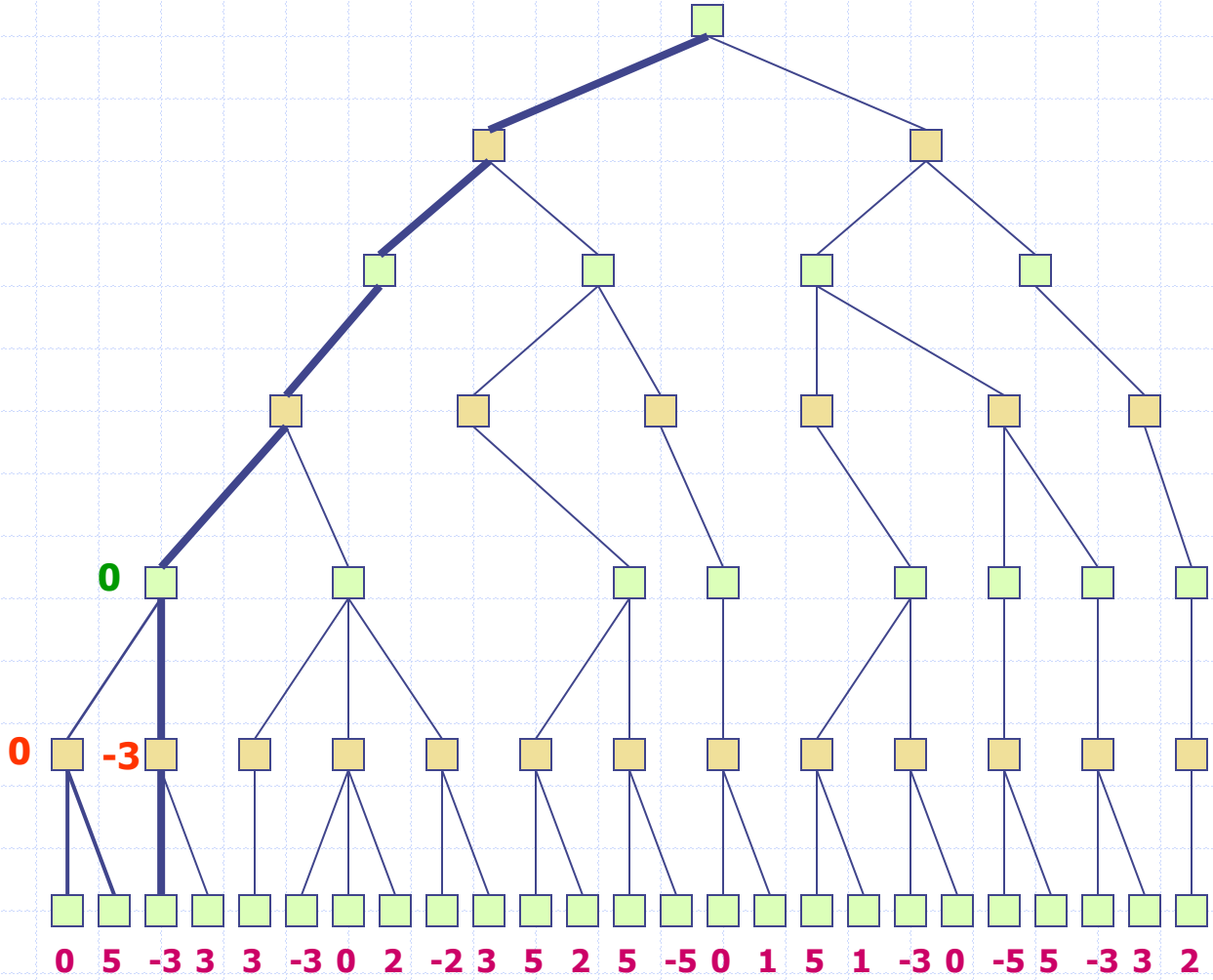
Alpha-Beta Example



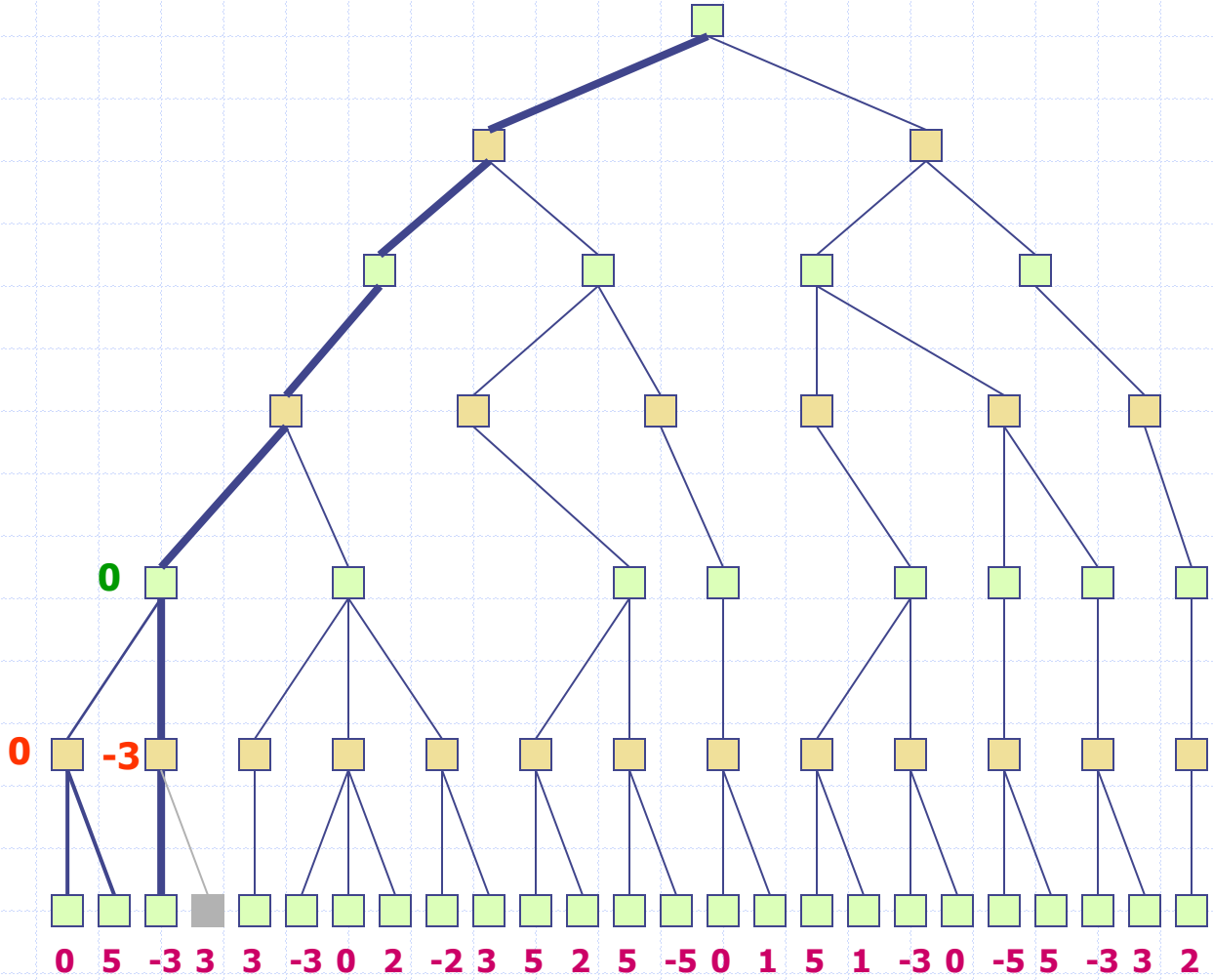
Alpha-Beta Example



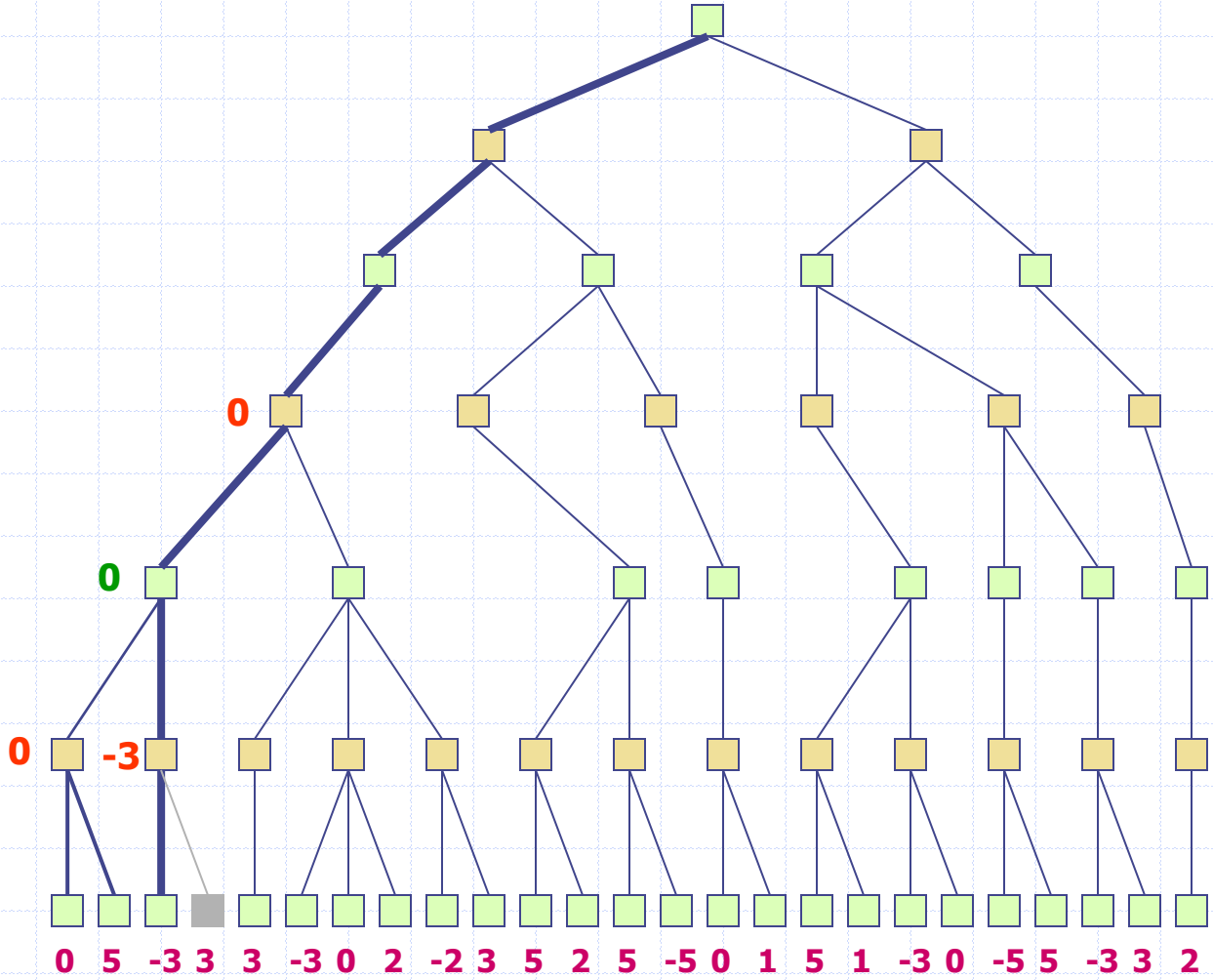
Alpha-Beta Example



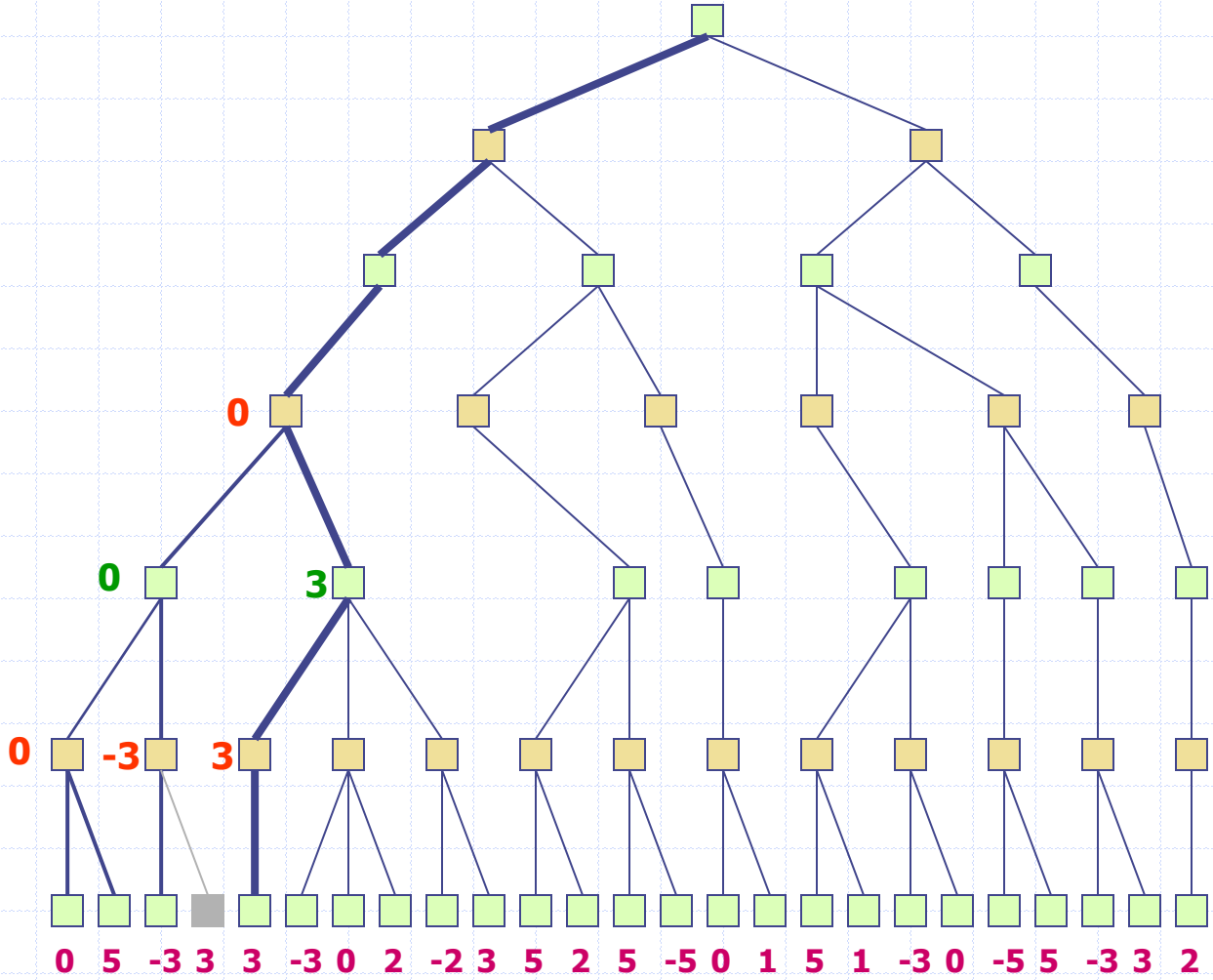
Alpha-Beta Example



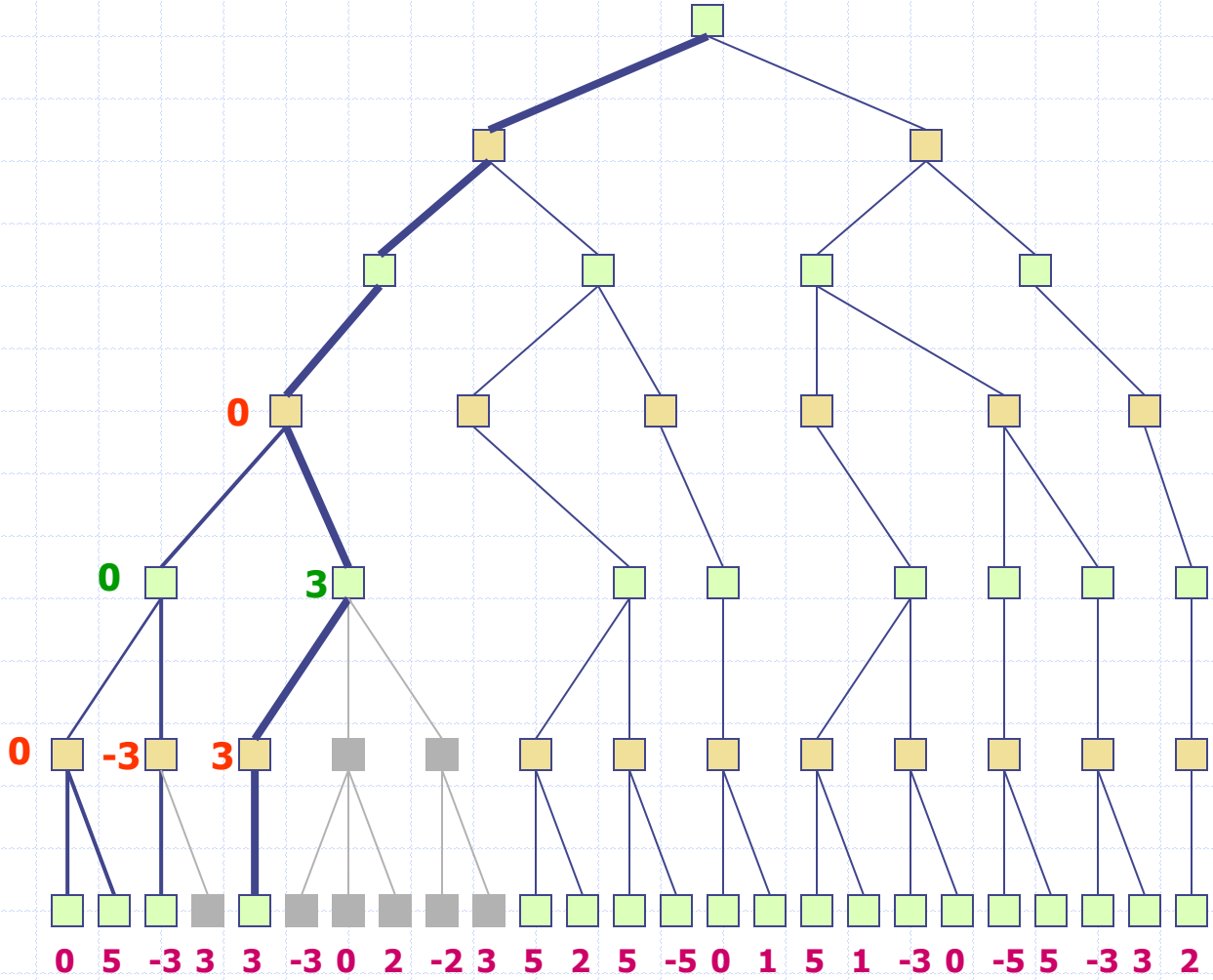
Alpha-Beta Example



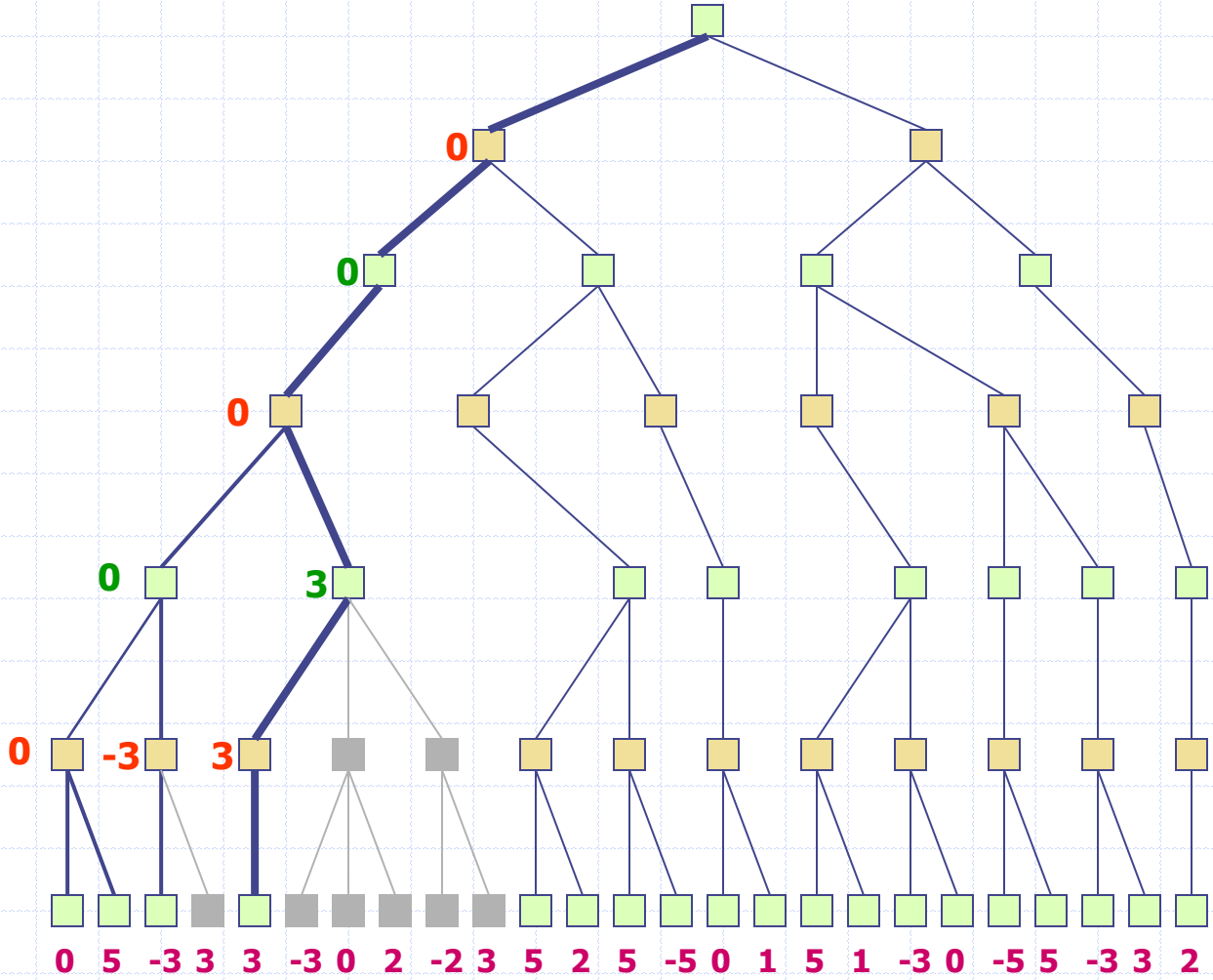
Alpha-Beta Example



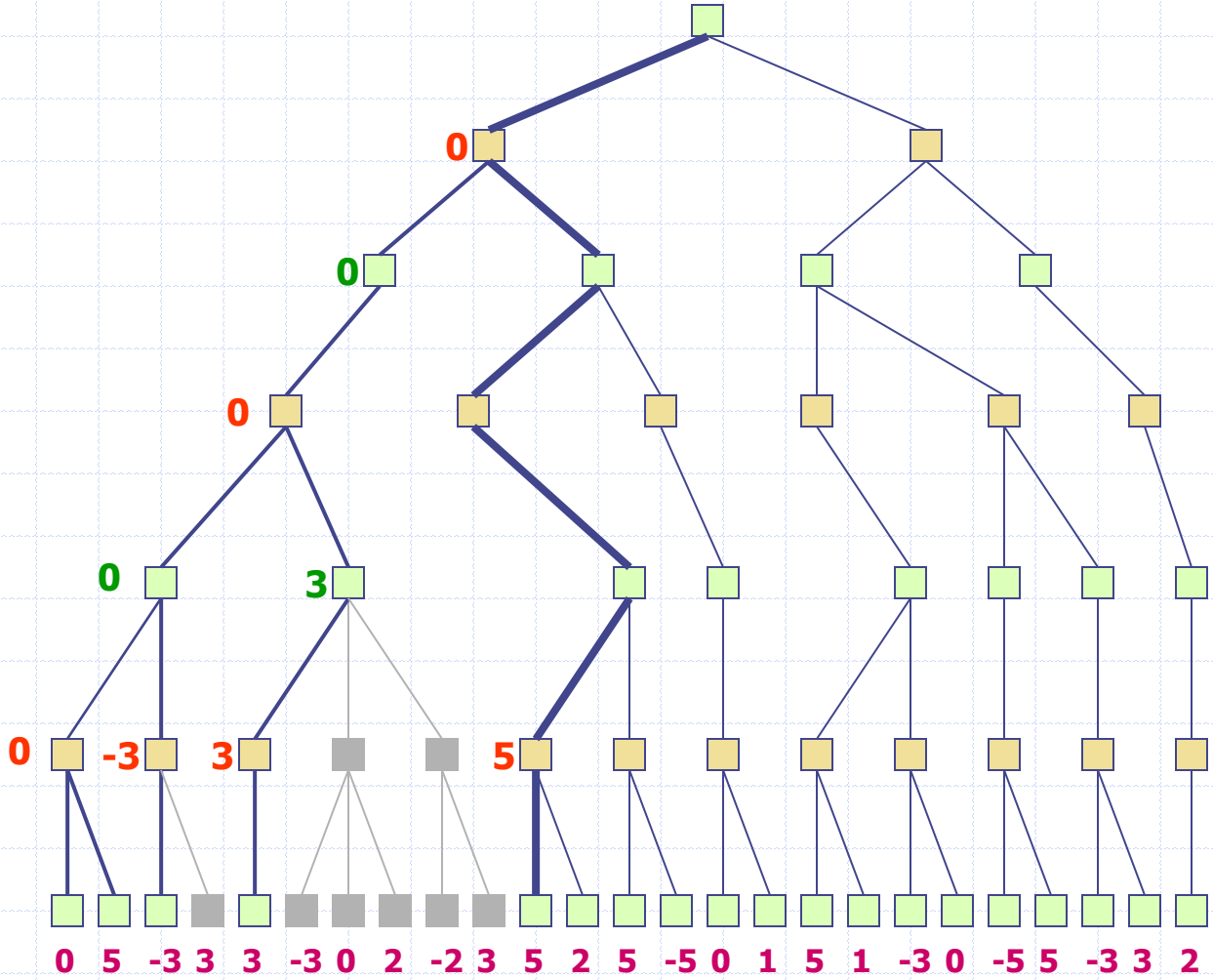
Alpha-Beta Example



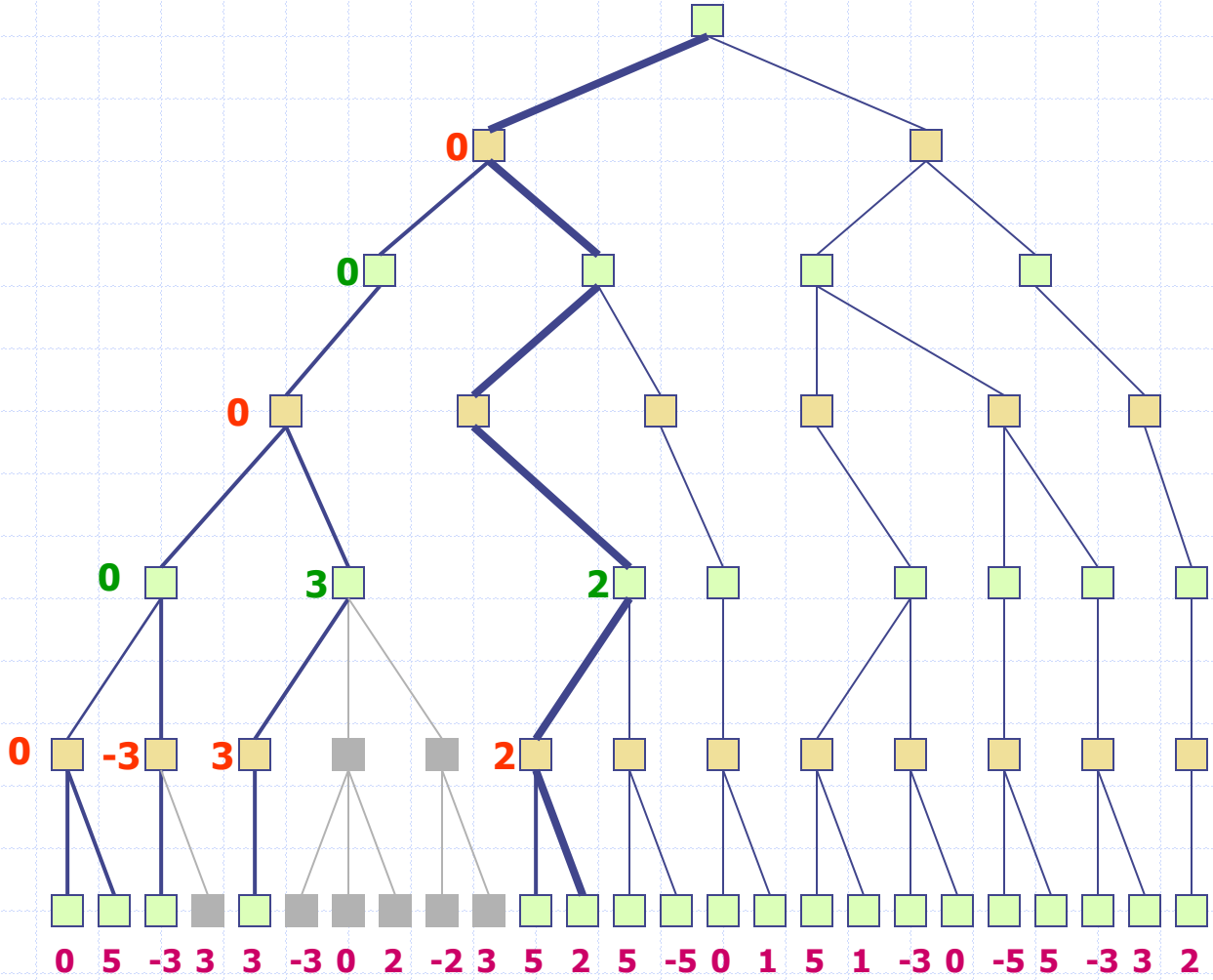
Alpha-Beta Example



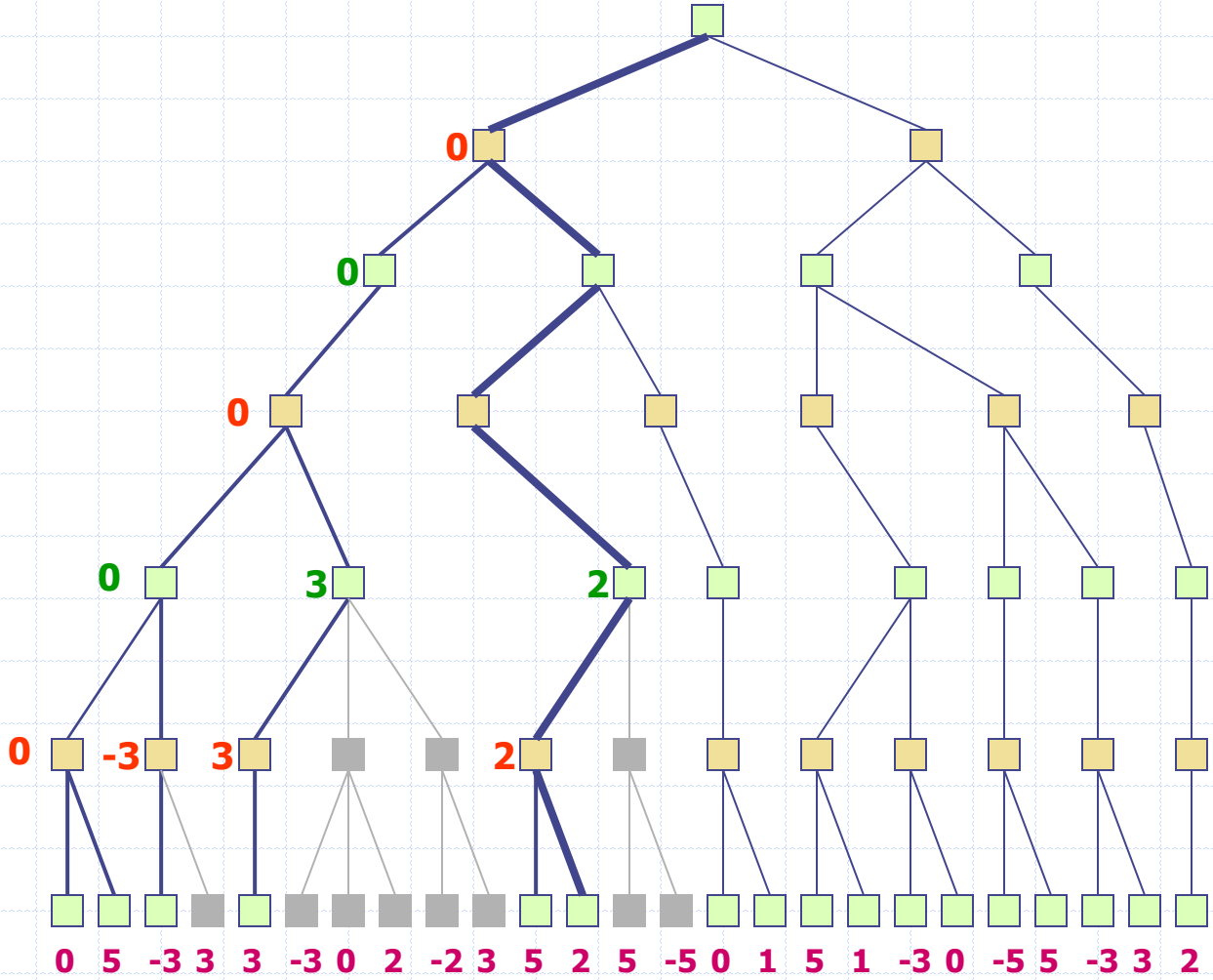
Alpha-Beta Example



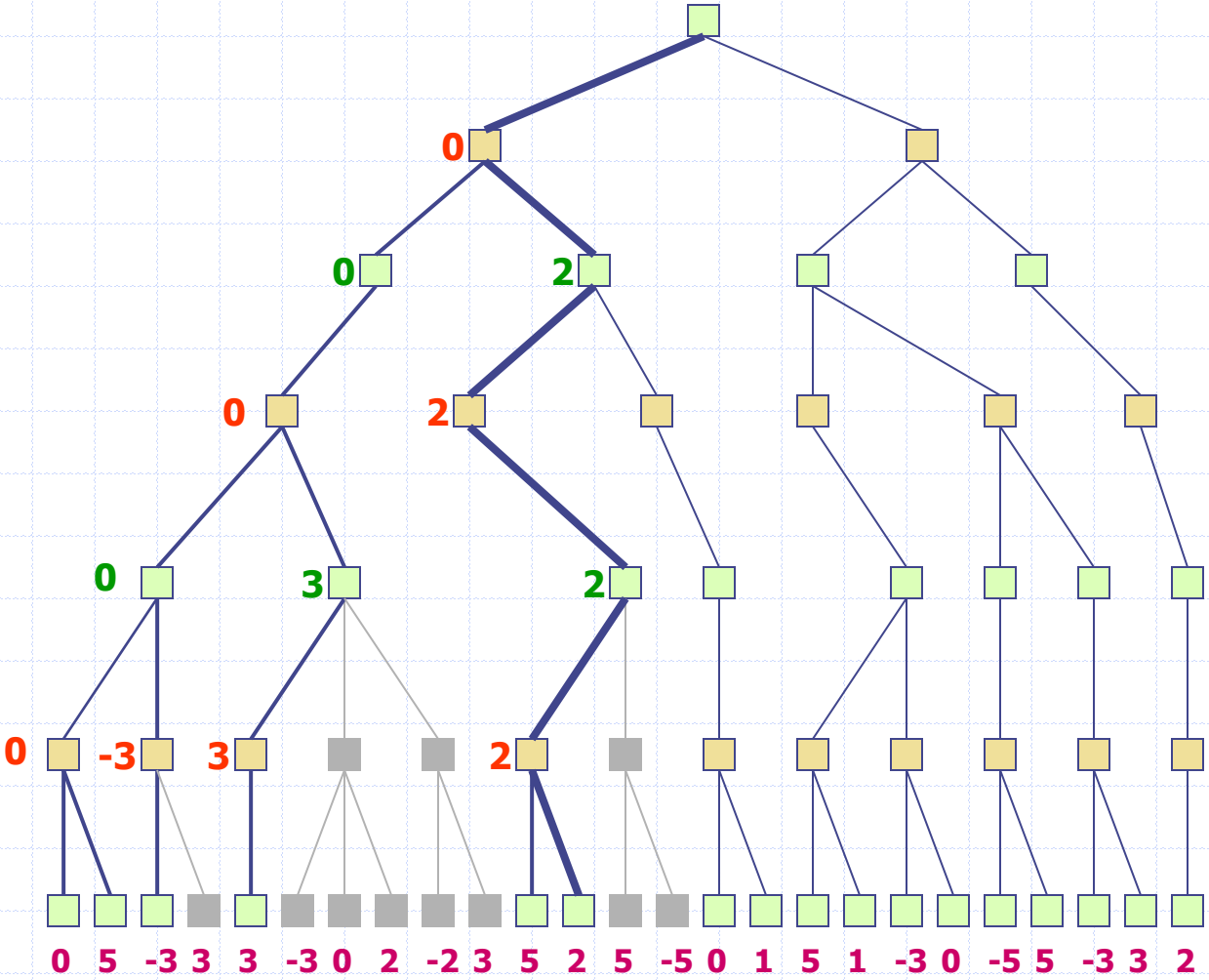
Alpha-Beta Example



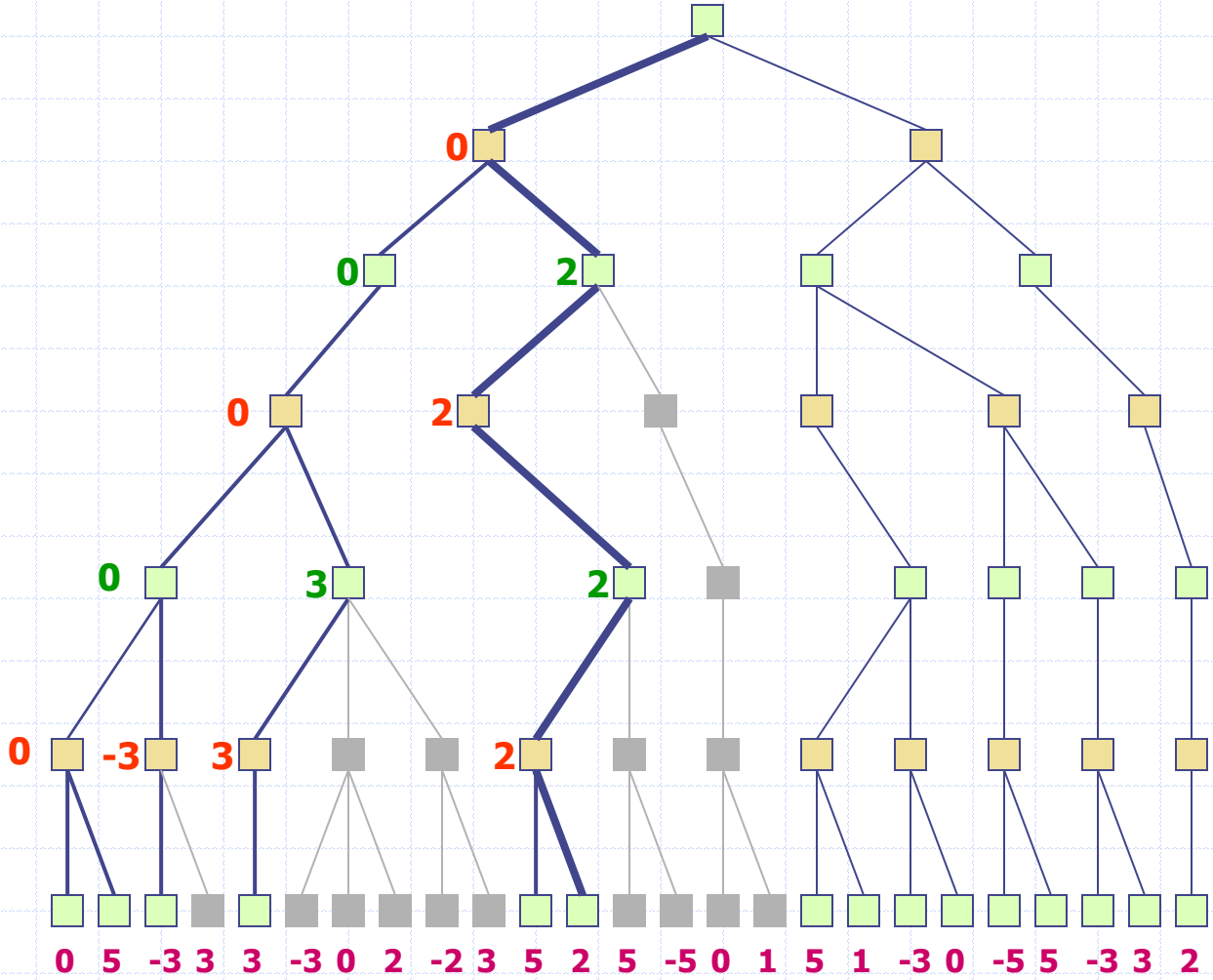
Alpha-Beta Example



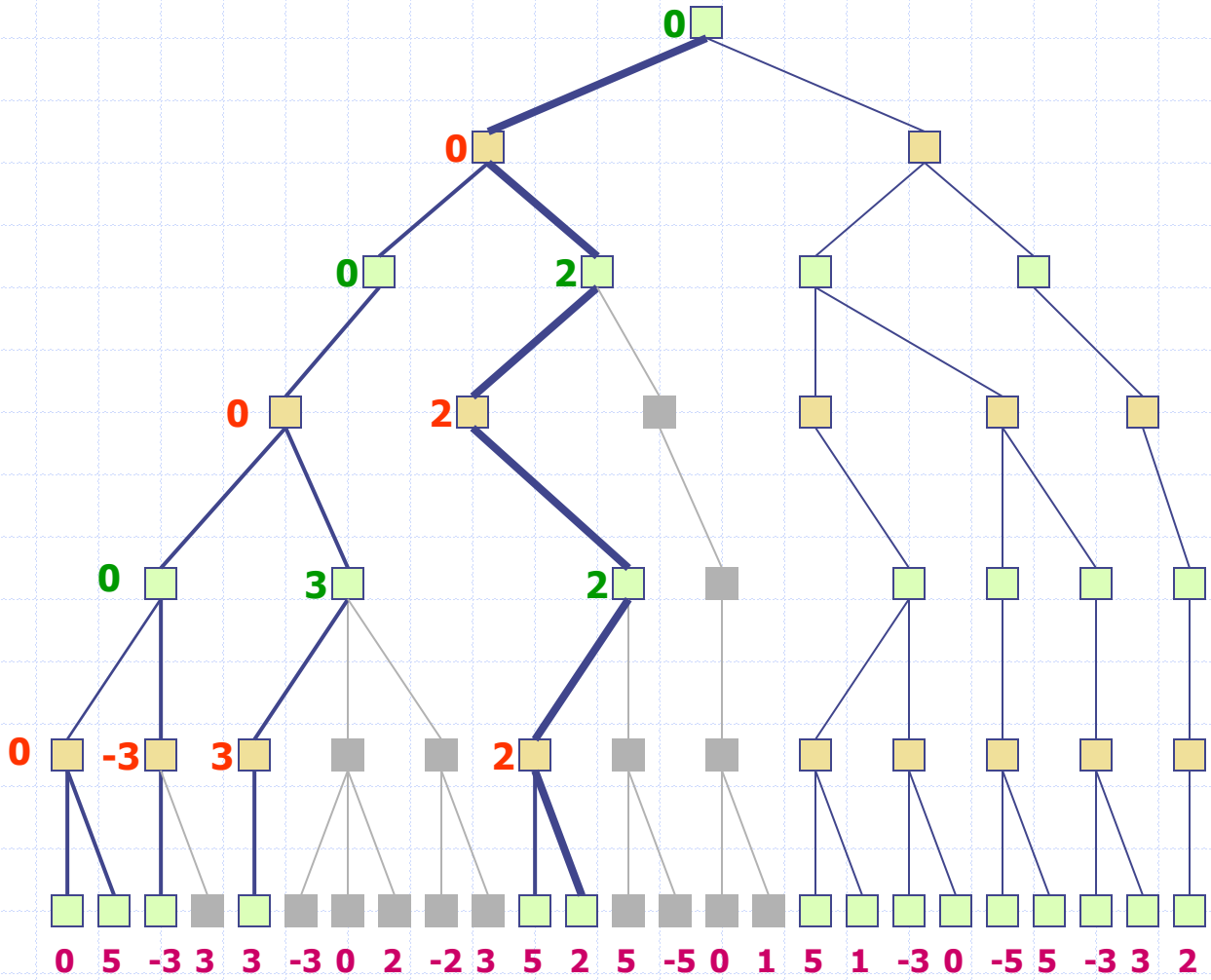
Alpha-Beta Example



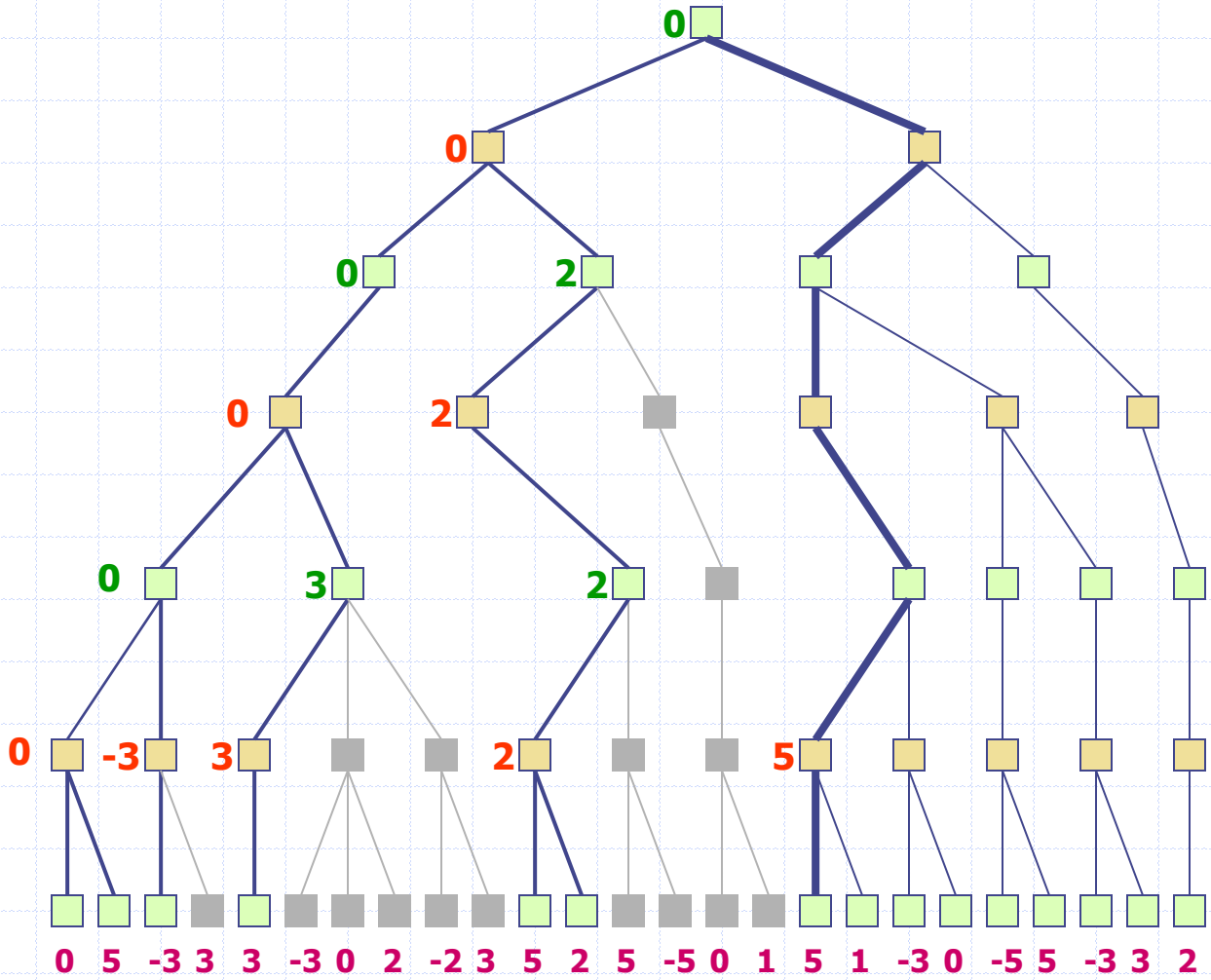
Alpha-Beta Example



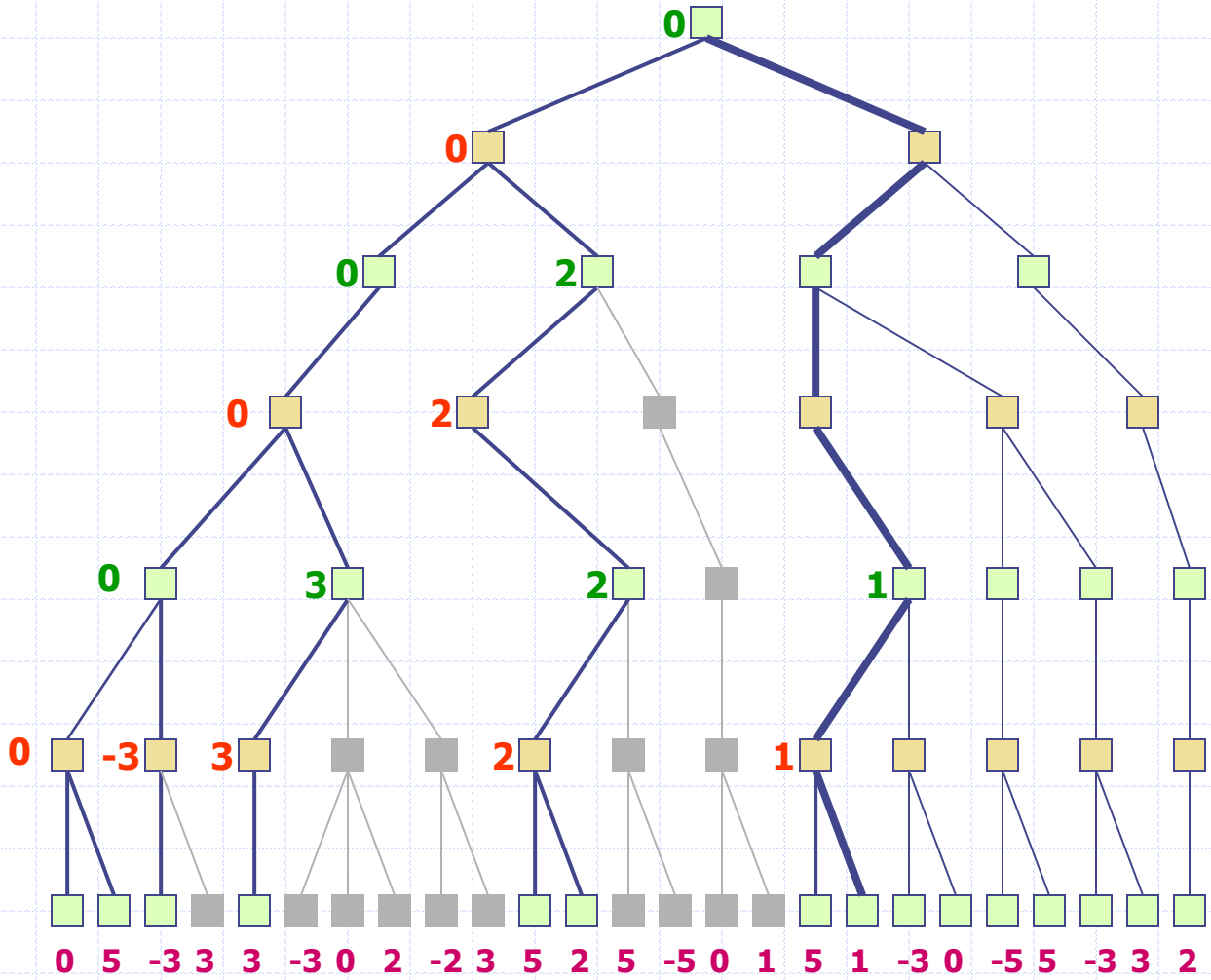
Alpha-Beta Example



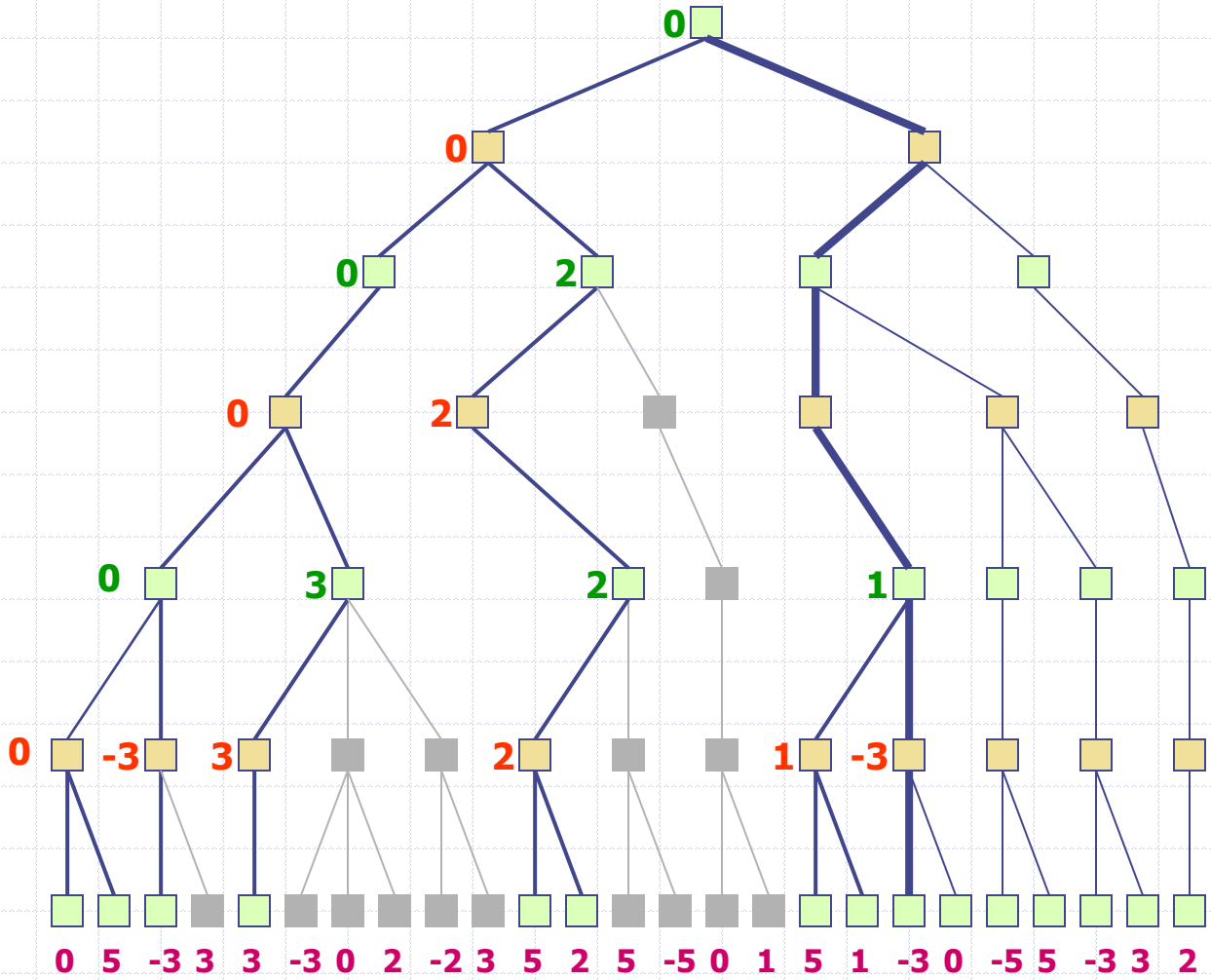
Alpha-Beta Example



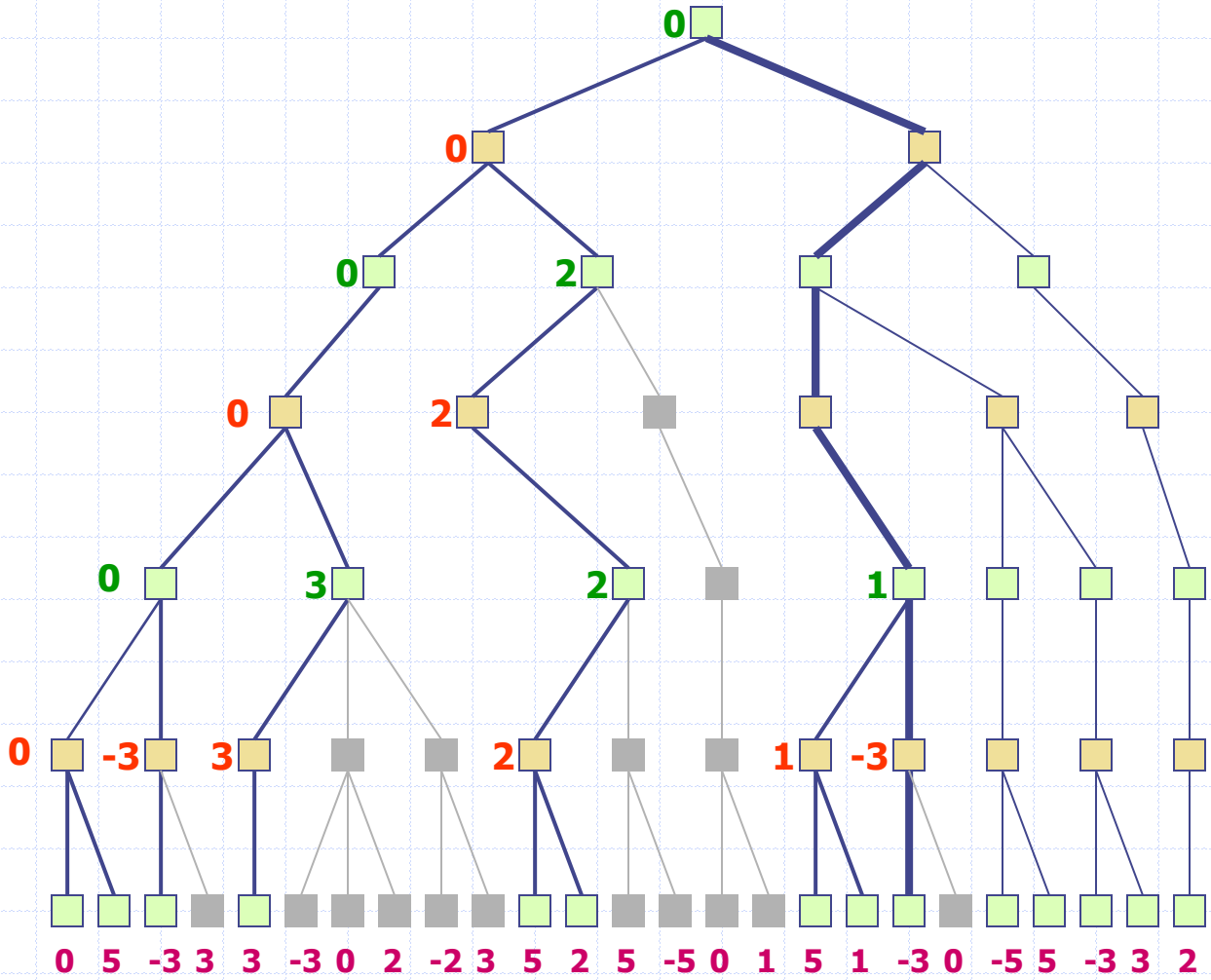
Alpha-Beta Example



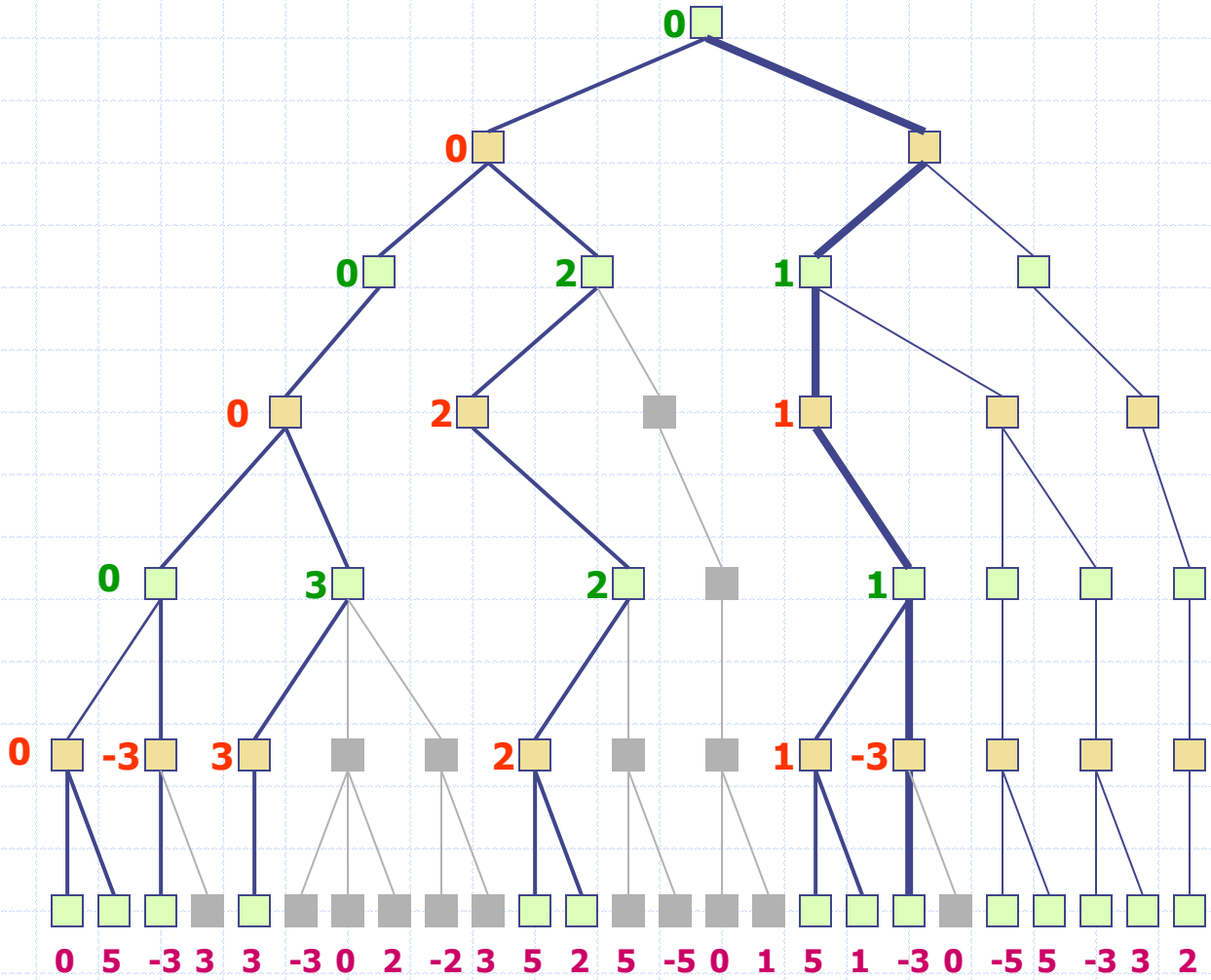
Alpha-Beta Example



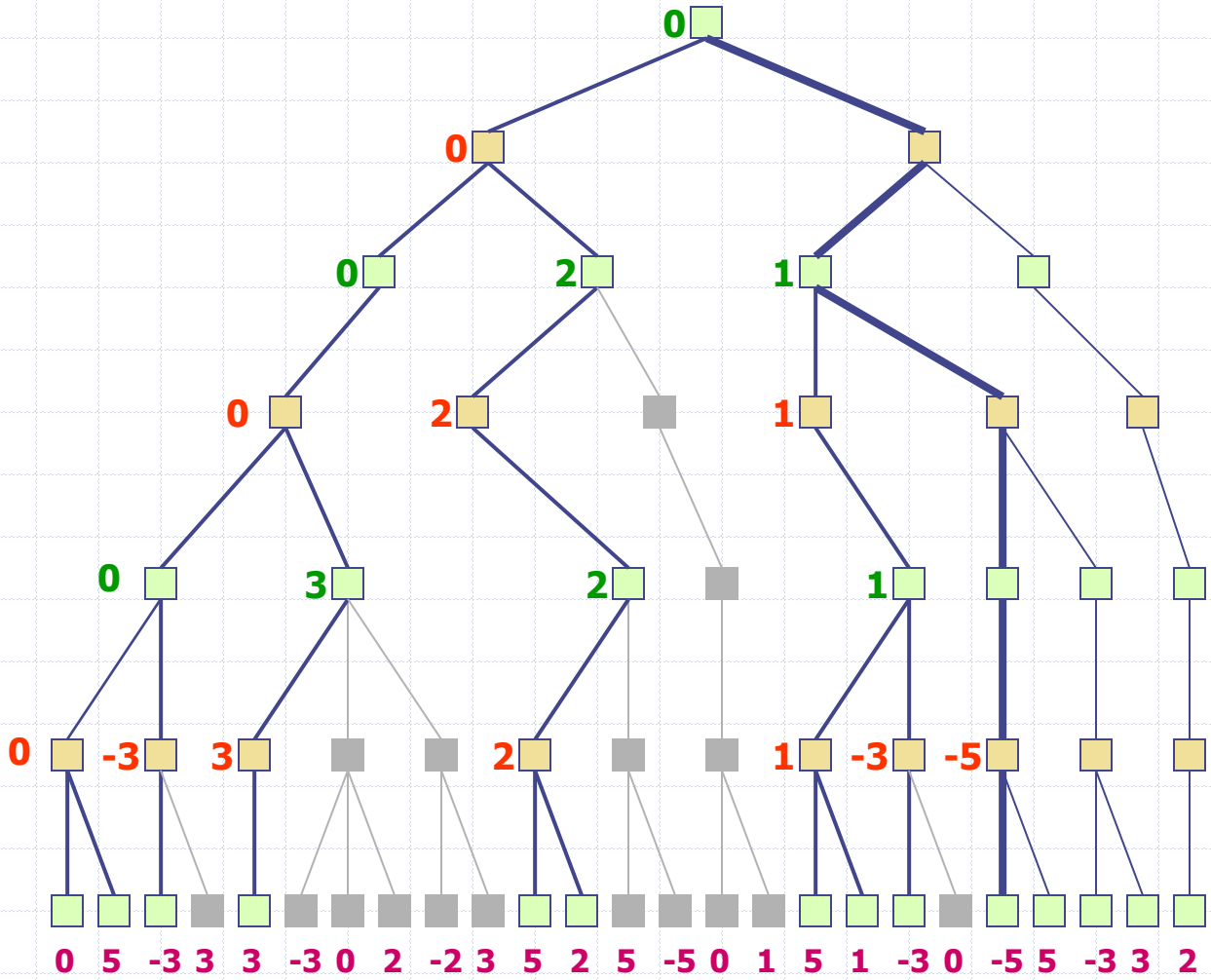
Alpha-Beta Example



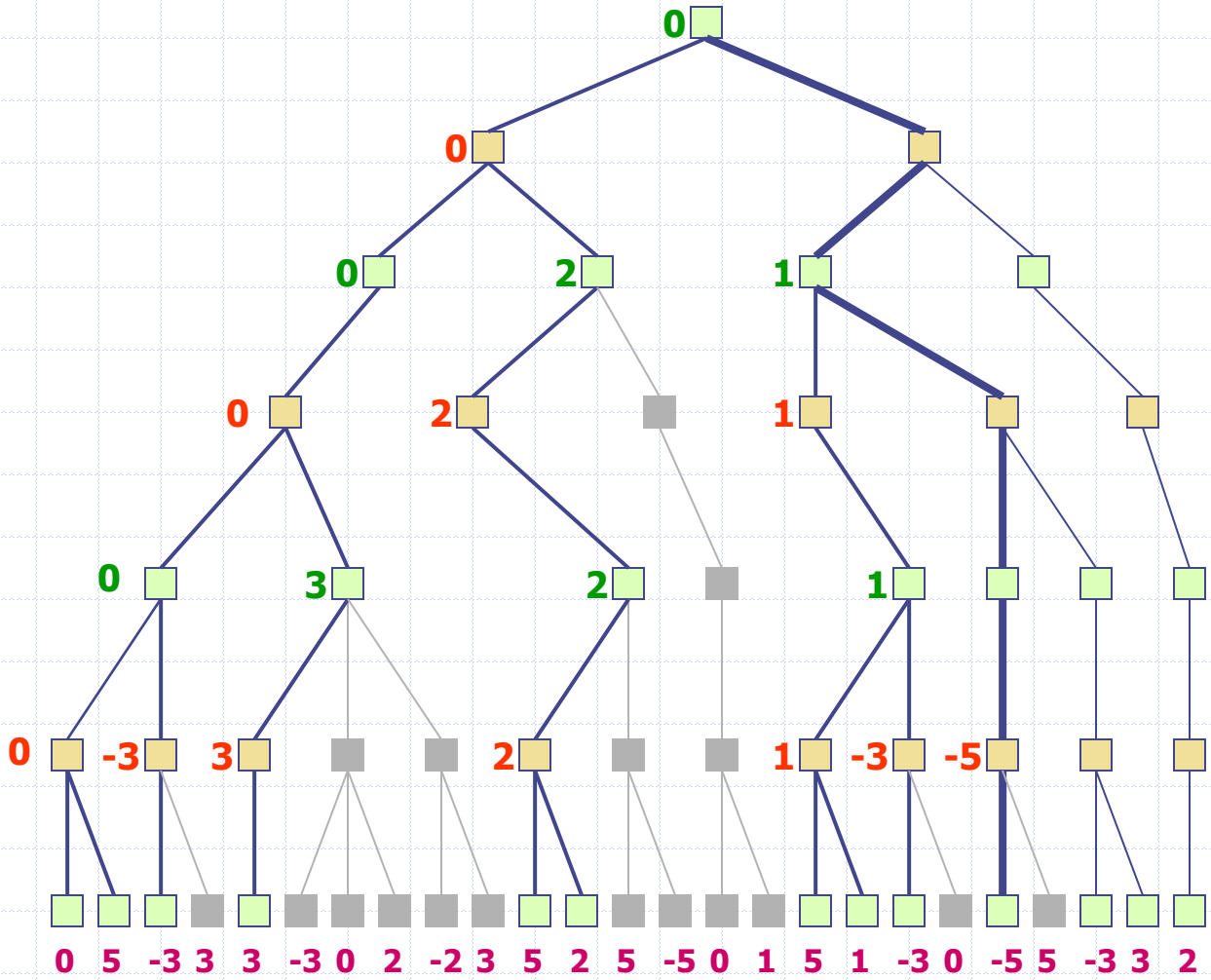
Alpha-Beta Example



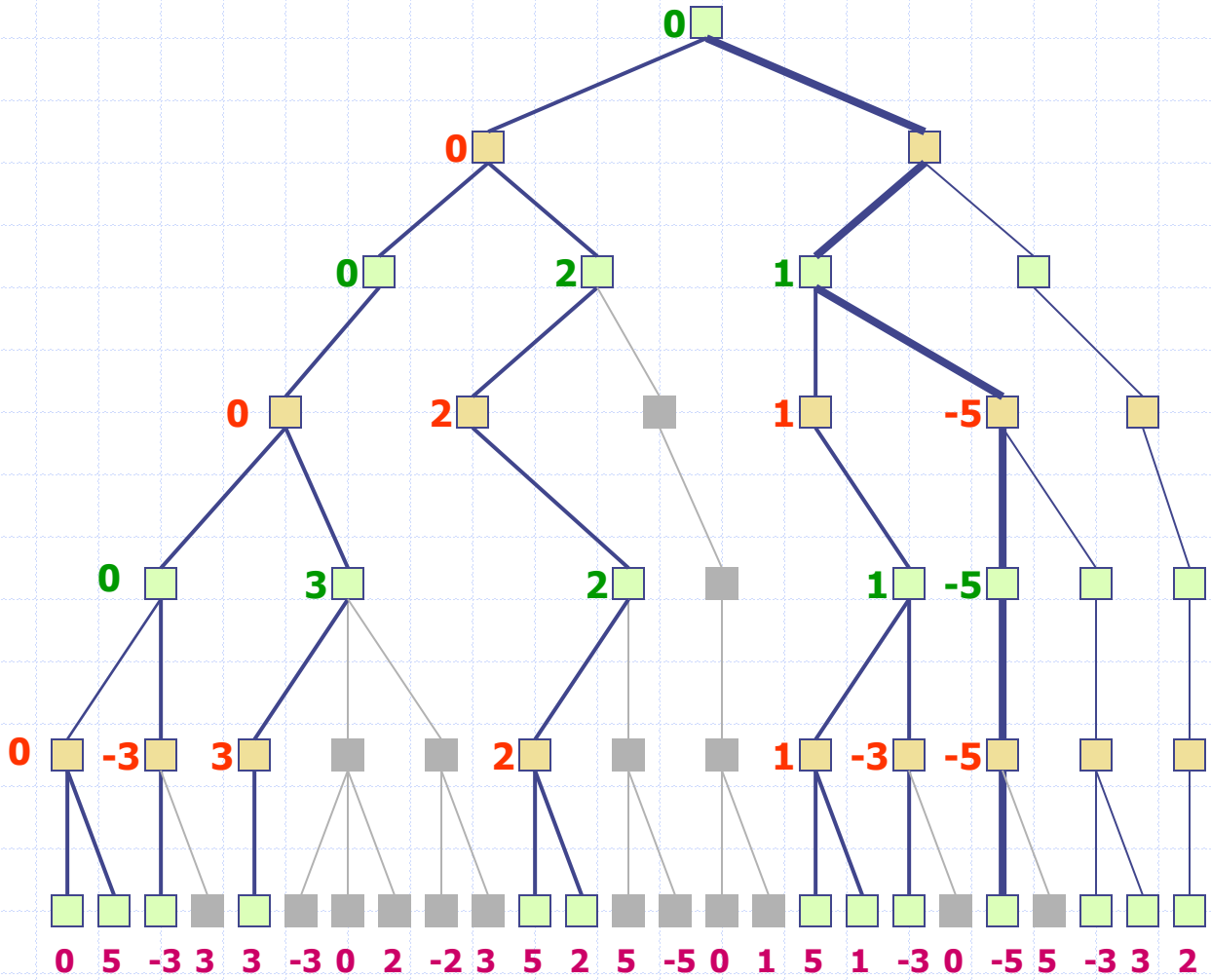
Alpha-Beta Example



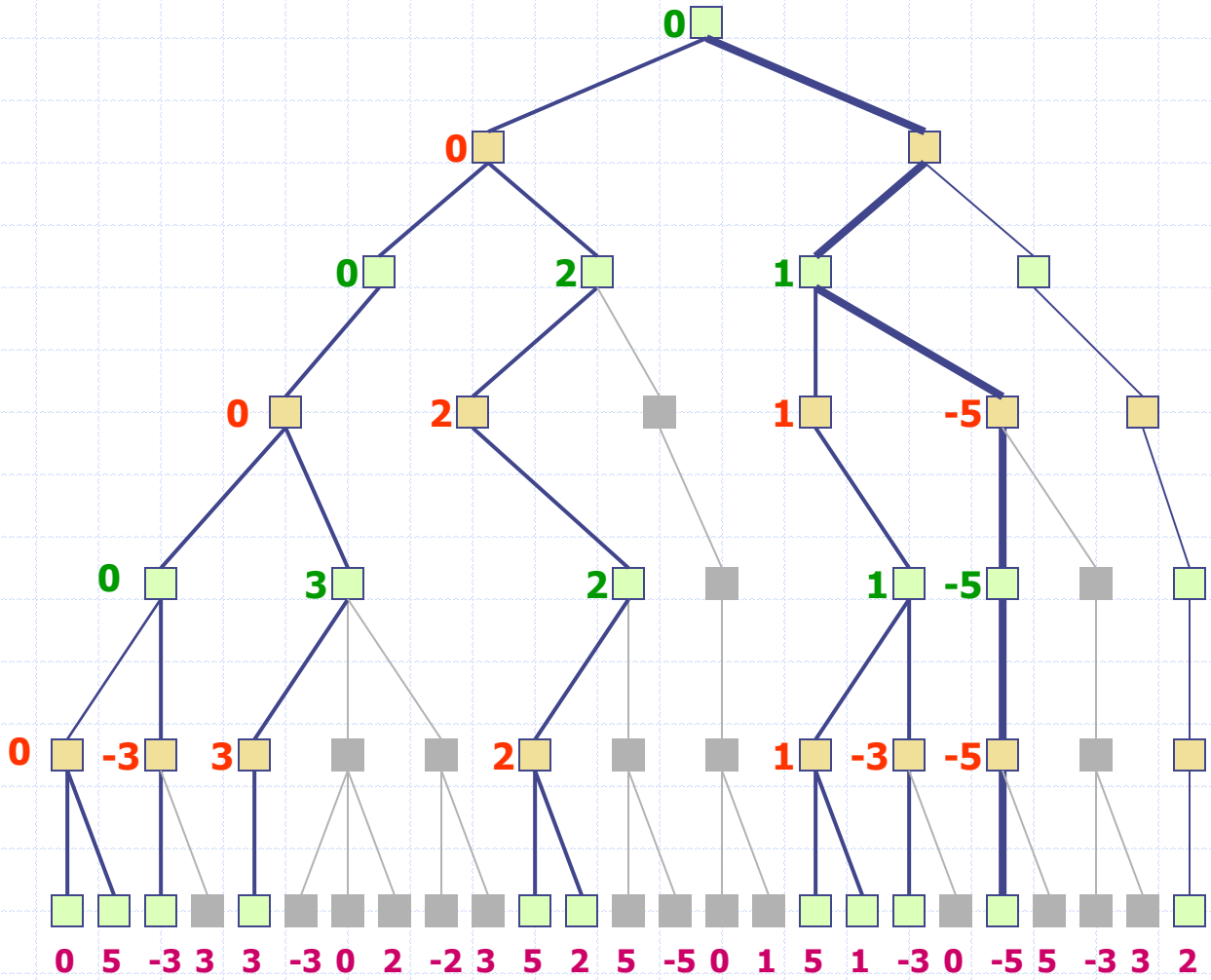
Alpha-Beta Example



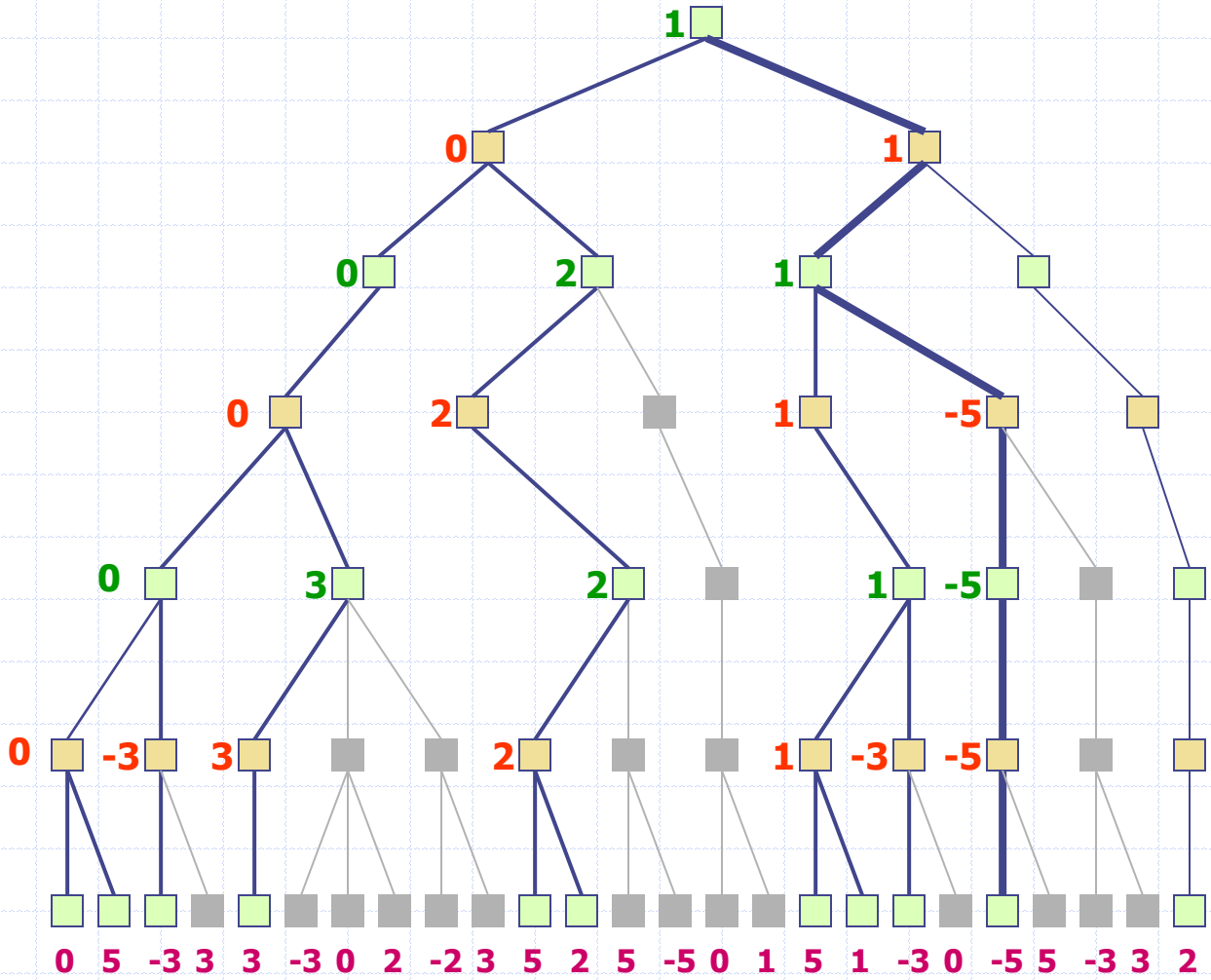
Alpha-Beta Example



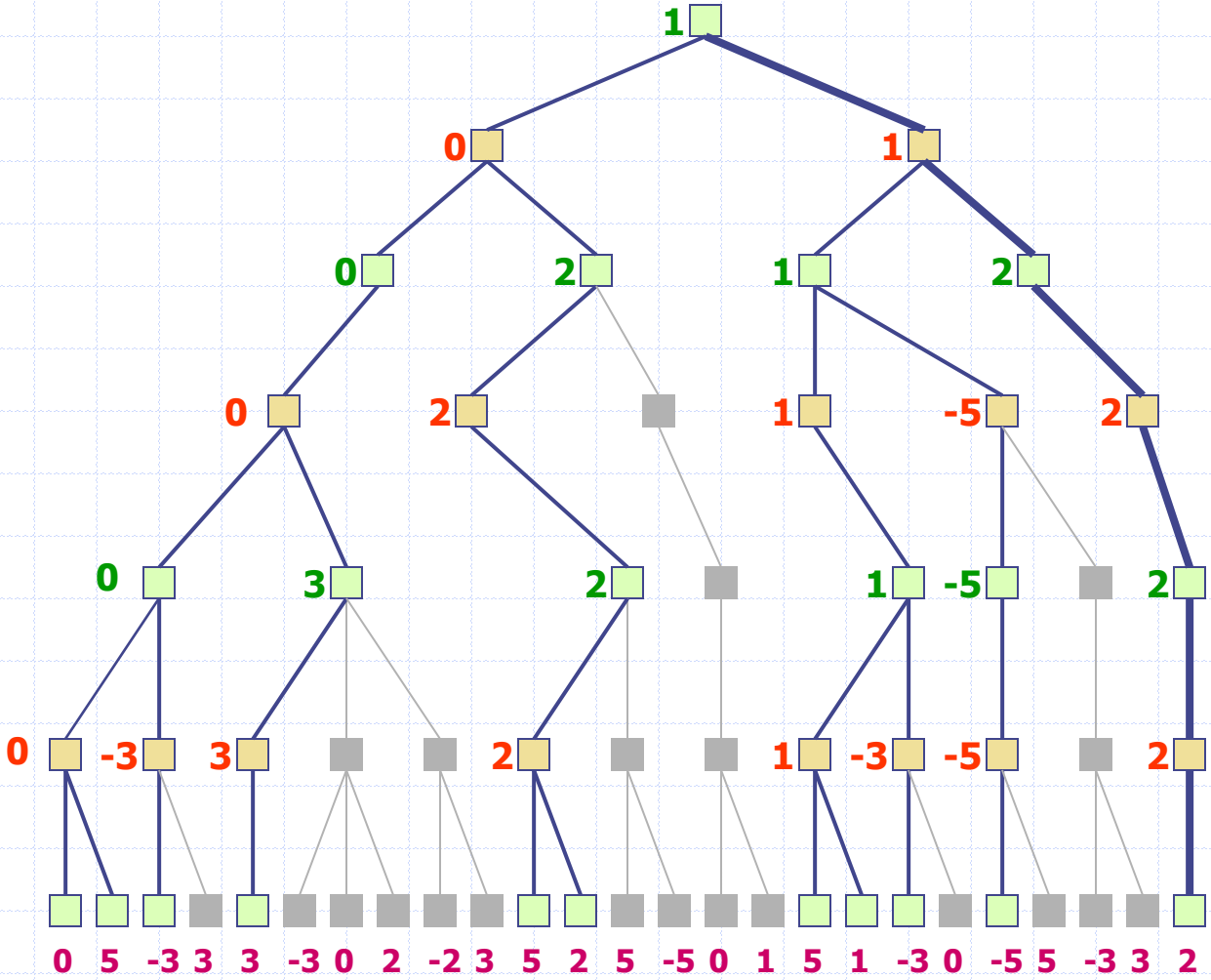
Alpha-Beta Example



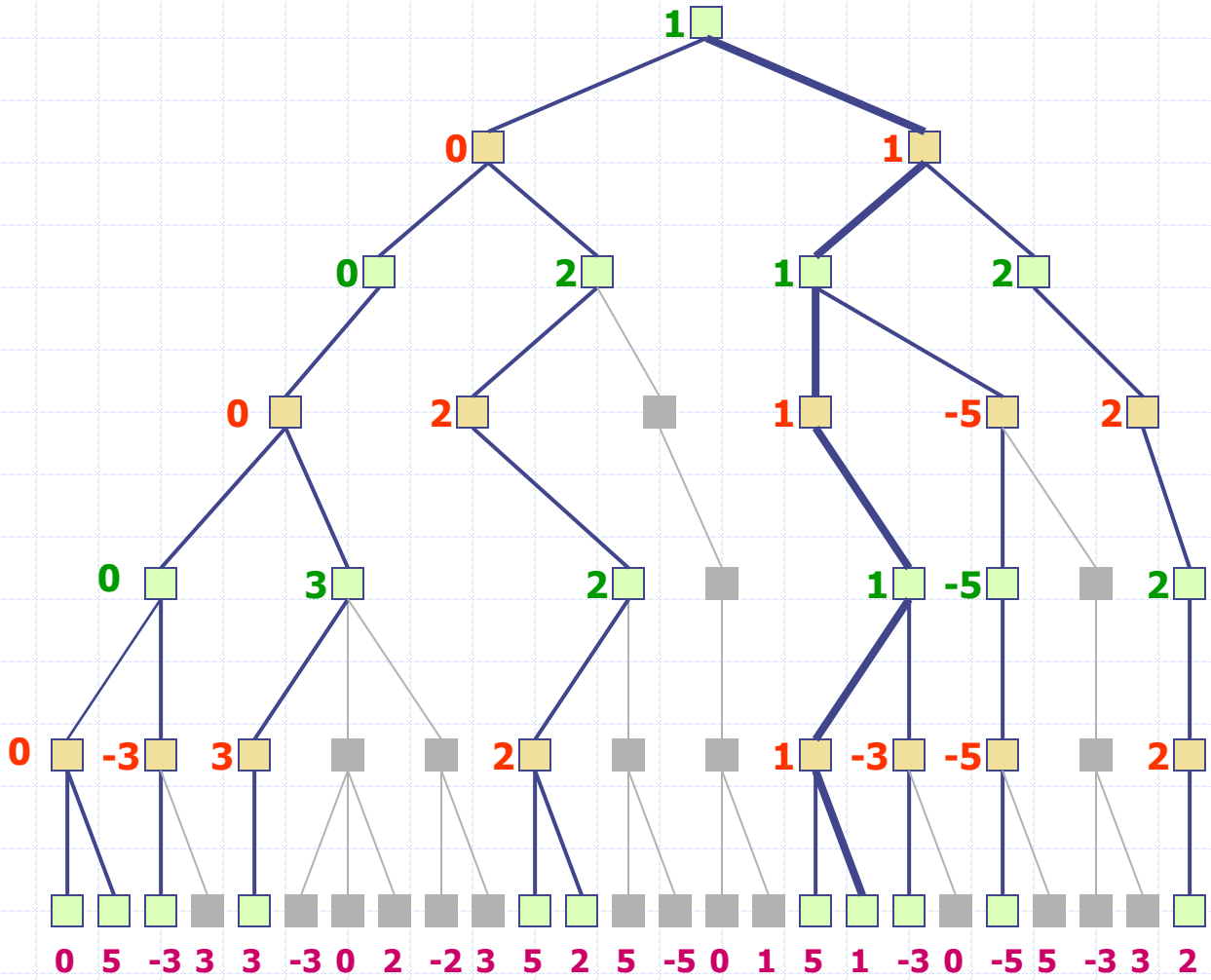
Alpha-Beta Example



Alpha-Beta Example



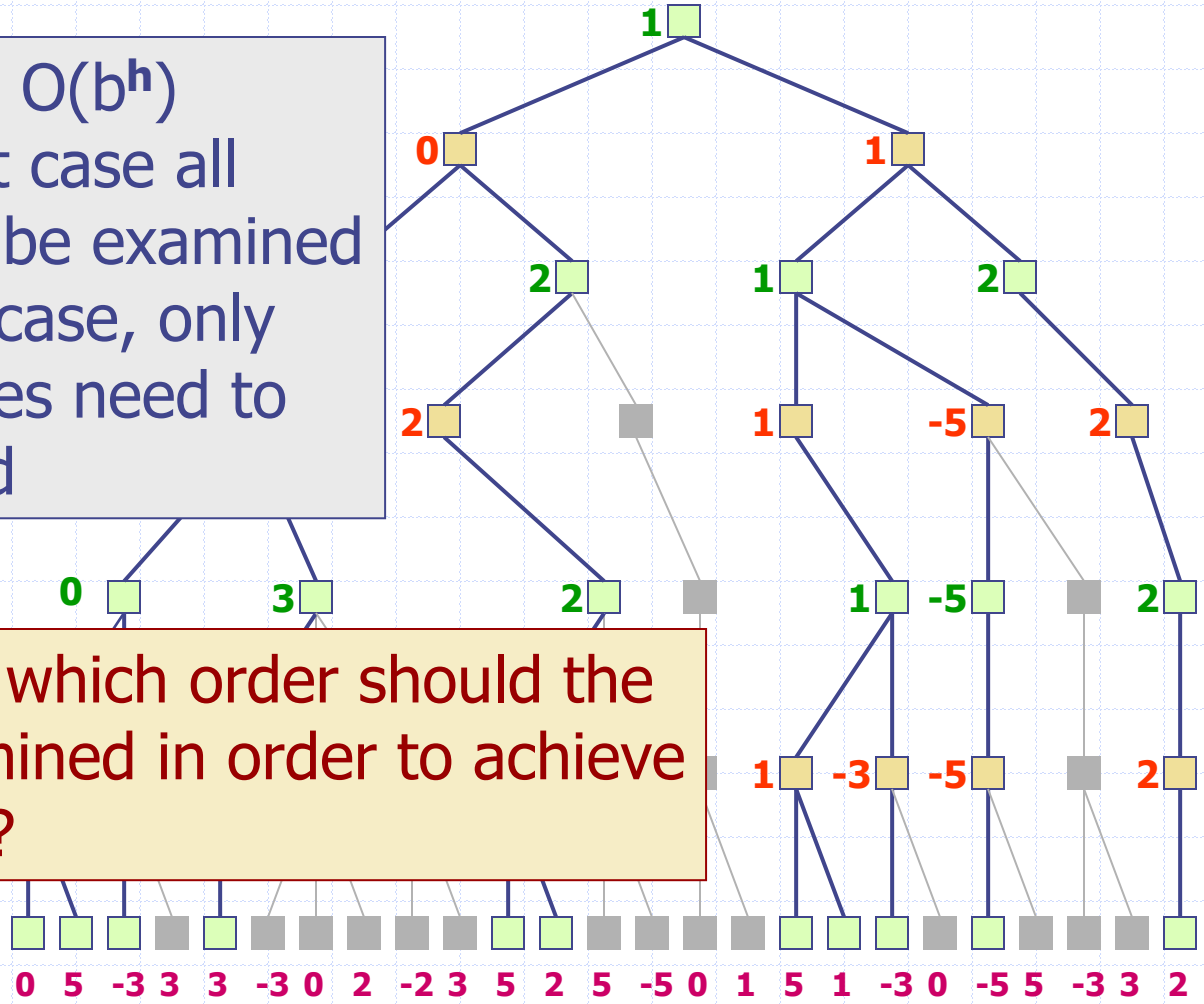
Alpha-Beta Example



How Much Do We Gain?

Size of tree = $O(b^h)$

- In the worst case all nodes must be examined
- In the best case, only $O(b^{h/2})$ nodes need to be examined



Exercise: In which order should the node be examined in order to achieve the best gain?

Alpha-Beta Procedure

- The alpha of a MAX node is a lower bound on the backed-up value
- The beta of a MIN node is a higher bound on the backed-up value
- Update the alpha/beta of the parent of a node N when all search below N has been completed or discontinued

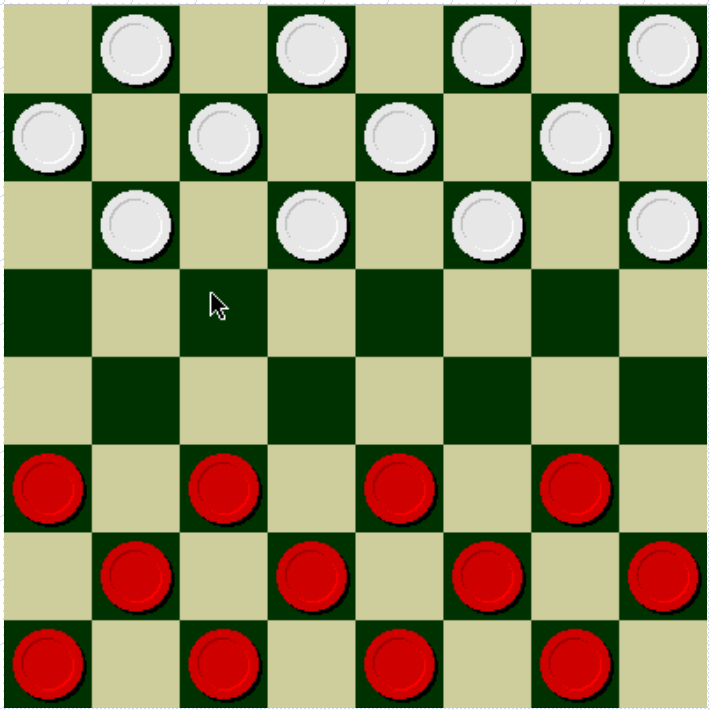
Alpha-Beta Procedure

- The alpha of a MAX node is a lower bound on the backed-up value
- The beta of a MIN node is a higher bound on the backed-up value
- Update the alpha/beta of the parent of a node N when all search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its alpha is ≥ beta of a MIN ancestor of N
- Discontinue the search below a MIN node N if its beta is ≤ alpha of a MAX ancestor of N

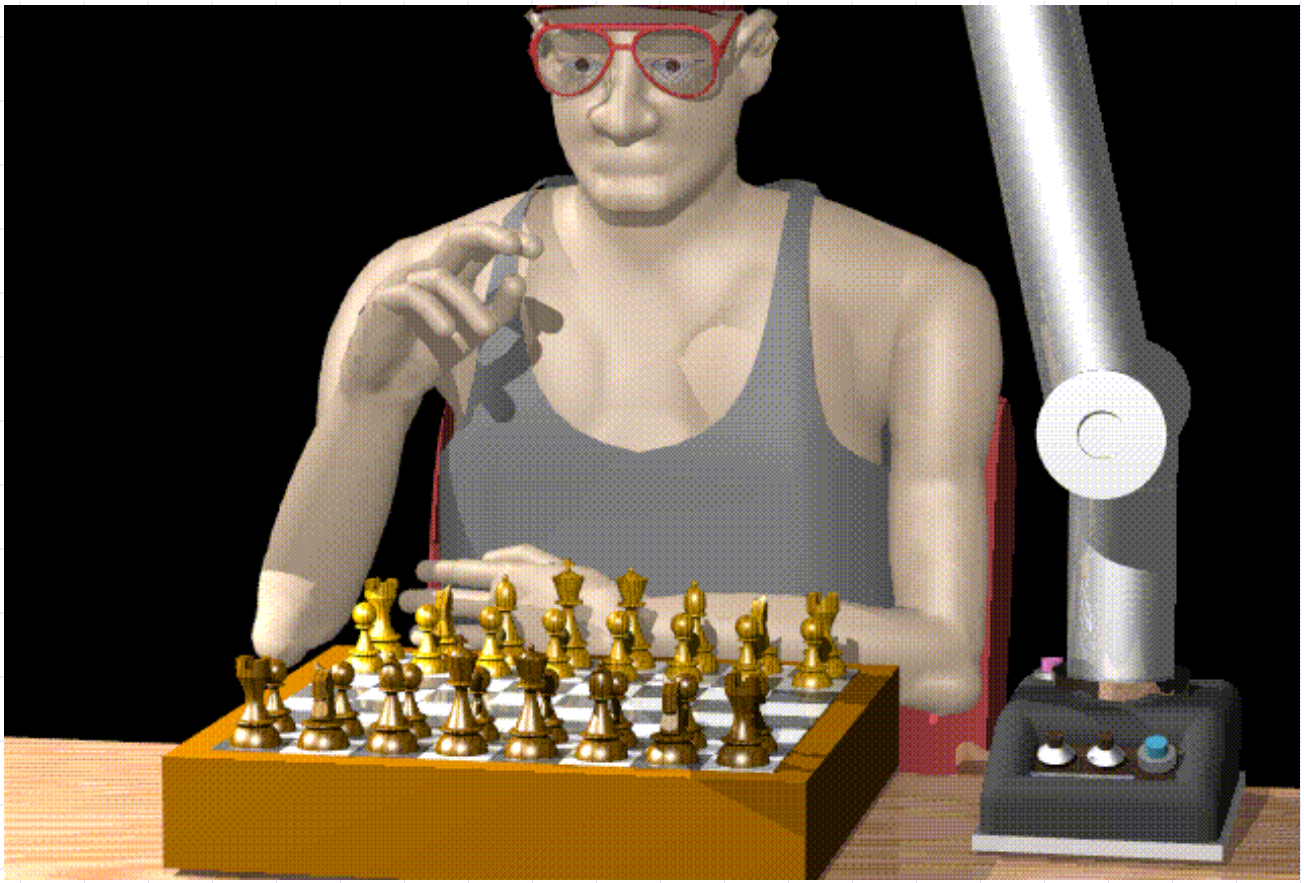
Alpha-Beta + ...

- Iterative deepening
- Singular extensions

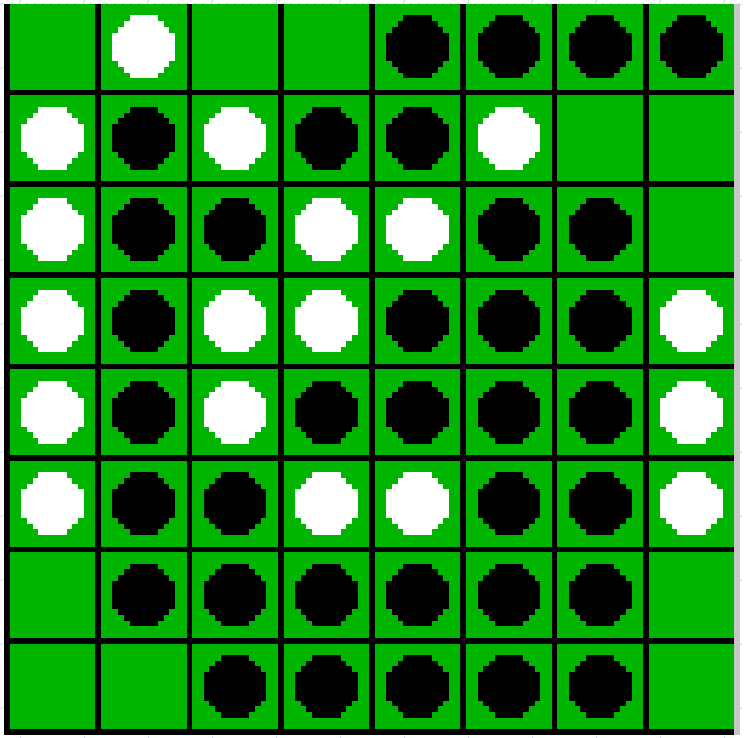
Checkers



Chess



Reversi/Othello



Summary

- Two-players game as a domain where action models are uncertain
- Optimal decision in the worst case
- Game tree
- Evaluation function / backed-up value
- Minimax procedure
- Alpha-beta procedure