CMSC 471: Games

Spring 2021 (Sections 01 & 03)

Slides courtesy Tim Finin. Some material adopted from notes by Andreas Geyer-Schulz and Chuck Dyer.

Overview

- Game playing
  - State of the art and resources
  - Framework
- Game trees
  - Minimax
  - Alpha-beta pruning
  - Adding randomness

Why study games?

- Interesting, hard problems requiring minimal "initial structure"
- Clear criteria for success
- Study problems involving {hostile, adversarial, competing} agents and uncertainty of interacting with the natural world
- People have used them to assess their intelligence
- Fun, good, easy to understand, PR potential
- Games often define very large search spaces, e.g. chess 35^{100} nodes in search tree, 10^{40} legal states

Chess early days

- **1948**: Norbert Wiener describes how chess program can work using minimax search with an evaluation function
- **1950**: Claude Shannon publishes Programming a Computer for Playing Chess
- **1951**: Alan Turing develops *on paper* 1st program capable of playing full chess games (Turochamp)
- **1958**: 1st program plays full game on IBM 704 (loses)
- **1962**: Kotok & McCarthy (MIT) 1st program to play credibly
- **1967**: Greenblatt’s Mac Hack Six (MIT) defeats a person in regular tournament play

State of the art

- **1979 Backgammon:** BKG (CMU) tops world champ
- **1994 Checkers**: Chinook is the world champion
- **1997 Chess**: IBM Deep Blue beat Gary Kasparov
- **2007 Checkers:** solved (it's a draw)
- **2016 Go**: AlphaGo beat champion Lee Sedol
- **2017 Poker:** CMU's Libratus won $1.5M from 4 top poker players in 3-week challenge in casino
- **20?? Bridge**: Expert bridge programs exist, but no world champions yet

Classical vs. Statistical/Neural Approaches

- We'll look first at the classical approach used from the 1940s to 2010
- Then at newer statistical approached of which AlphaGo is an example
- These share some techniques

Typical simple case for a game

- **2-person** game, with **alternating moves**
- **Zero-sum**: one player's loss is the other's gain
- **Perfect information**: both players have access to complete information about state of game. No information hidden from either player.
- **No chance** (e.g., using dice) involved

Typical simple case for a game

- **2-person** game, with **alternating moves**
- **Zero-sum**: one player's loss is the other's gain
- **Perfect information**: both players have access to complete information about state of game. No information hidden from either player.
- **No chance** (e.g., using dice) involved
- Examples: Tic-Tac-Toe, Checkers, Chess, Go, Nim, Othello
- But not: Bridge, Solitaire, Backgammon, Poker, Rock-Paper-Scissors, ...

Can we use …

- Uninformed search?
- Heuristic search?
- Local search?
- Constraint based search?

None of these model the fact that we have an **adversary** …

How to play a game

- A way to play such a game is to:
  - Consider all the legal moves you can make
  - Compute new position resulting from each move
  - Evaluate each to determine which is best
  - Make that move
  - Wait for your opponent to move and repeat
- Key problems are:
  - Representing the "board" (i.e., game state)
  - Generating all legal next boards
  - Evaluating a position

Evaluation function

- **Evaluation function** or **static evaluator** used to evaluate the "goodness" of a game position

  Contrast with heuristic search, where evaluation function estimates **cost** from start node to goal passing through given node

- Zero-sum assumption permits single function to describe goodness of board for both players
  - **f(n) >> 0**: position n good for me; bad for you
  - **f(n) << 0**: position n bad for me; good for you
  - **f(n) near 0**: position n is a neutral position
  - **f(n) = +infinity**: win for me
  - **f(n) = -infinity**: win for you

Evaluation function examples

- **For Tic-Tac-Toe**

  f(n) = [# my open 3lengths] - [# your open 3lengths]

  Where 3length is complete row, column or diagonal that has no opponent marks

- **Alan Turing's function for chess**
  - **f(n) = w(n)/b(n)** where w(n) = sum of point value of white's pieces and b(n) = sum of black's
  - Traditional piece values: pawn:1; knight:3; bishop:3; rook:5; queen:9

Evaluation function examples

- Most evaluation functions specified as a weighted sum of positive features

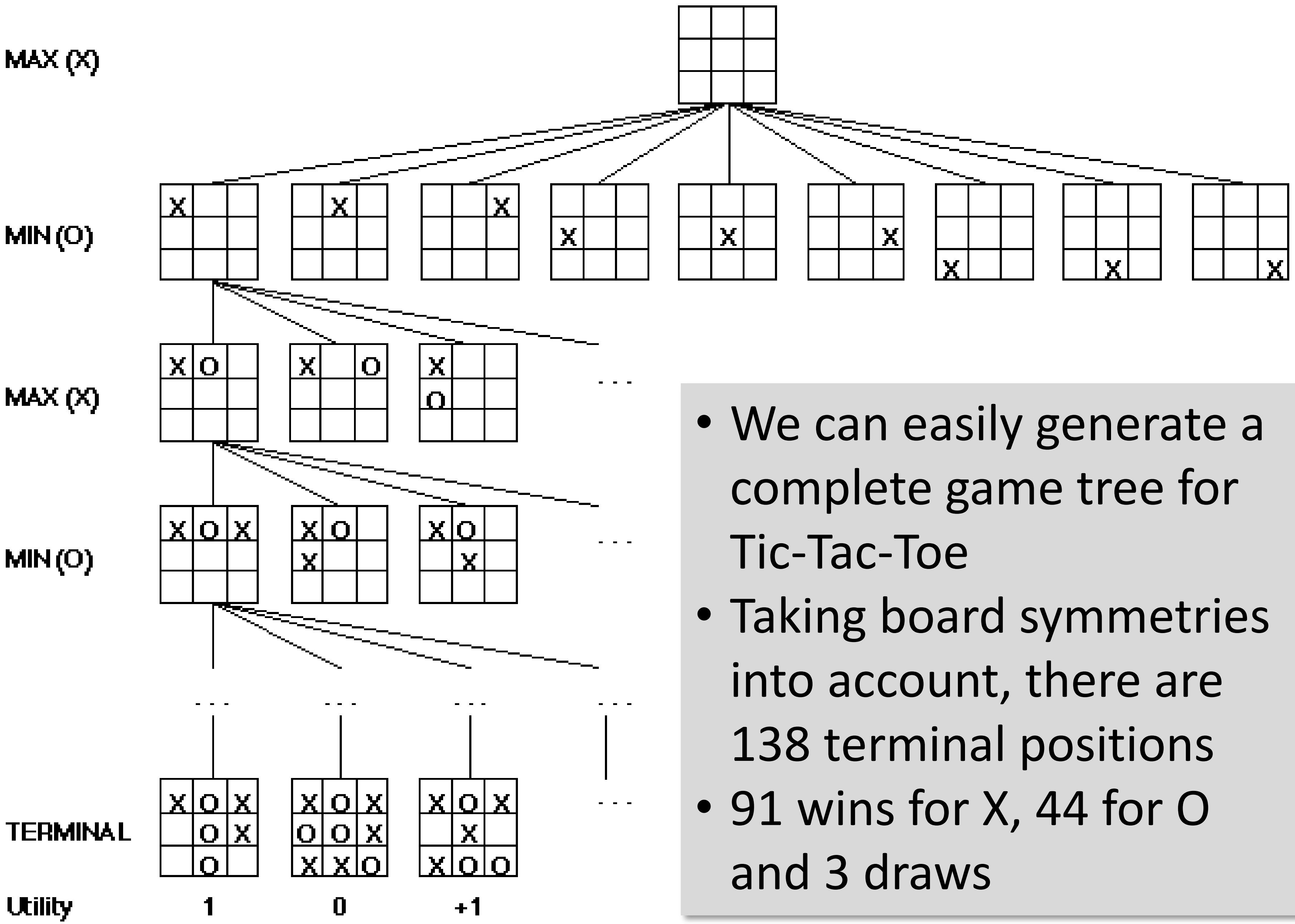
  $f(n) = w_1*feat_1(n) + w_2*feat_2(n) + ... + w_n*feat_k(n)$

- Example chess features are piece count, piece values, piece placement, squares controlled, etc.
- IBM’s chess program Deep Blue (circa 1996) had >8K features in its evaluation function

But, that’s not how people play

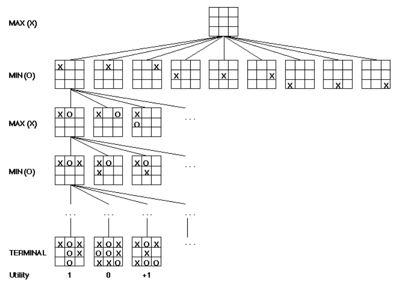
- People also use *look ahead*

  i.e., enumerate actions, consider opponent’s possible responses, REPEAT
- Producing a *complete* **game tree** is only possible for simple games
- So, generate a partial game tree for some number of plys
  - Move = each player takes a turn
  - Ply = one player’s turn
- What do we do with the game tree?



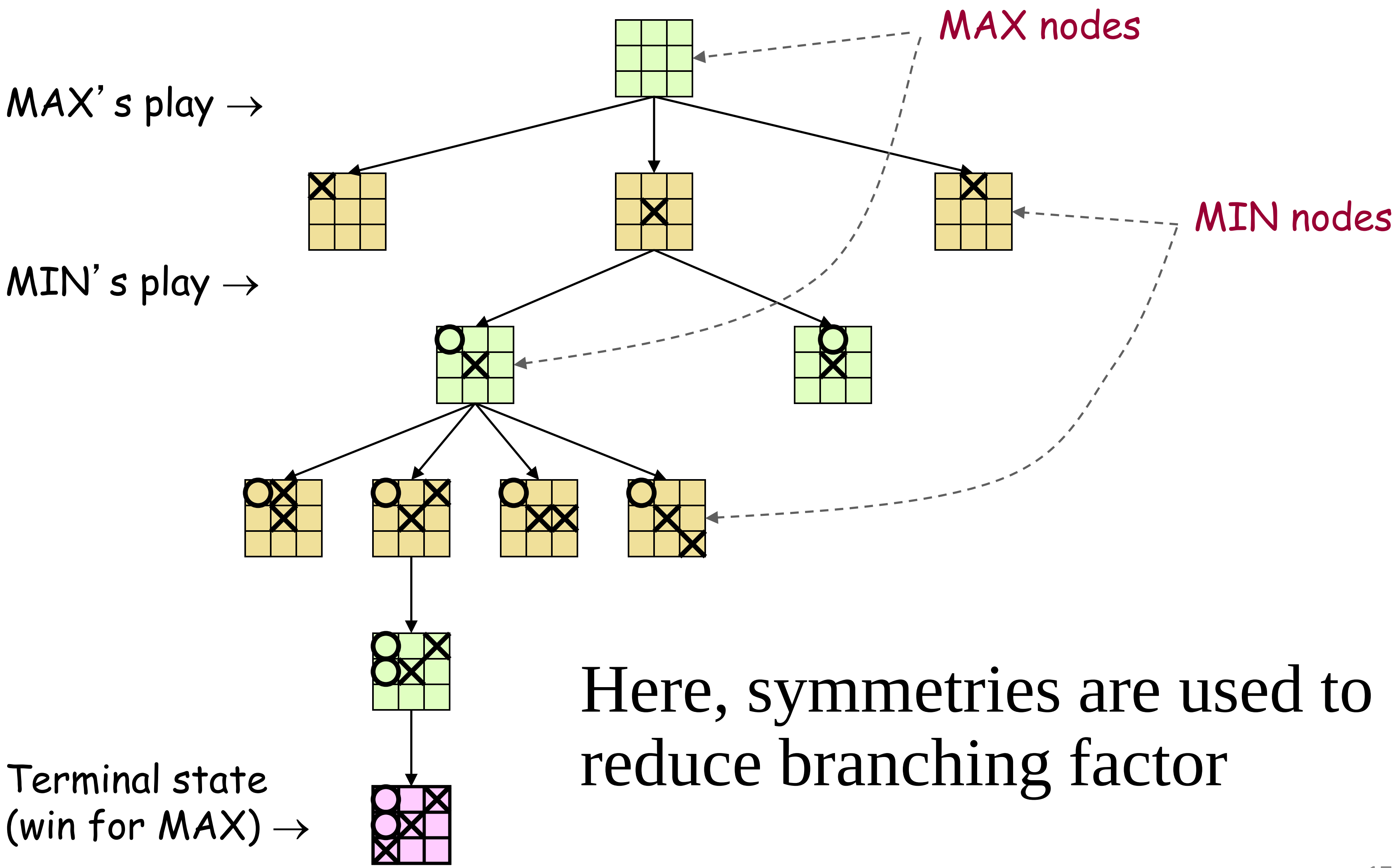
- We can easily generate a complete game tree for Tic-Tac-Toe
- Taking board symmetries into account, there are 138 terminal positions
- 91 wins for X, 44 for O and 3 draws

Game trees



- Problem spaces for typical games are trees
- Root node is current board configuration; player must decide best single move to make next
- **Static evaluator function** rates board position **f(board):**real, >0 for me; <0 for opponent
- Arcs represent possible legal moves for a player
- If **my turn** to move, then root is labeled a "**MAX**" node; otherwise it's a "**MIN**" node
- Each tree level's nodes are all MAX or all MIN; nodes at level i are of opposite kind from those at level i+1

Game Tree for Tic-Tac-Toe



Here, symmetries are used to reduce branching factor

Minimax procedure

- Create MAX node with current board configuration
- Expand nodes to some **depth** (a.k.a. **plys**) of lookahead in game
- Apply evaluation function at each **leaf** node
- *Back up* values for each non-leaf node until value is computed for the root node
  - At MIN nodes: value is **minimum** of children's values
  - At MAX nodes: value is **maximum** of children's values
- Choose move to child node whose backed-up value determined value at root

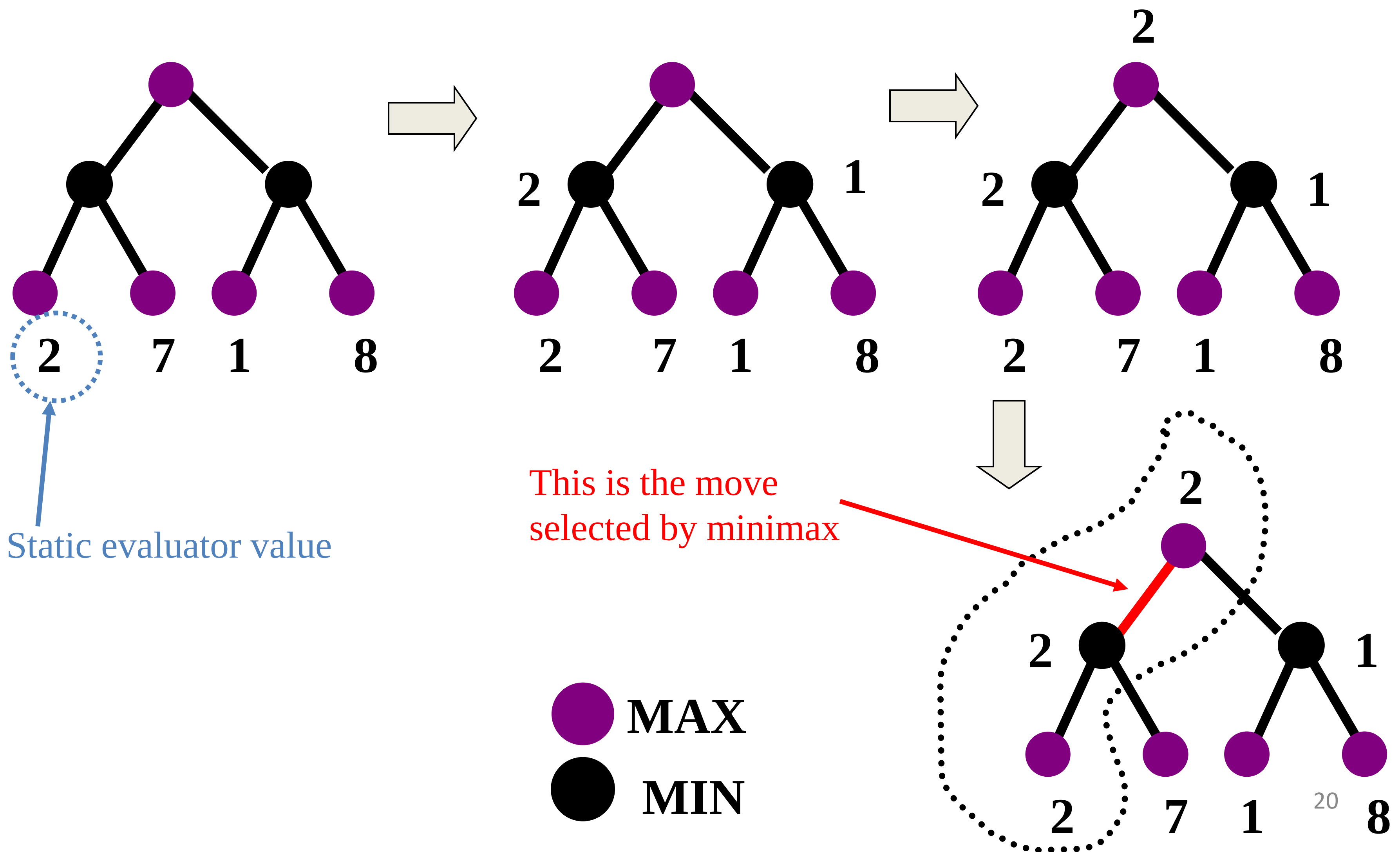
Minimax theorem

- Intuition: assume your opponent is at least as smart as you and play accordingly
  - If she's not, you can only do better!
- Von Neumann, J: *Zur Theorie der Gesellschafts-spiele* Math. Annalen. **100** (1928) 295-320

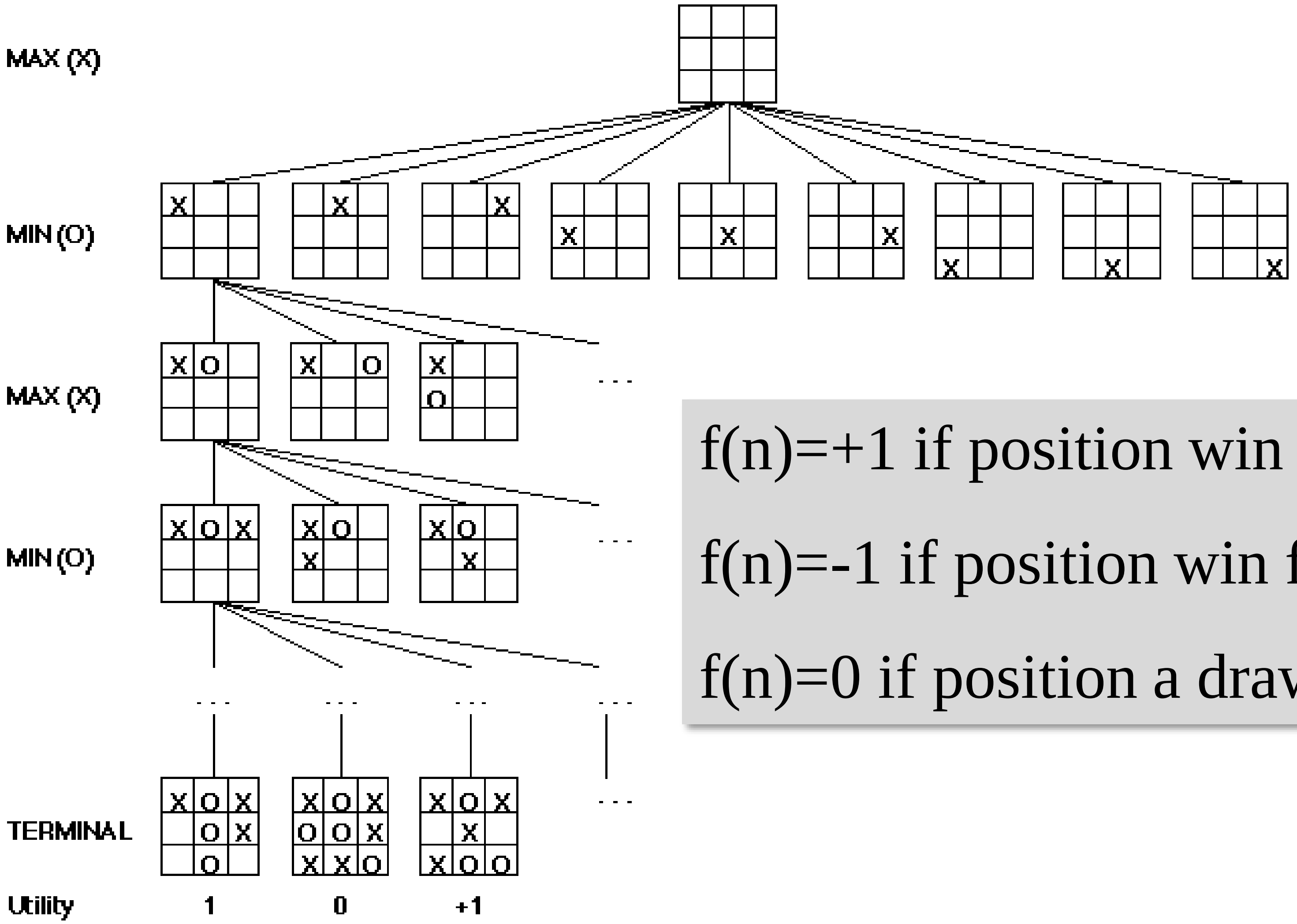
For every 2-person, 0-sum game with finite strategies, there is a value V and a mixed strategy for each player, such that (a) given player 2's strategy, best payoff possible for player 1 is V, and (b) given player 1's strategy, best payoff possible for player 2 is –V.

- You can think of this as:
  - Minimizing your maximum possible loss
  - Maximizing your minimum possible gain

Minimax Algorithm



Partial Game Tree for Tic-Tac-Toe



f(n)=+1 if position win for X

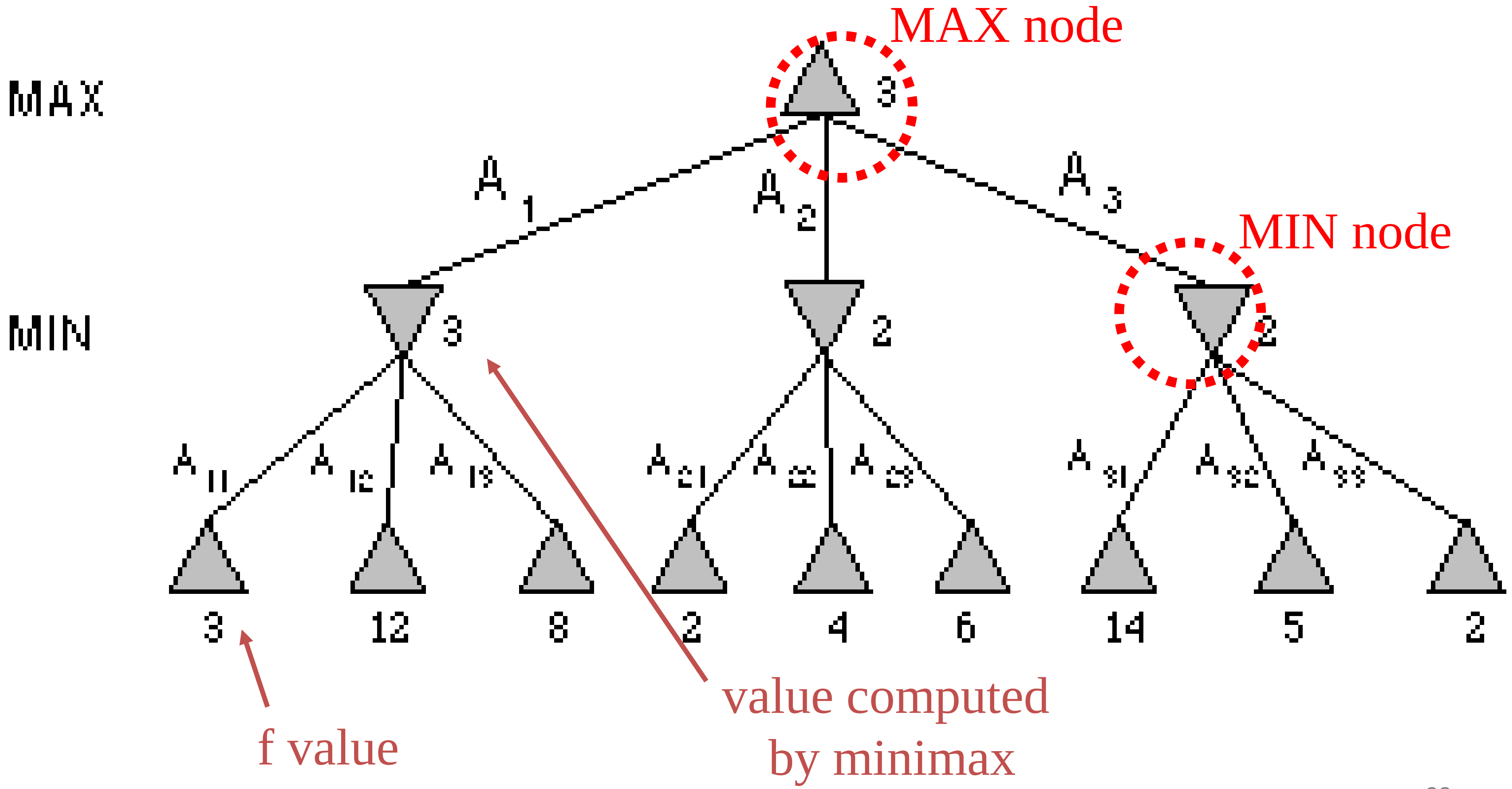
f(n)=-1 if position win for O

f(n)=0 if position a draw

Why backed-up values?

- Why not just use a good static evaluator metric on immediate children
- **Intuition:** if metric is good, doing look ahead and backing up values with Minimax should be better
- Non-leaf node N's backed-up value is value of best state MAX can reach at depth **h** if MIN plays *well*
  - "plays well": same criterion as MAX applies to itself
- If e is good, then backed-up value is better estimate of STATE(N) goodness than e(STATE(N))
- Use lookahead horizon **h** because time to choose move is limited

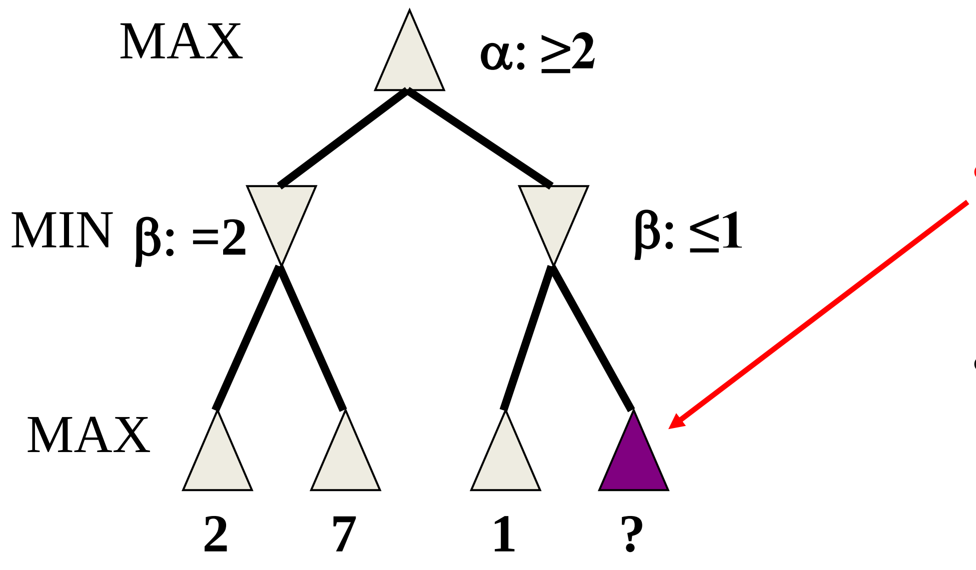
Minimax Tree



Is that all there is to simple games?

Alpha-beta pruning

- Improve performance of the minimax algorithm through **alpha-beta pruning**
- *"If you have an idea that is surely bad, don't take the time to see how truly awful it is"* -Pat Winston (MIT)



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

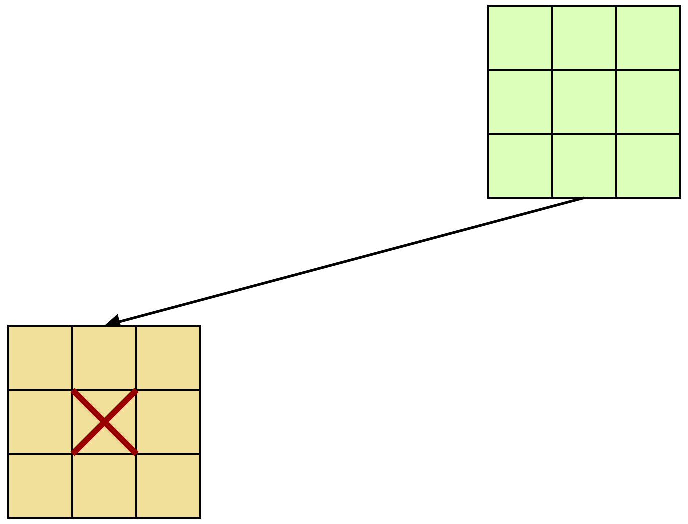
Alpha-beta pruning

- Traverse search tree in depth-first order
- At **MAX** node n, **alpha(n)** = max value found so far

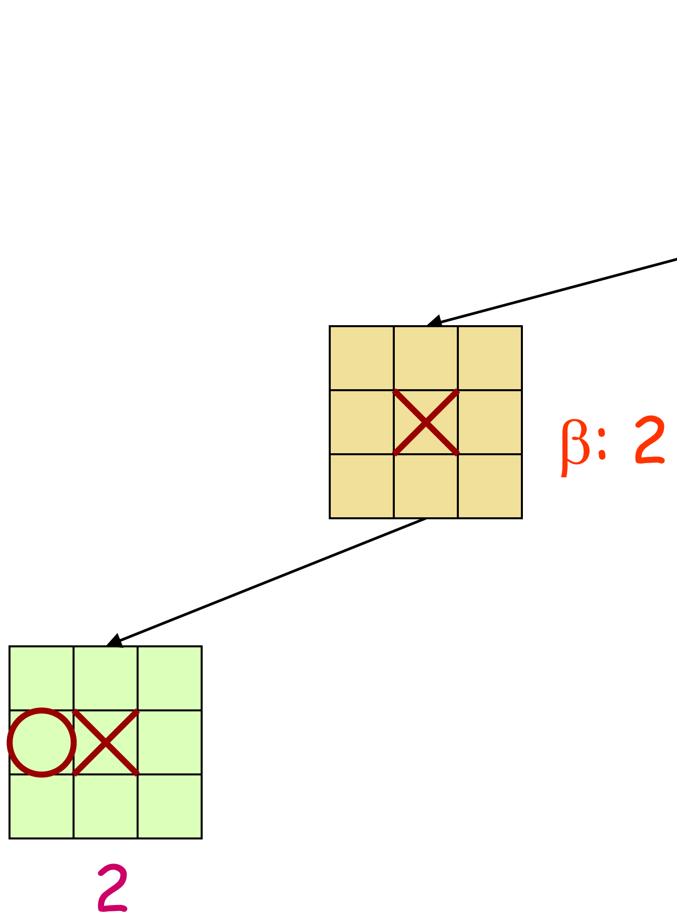
  Alpha values start at -∞ and only increase
- At **MIN** node n, **beta(n)** = min value found so far

  Beta values start at +∞ and only decrease
- **Beta cutoff**: stop search below MAX node N (i.e., don’t examine more descendants) if alpha(N) >= beta(i) for some MIN node ancestor i of N
- **Alpha cutoff:** stop search below MIN node N if beta(N)<=alpha(i) for a MAX node anceastor i of N

Alpha-Beta Tic-Tac-Toe Example

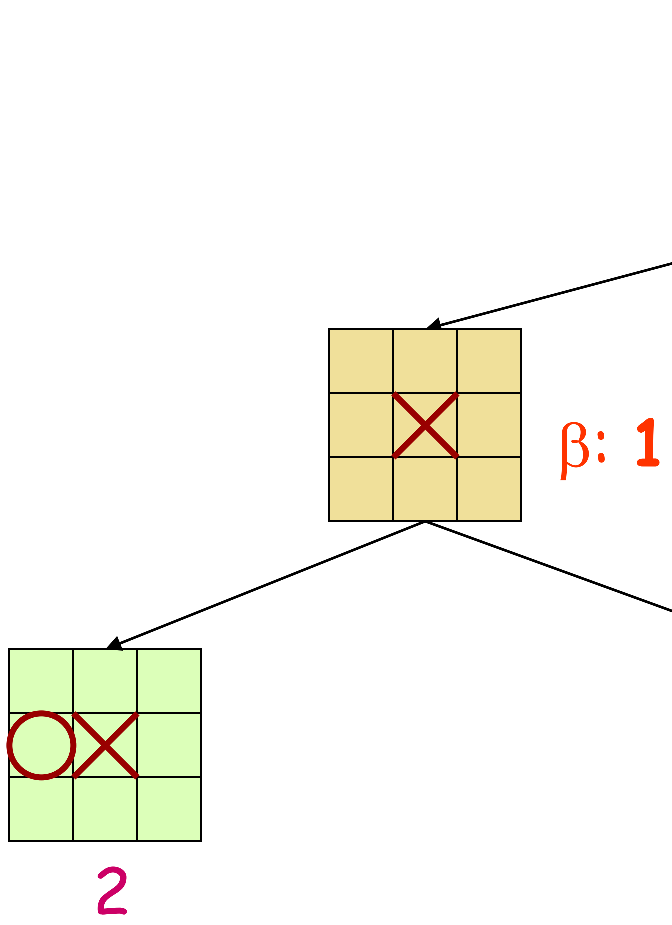


Alpha-Beta Tic-Tac-Toe Example



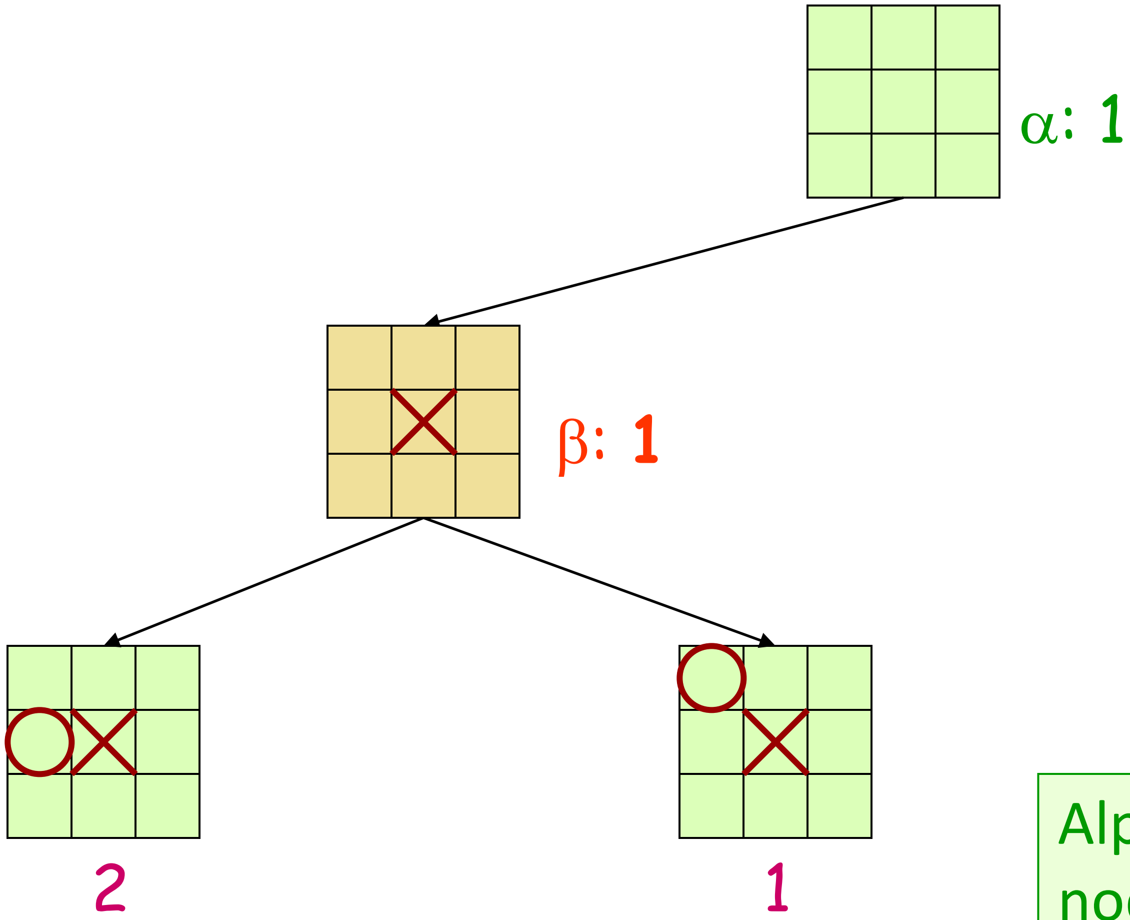
Beta value of a MIN node is **upper** bound on final backed-up value; it can never increase

Alpha-Beta Tic-Tac-Toe Example



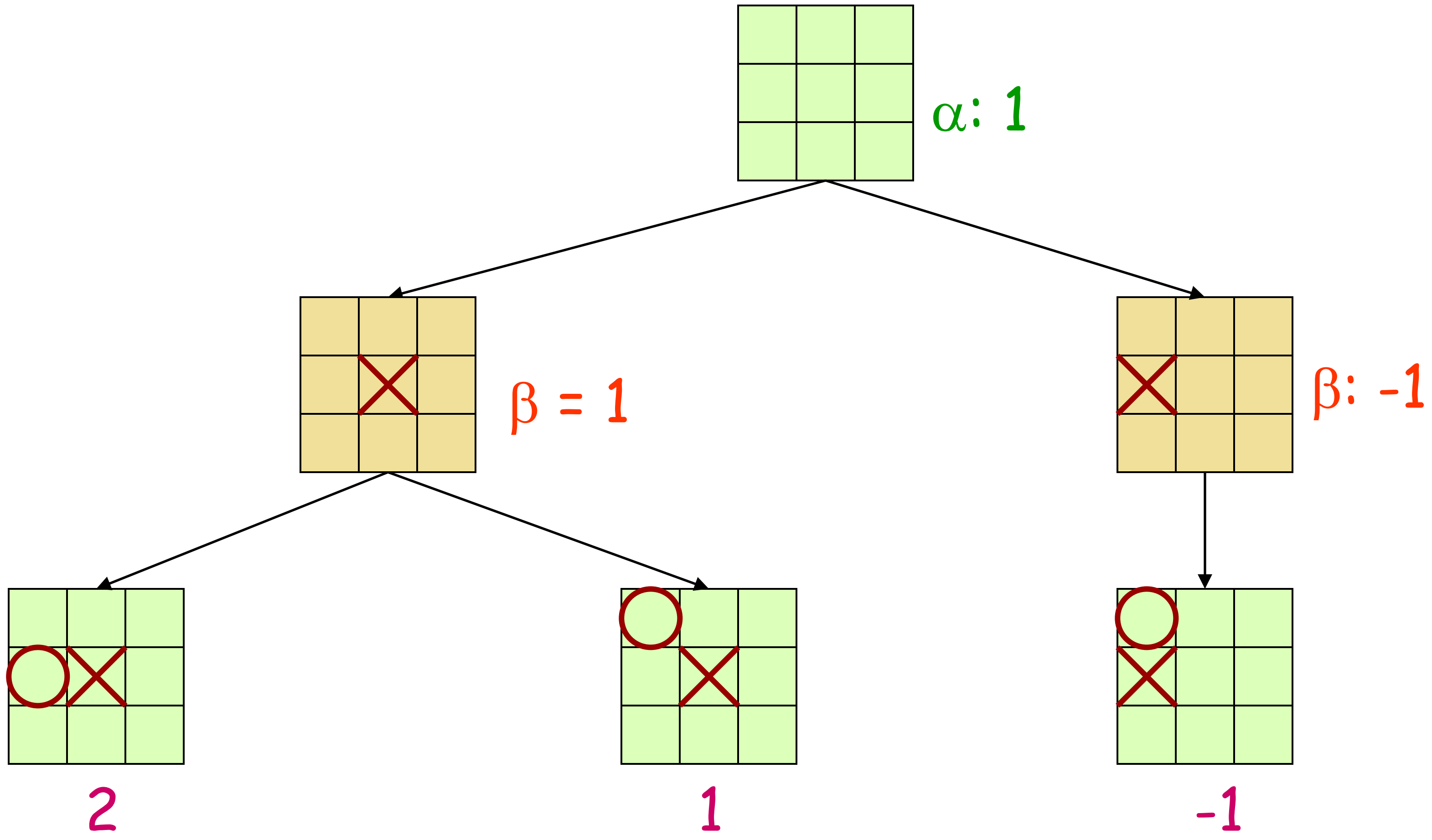
Beta value of a MIN node is **upper** bound on final backed-up value; it can never increase

Alpha-Beta Tic-Tac-Toe Example

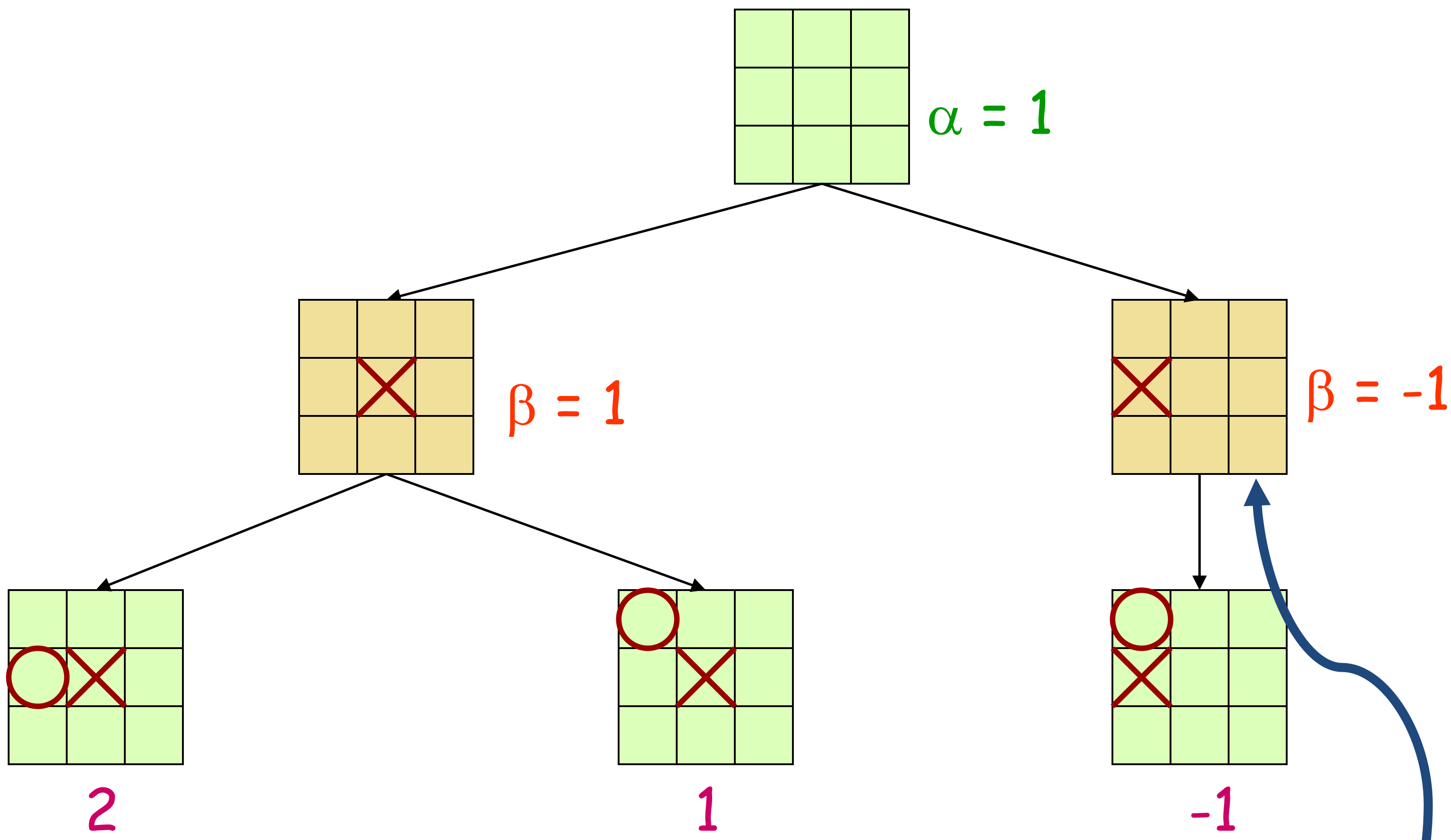


Alpha value of MAX node is **lower** bound on final backed-up value; it can never decrease

Alpha-Beta Tic-Tac-Toe Example



Alpha-Beta Tic-Tac-Toe Example



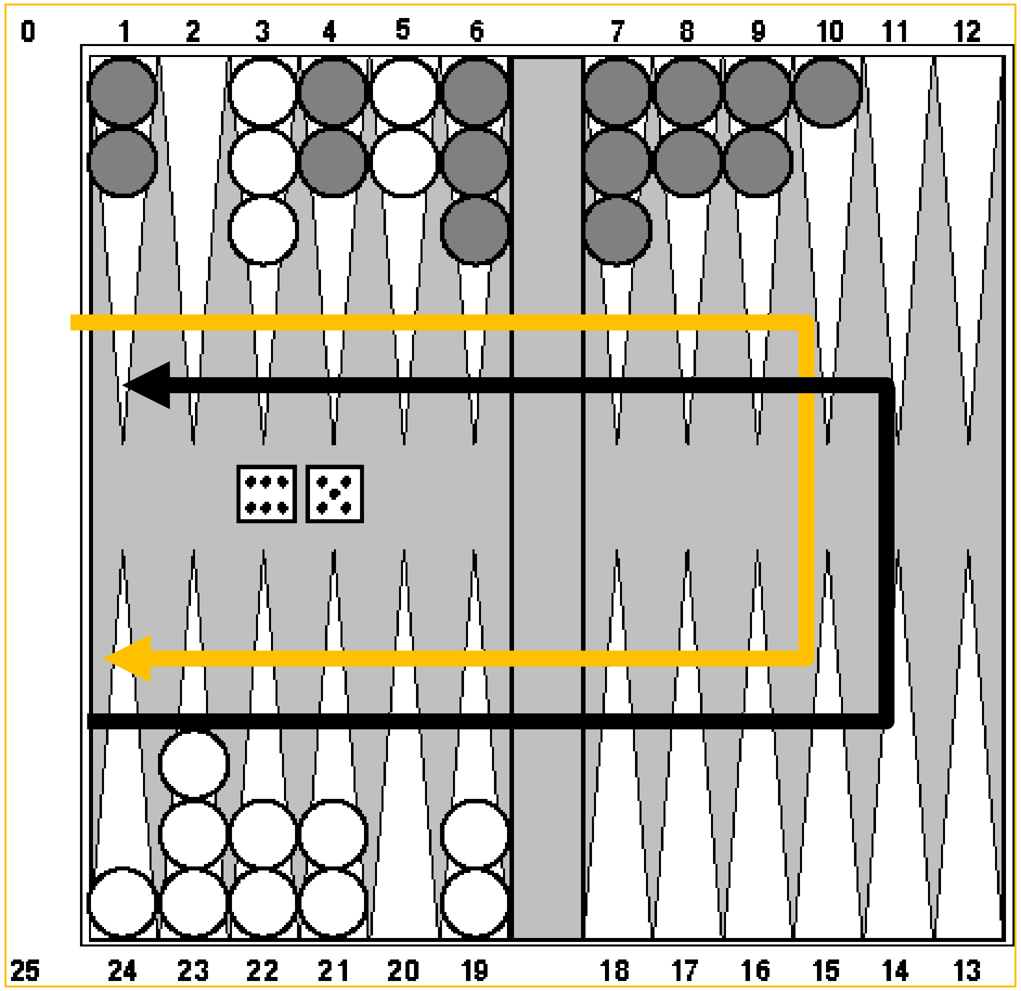
Discontinue search below a MIN node whose beta value ≤ alpha value of one of its MAX ancestors

Stochastic Games

- In real life, unpredictable external events can put us into unforeseen situations
- Many games introduce unpredictability through a random element, such as the throwing of dice
- These offer simple scenarios for problem solving with adversaries and uncertainty

Example: Backgammon

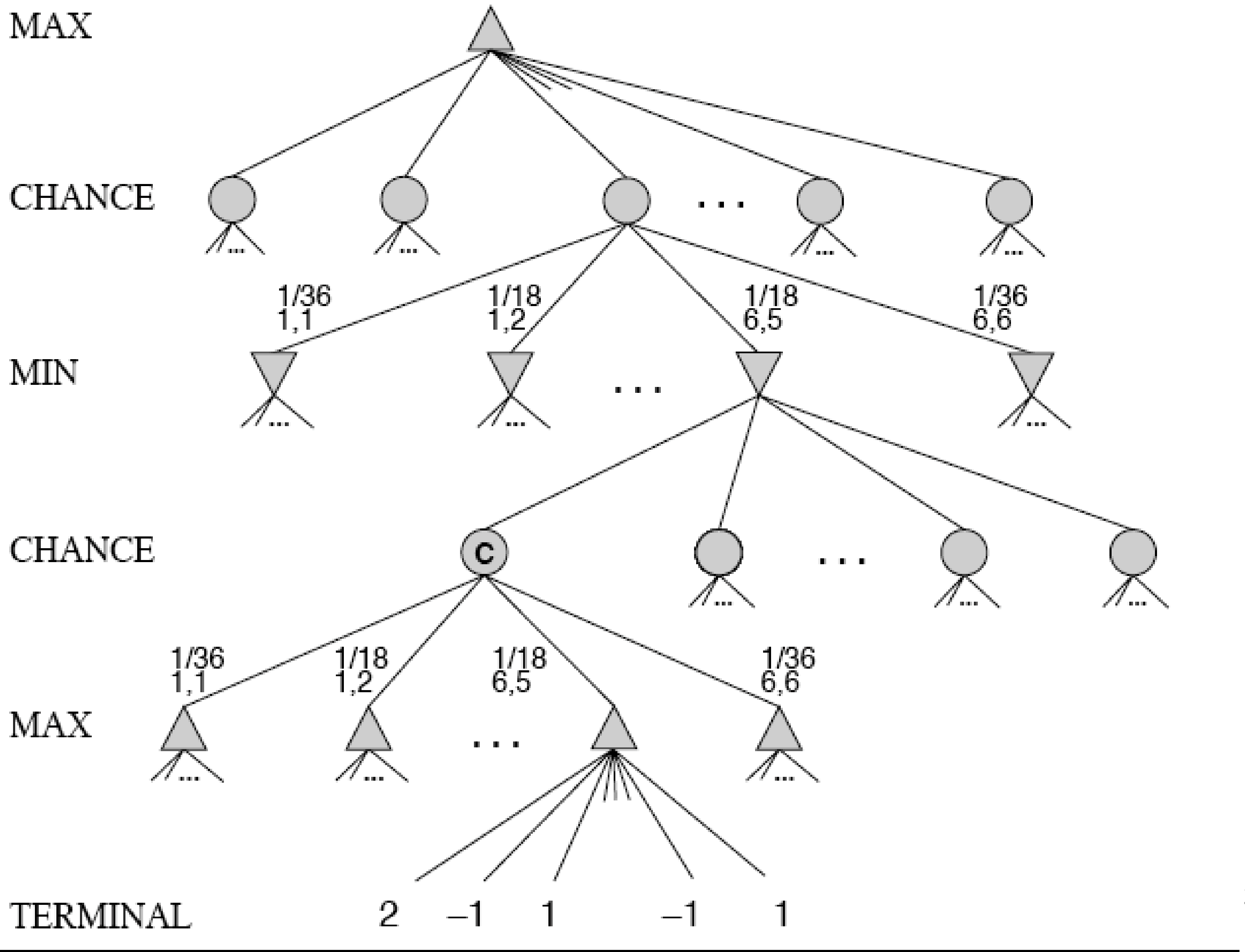
- Popular two-player game with uncertainty
- Players roll dice to determine what moves can be made
- White has just rolled 5 & 6, giving four legal moves:
  - 5-10, 5-11
  - 5-11, 19-24
  - 5-10, 10-16
  - 5-11, 11-16
- Good for exploring decision making in adversarial prob-lems involving skill **and** luck



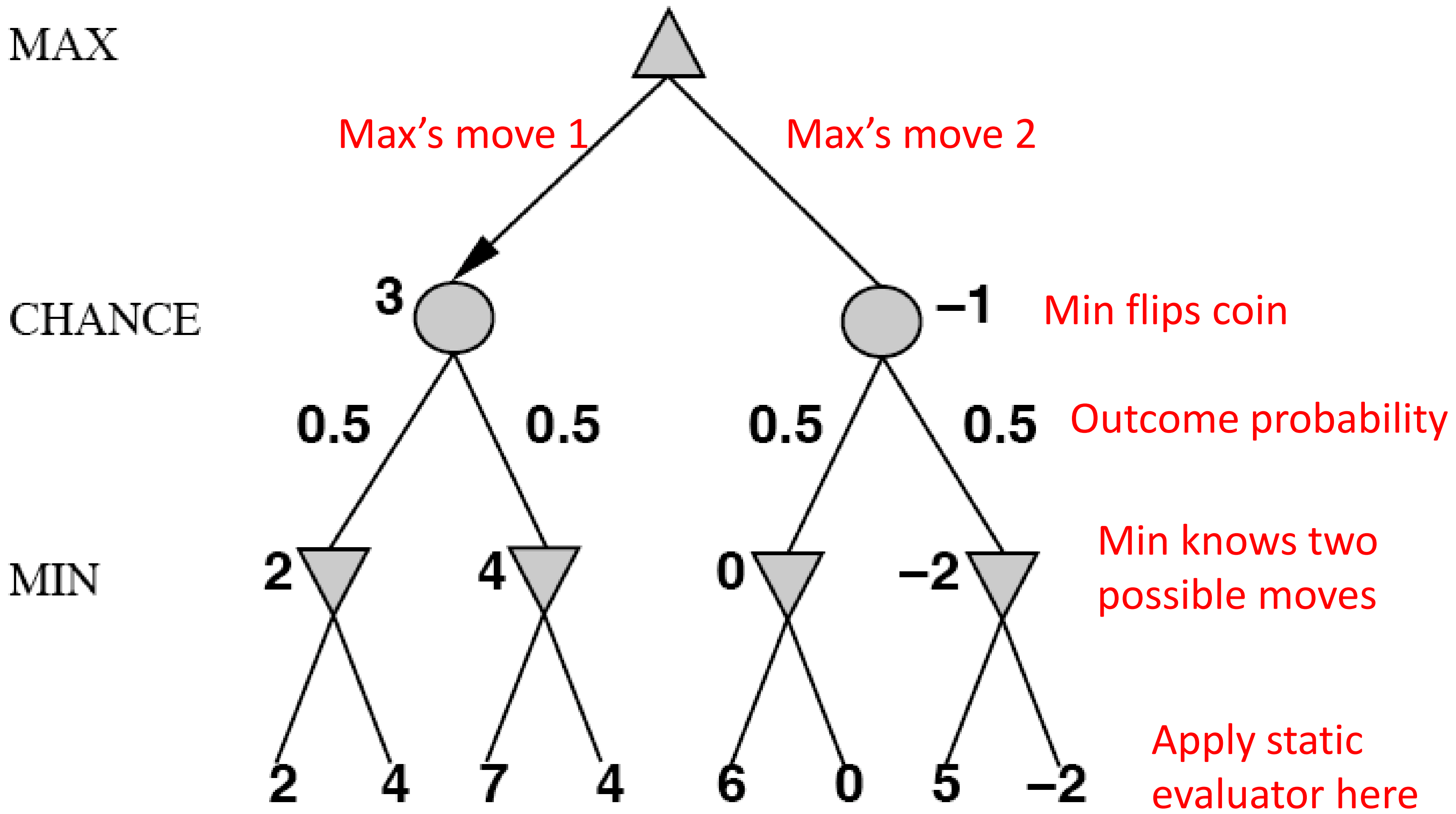
Why can’t we use MiniMax?

- Before a player chooses a move, she rolls dice and only then knows exactly what moves are possible
- The immediate outcome of each move is also known
- But she does not know what moves she or her opponent will have available in the future
- Need to adapt MiniMax to handle this

MiniMax trees with Chance Nodes



Understanding the notation

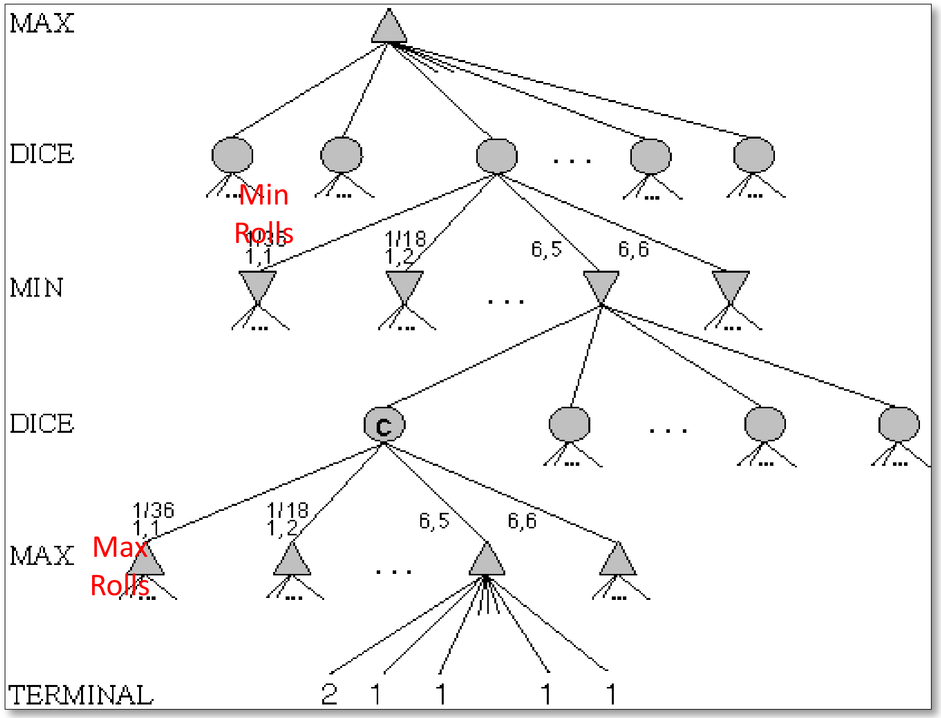


Board state includes chance outcome determining available moves

Game trees with chance nodes

- Chance nodes (circles) represent random events
- For random event with N outcomes, chance node has N children, each with a probability
- 2 dice: 21 distinct outcomes
- Use minimax to compute values for MAX and MIN nodes
- Use expected values for chance nodes

- Chance nodes over max node: expectimax(C) = $\sum_i(P(d_i)*\text{maxval}(i))$
- Chance nodes over min node: expectimin(C) = $\sum_i(P(d_i)*\text{minval}(i))$

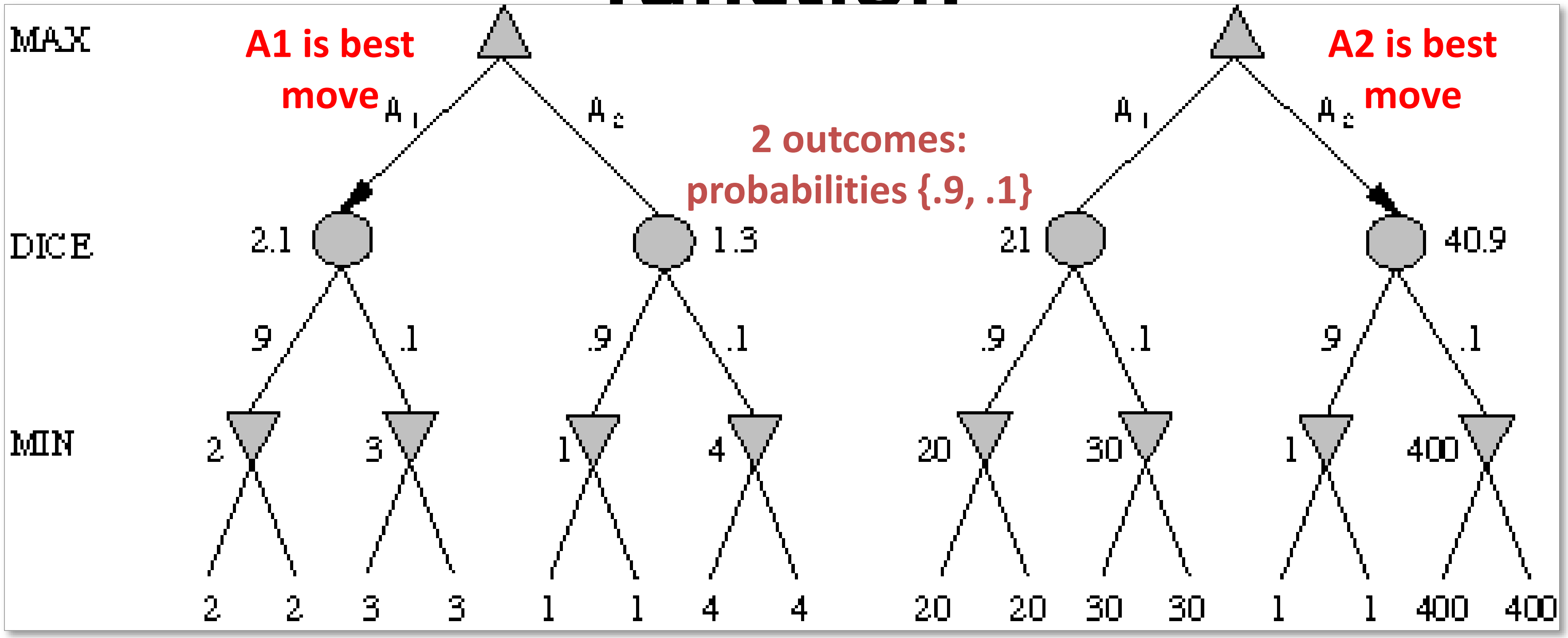


Impact on lookahead

- Dice rolls **increase branching factor**
  - There are 21 possible rolls with two dice
- Backgammon: ~20 legal moves for given roll

  ~6K with 1-1 roll (get to roll again!)
- At depth 4: 20 * (21 * 20)**3 ≈ 1.2B boards
- As depth increases, probability of reaching a given node shrinks
  - lookahead's value diminished and alpha-beta pruning is much less effective
- TDGammon used depth-2 search + good static evaluator to achieve world-champion level

Meaning of the evaluation function



- With probabilities & expected values we must be careful about meaning of values returned by static evaluator
- Relative-order preserving change of static evaluation values doesn't change minimax decision, but could here
- Linear transformations are OK

Games of imperfect information

- E.g. card games where opponent's initial hanc unknown
  - Can calculate probability for each possible deal
  - Like having one big dice roll at beginning of game
- Possible approach: minimax over each action in each deal; choose action with highest expected value over all deals
- Special case: if action optimal for all deals, it's optimal
- GIB bridge program, approximates this idea by

1. Generating 100 deals consistent with bidding
2. Picking action that wins most tricks on average

High-Performance Game Programs

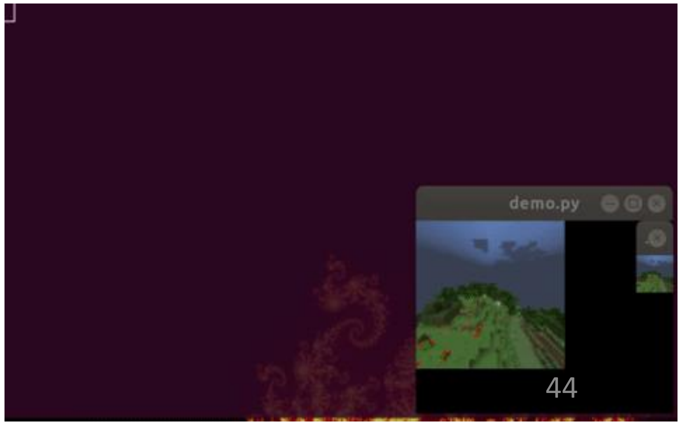
- Many programs based on alpha-beta + iterative deepening + extended/singular search + transposition tables + huge databases + ...
- Chinook searched all checkers configurations with ≤ 8 pieces to create endgame database of 444 billion board configurations
- Methods general, but implementations improved via many specifically tuned-up enhancements (e.g., the evaluation functions)

Other Issues

- Multi-player games, no alliances
  - E.g., many card games, like Hearts
- Multi-player games with alliances
  - E.g., Risk
  - More on this when we discuss game theory
  - Good model for a social animal like humans, where we must balance cooperation and competition

AI and video Games

- Many games include agents run by the game program as
  - Adversaries, in first person shooter games
  - Collaborators, in a virtual reality game
  - E.g.: AI bots in Fortnite Chapter 2
- Some games used as AI/ML challenges or learning environments
  - MineRL: train bots to play Minecraft
  - MarioAI: train bots for Super Mario Bros



General Game Playing

- General Game Playing is an idea developed by Michael Genesereth of Stanford
- See his site for more information
- Idea: don’t develop specialized systems to play specific games (e.g., Checkers) well
- Goal: design AI programs to be able to play more than one game successfully
- **Work from a description of a novel game**

A example of General Intelligence

- Artificial General Intelligence describes research that aims to create machines capable of general intelligent action
- Harkens back to early visions of AI, like McCarthy's Advise Taker
  - See Programs with Common Sense (1959)
- A response to frustration with narrow specialists, often seen as "hacks"
  - See On Chomsky and the Two Cultures of Statistical Learning

Perspective on Games: Con and Pro

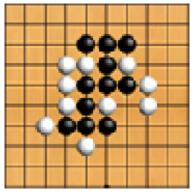
"Chess is the Drosophila of artificial intelligence. However, computer chess has developed much as genetics might have if the geneticists had concentrated their efforts starting in 1910 on breeding racing Drosophila. We would have some science, but mainly we would have very fast fruit flies."

John McCarthy, Stanford

"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, Yale

AlphaGO



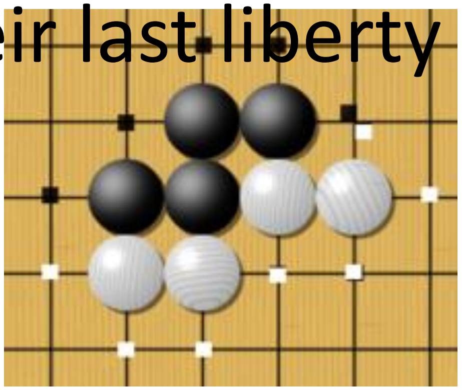
- Developed by Google's DeepMind
- Beat top-ranked human grandmasters in 2016
- Used Monte Carlo tree search over game tree

  expands search tree via random sampling of search space
- *Science* Breakthrough of the year runner-up

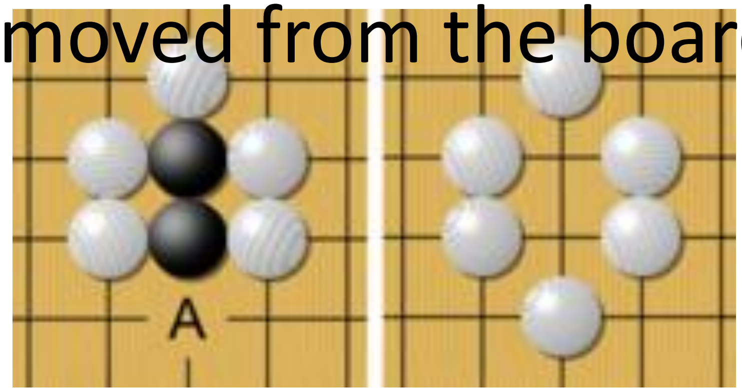
  Mastering the game of Go with deep neural networks and tree search, Silver et al., *Nature*, 529:484–489, 2016
- Match with grandmaster Lee Sedol in 2016

Go - the game

- Played on 19x19 board; black vs. white stones
- Huge state space $O(b^d)$: chess:~35^{80}, go: ~250^{150}
- Rule: Stones on board must have an adjacent open point ("liberty") or be part of connected group with a liberty. Groups of stones losing their last liberty are removed from the board.



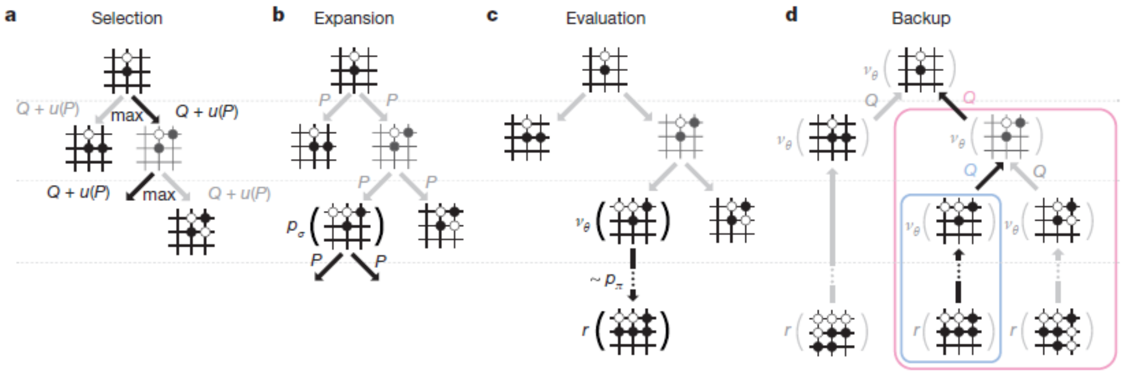
liberties



capture

AlphaGo implementation

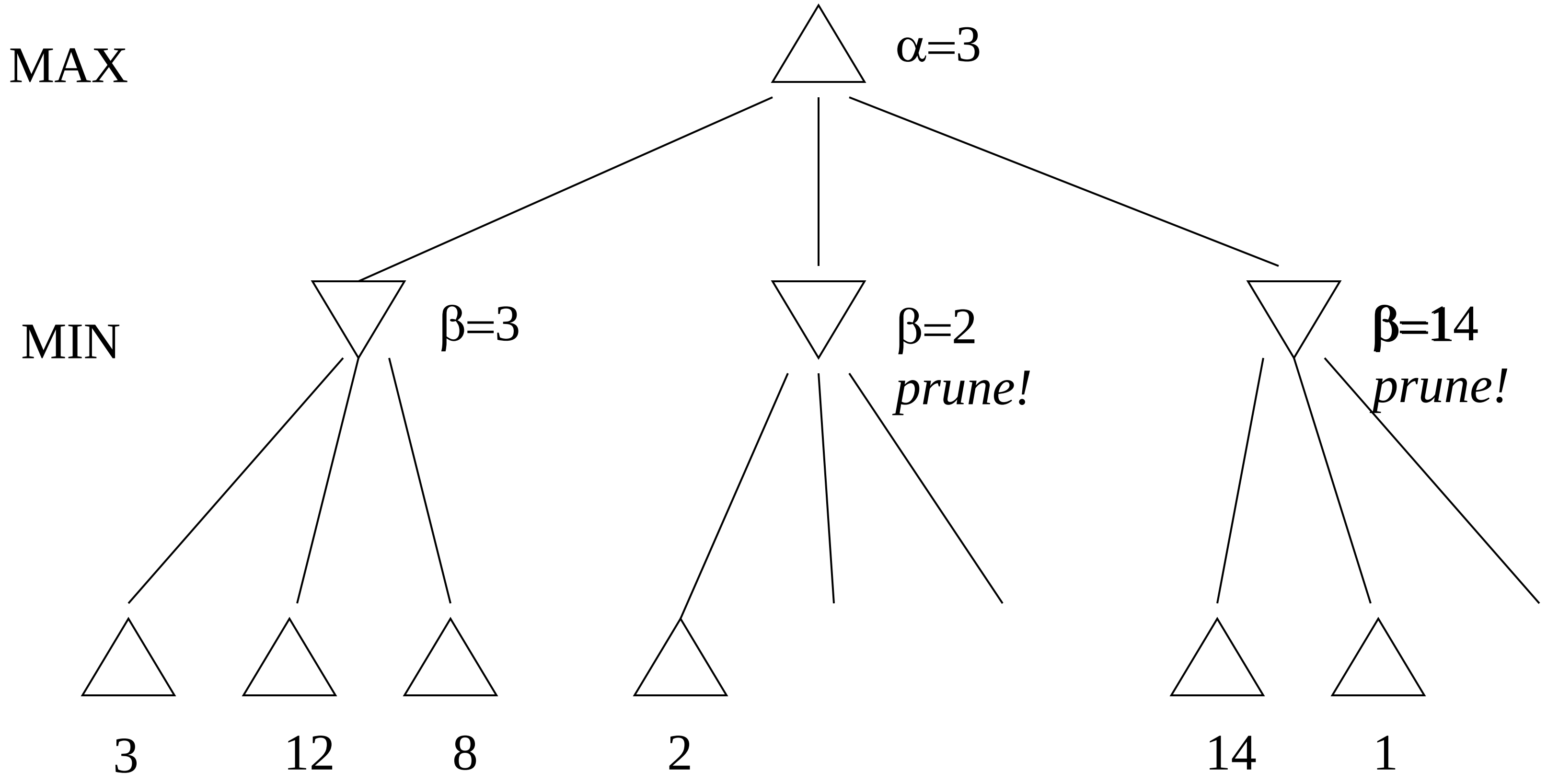
- Trained deep neural networks (13 layers) to learn **value function** and **policy function**
- Performs Monte Carlo game search
  - –explore state space like minimax
  - –random "rollouts"
  - –simulate probable plays by opponent according to policy function



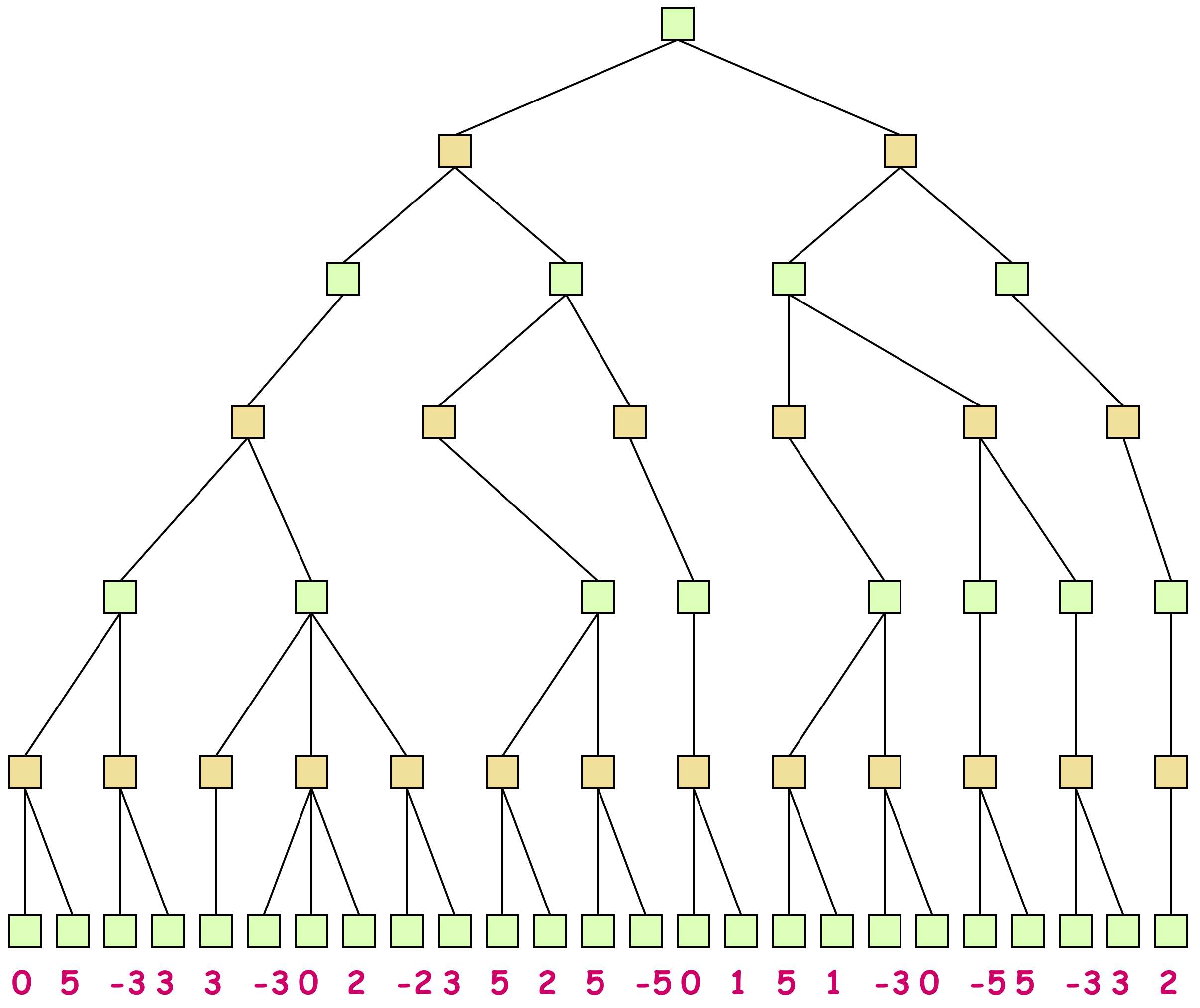
AlphaGo implementation

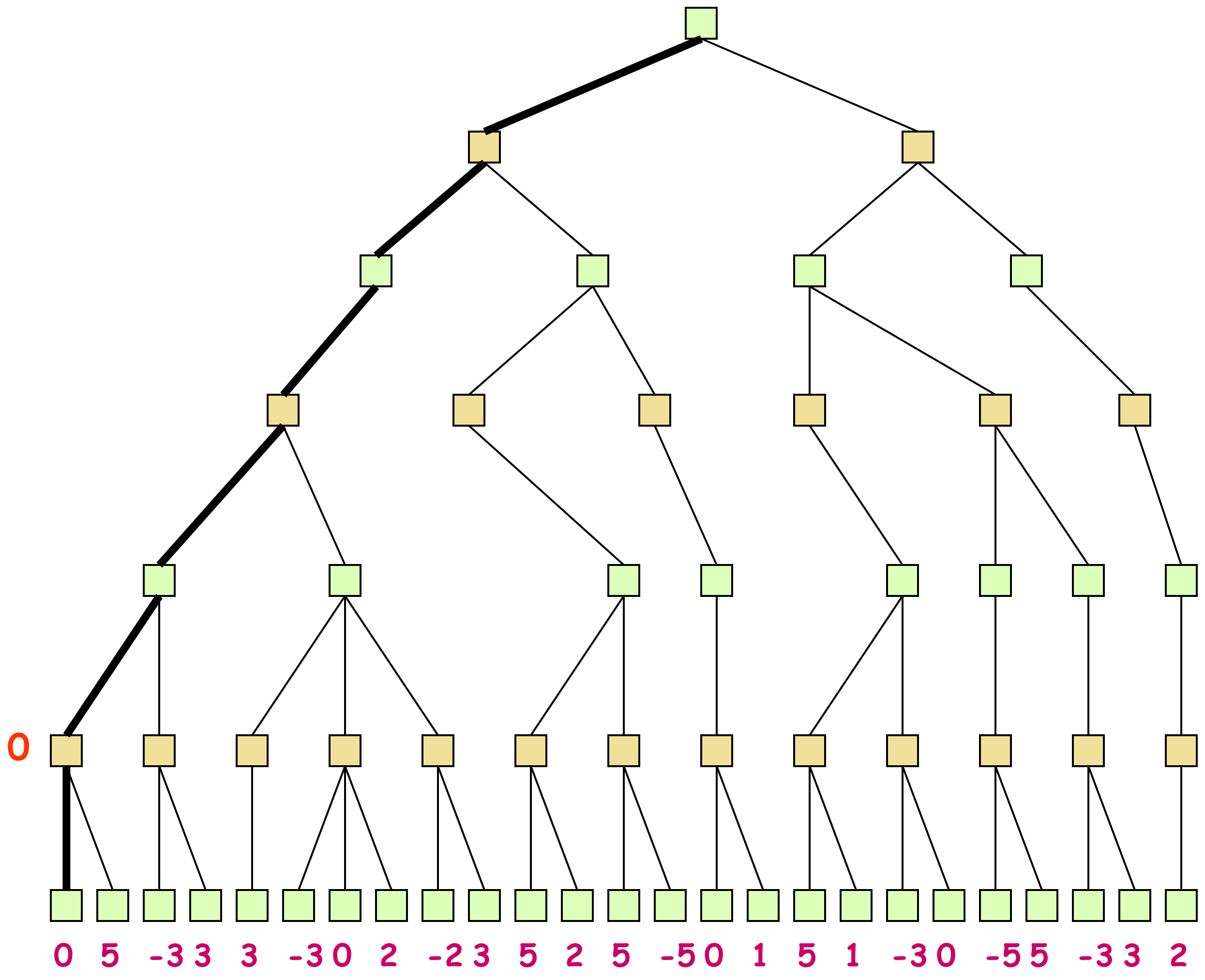
- Hardware: 1920 CPUs, 28O GPUs
- Neural networks trained in two phases over 4-6 weeks
- **Phase 1:** supervised learning from database of 30 million moves in games between two good human players
- **Phase 2:** play against versions of self using reinforcement learning to improve performance

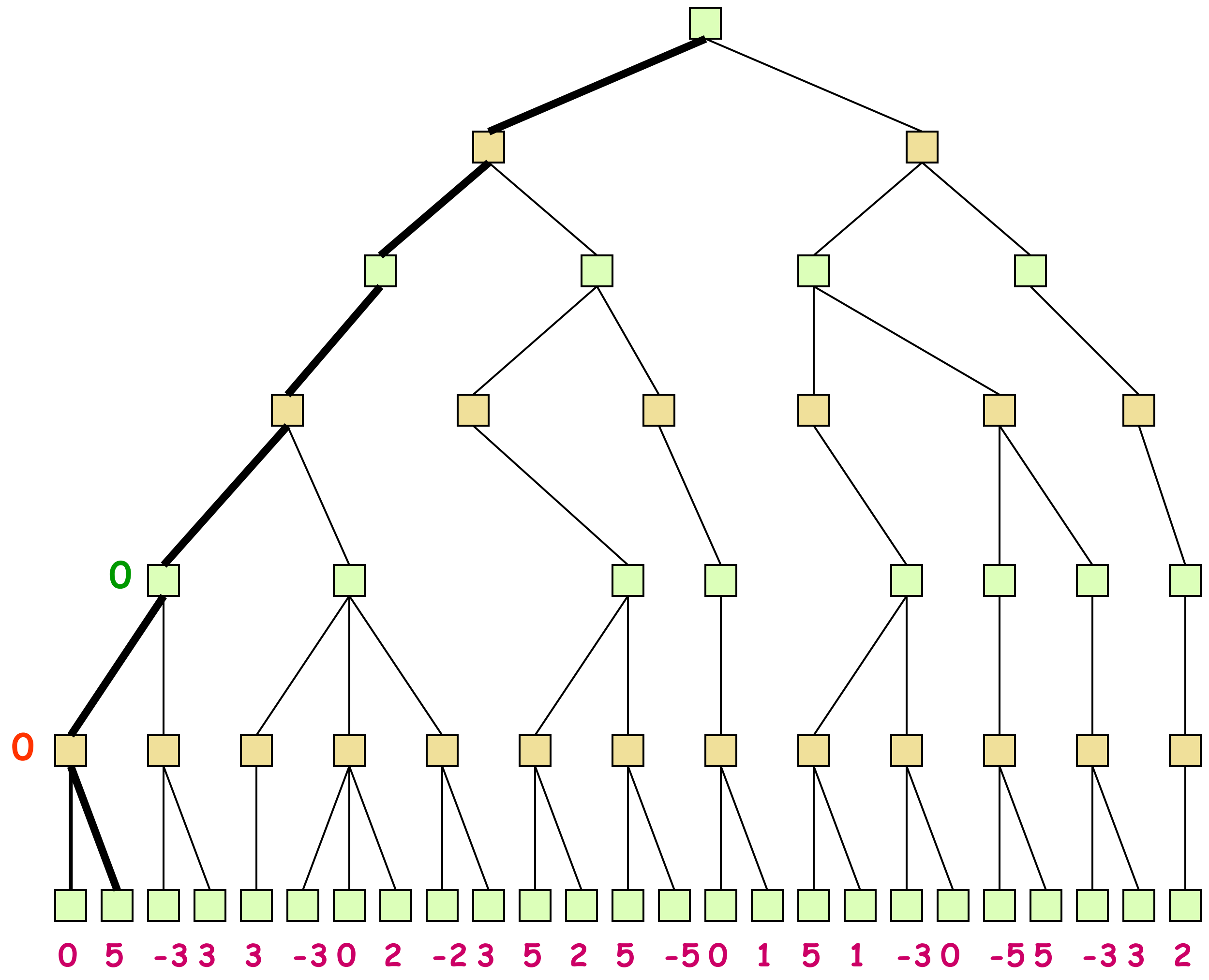
Another alpha-beta example

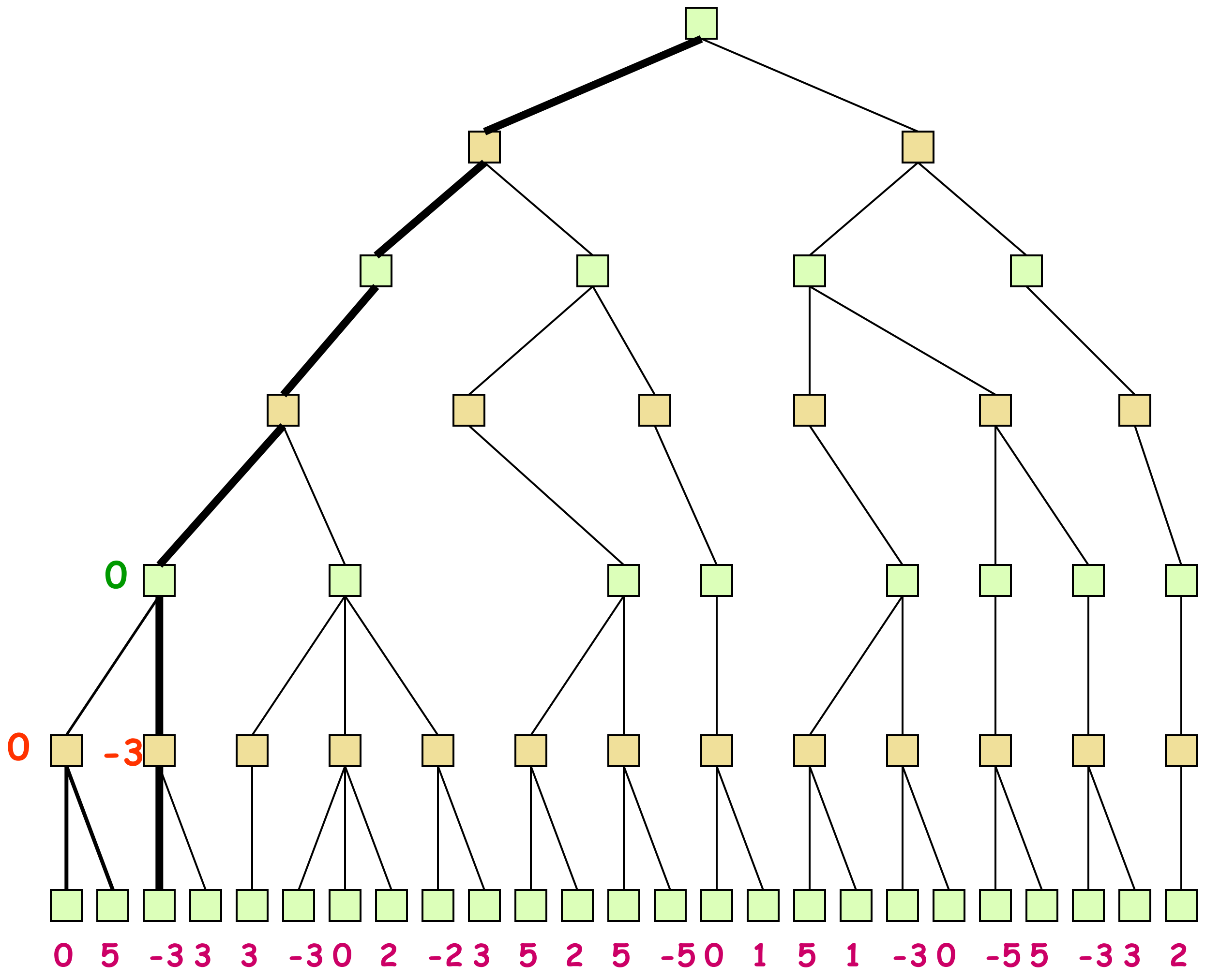


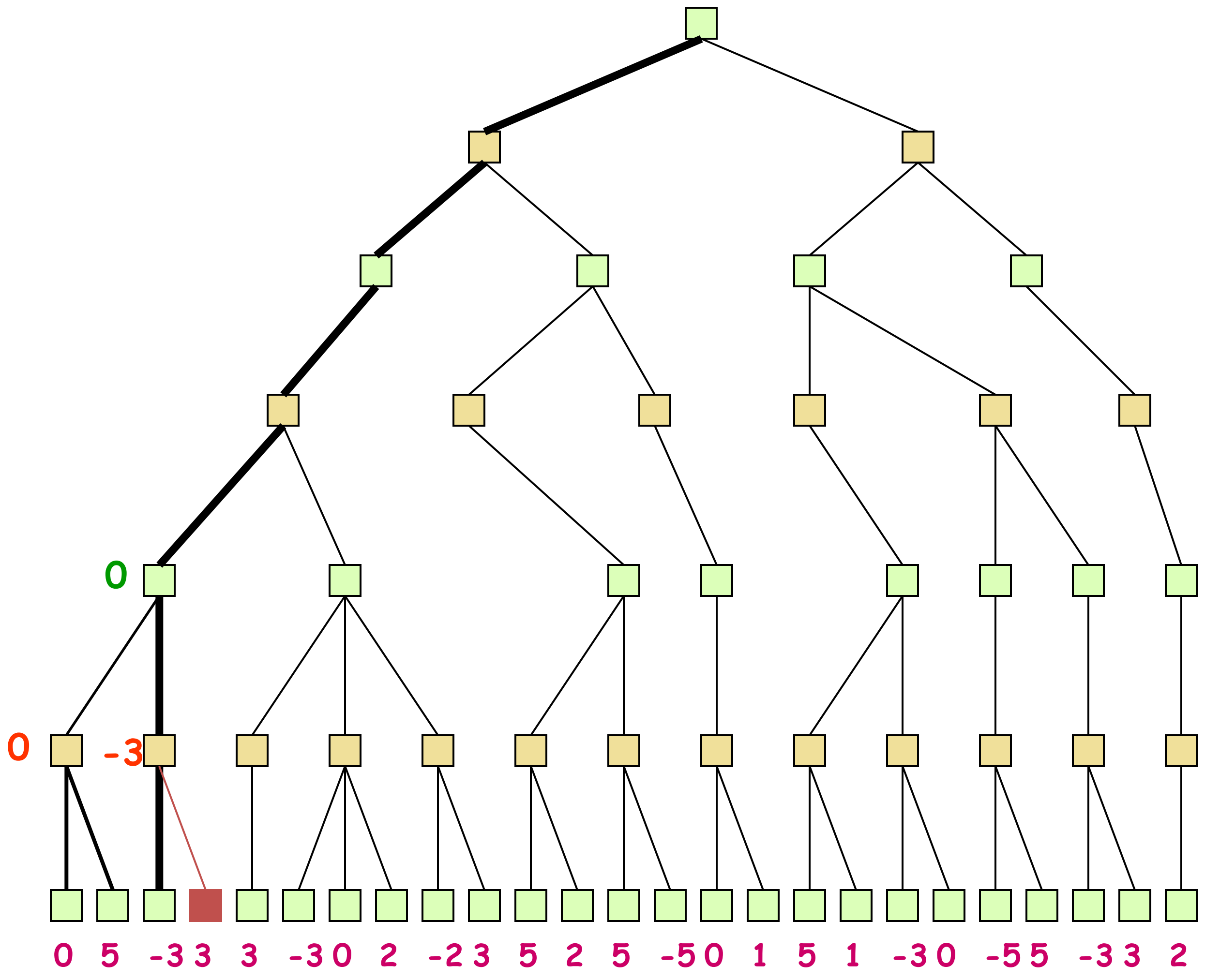
Alpha-Beta Tic-Tac-Toe Example 2

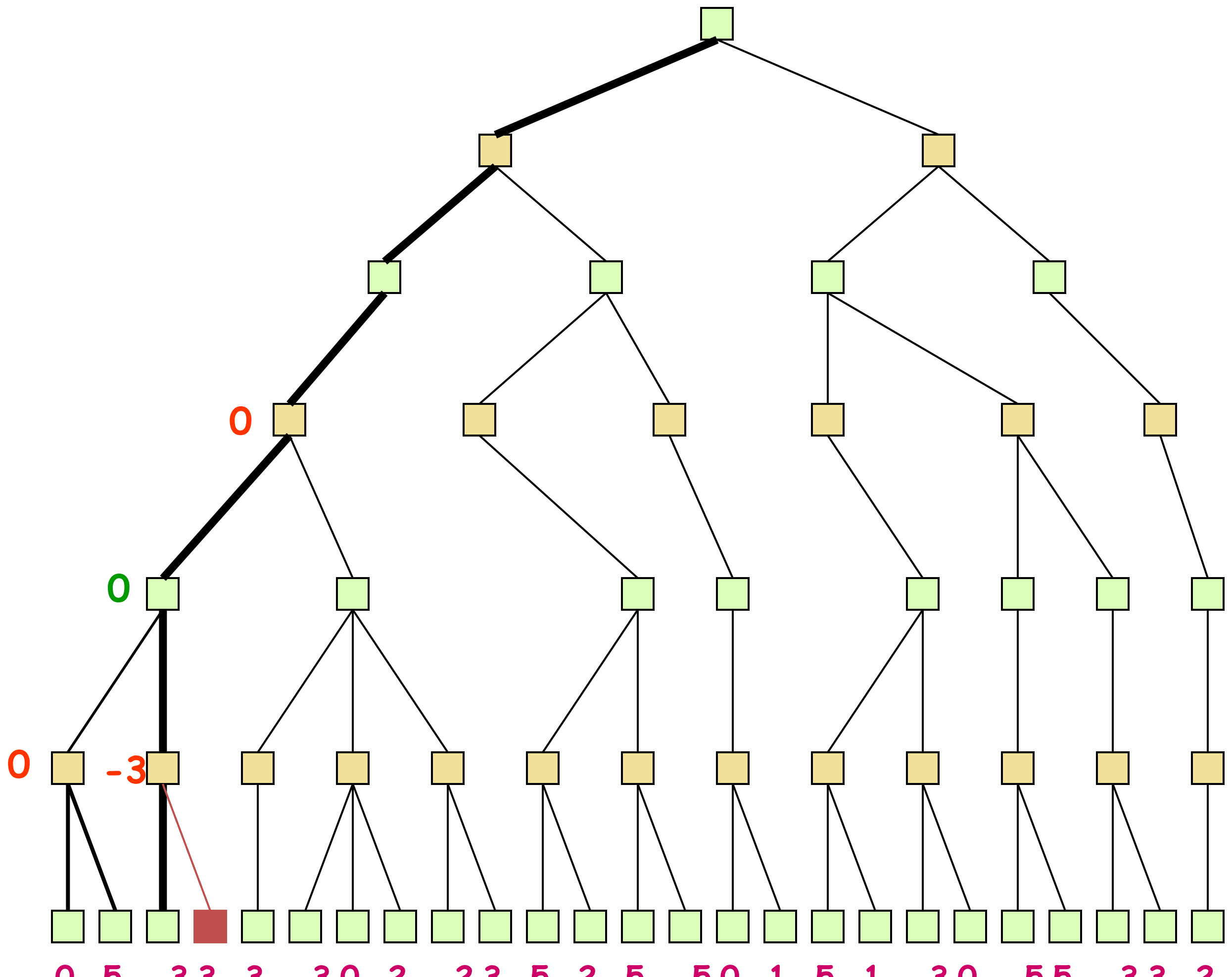


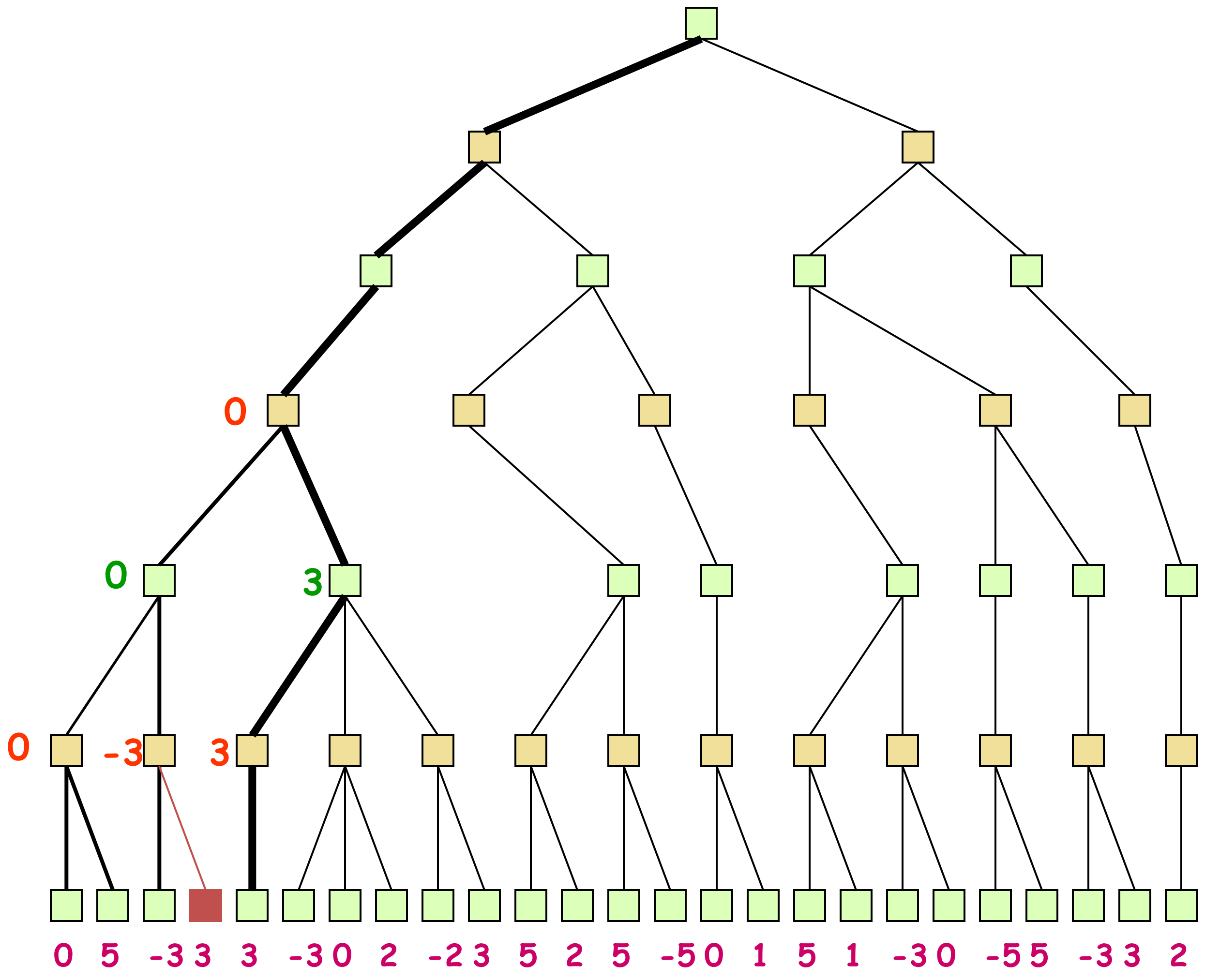

0
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2


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

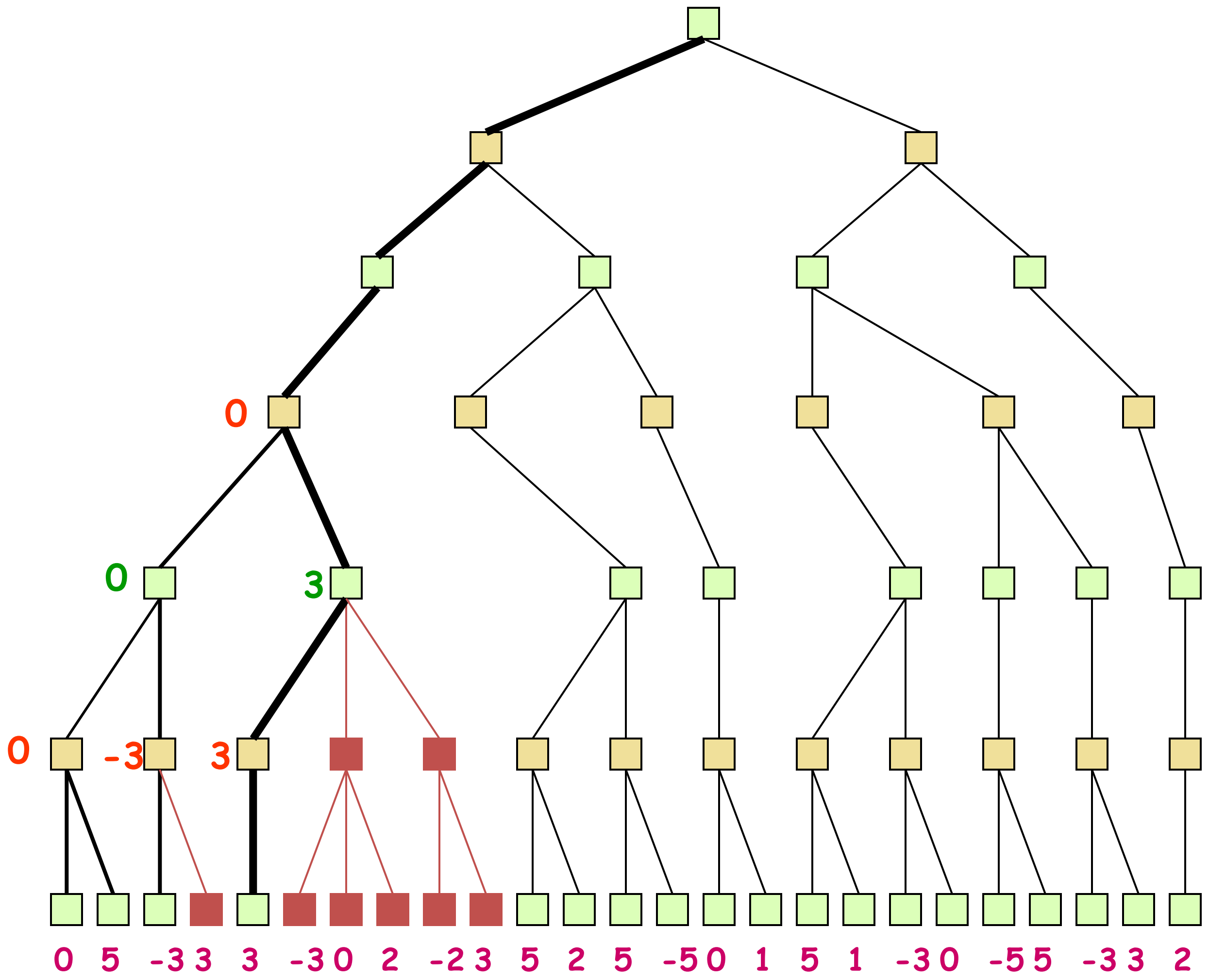

0
0
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

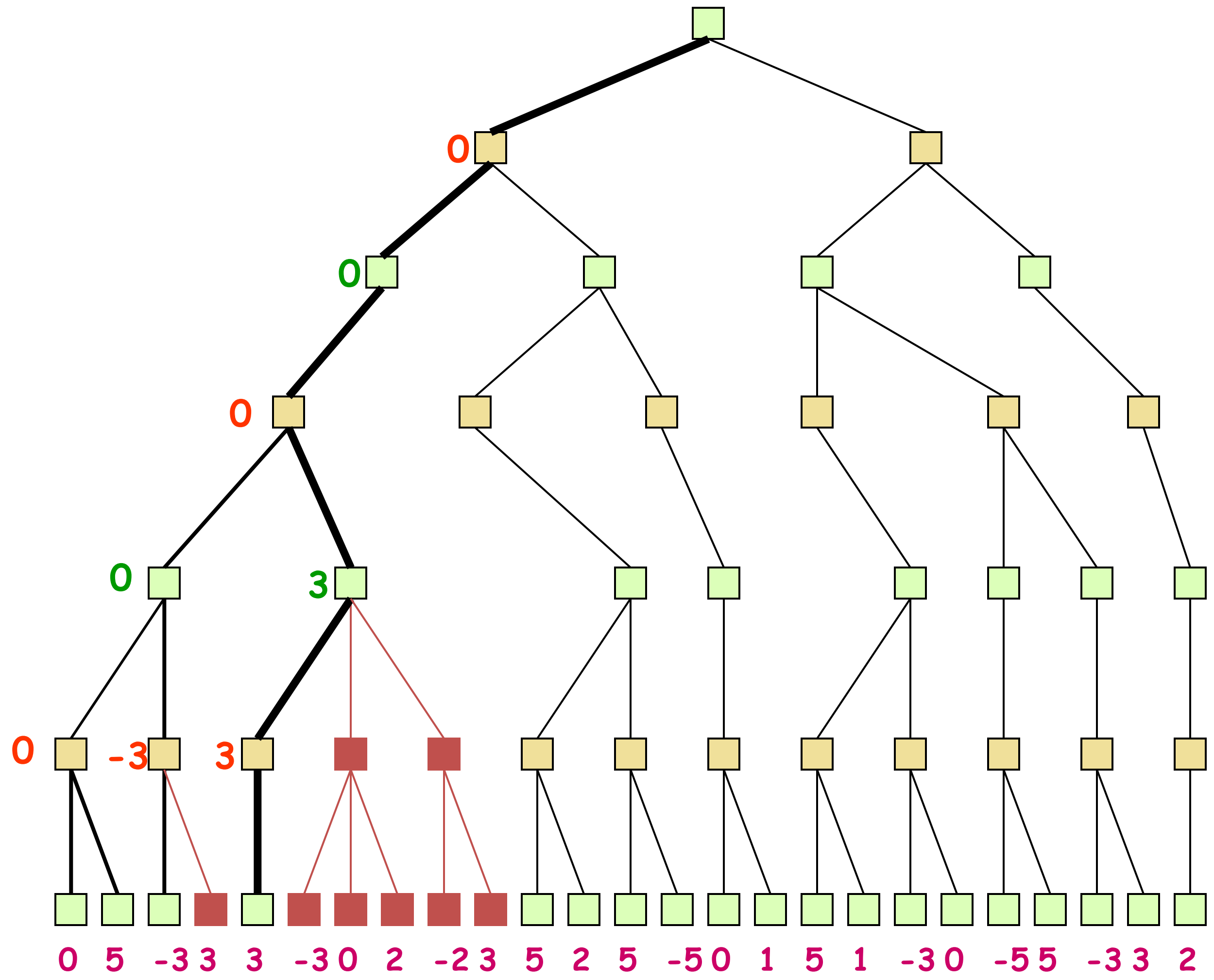

0
0
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

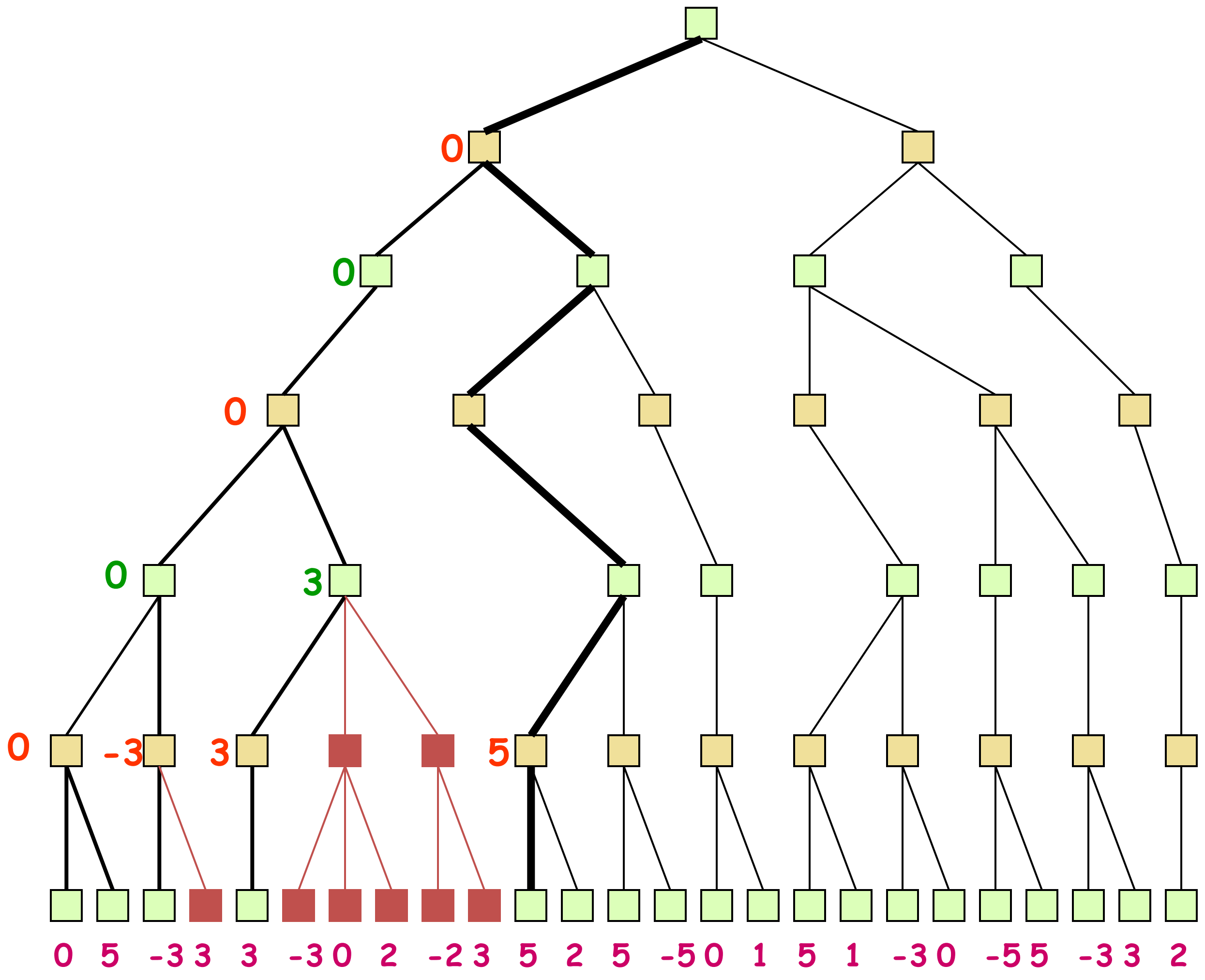

0
0
0
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

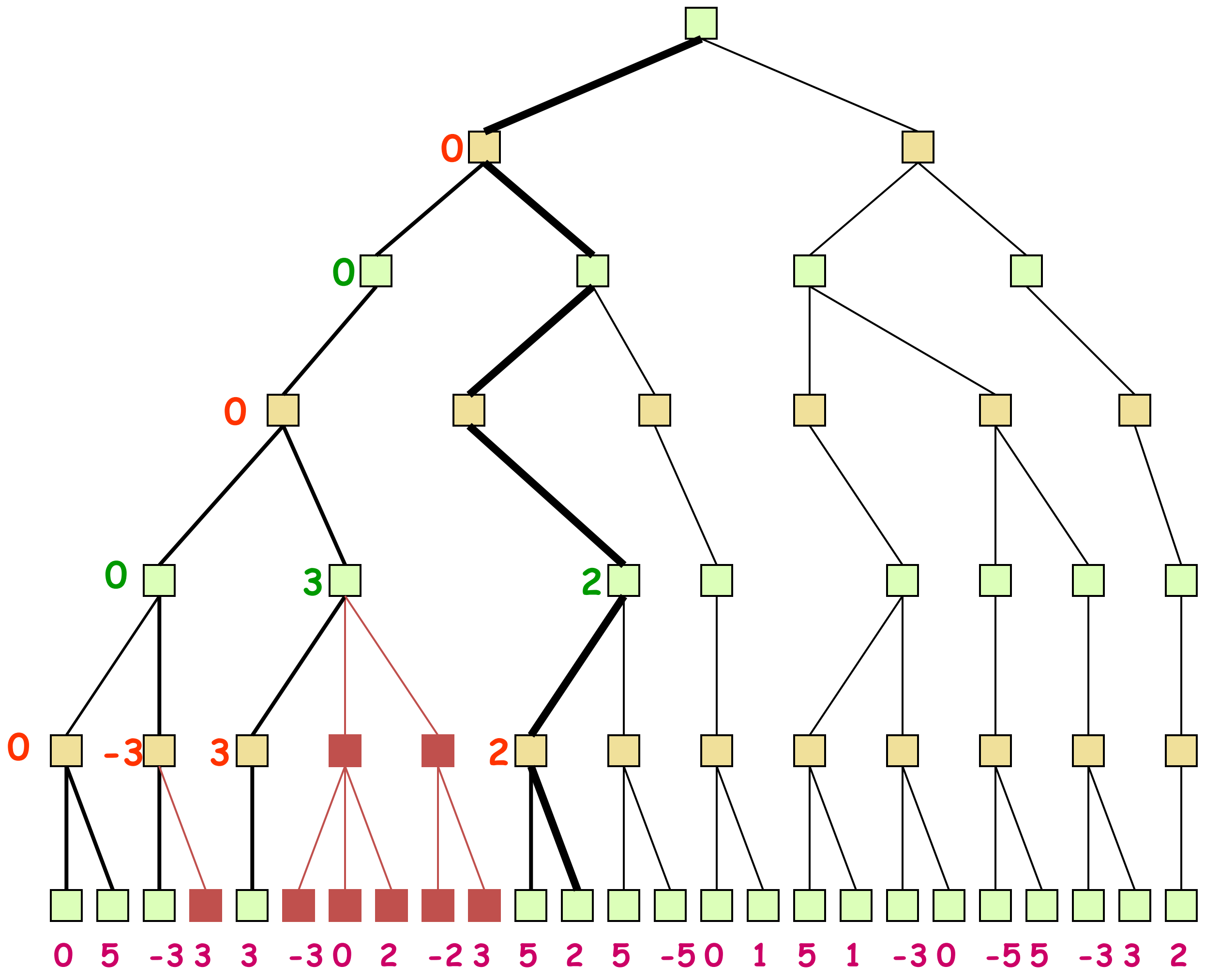


0
0
3
0
-3
3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

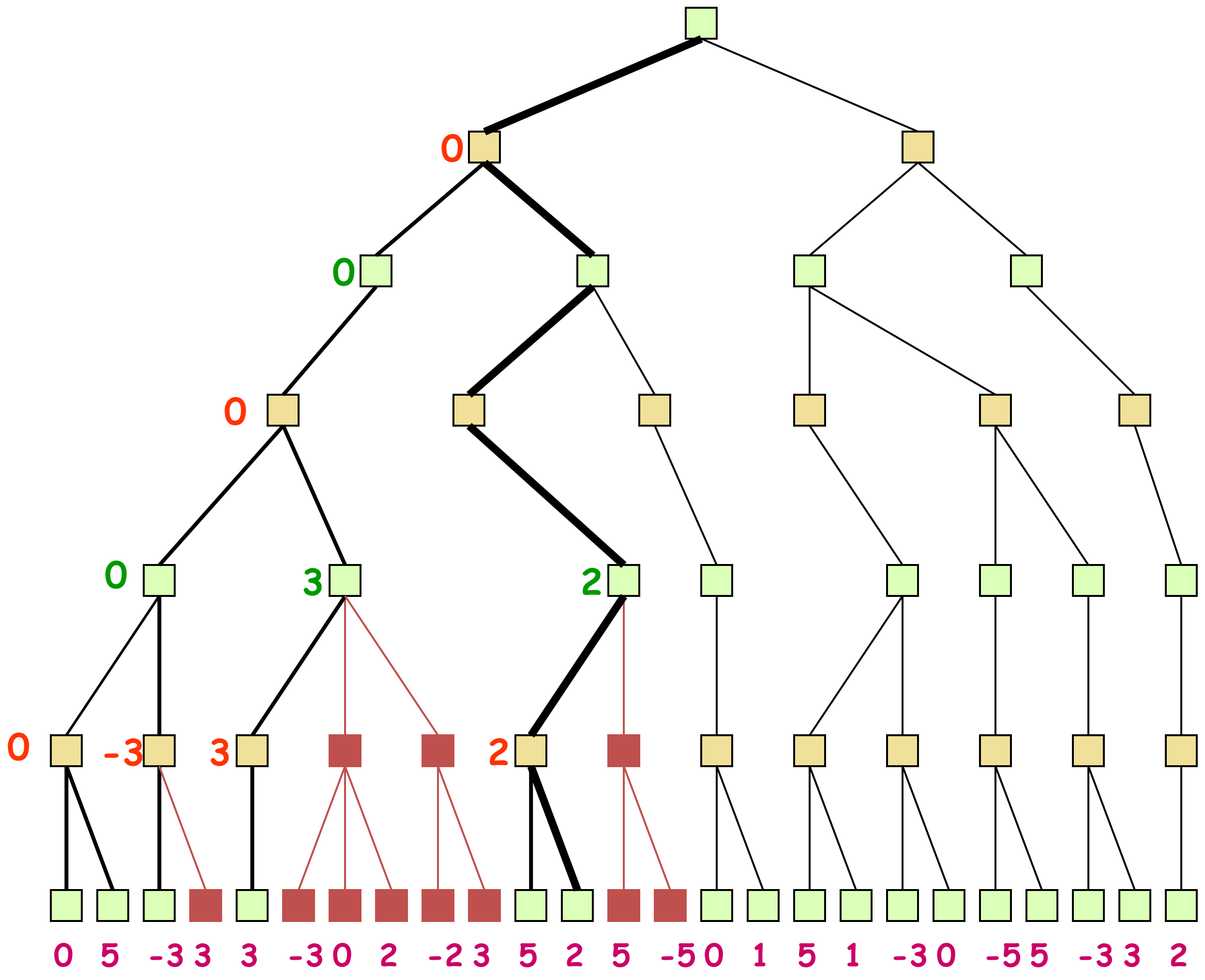

0
0
3
0
-3
3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

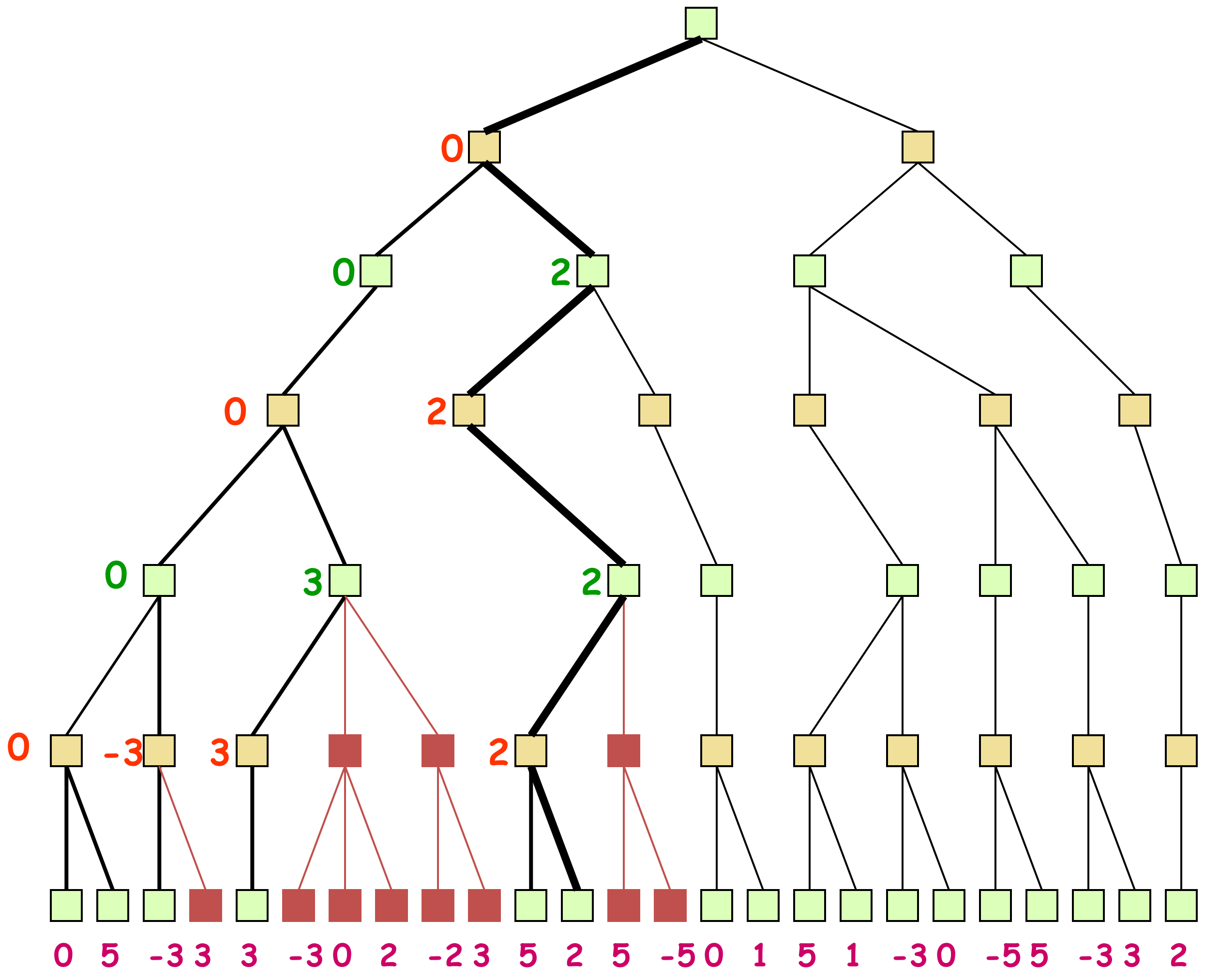

0
0
0
0
3
0
-3
3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

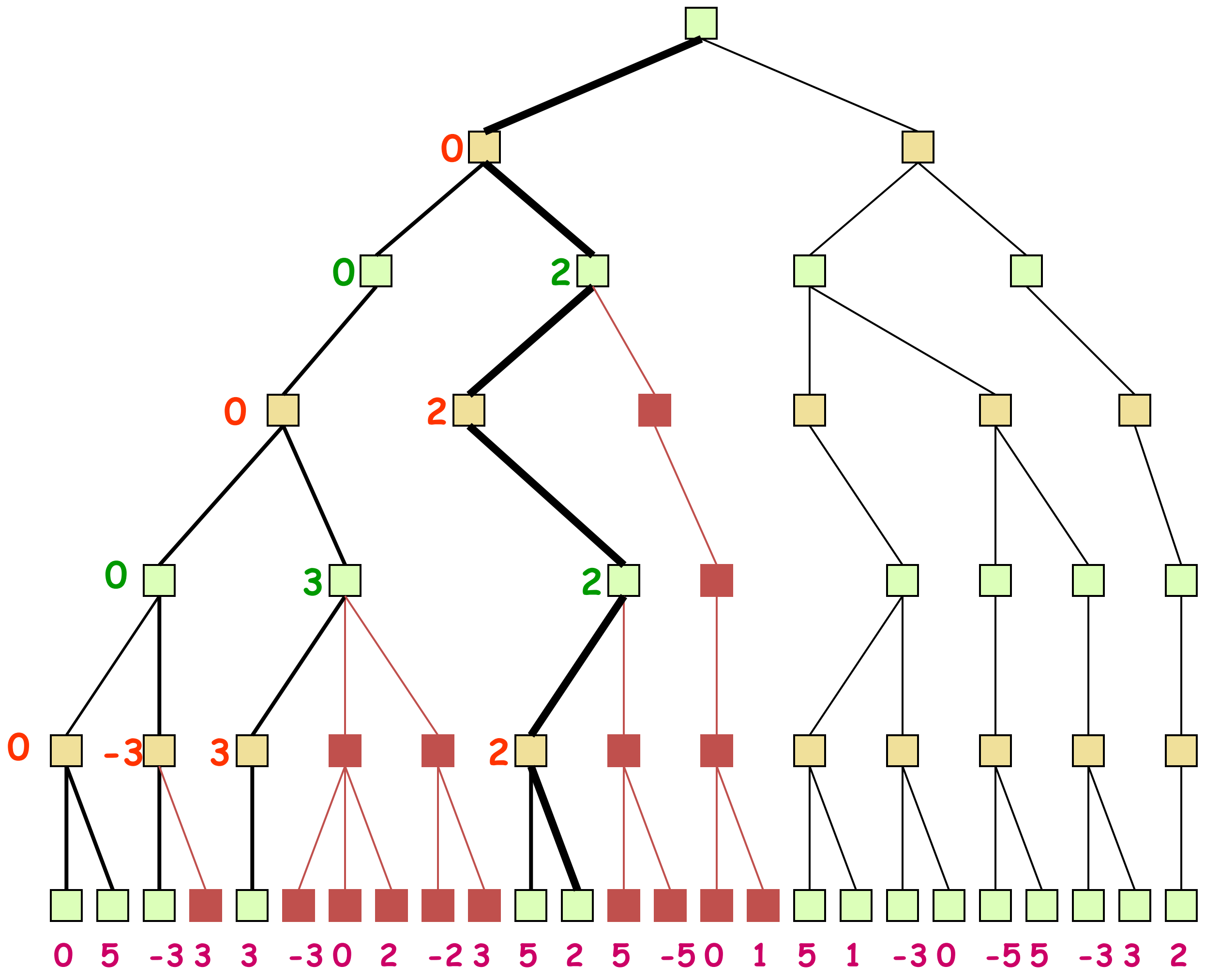

0
0
0
0
3
0
-3
3
5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

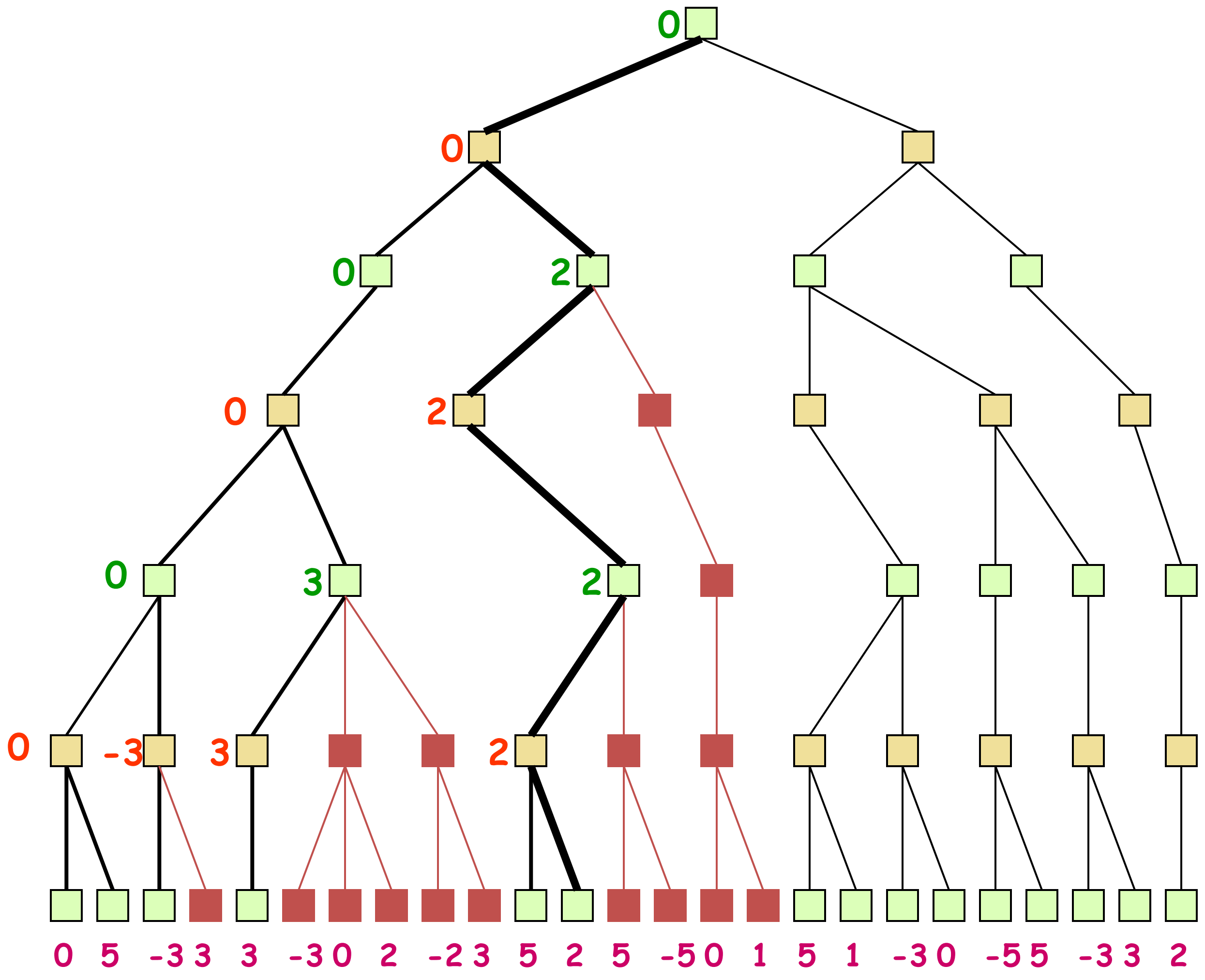


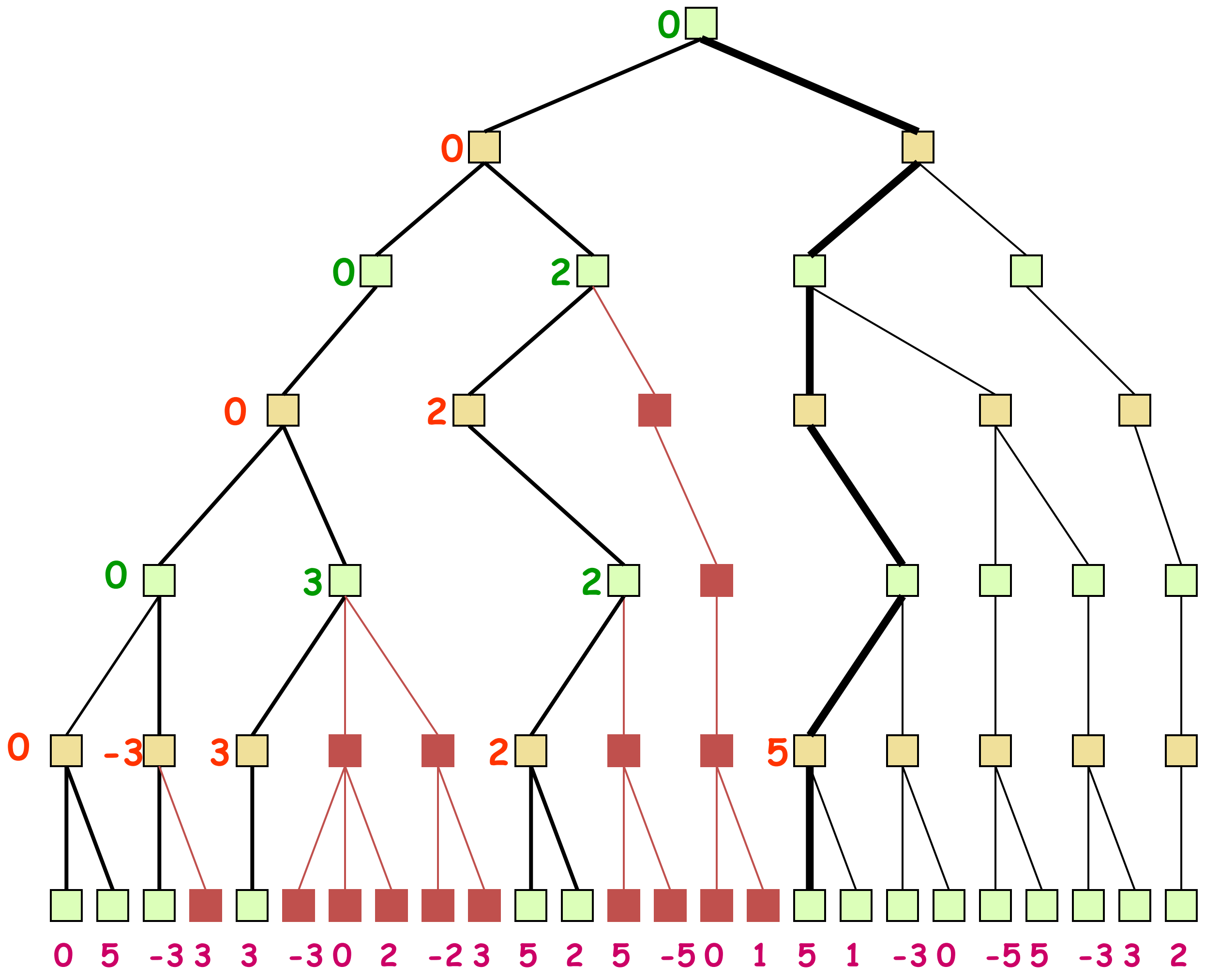
0
0
0
0
3
2
0
-3
3
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2


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

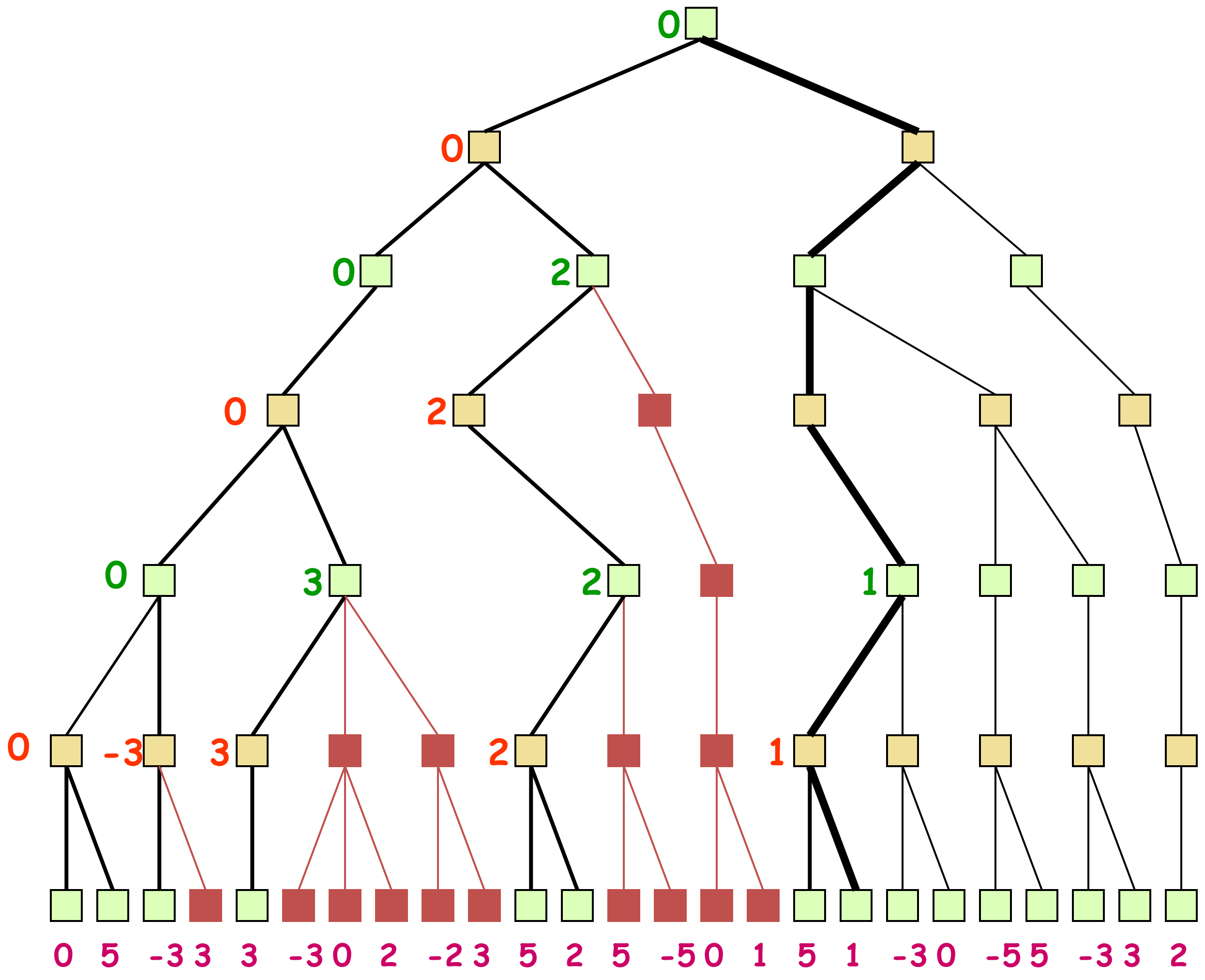

0
0
2
0
2
0
3
2
0
-3
3
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

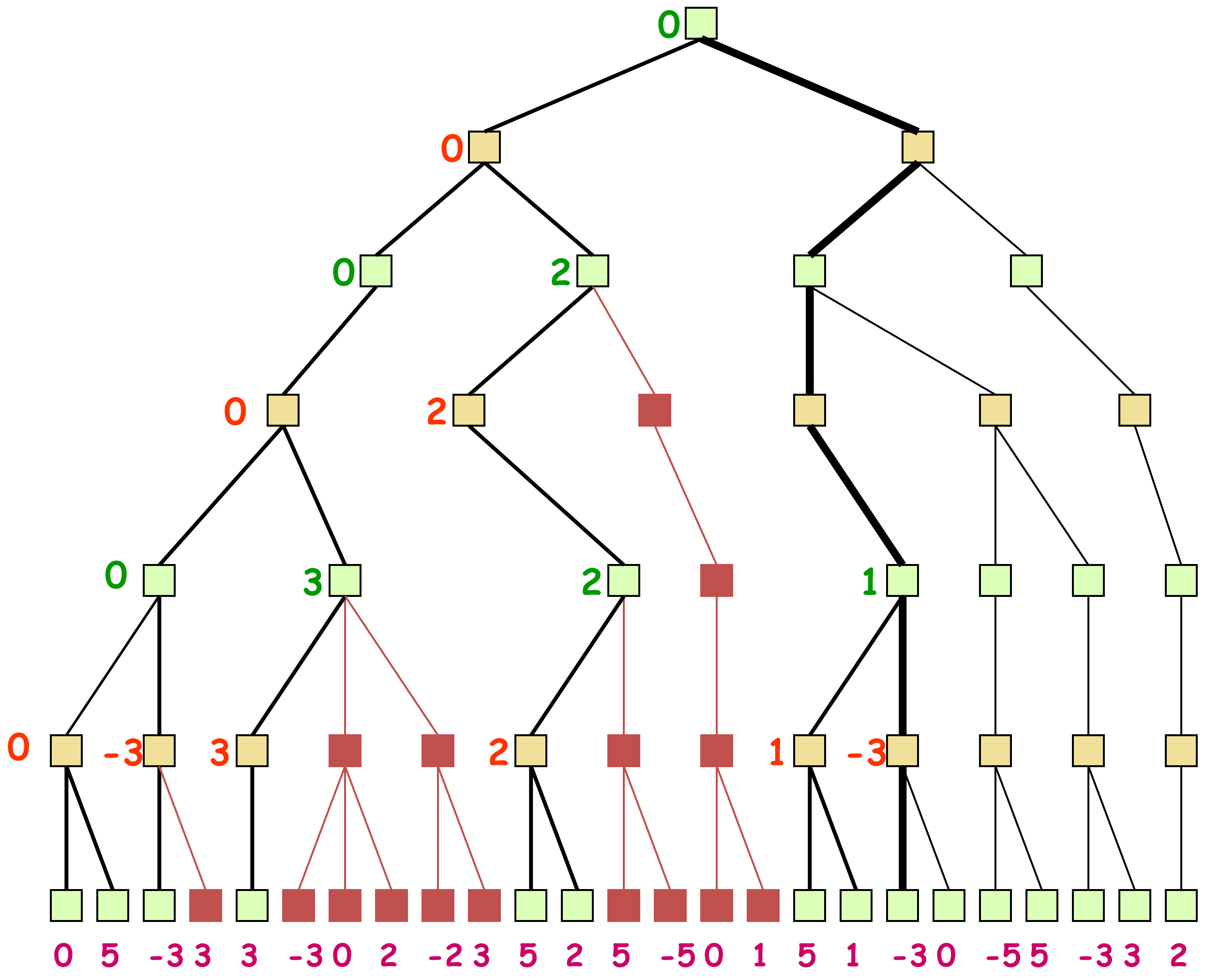

0
0
2
0
2
0
3
2
0
-3
3
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

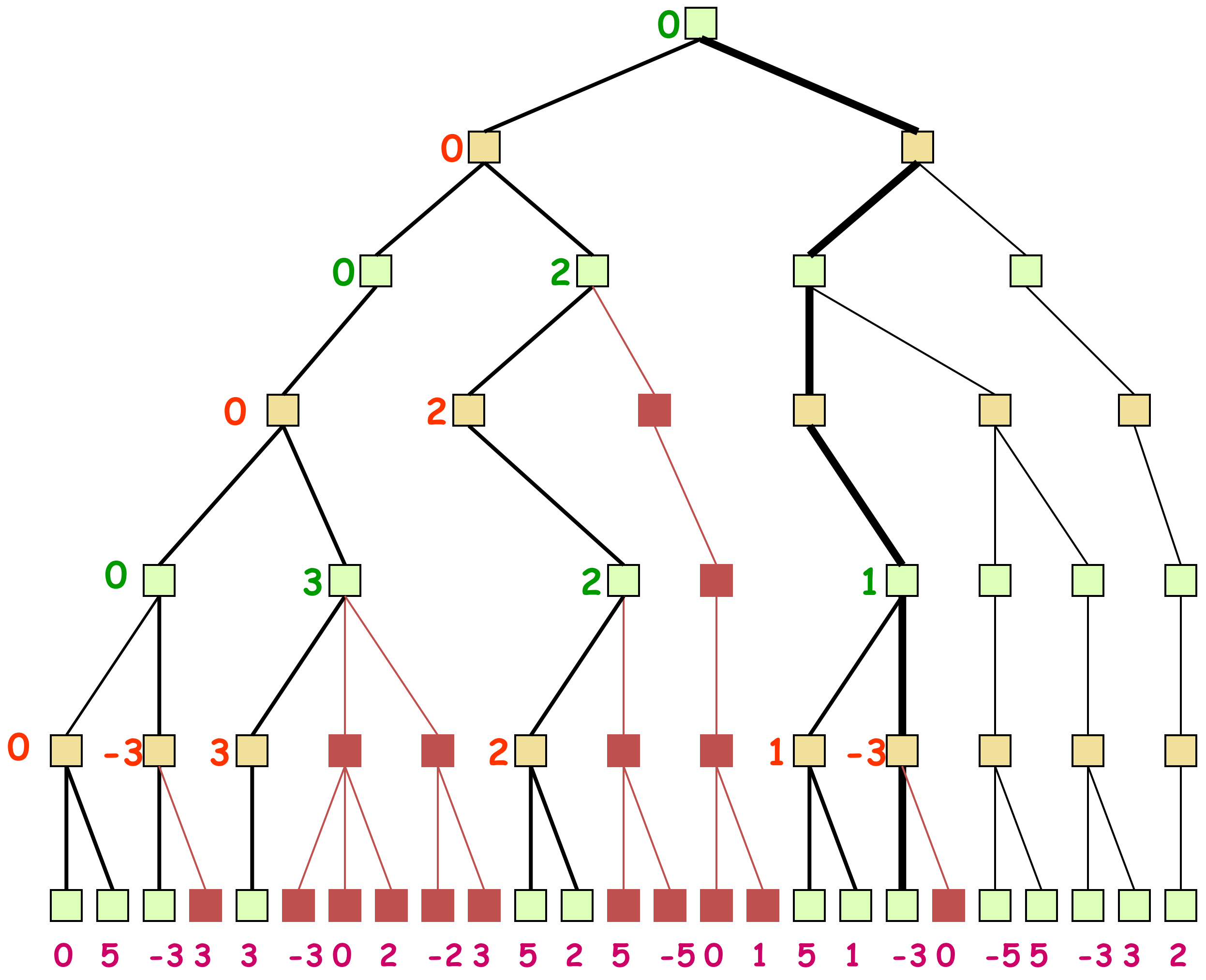

0
0
0
2
0
2
0
3
2
0
-3
3
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

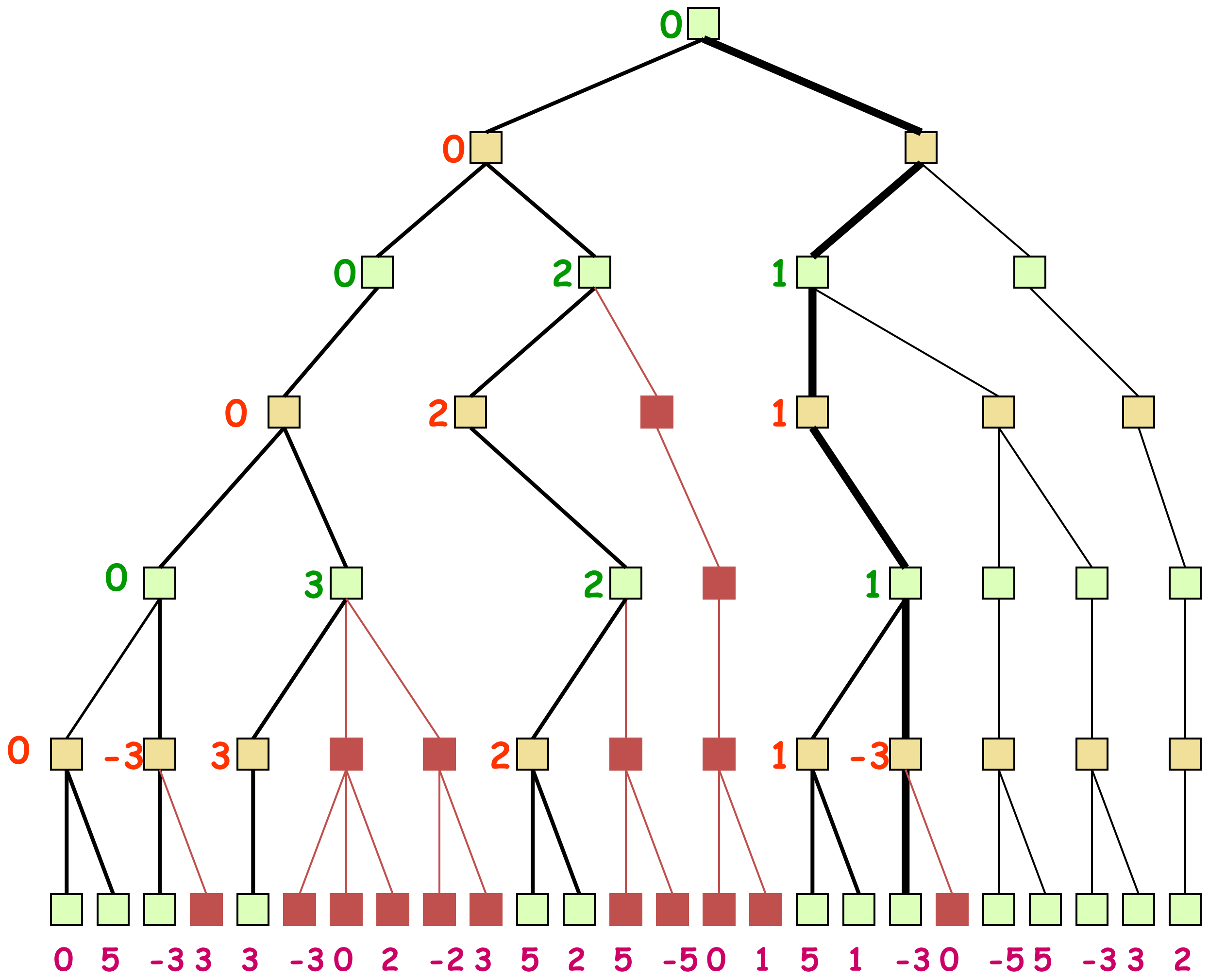


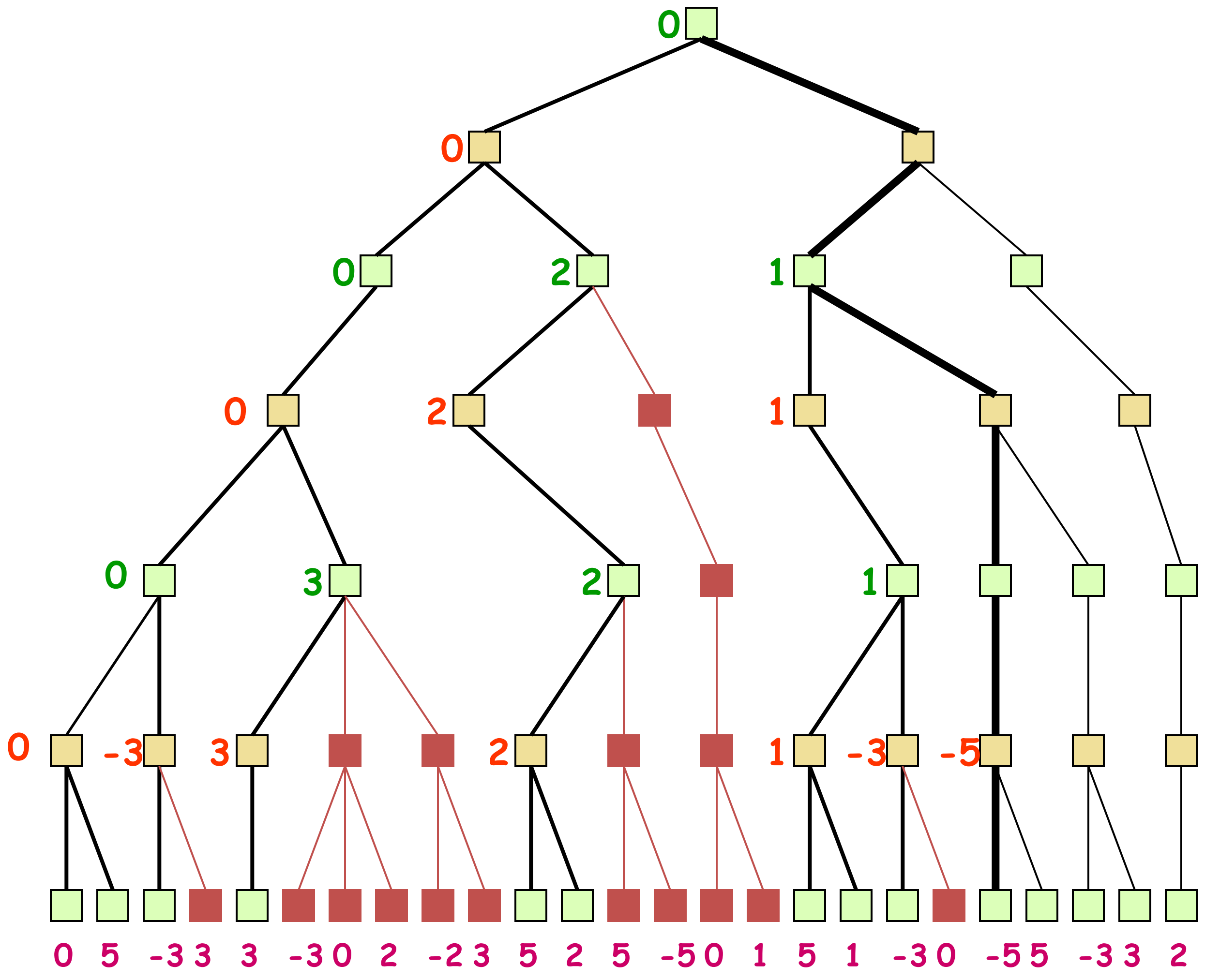
0
0
0
2
0
2
0
3
2
0
-3
3
2
5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

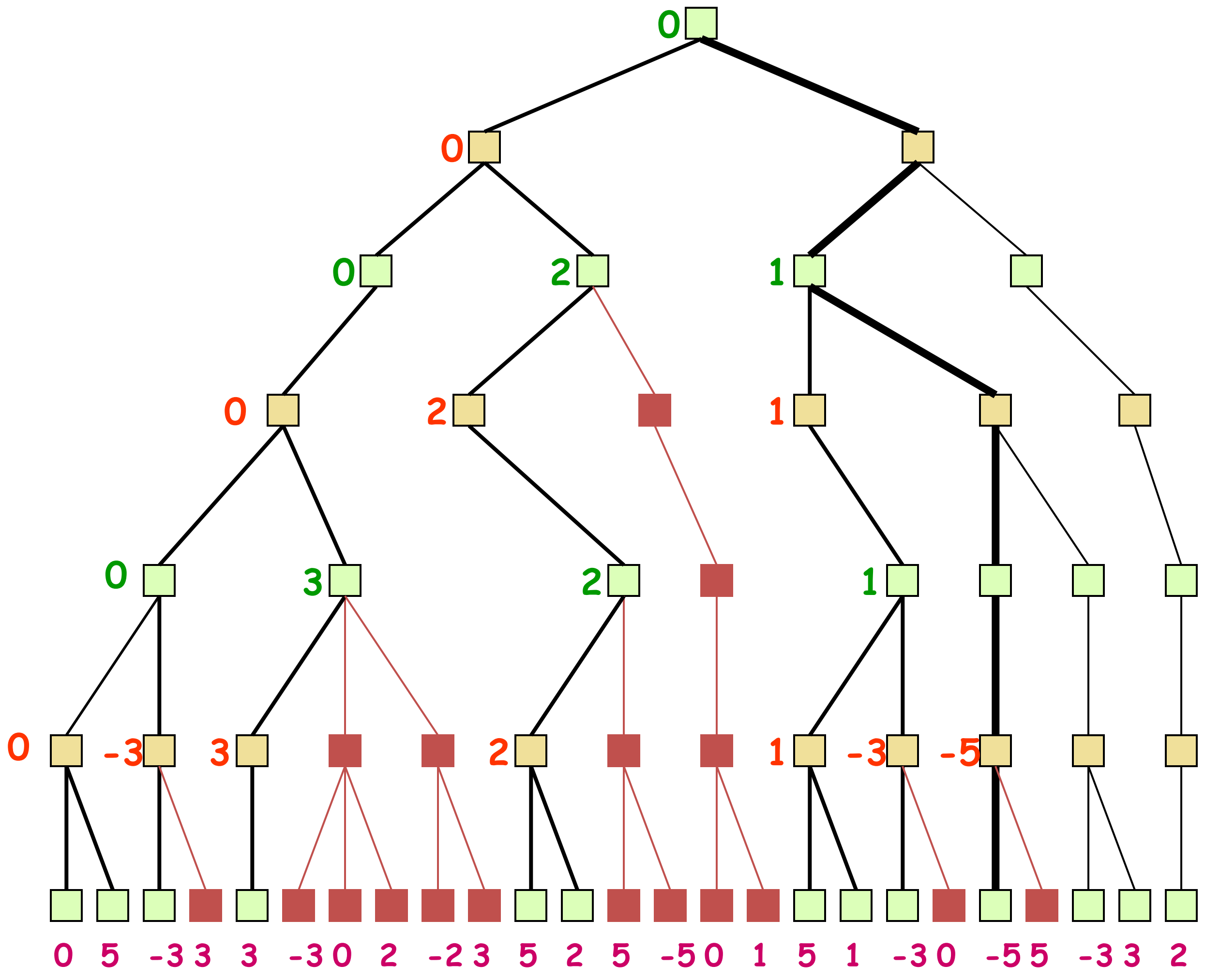

0
0
0
2
0
2
0
3
2
1
0
-3
3
2
1
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

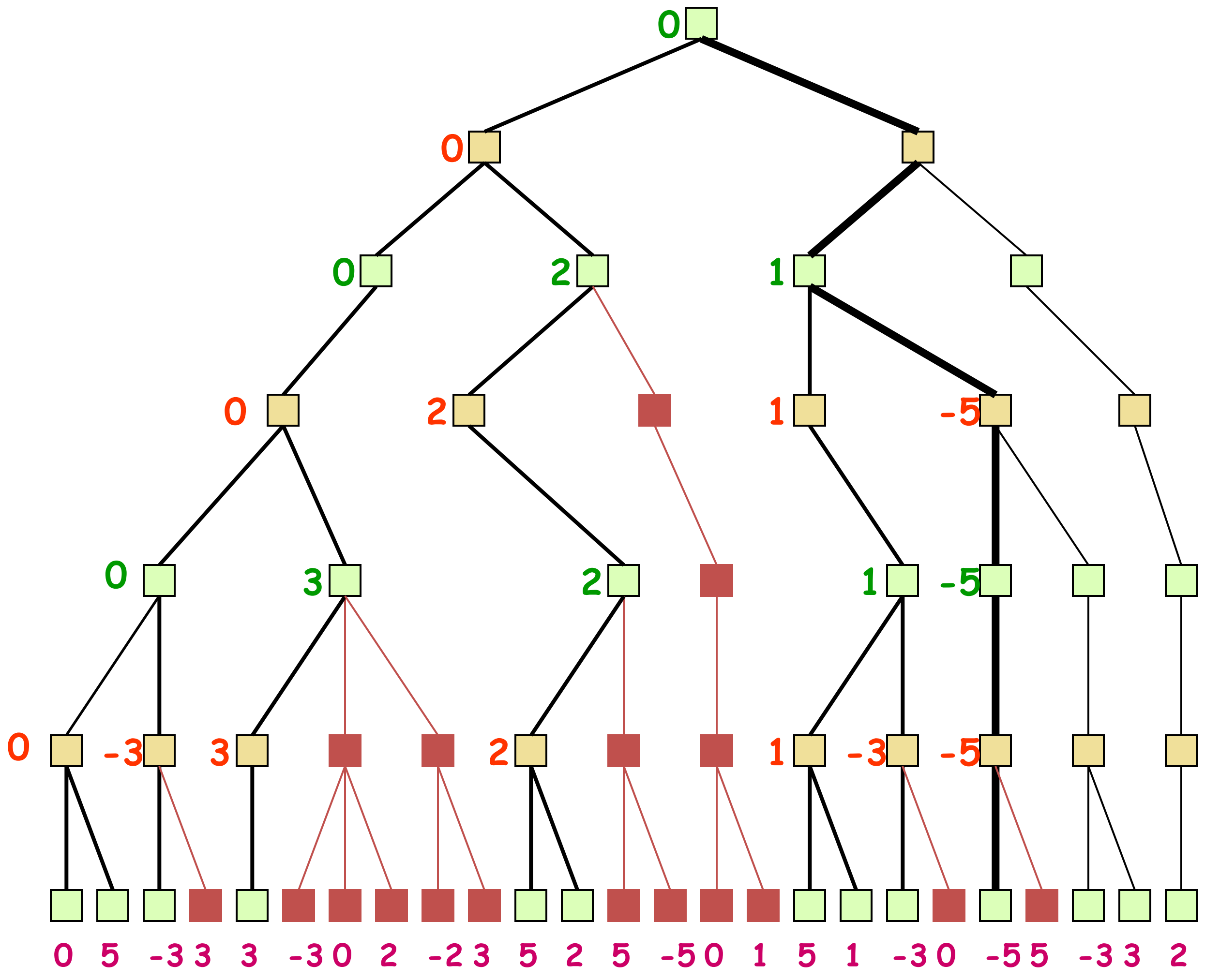

0
0
0
2
0
2
0
3
2
1
0
-3
3
2
1
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2


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

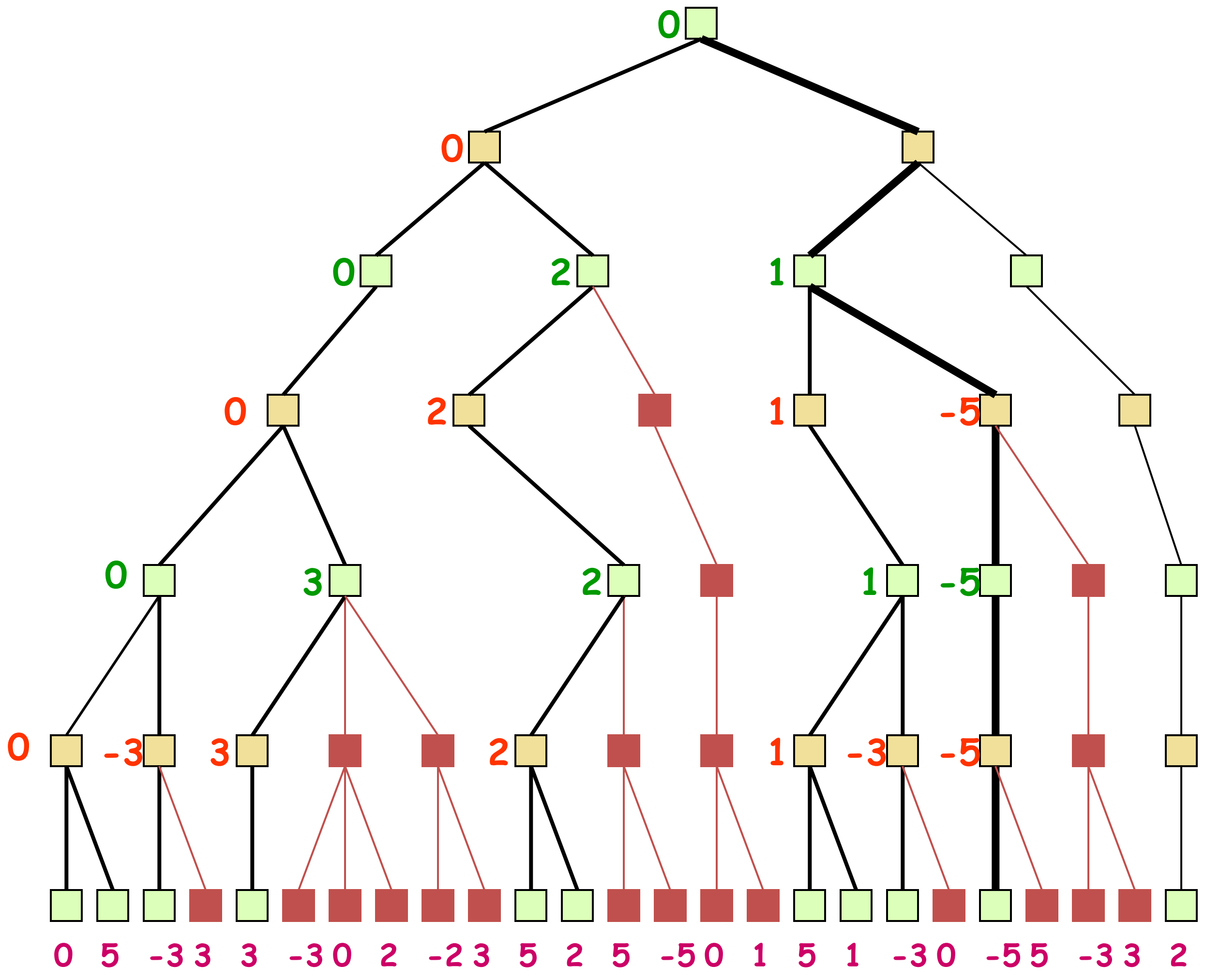

0
0
0
2
1
0
2
1
0
3
2
1
0
-3
3
2
1
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

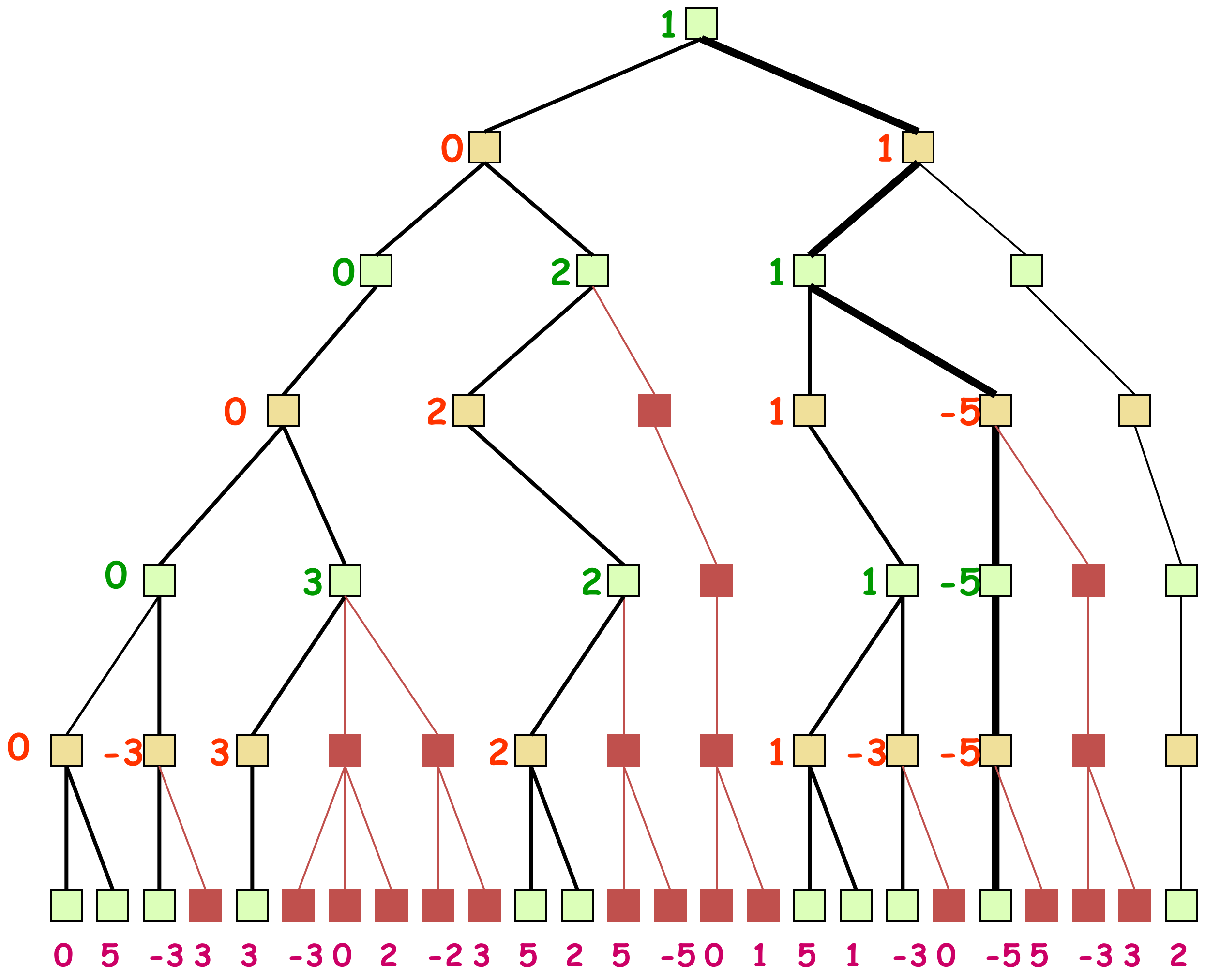

0
0
0
2
1
0
2
1
0
3
2
1
0
-3
3
2
1
-3
-5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

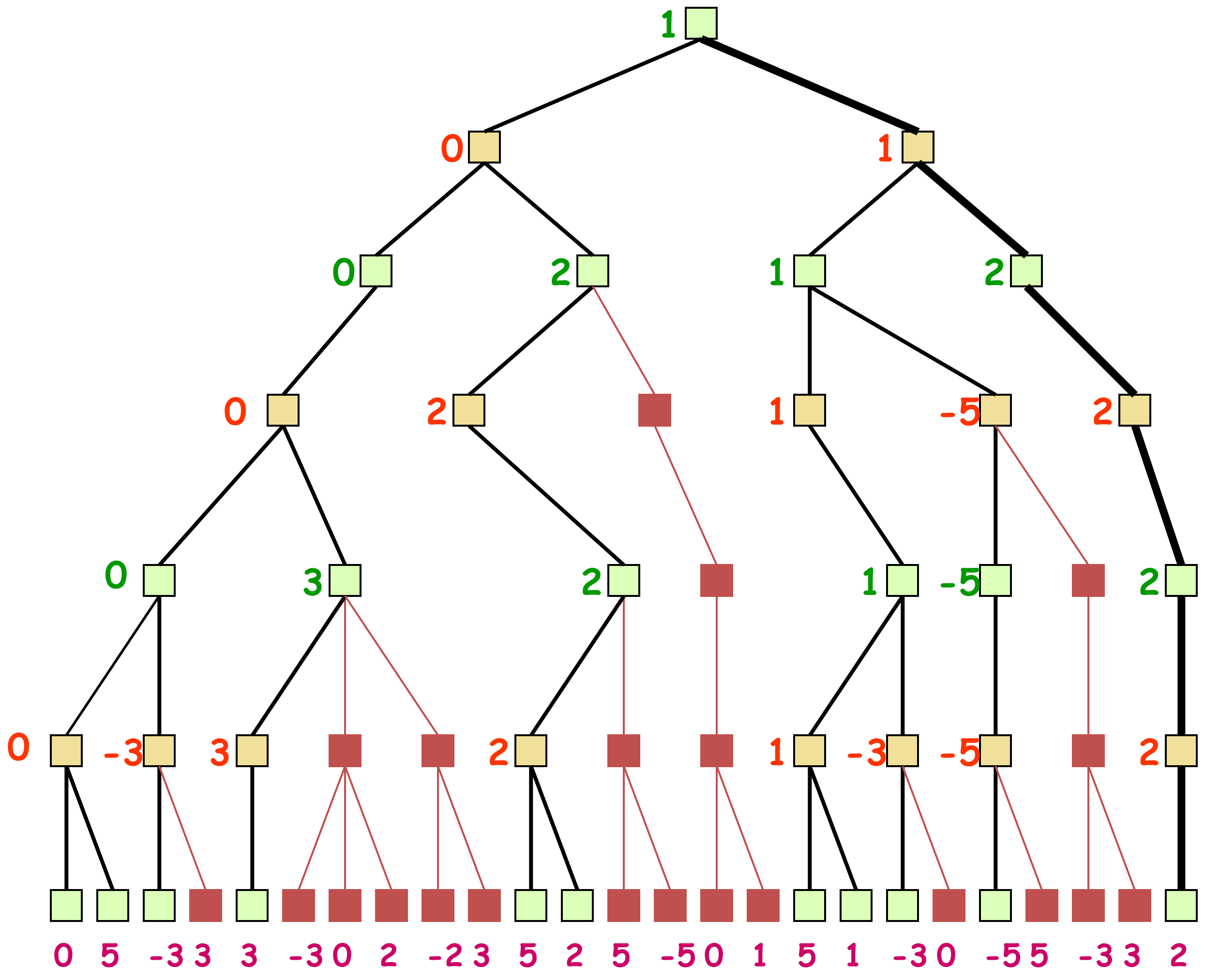

0
0
0
2
1
0
2
1
0
3
2
1
0
-3
3
2
1
-3
-5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2



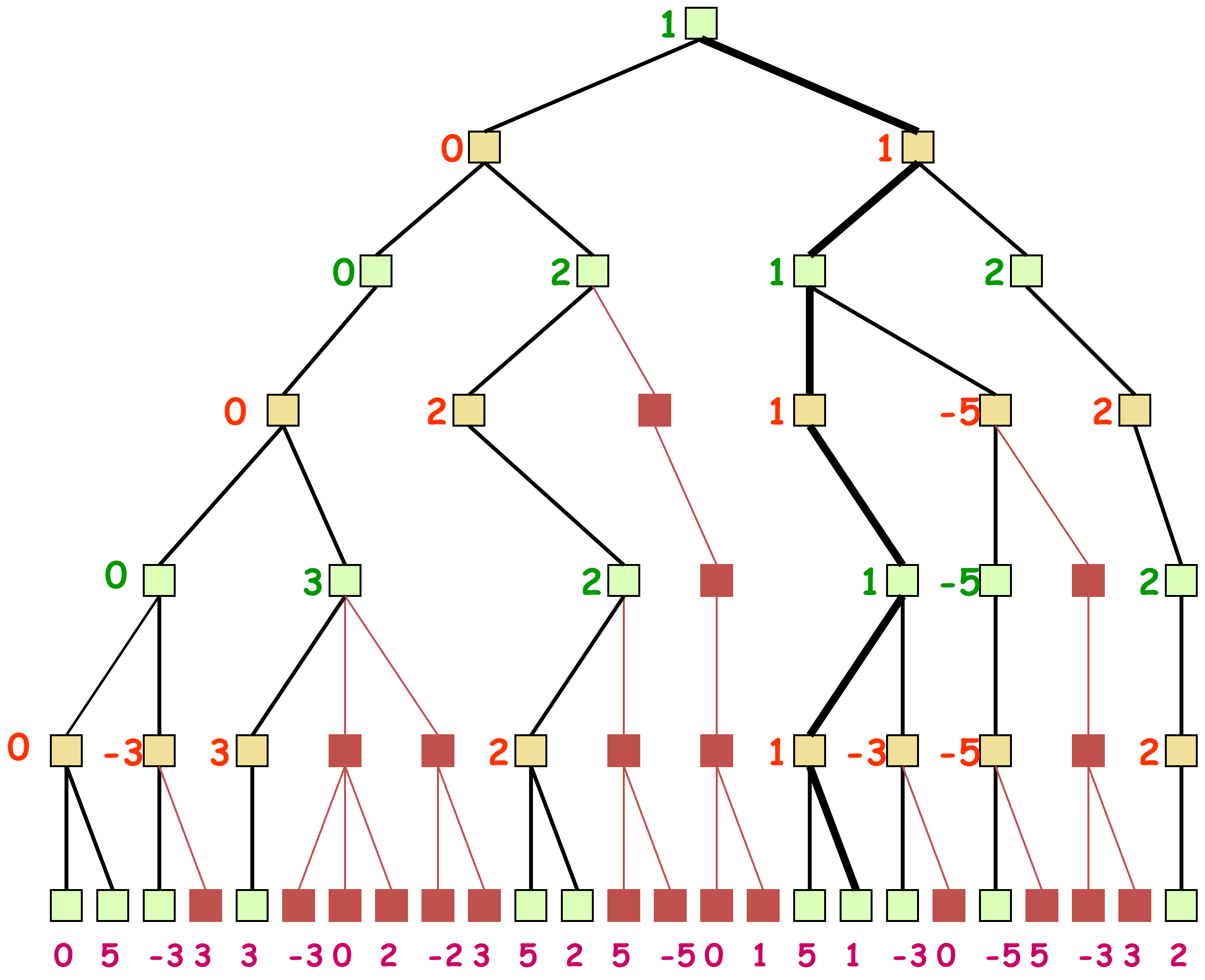
0
0
0
2
1
0
2
1
-5
0
3
2
1
-5
0
-3
3
2
1
-3
-5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2


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


1
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


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

With alpha-beta we avoided computing a static evaluation metric for 14 of the 25 leaf nodes



Effectiveness of alpha-beta

- Alpha-beta guaranteed to compute same value for root node as minimax, but with ≤ computation
- **Worst case:** no pruning, examine b^d leaf nodes, where nodes have b children & d-ply search is done
- **Best case:** examine only $(2b)^{d/2}$ leaf nodes
  - You can search twice as deep as minimax!
  - **Occurs if each player's best move is 1st alternative**
- In Deep Blue, alpha-beta pruning reduced effective branching factor from ~35 to ~6

Many other improvements

- **Adaptive horizon + iterative deepening**
- **Extended search**: retain k>1 best paths (not just one) extend tree at greater depth below their leaf nodes to help dealing with "horizon effect"
- **Singular extension**: If move is obviously better than others in node at horizon h, expand it
- Use **transposition tables** to deal with repeated states