Game Playing

Game Playing

Why do AI researchers study game playing?

1. It’s a good reasoning problem, formal and nontrivial.

2. Direct comparison with humans and other computer programs is easy.

Why study games?

- Interesting, hard problems which require minimal “initial structure”
- Clear criteria for success
- Offer an opportunity to study problems involving {hostile, adversarial, competing} agents and the uncertainty of interacting with the natural world
- Historical reasons: For centuries humans have used them to exert their intelligence
- Fun, good, easy to understand
- Games often define very large search spaces
  - chess 35^{100} nodes in search tree, 10^{40} legal states

What Kinds of Games?

Mainly games of strategy with the following characteristics:

1. Sequence of moves to play
2. Rules that specify possible moves
3. Rules that specify a payment for each move
4. Objective is to maximize your payment

Games vs. Search Problems

- Unpredictable opponent → specifying a move for every possible opponent reply
- Time limits → unlikely to find goal, must approximate

Games

A game can be formally defined as a kind of search problem with 04 components:

1. The *initial state*, which includes the board position & identifies the player to move
2. A *successor function*, which returns a list of (move, state) pairs, each indicating a legal move & resulting state
3. A *terminal test*, which determines when the game is over. States where the game has ended are called terminal states
4. A *utility function*, which gives a numeric values for the terminal states.

   - In chess, the outcome is a win, loss, or draw, with values +1, -1, or 0.

   -Backgammon (+192~-192)

Game Trees

- Game trees are used to represent two-player games.
- Alternate moves in the game are represented by alternate levels in the tree.
- Nodes in the tree represent positions.
- Edges between nodes represent moves.
- Leaf nodes represent won, lost or drawn positions.

Game Tree (2-player, Deterministic, Turns)

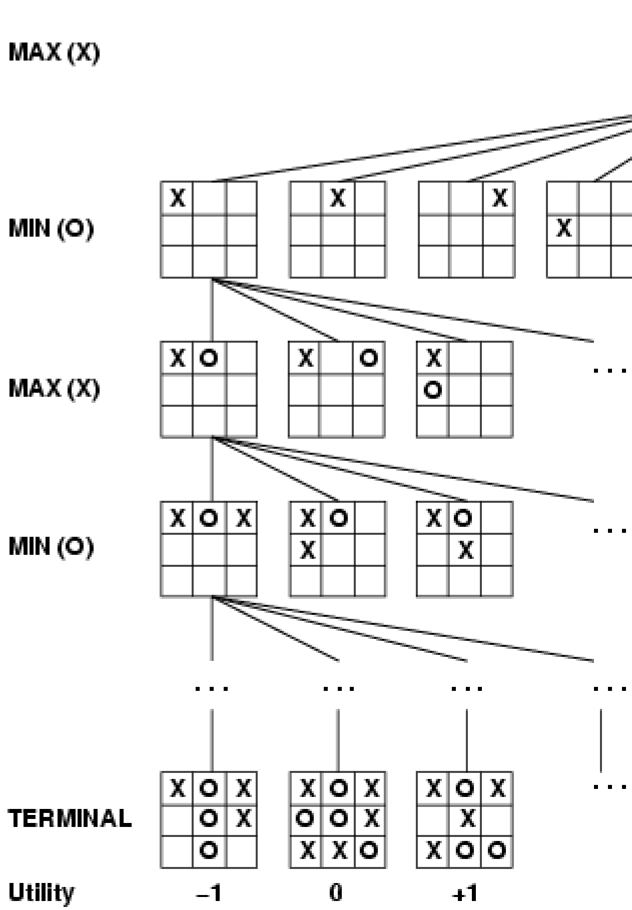
computer's turn

opponent's turn

computer's turn

opponent's turn

leaf nodes are evaluated



The computer is **Max**.
The opponent is **Min**.

At the leaf nodes, the **utility function** is employed.

- f(n) = +1 if the position is a win for X.
- f(n) = -1 if the position is a win for O.
- f(n) = 0 if the position is a draw.

Assumptions

In talking about game playing systems, we make a number of assumptions:

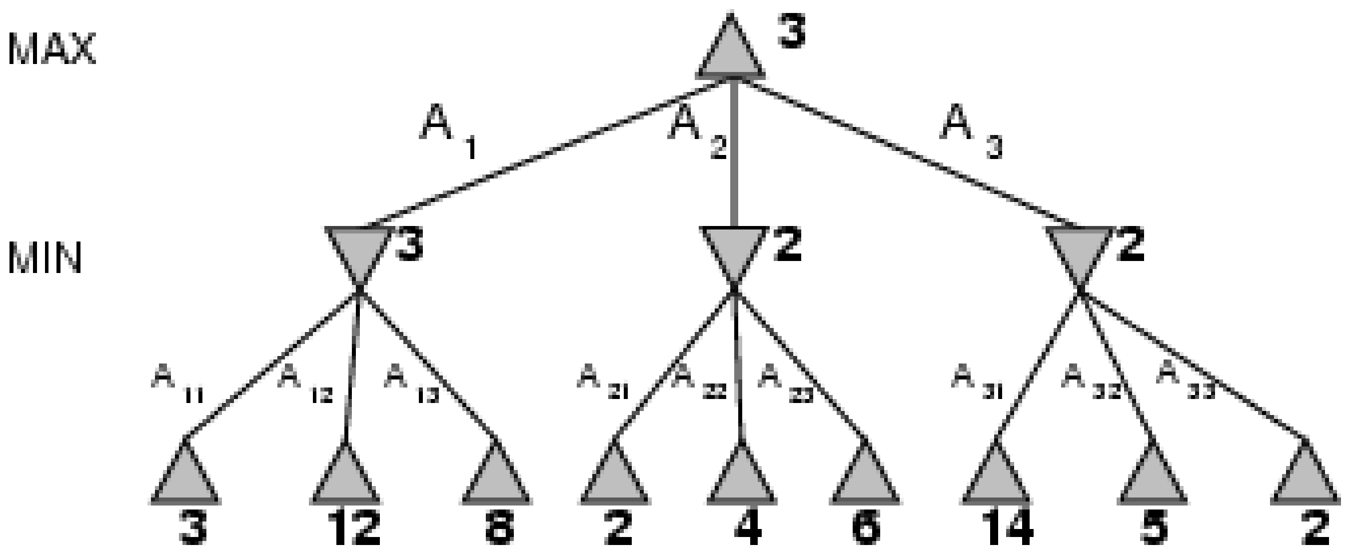
- The opponent is rational – will play to win.
- The game is zero-sum – if one player wins, the other loses.
- Usually, the two players have complete knowledge of the game. For games such as poker, this is clearly not true.

Mini-Max Terminology

- utility function: the function applied to leaf nodes
- backed-up value
  - of a max-position: the value of its largest successor
  - of a min-position: the value of its smallest successor
- minimax procedure: search down several levels; at the bottom level apply the utility function, back-up values all the way up to the root node, and that node selects the move.

Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game:



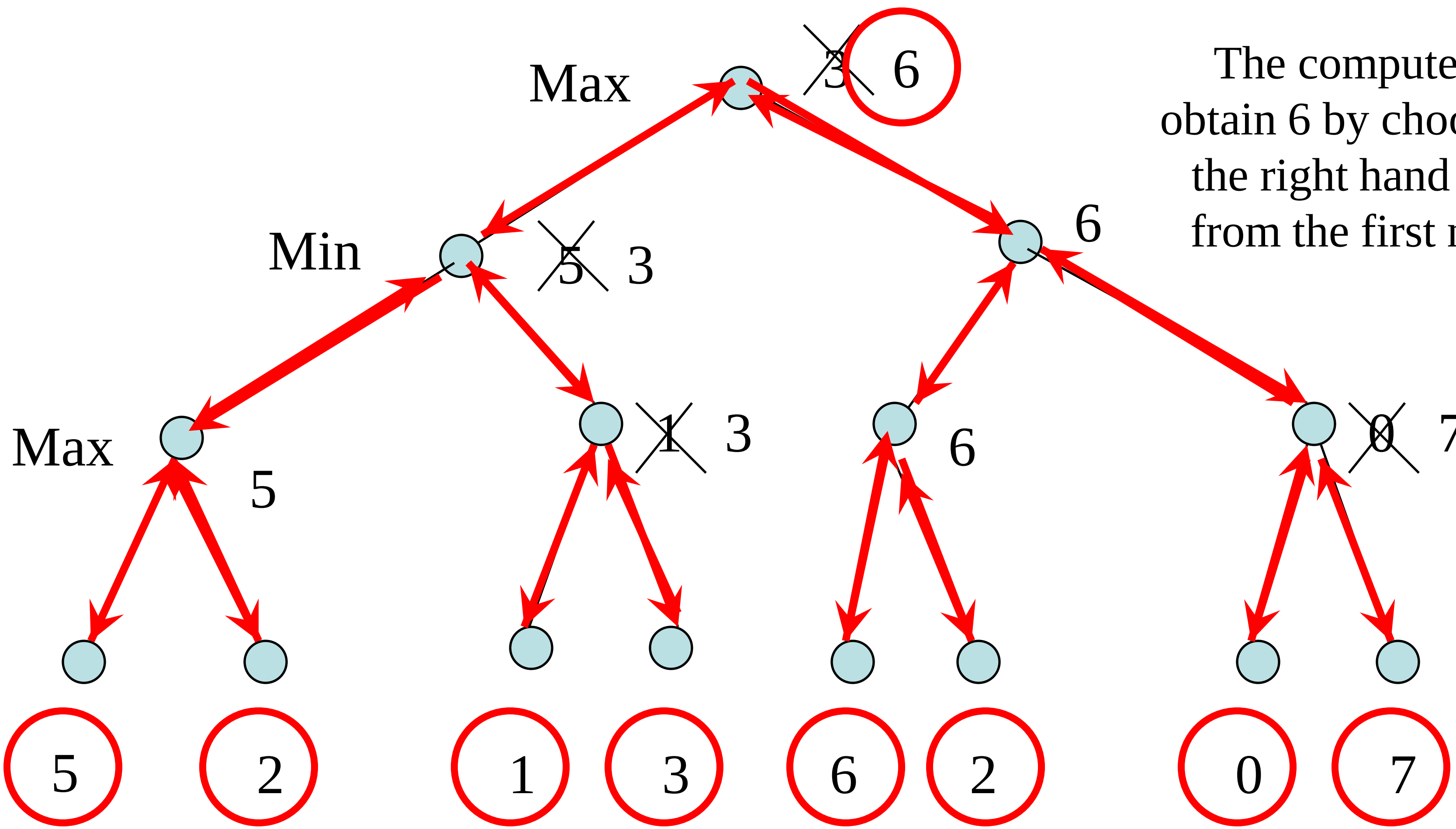
Minimax Strategy

- Why do we take the min value every other level of the tree?
  - These nodes represent the opponent's choice of move.
  - The computer assumes that the human will choose that move that is of least value to the computer.

Minimax Function

- MINIMAX-VALUE(n) =
  - UTILITY(n) if n is a terminal state
  - $\max_{s \in Successors(n)}$ MINIMAX-VALUE(s) if n is a MAX node
  - $\min_{s \in Successors(n)}$ MINIMAX-VALUE(s) if n is a MIN node

Minimax – Animated Example



The computer can obtain 6 by choosing the right hand edge from the first node.

Minimax algorithm

function MINIMAX-DECISION(*state*) **returns** *an action*

  $v \leftarrow$ MAX-VALUE(*state*)
  return the *action* in SUCCESSORS(*state*) with value v

---

function MAX-VALUE(*state*) **returns** *a utility value*

  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))
  return v

---

function MIN-VALUE(*state*) **returns** *a utility value*

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

Properties of Minimax

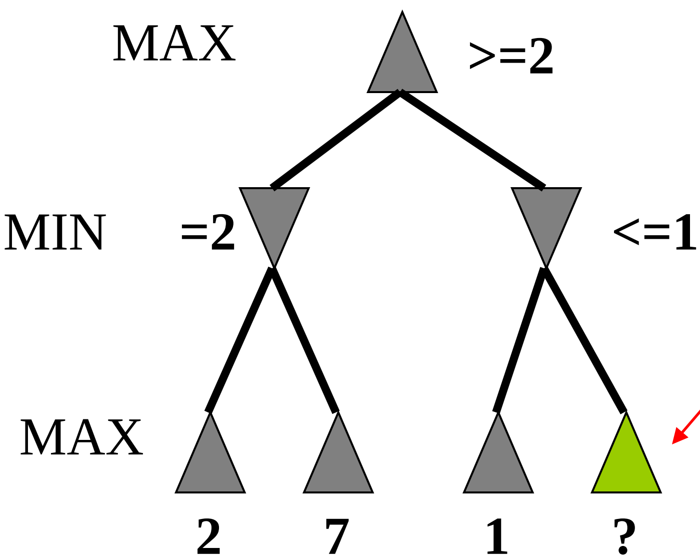
- Complete? Yes (if tree is finite)
- Optimal? Yes (against an optimal opponent)
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$ (depth-first exploration)

- For chess, $b \approx 35$, $m \approx 100$ for "reasonable" games
  → exact solution completely infeasible

Need to speed it up.

Alpha-Beta pruning

- We can improve on the performance of the minimax algorithm through **alpha-beta pruning**
- Basic idea: *“If you have an idea that is surely bad, don't take the time to see how truly awful it is.”* -- Pat Winston

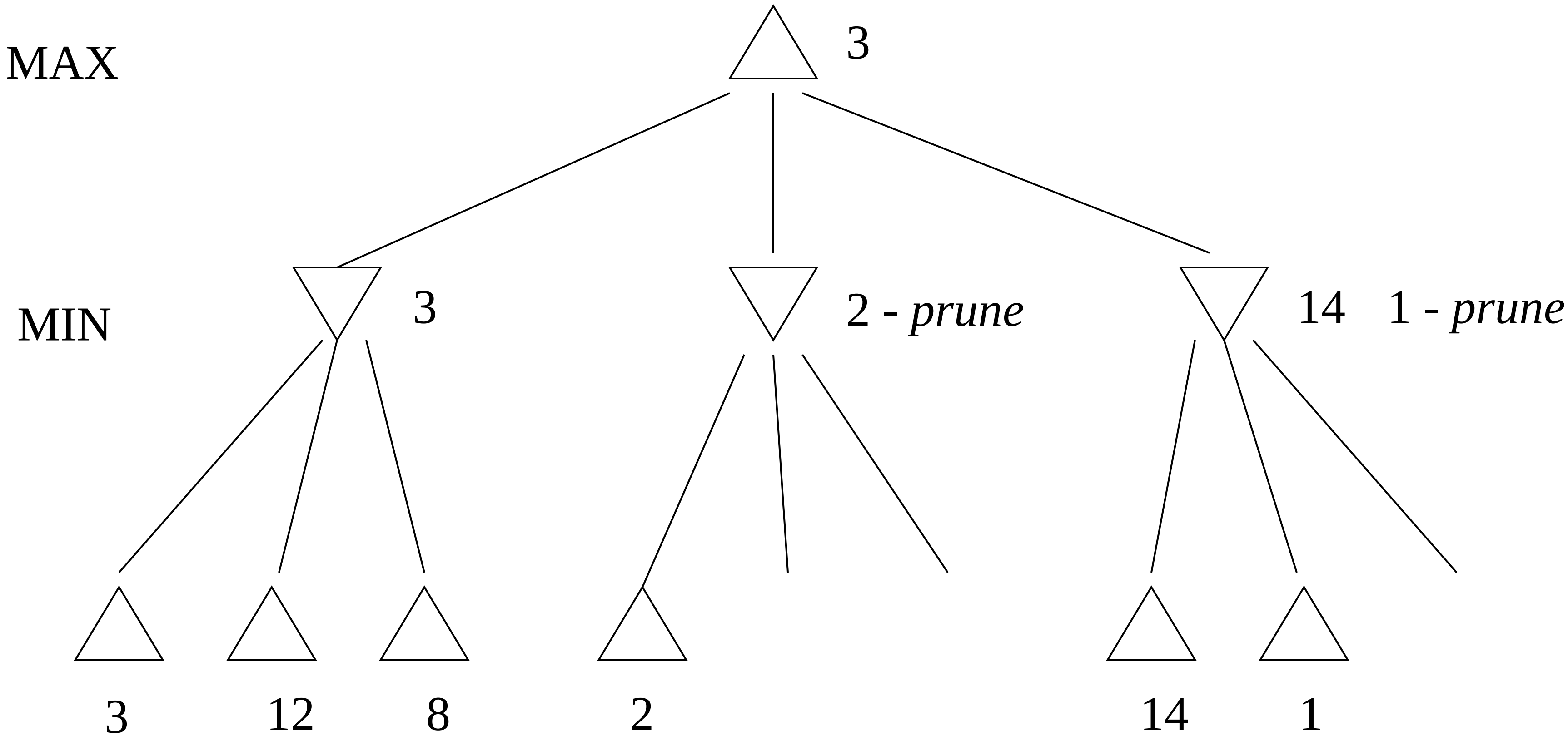


- We don’t need to compute the value at this node.
- No matter what it is, it can’t affect the value of the root node.

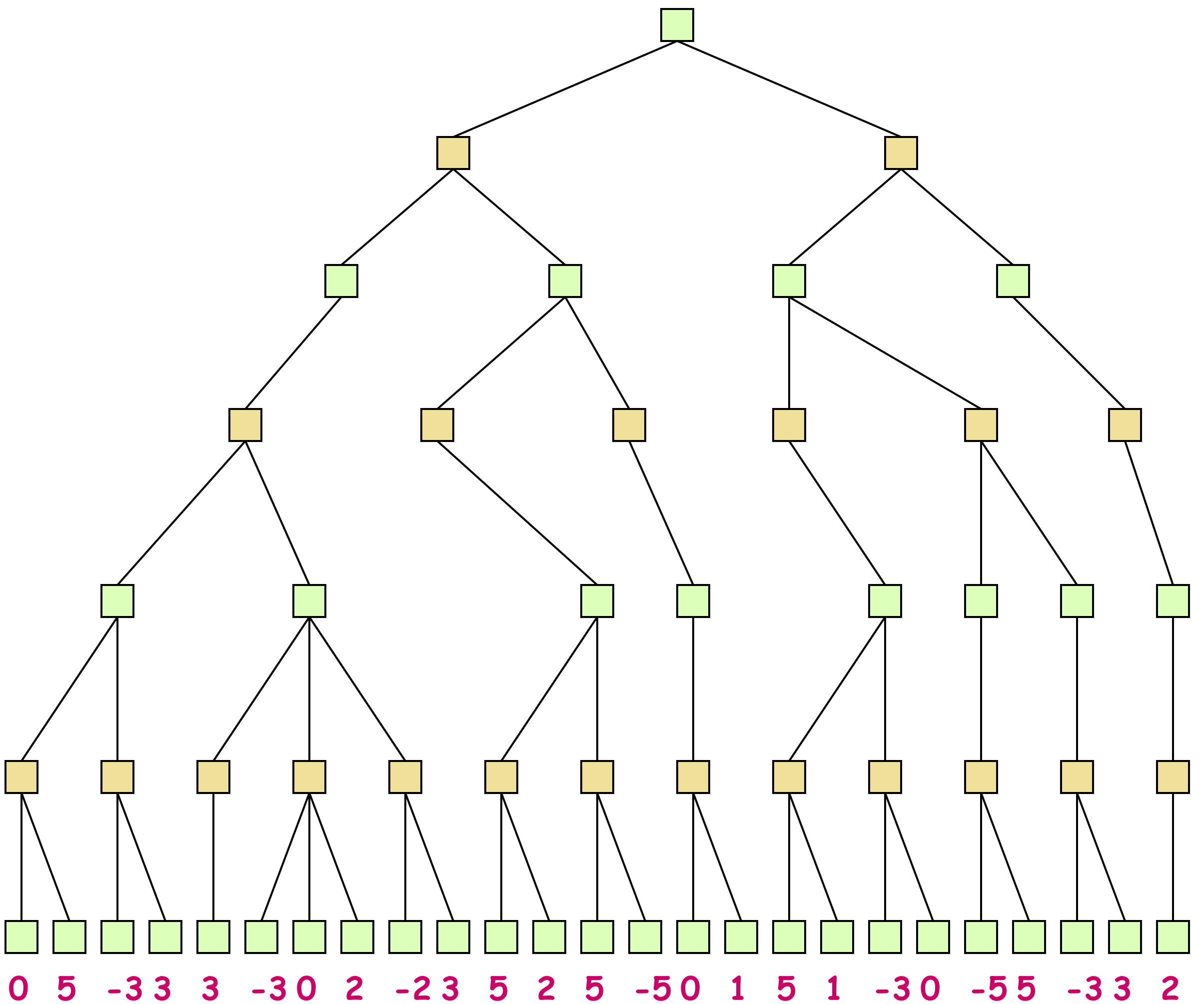
Alpha-Beta pruning

- Traverse the search tree in depth-first order
- At each **MAX** node n, **alpha(n)** = maximum value found so far
- At each **MIN** node n, **beta(n)** = minimum value found so far
  - The alpha values start at $-\infty$ and only increase, while beta values start at $+\infty$ and only decrease
- **Beta cutoff**: Given MAX node n, cut off search below n (i.e., don’t generate/examine any more of n’s children) if alpha(n) >= beta(i) for some MIN node ancestor i of n.
- **Alpha cutoff:** stop searching below MIN node n if beta(n) <= alpha(i) for some MAX node ancestor i of n.

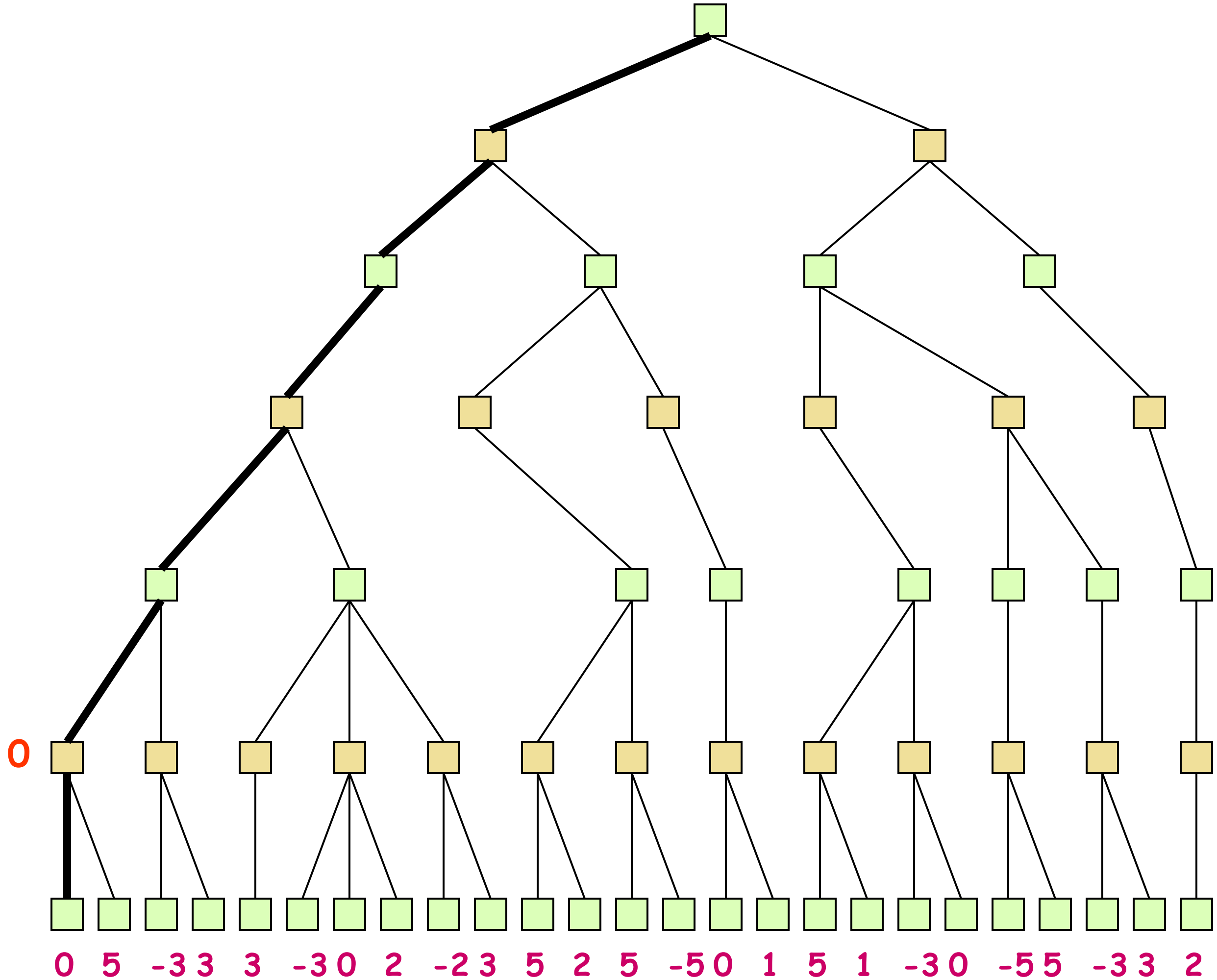
Alpha-Beta General example



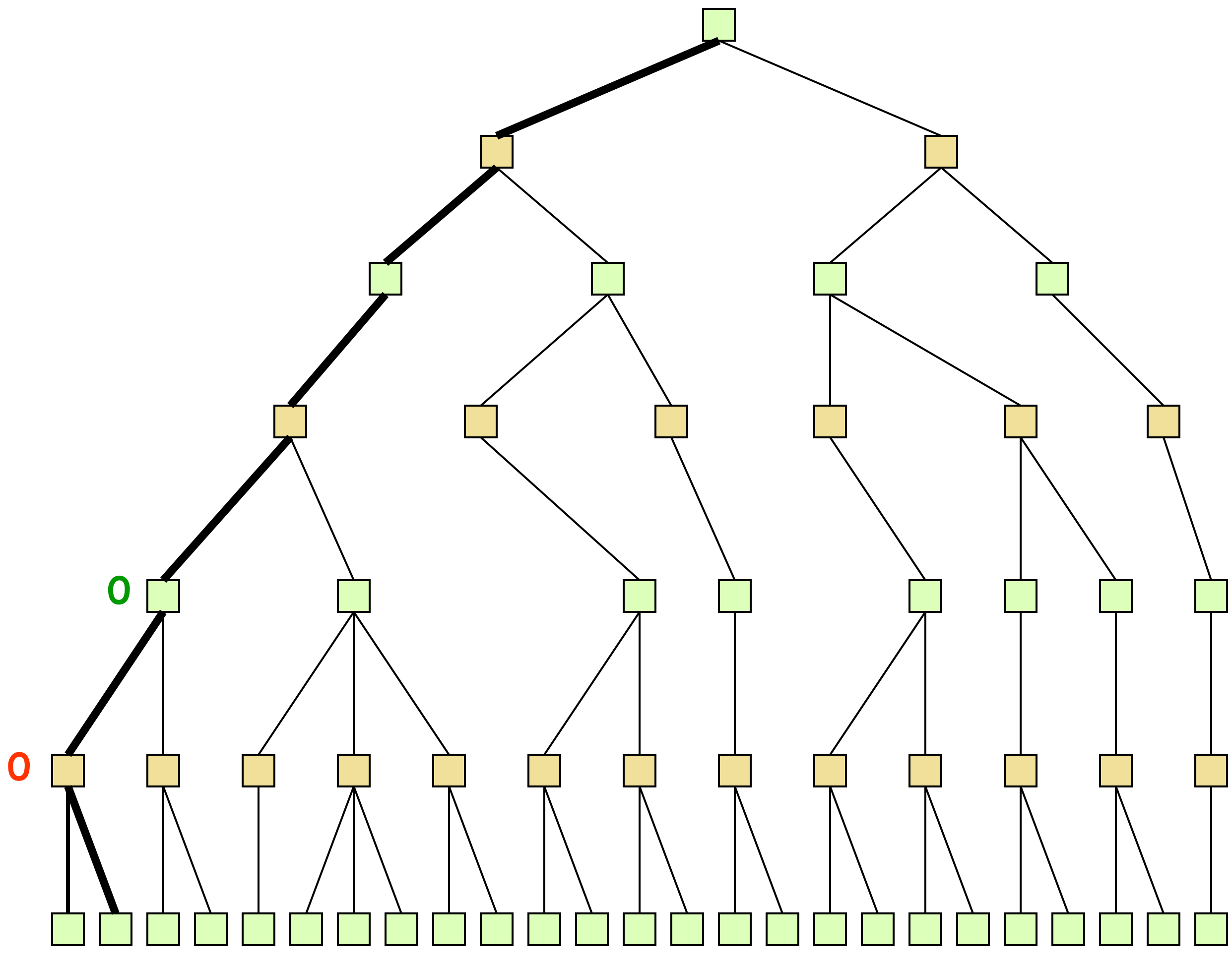
Alpha-Beta Pruning Example-2



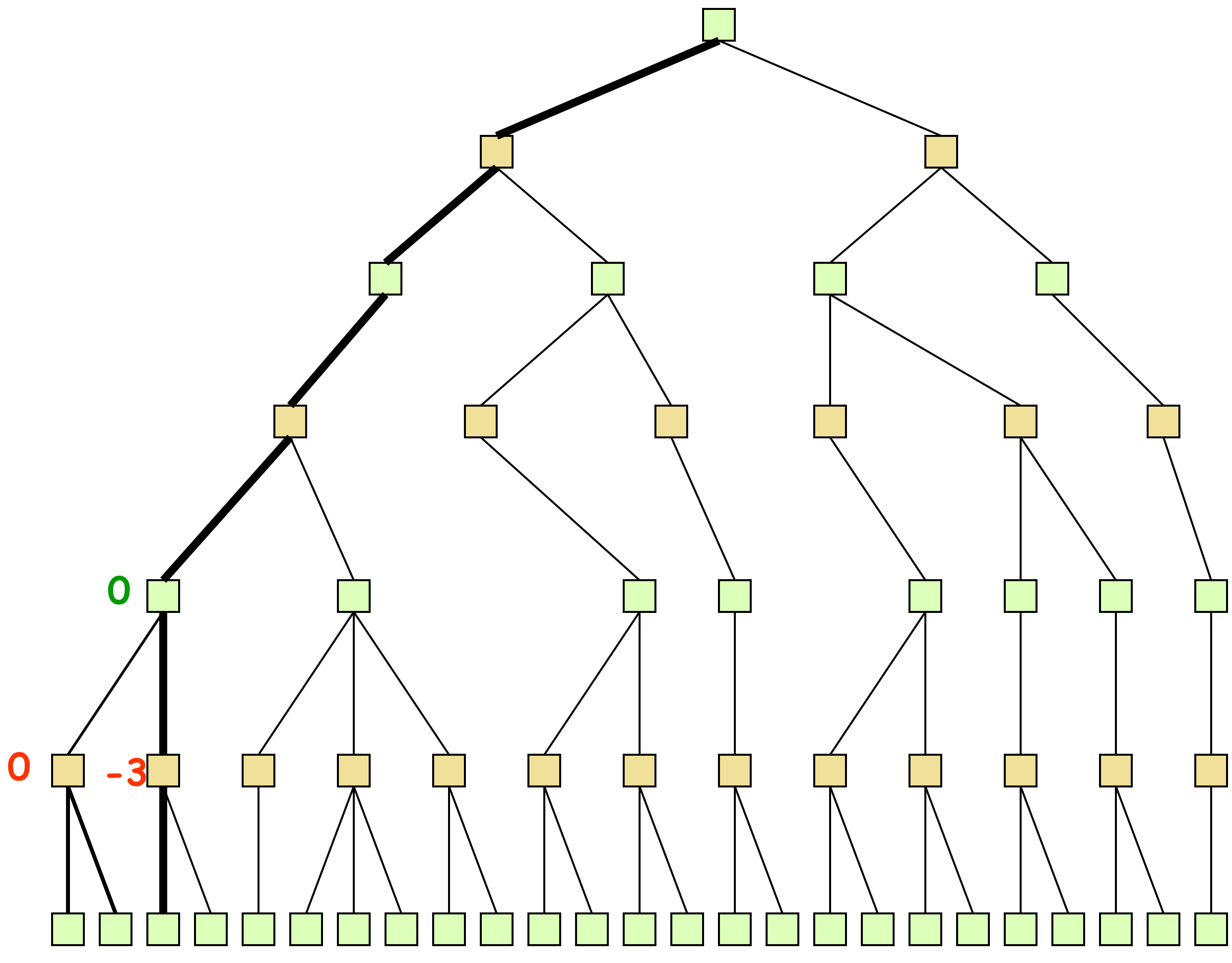
Alpha-Beta Pruning Example-2



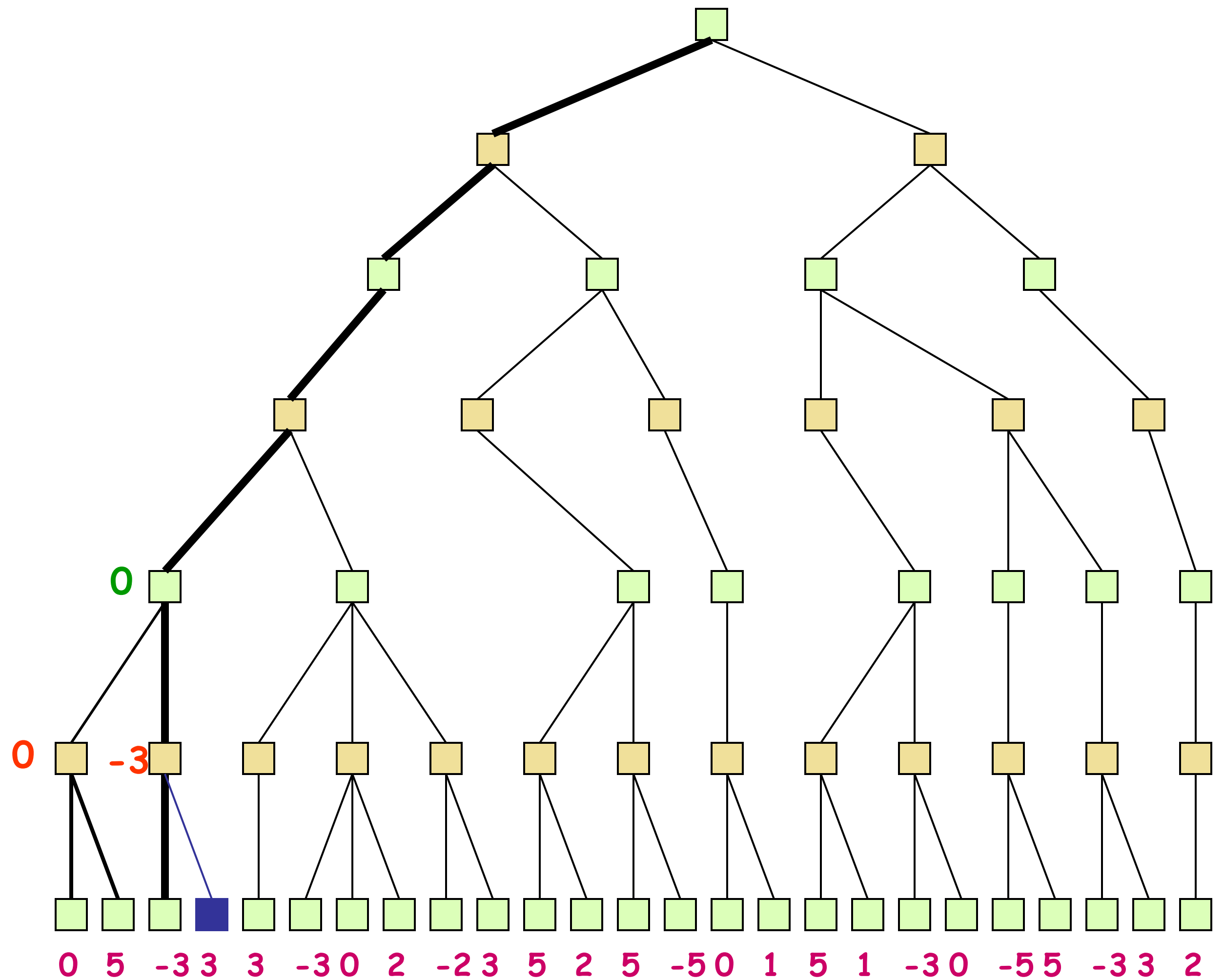
Alpha-Beta Pruning Example-2



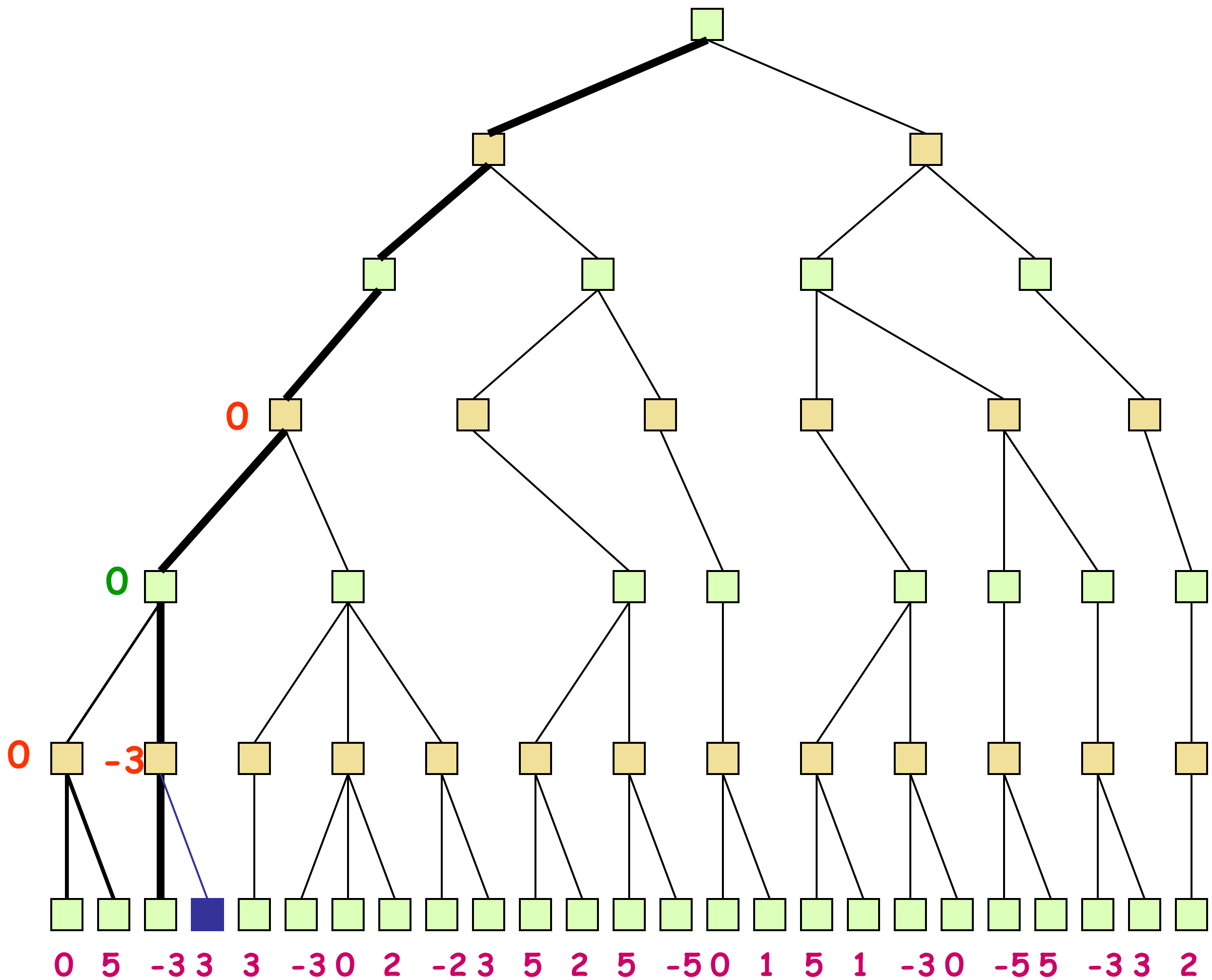
Alpha-Beta Pruning Example-2



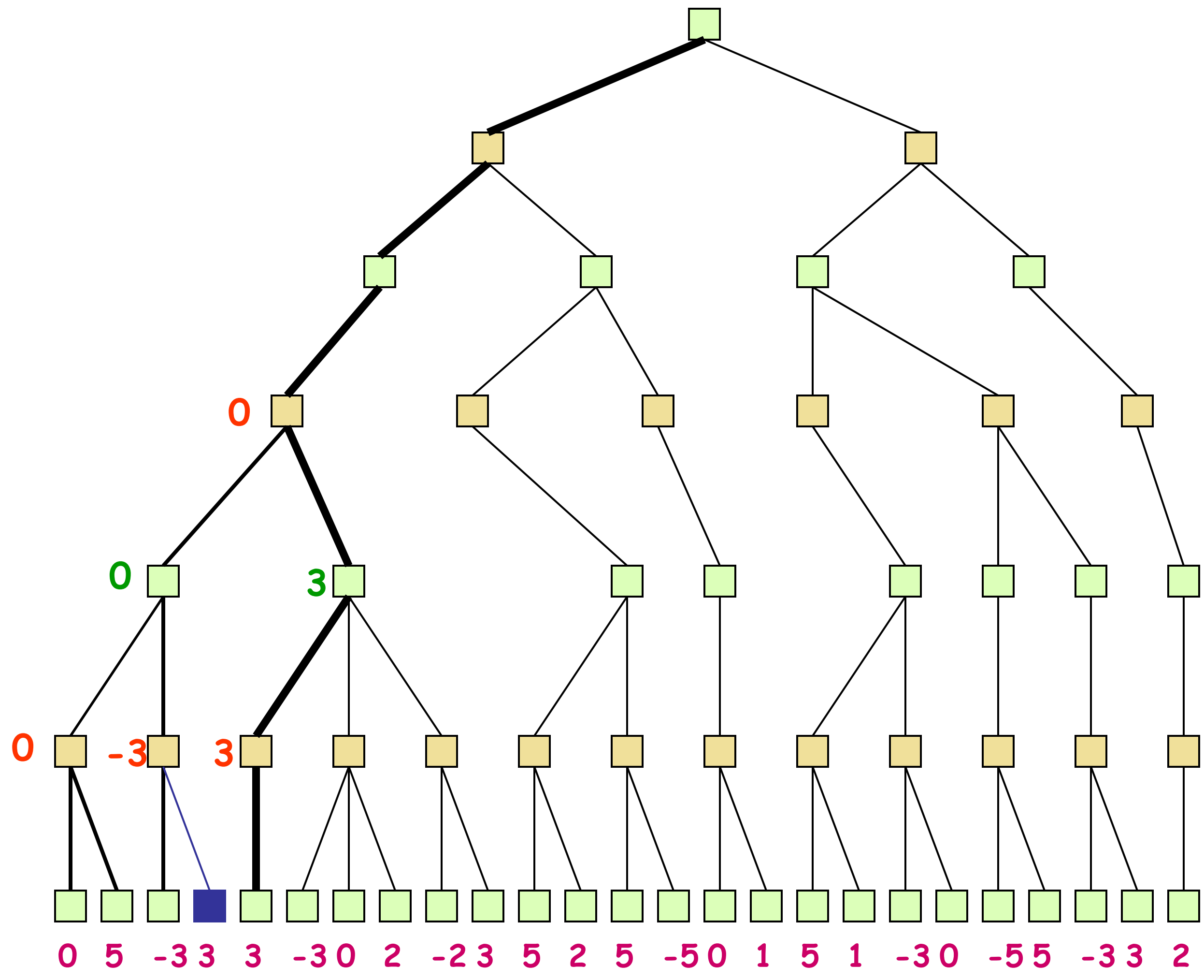
Alpha-Beta Pruning Example-2



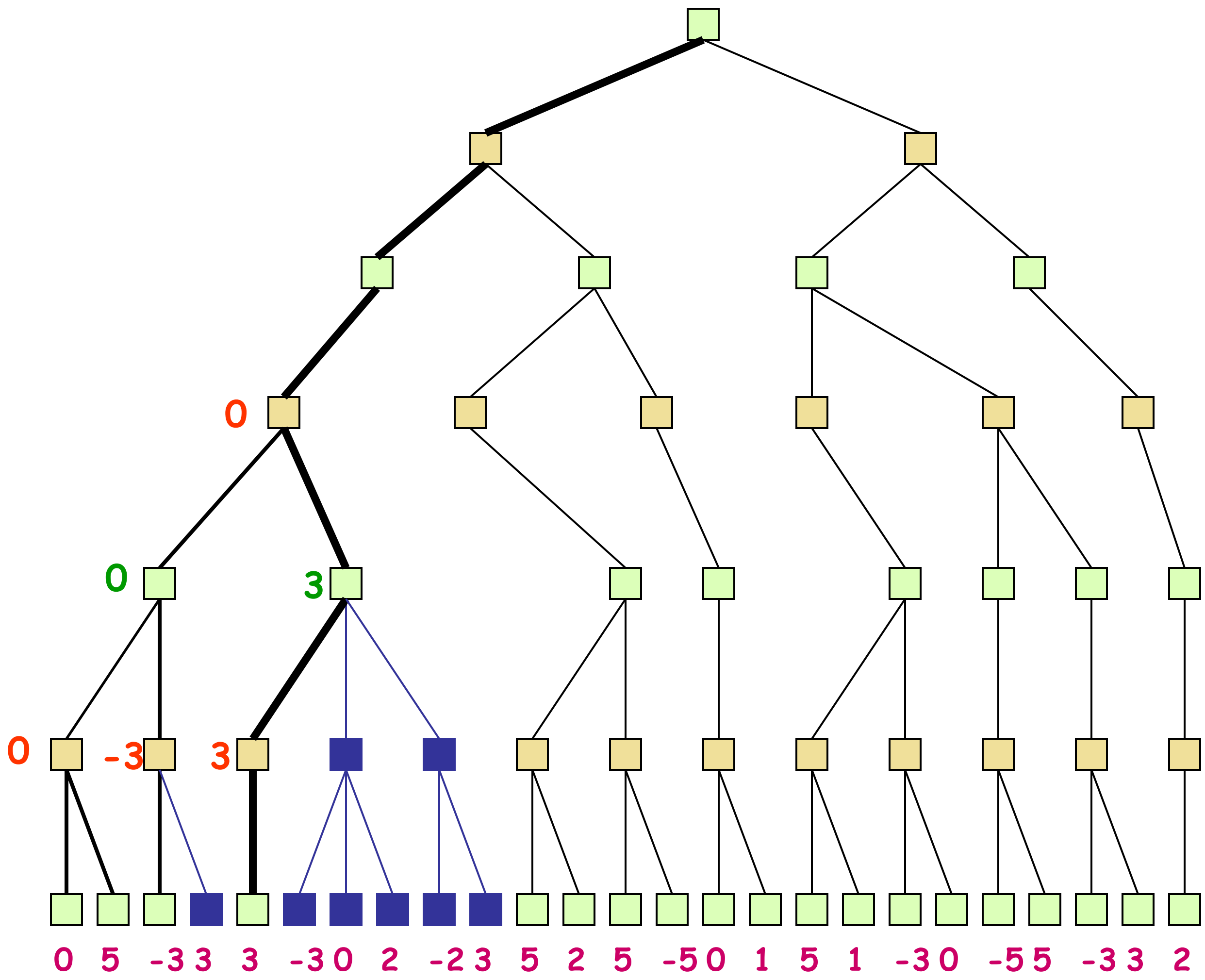
Alpha-Beta Pruning Example-2



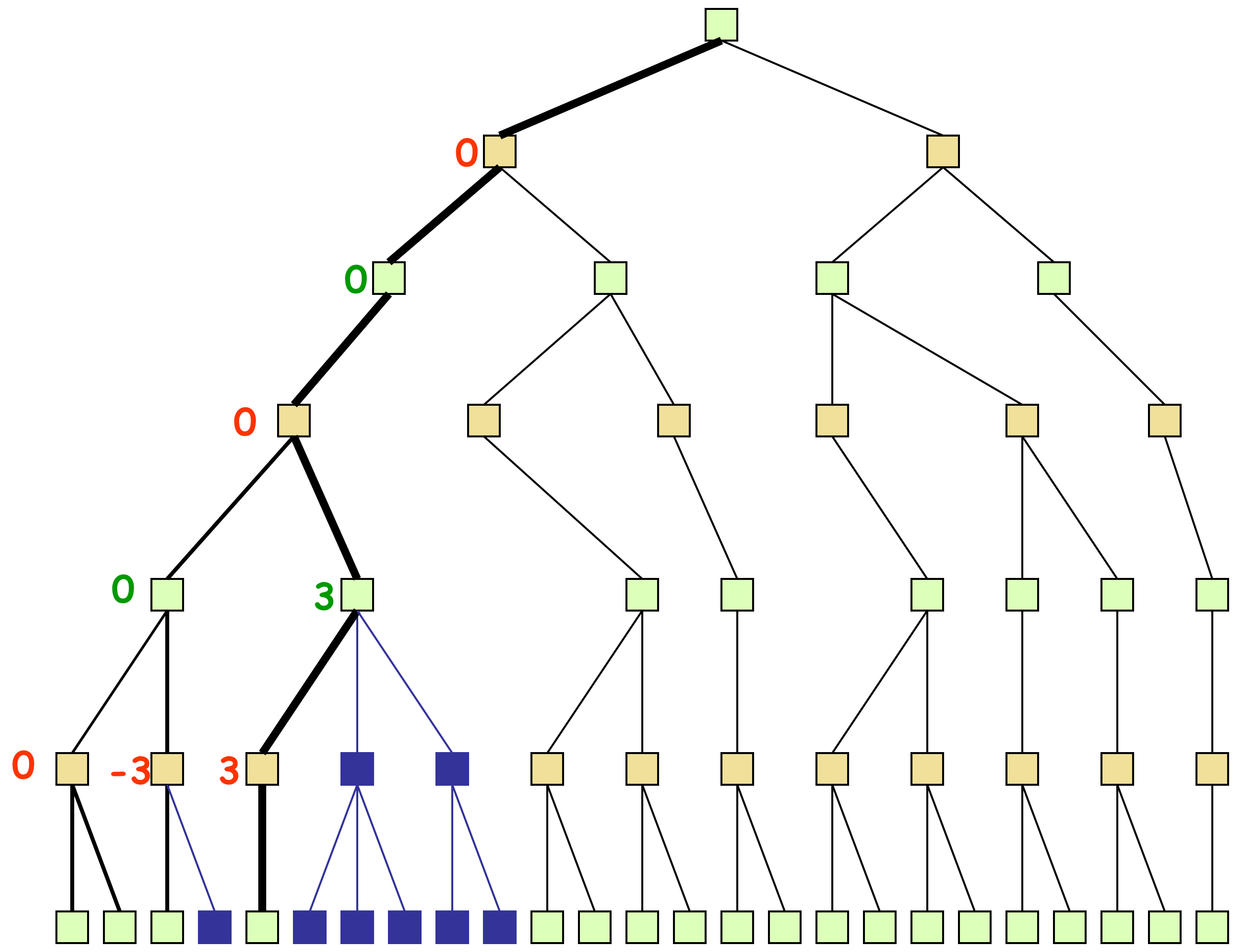
Alpha-Beta Pruning Example-2



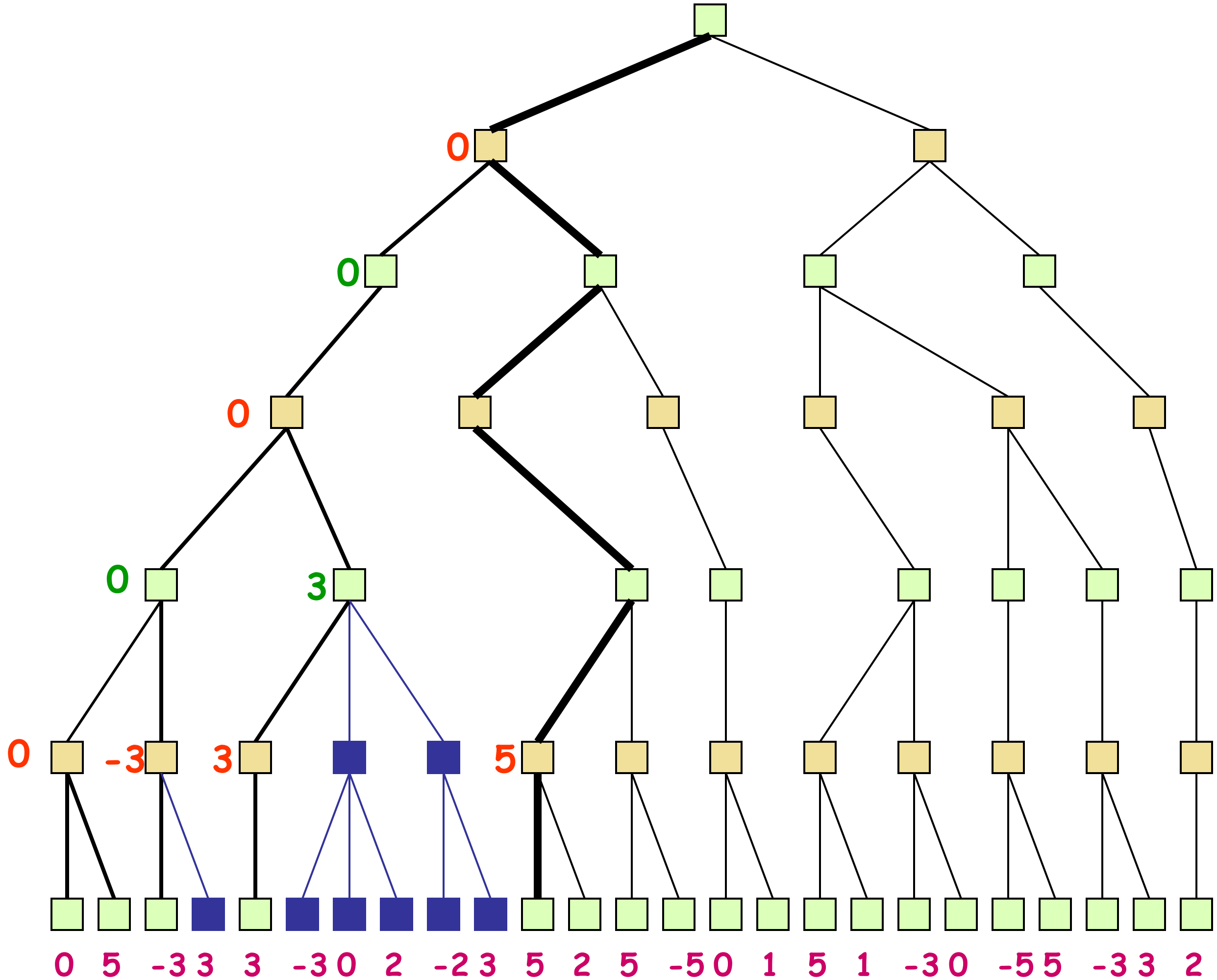
Alpha-Beta Pruning Example-2



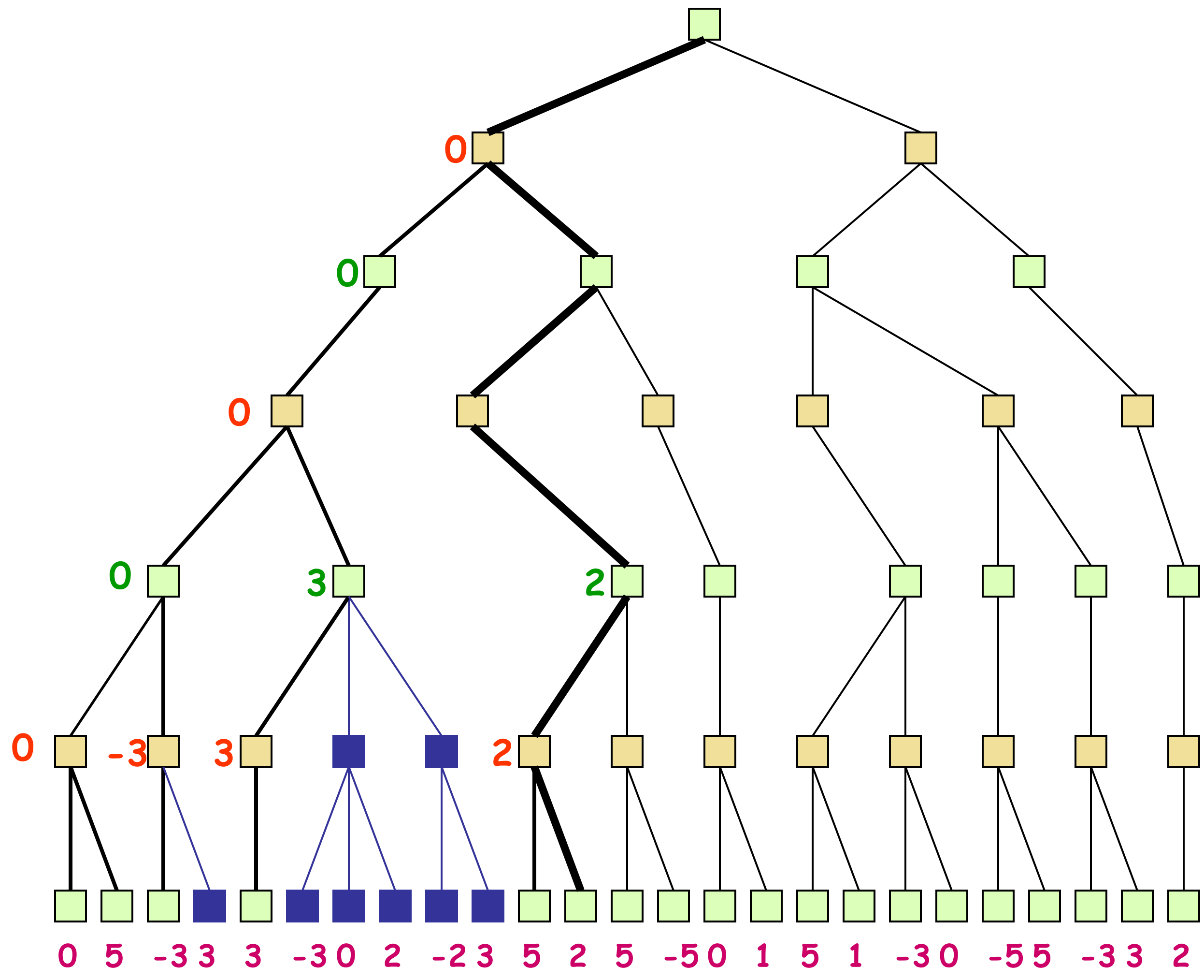
Alpha-Beta Pruning Example-2



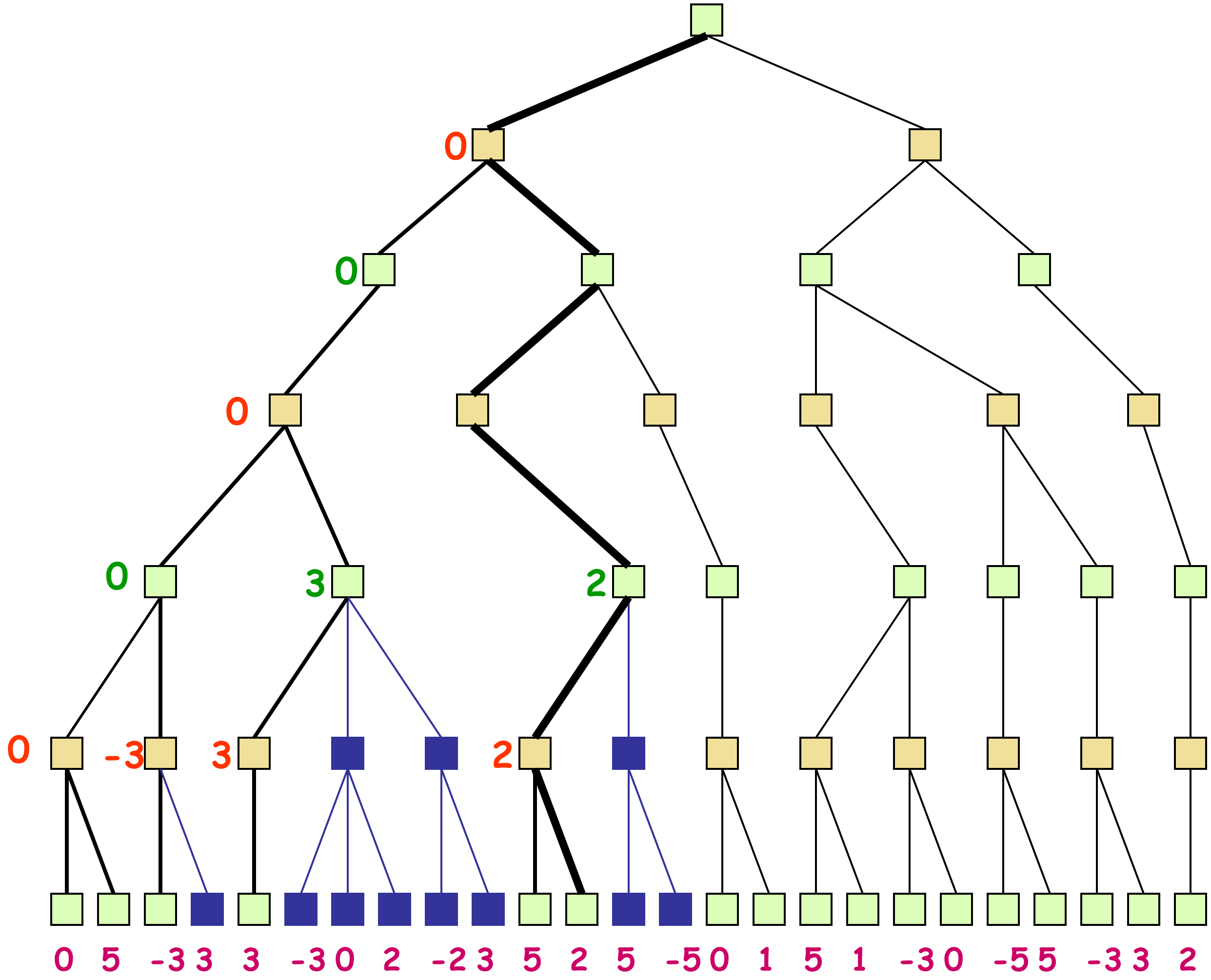
Alpha-Beta Pruning Example-2



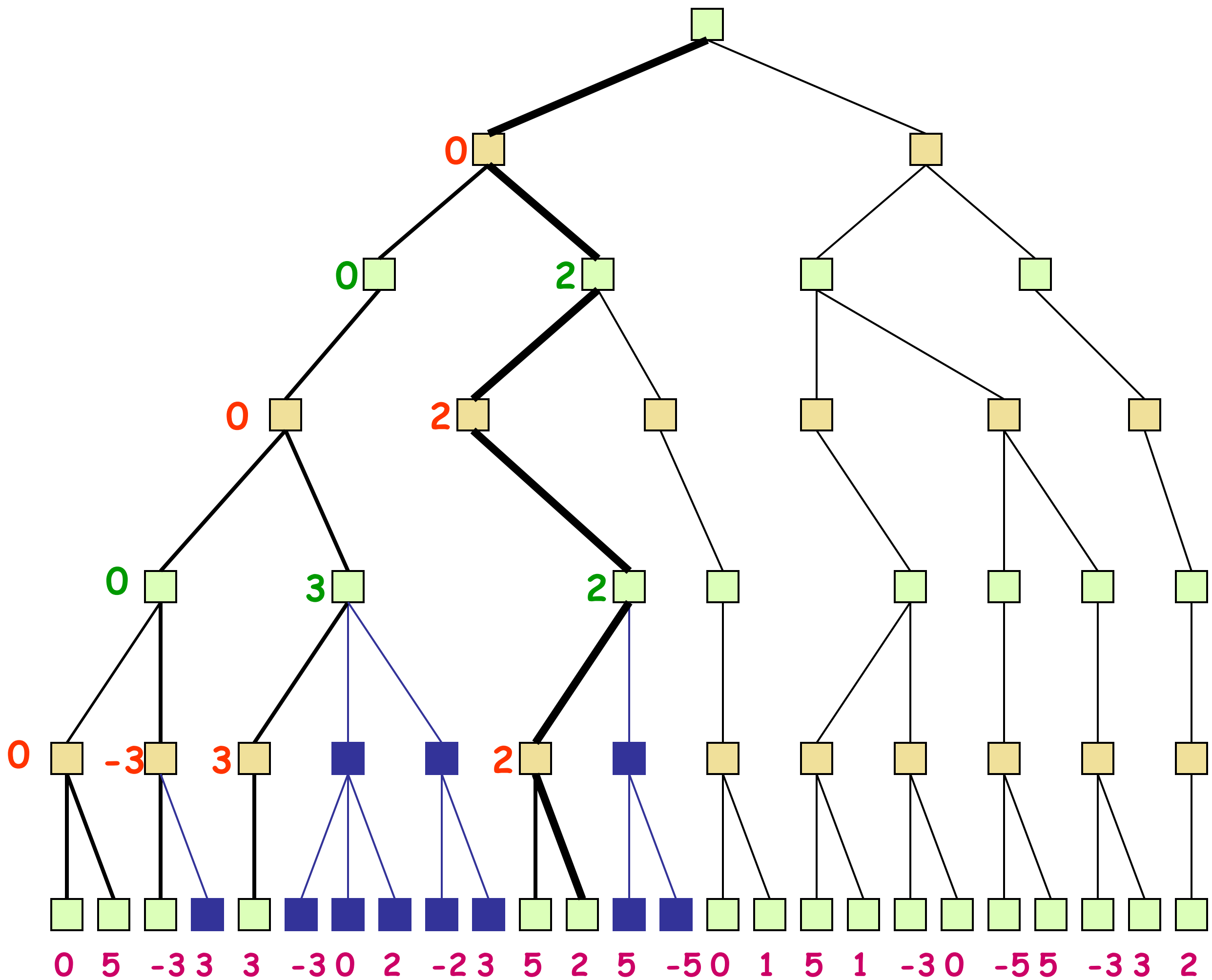
Alpha-Beta Pruning Example-2



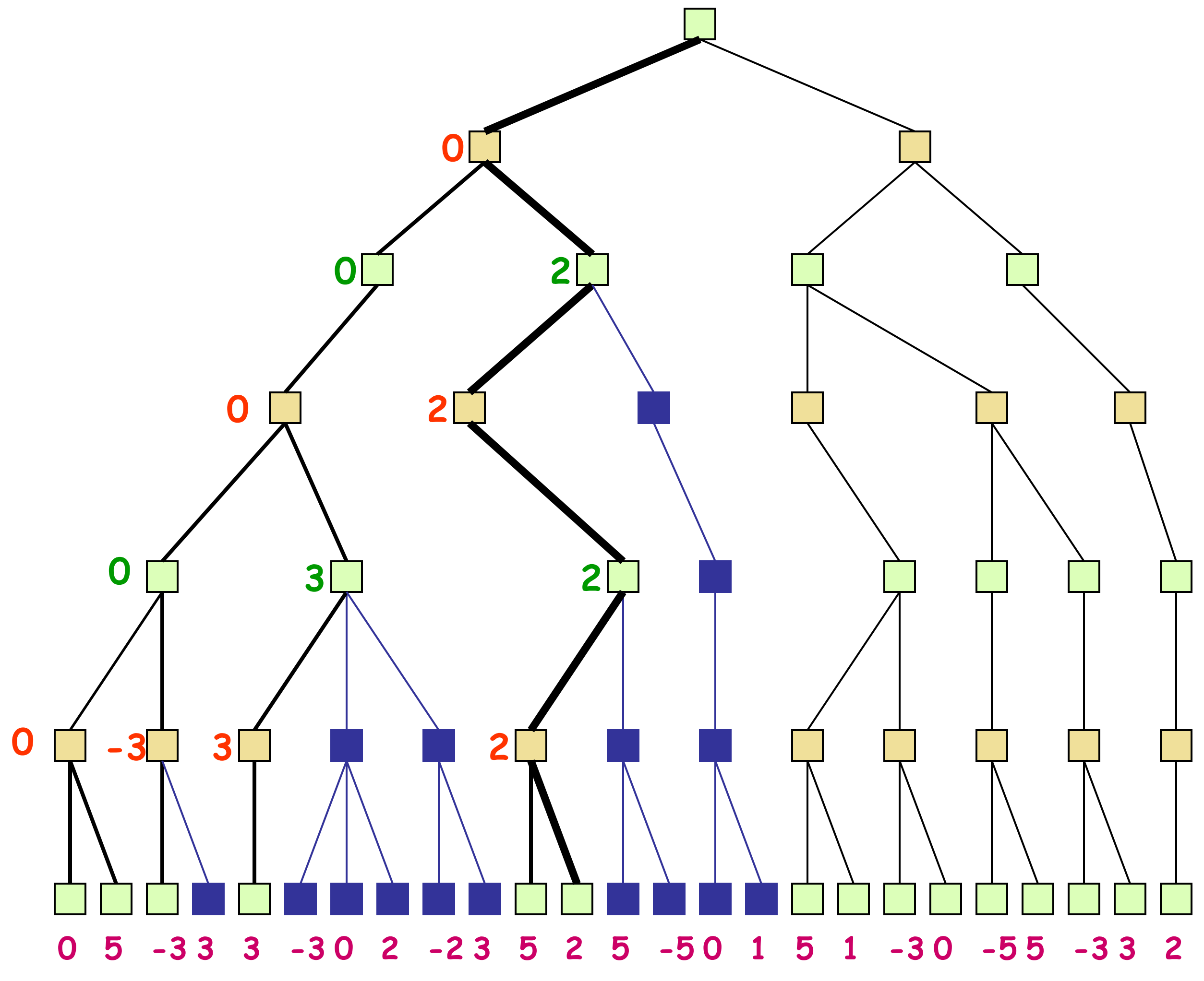
Alpha-Beta Pruning Example-2



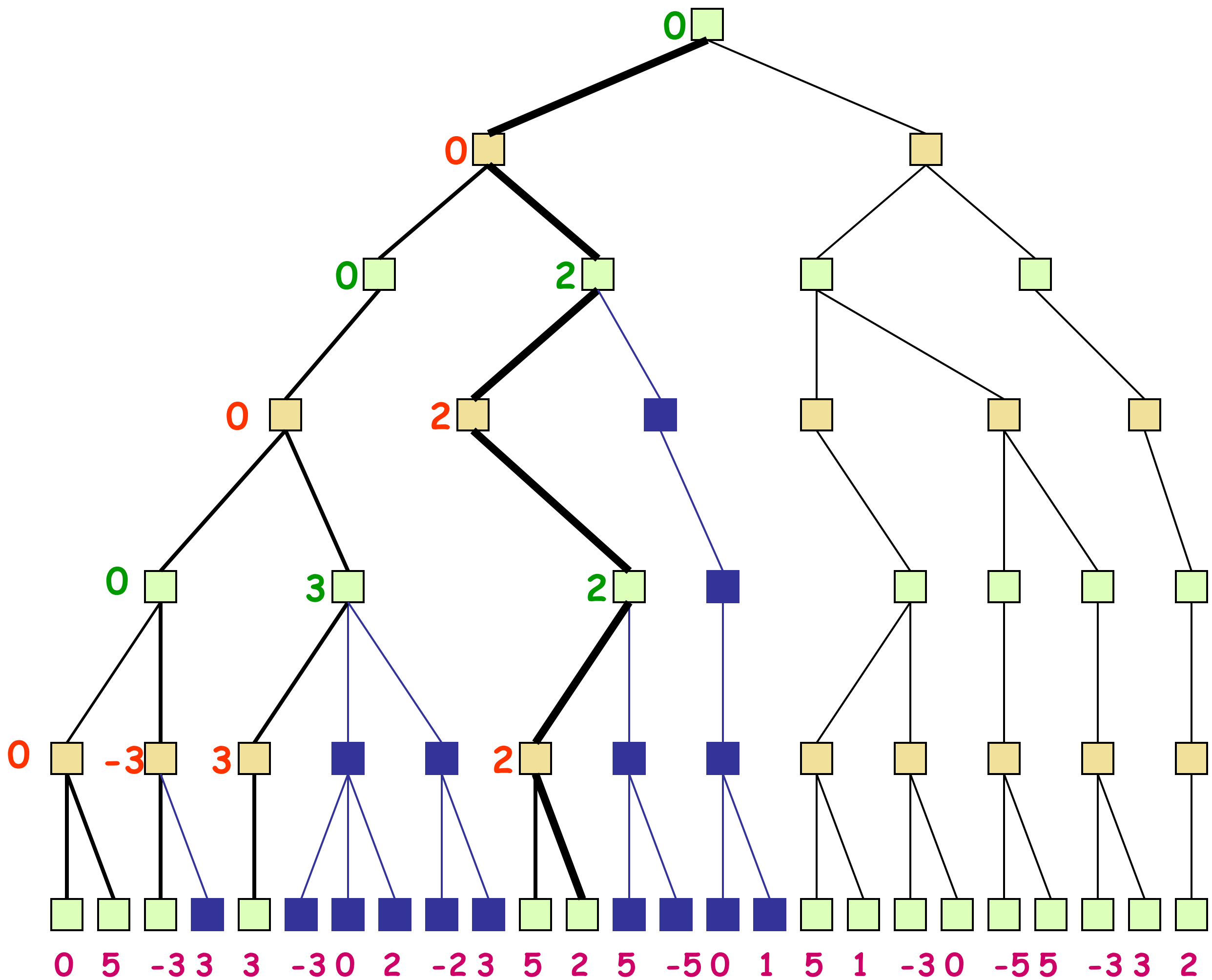
Alpha-Beta Pruning Example-2



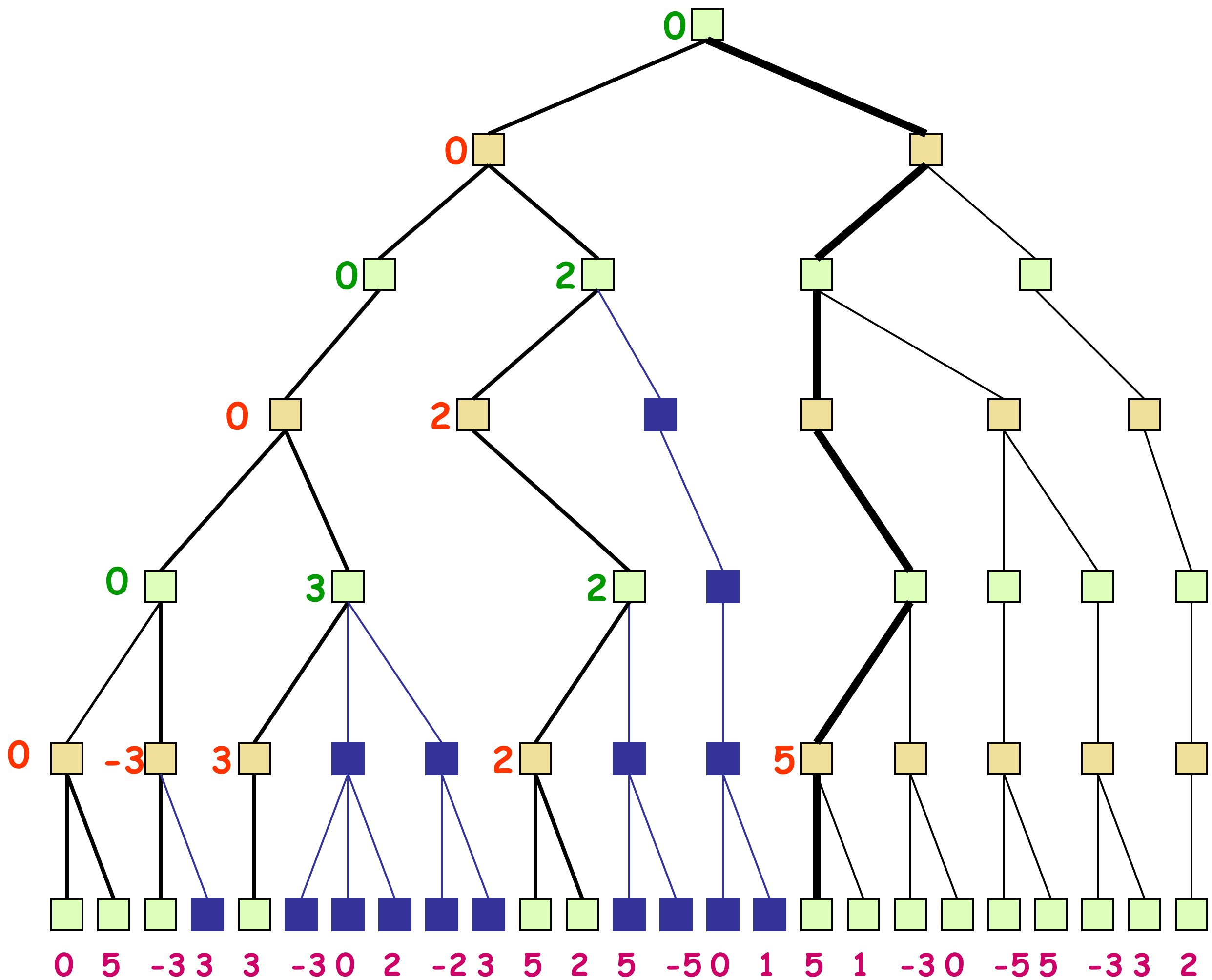
Alpha-Beta Pruning Example-2



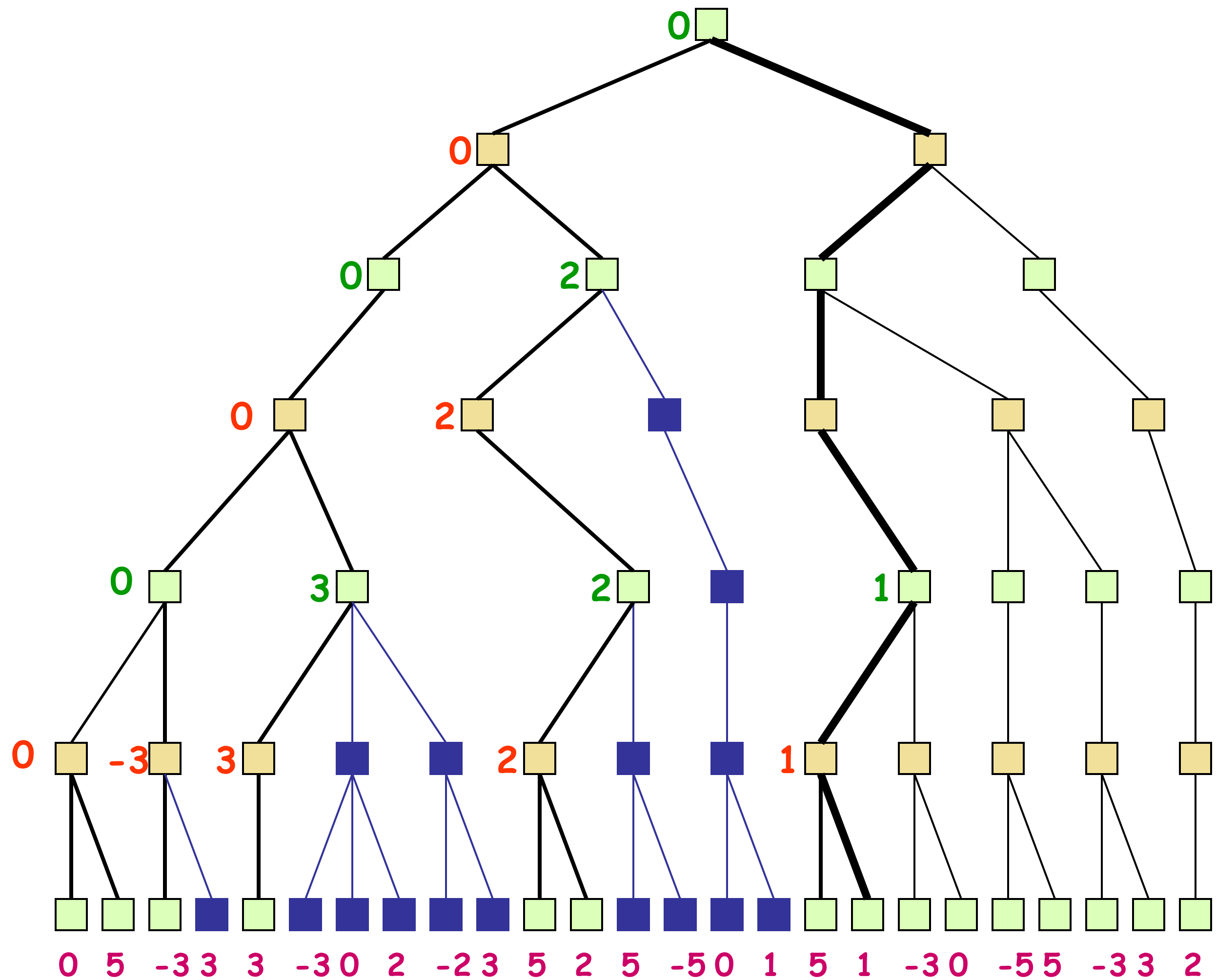
Alpha-Beta Pruning Example-2



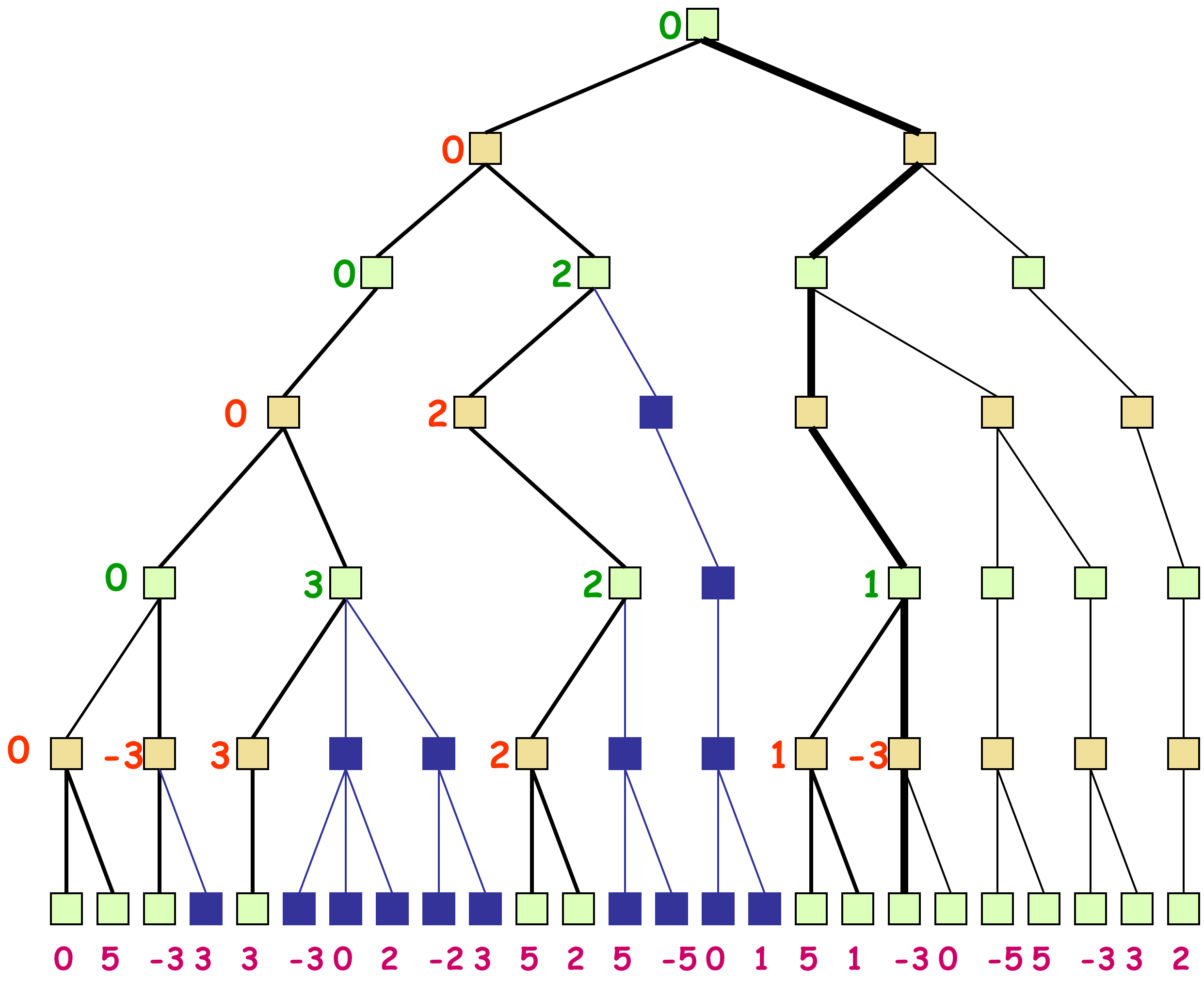
Alpha-Beta Pruning Example-2



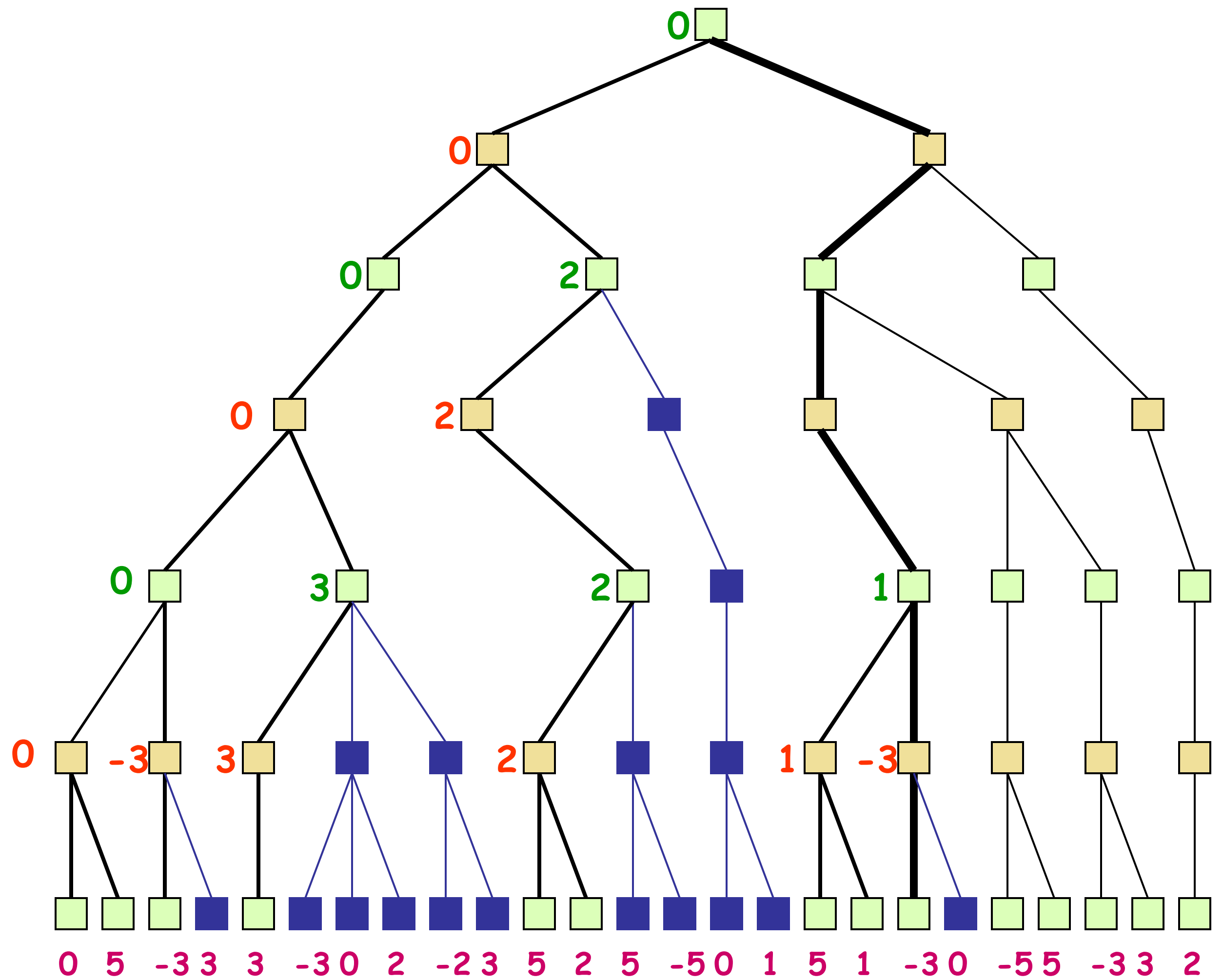
Alpha-Beta Pruning Example-2



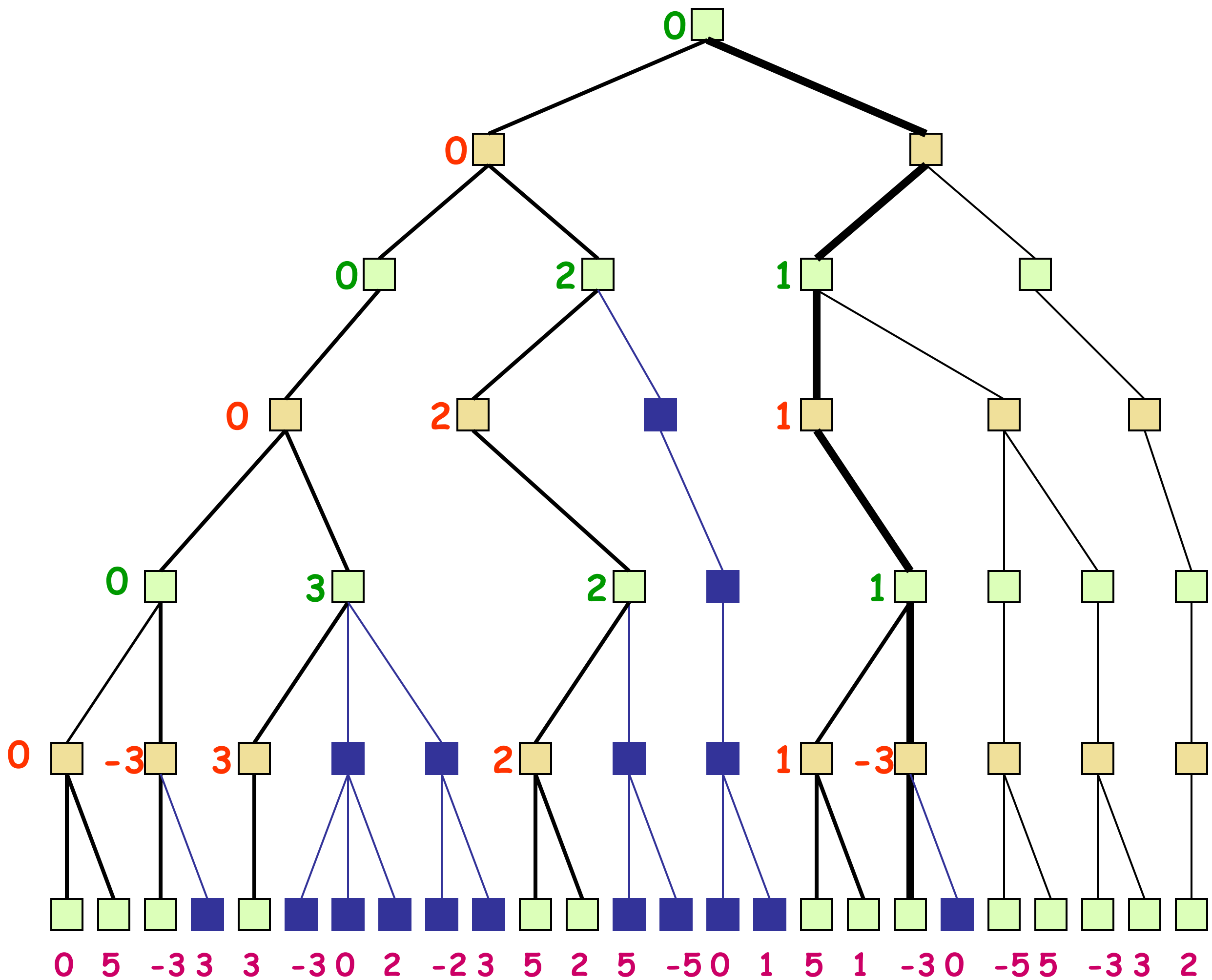
Alpha-Beta Pruning Example-2



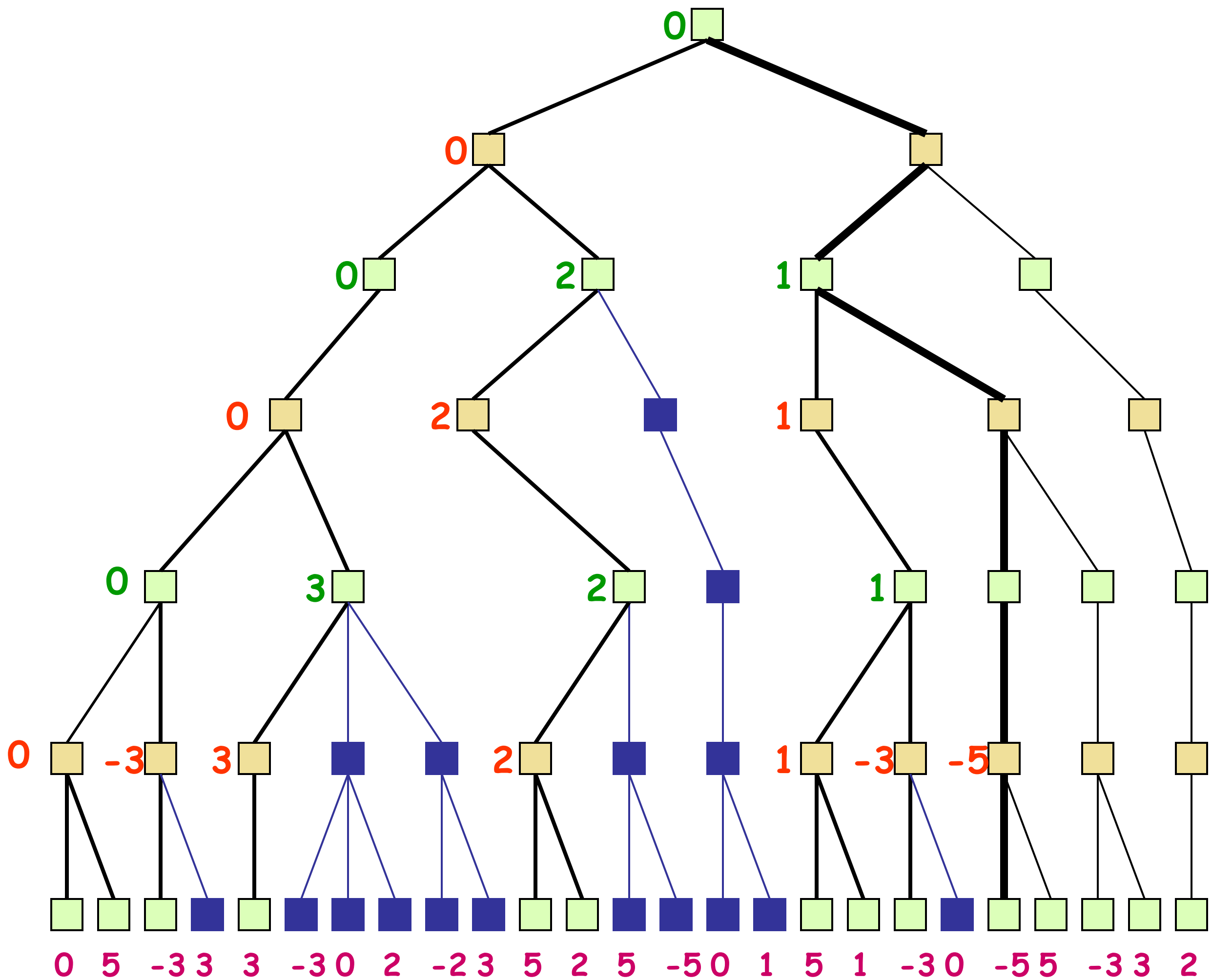
Alpha-Beta Pruning Example-2



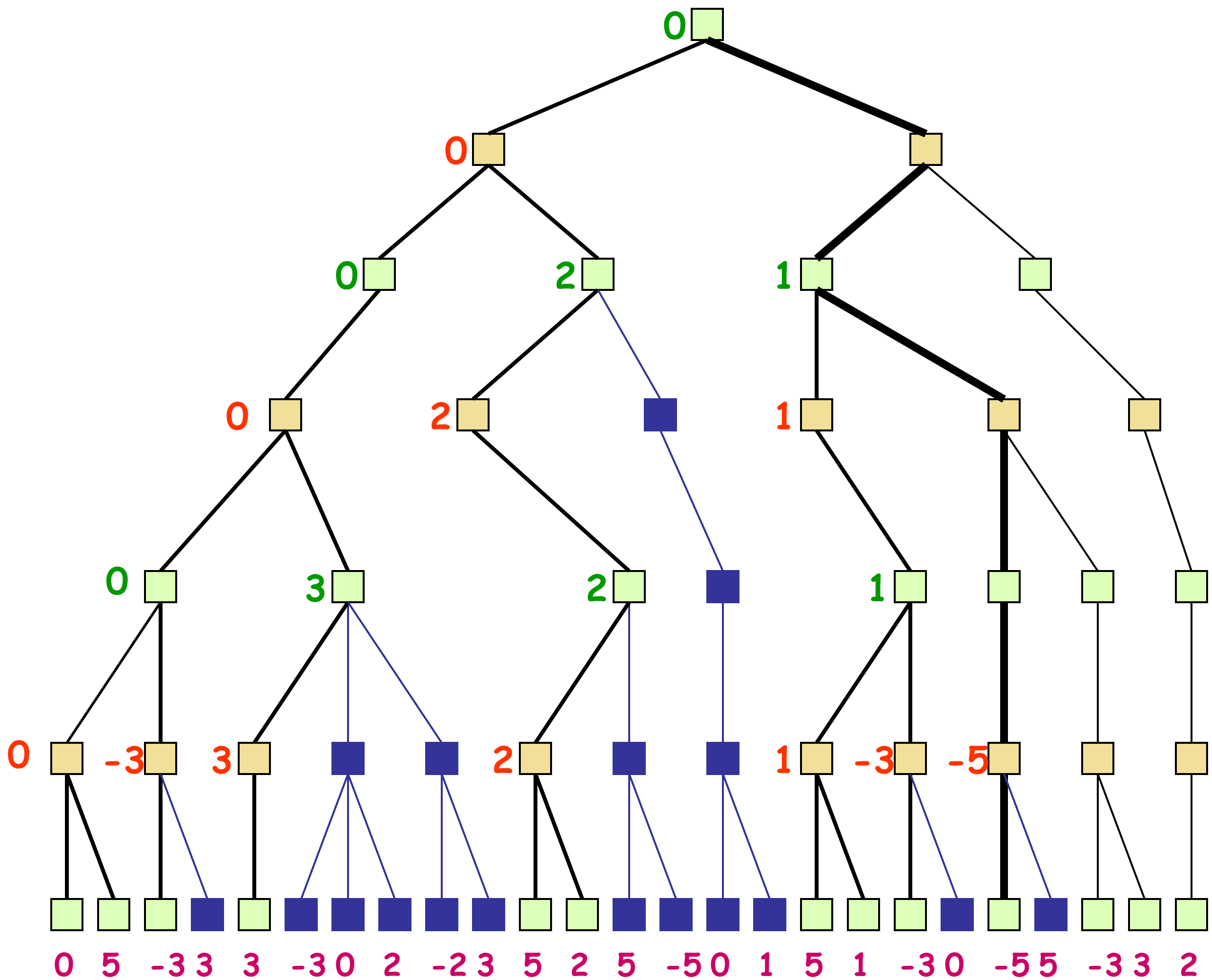
Alpha-Beta Pruning Example-2



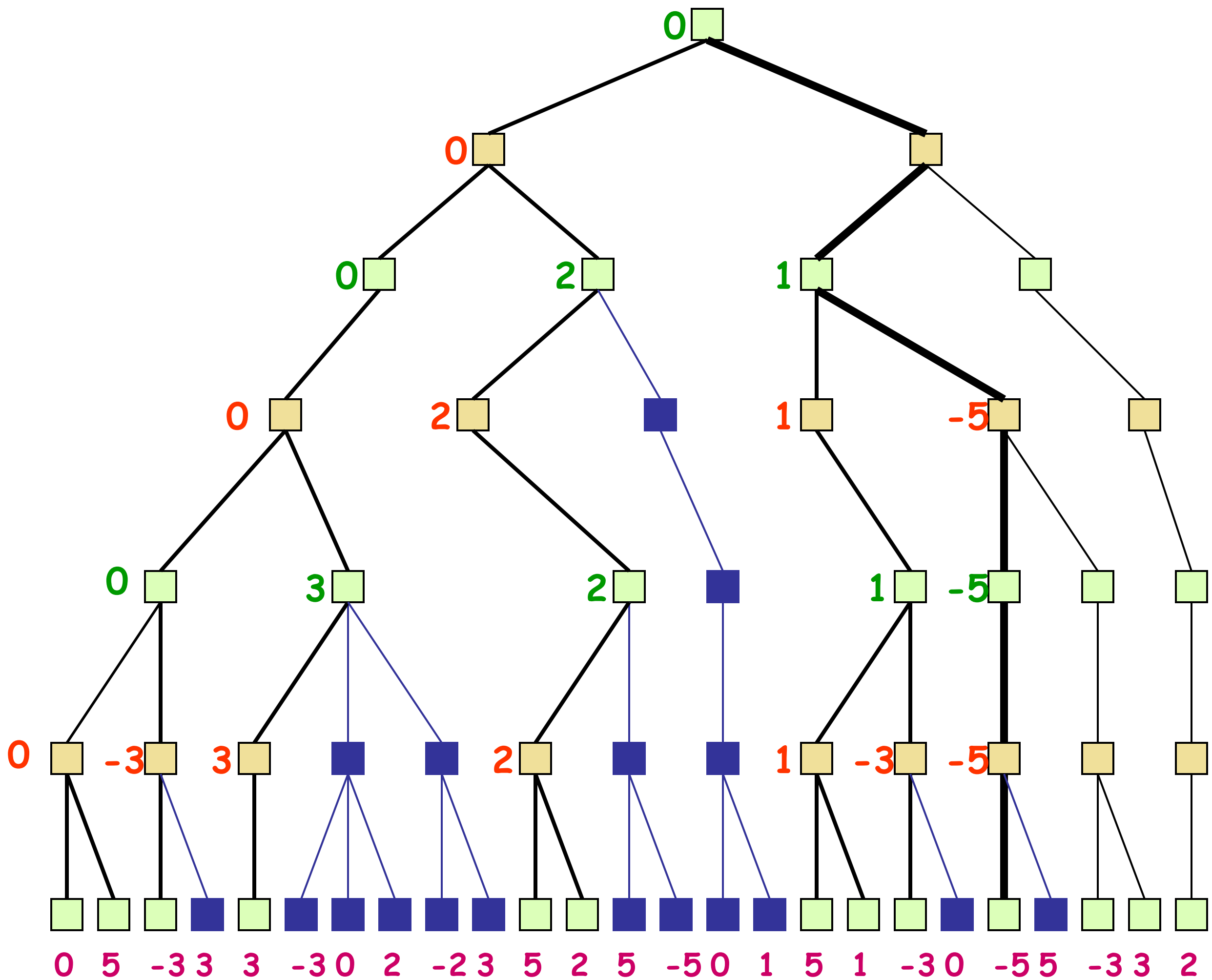
Alpha-Beta Pruning Example-2



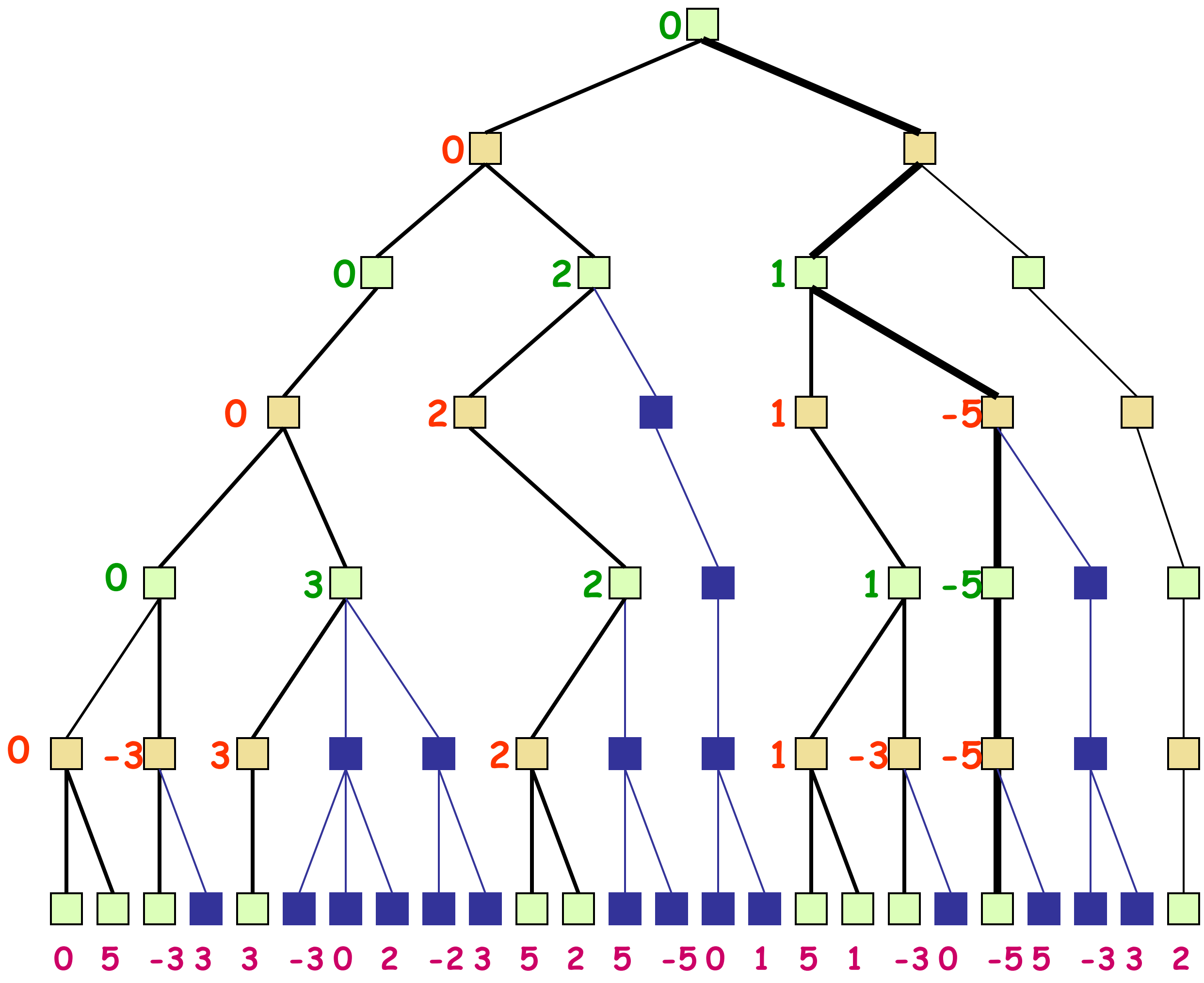
Alpha-Beta Pruning Example-2



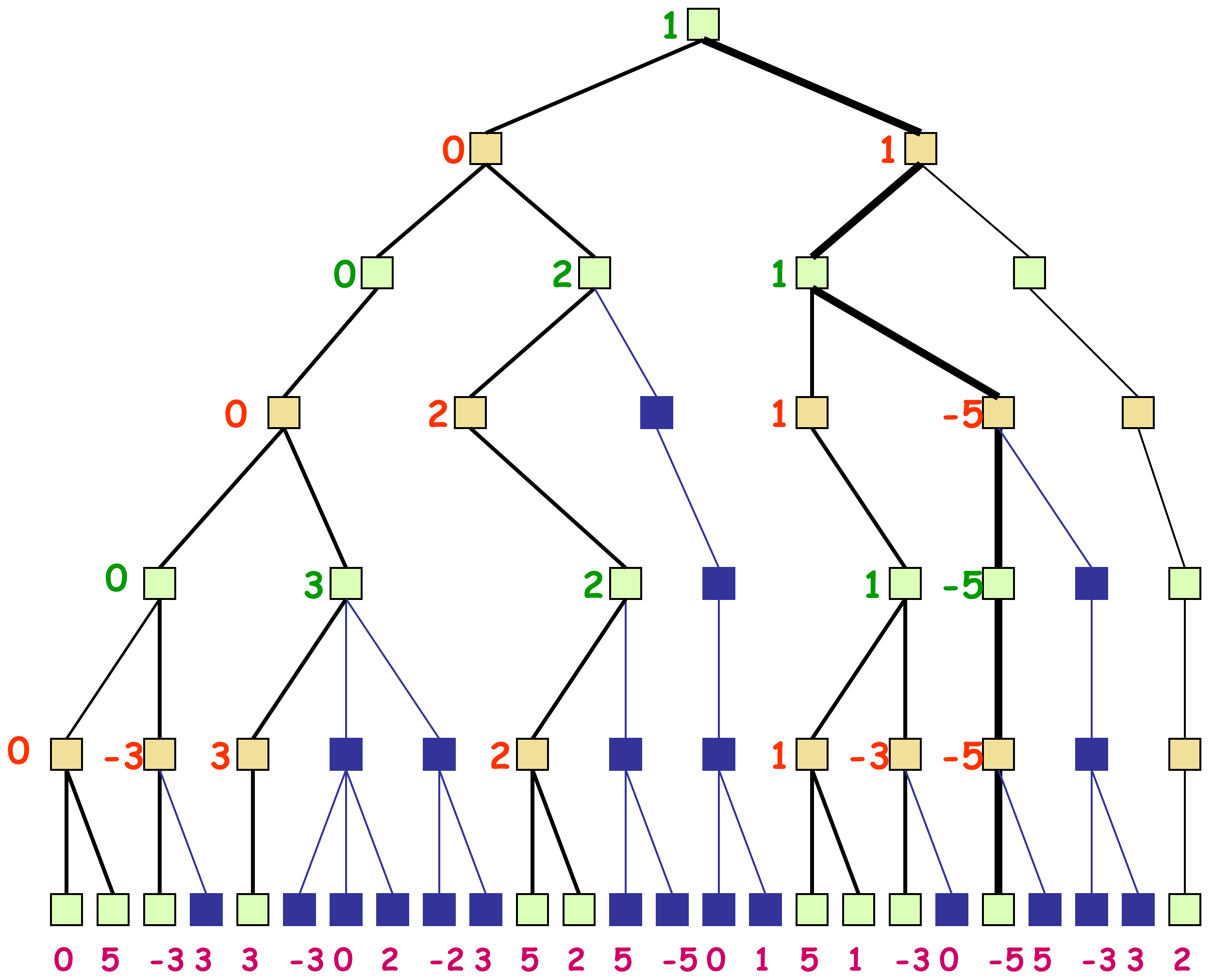
Alpha-Beta Pruning Example-2



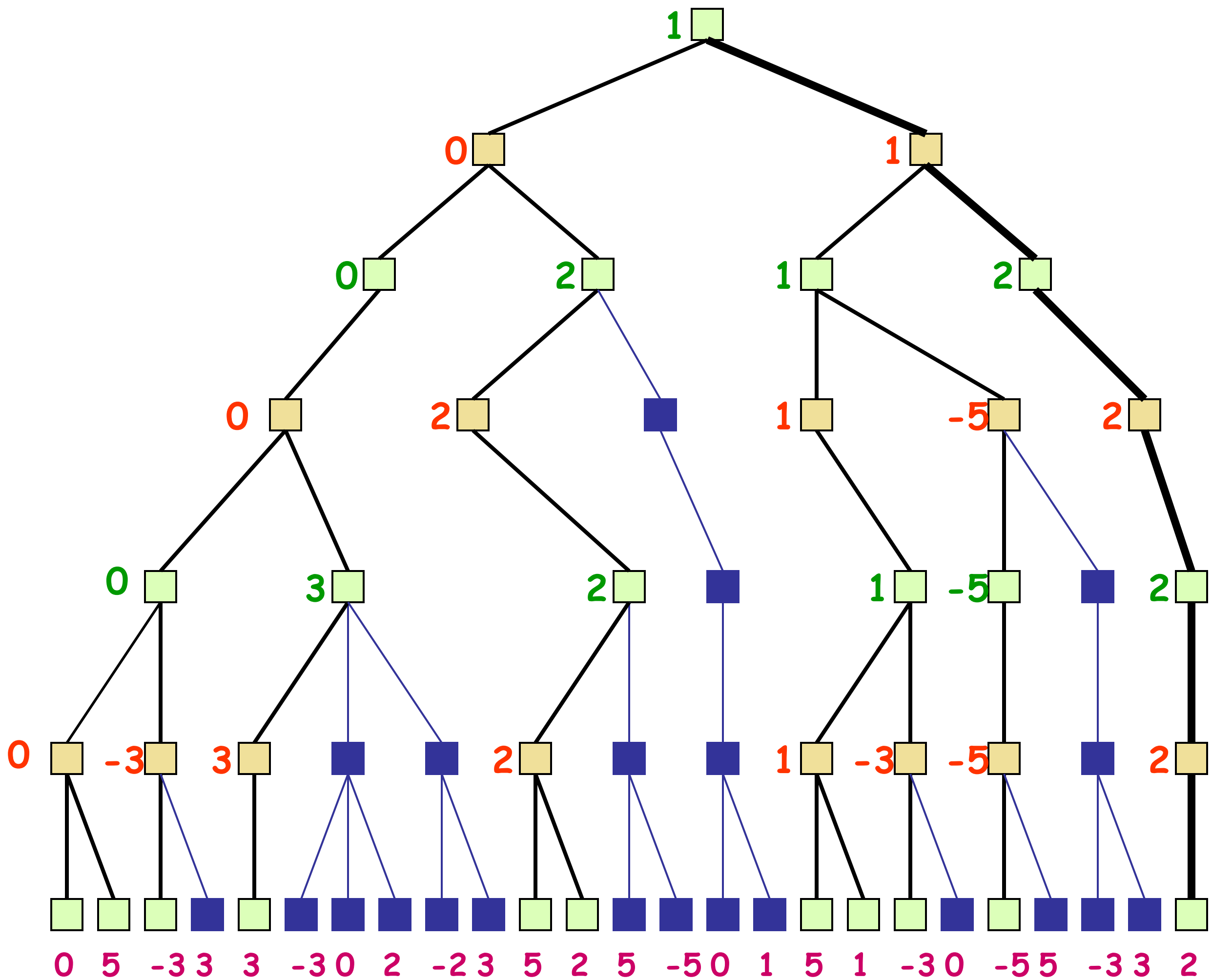
Alpha-Beta Pruning Example-2



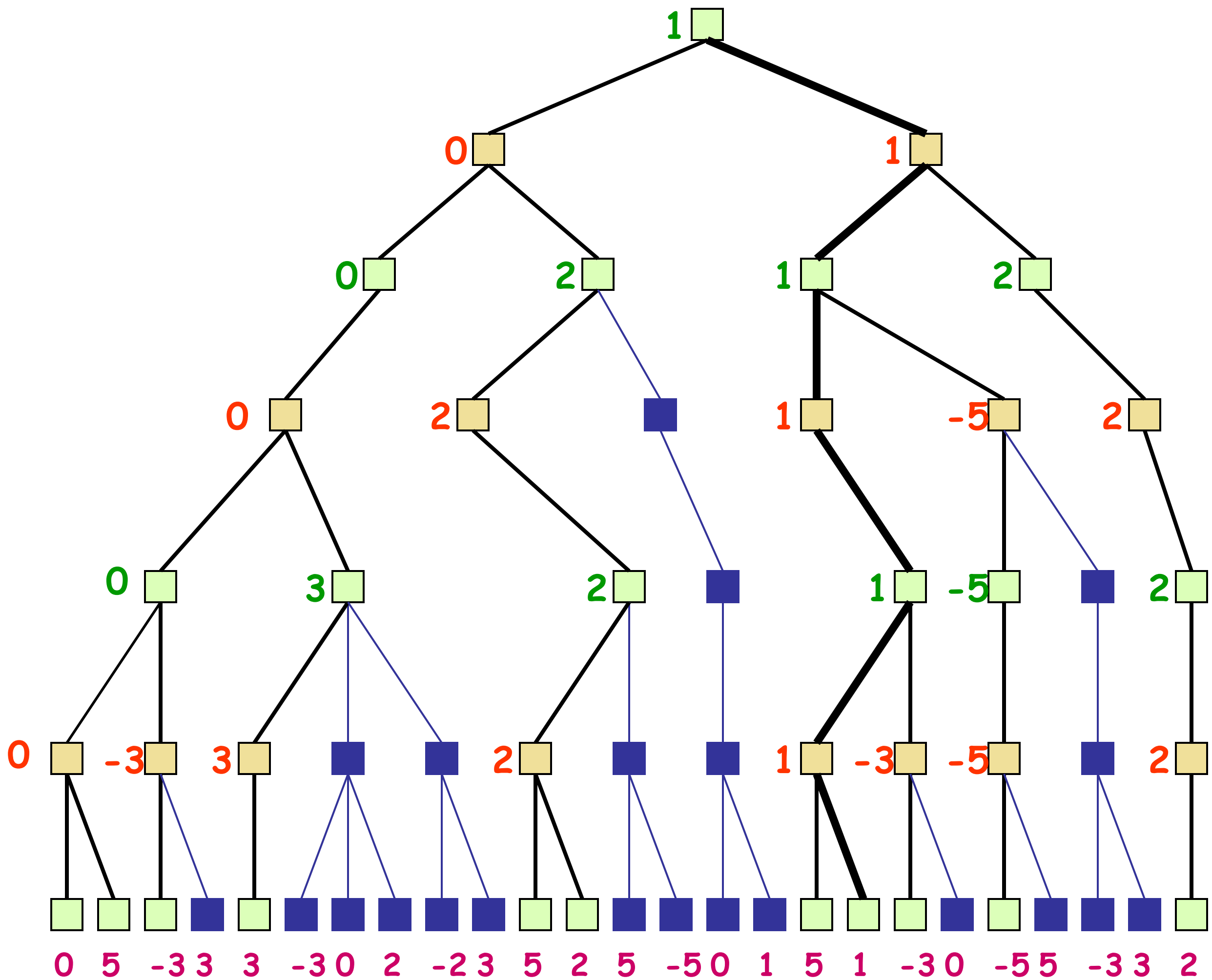
Alpha-Beta Pruning Example-2



Alpha-Beta Pruning Example-2



Alpha-Beta Pruning Example-2



The α-β algorithm

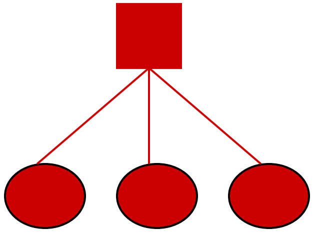
function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game

  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  return the *action* in SUCCESSORS(*state*) with value v

---

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*

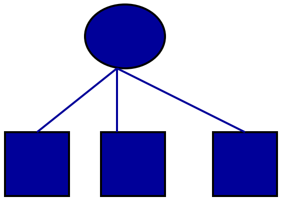
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v cutoff
    $\alpha \leftarrow$ MAX(α, v)
  return v



The α-β algorithm

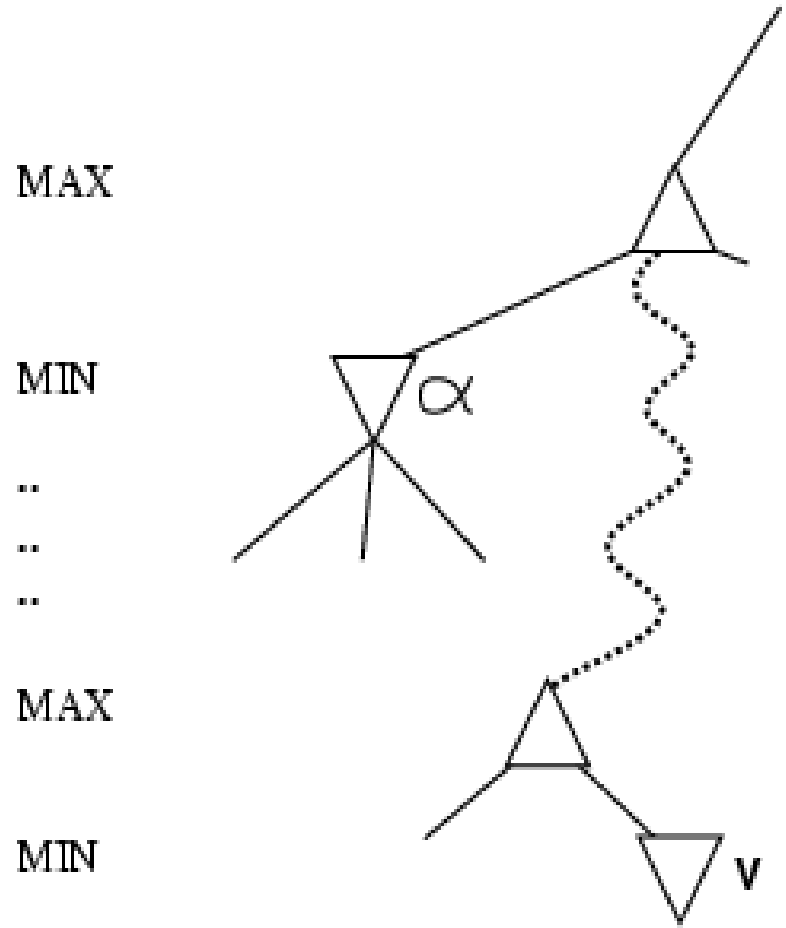
function MIN-VALUE($state, \alpha, \beta$) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

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



Why is it called α-β?

- **α** is the value of the best (i.e., highest-value) choice found so far for *MAX* at any choice point along the path to the root.
  - If *v* is worse than **α**, *MAX* will avoid it
    - → prune that branch
- **β** is the value of the best (i.e., lowest-value) choice found so far for MIN at any choice point along the path for to the root.



Properties of α-β

- Pruning does not affect final result. This means that it gets the exact same result as does full minimax.
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  → doubles depth of search
- A simple example of the value of reasoning about which computations are relevant (a form of metareasoning)

State of the art

- **Chess**:
  - Deep Blue beat Gary Kasparov in 1997
  - Garry Kasparav vs. Deep Junior (Feb 2003): tie!
  - Kasparov vs. X3D Fritz (November 2003): tie!
- **Checkers**: Chinook is the world champion
- **Go**: Computer players are decent, at best
- **Bridge**: “Expert” computer players exist, but no world champions yet
- **Poker:** CPRG regularly beats human experts
- Check out: http://www.cs.ualberta.ca/~games/

Summary

- Games are fun to work on!
- They illustrate several important points about AI.
- Perfection is unattainable → must approximate.
- Game playing programs have shown the world what AI can do.