Adversarial Search
Aka Games

Chapter 6

Some material adopted from notes by Charles R. Dyer, University of Wisconsin-Madison

Overview

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

Why study games?

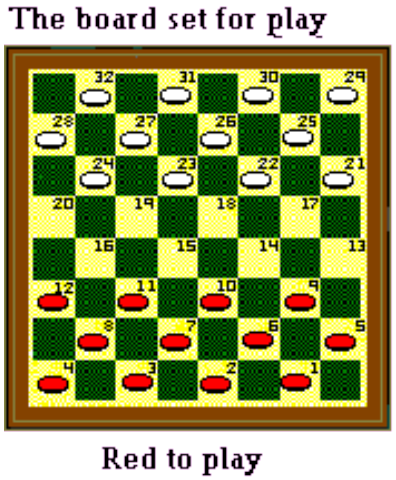
- Interesting, hard problems that require minimal "initial structure"
- Clear criteria for success
- A way to study problems involving {hostile, adversarial, competing} agents and the uncertainty of interacting with the natural world
- People have used them to asses their intelligence
- Fun, good, easy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

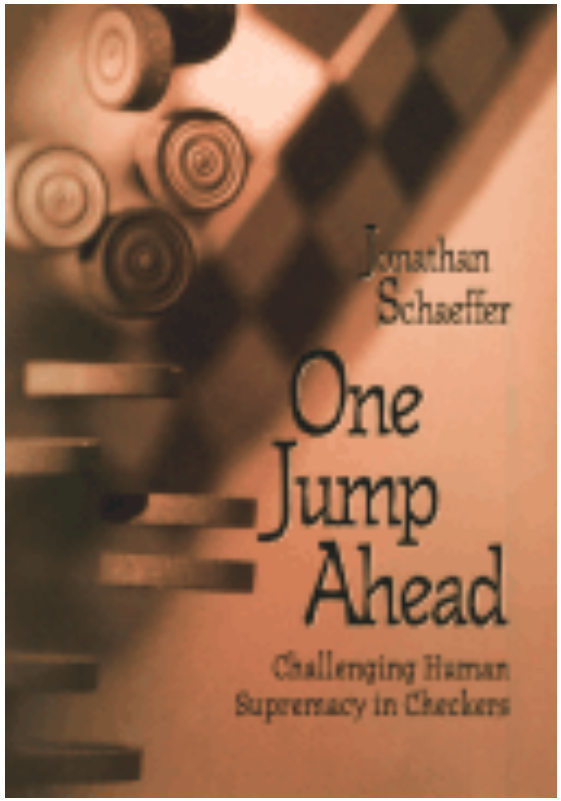
State of the art

- **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
- **Checkers:** has been solved exactly – it's a draw!
- **Go**: Computer players are decent, at best
- **Bridge**: "Expert" computer players exist, but no world champions yet
- **Poker:** Poki regularly beats human experts
- Check out the U. Alberta Games Group

Chinook

- Chinook is the World Man-Machine Checkers Champion, developed by researchers at the University of Alberta
- It earned this title by competing in human tournaments, winning the right to play for the (human) world championship, and eventually defeating the best players in the world
- Visit http://www.cs.ualberta.ca/~chinook/ to play a version of Chinook over the Internet.
- "One Jump Ahead: Challenging Human Supremacy in Checkers", Jonathan Schaeffer, 1998
- See Checkers Is Solved, J. Schaeffer, et al., Science, v317, n5844, pp1518-22, AAAS, 2007.

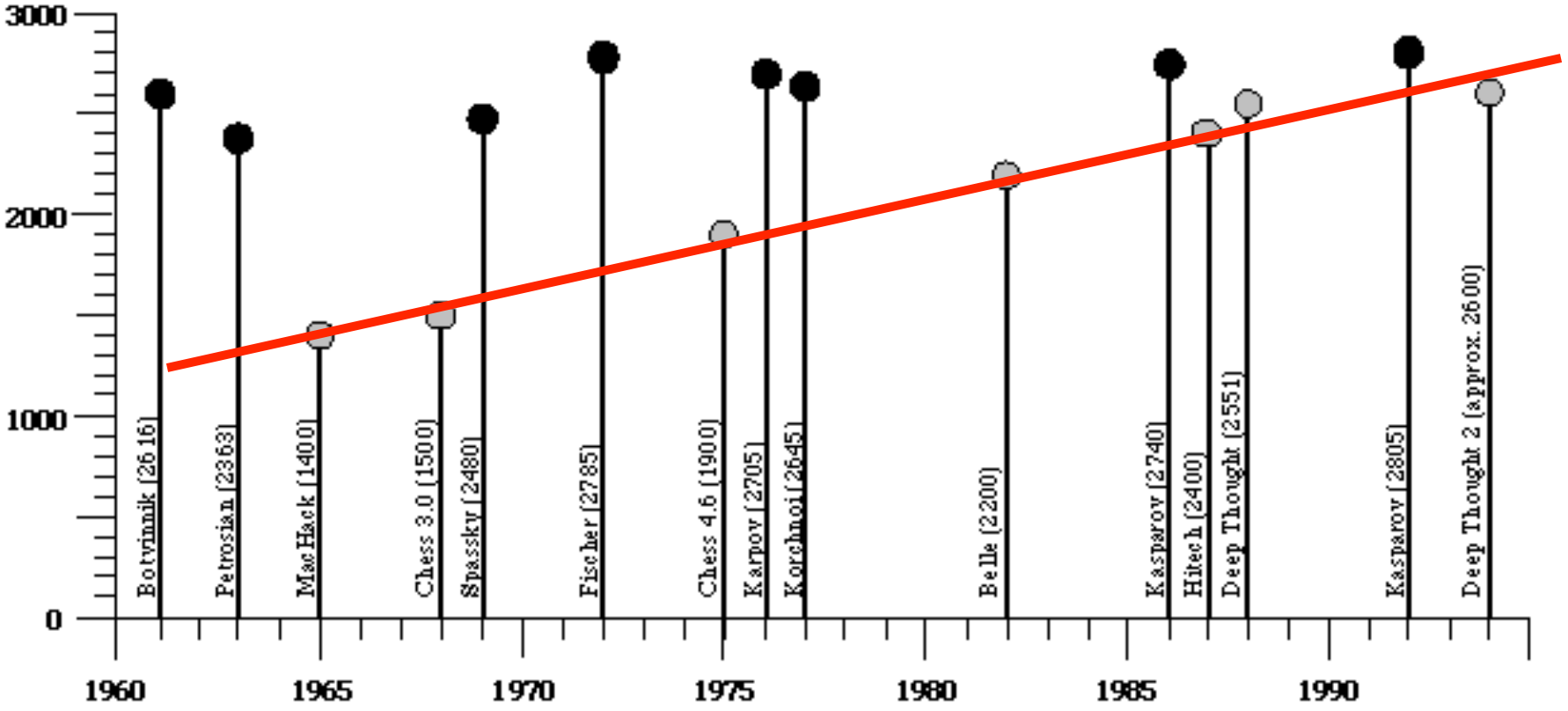




Chess early days

- **1948**: Norbert Wiener's book *Cybernetics* describes how a chess program could be developed using a depth-limited minimax search with an evaluation function
- **1950**: Claude Shannon publishes one of first papers on playing chess "Programming a Computer for Playing Chess"
- **1951**: Alan Turing develops on paper the first program capable of playing a full game of chess
- **1962**: Kotok and McCarthy (MIT) develop first program to play credibly
- **1967**: Mac Hack Six, by Richard Greenblatt et al. (MIT) idefeats a person in regular tournament play

Ratings of human & computer chess champions



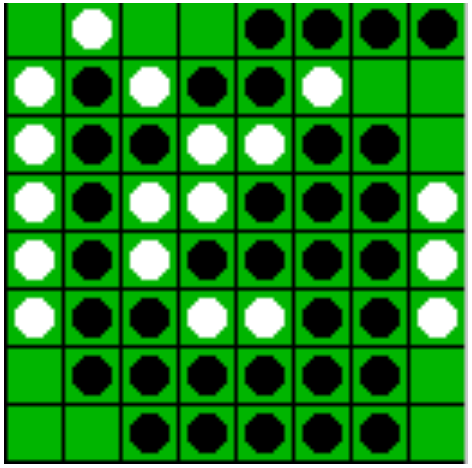
1997


Kasparov vs. Deep Blue: The Rematch - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://www.chess.ibm.com/home/html/b.html
may 11th
game 6 : may 11 @ 3:00PM EDT | 19:00 GMT
kasparov 2.5 deep blue 2.5
Home
The match
The players
The technology
Community
Deep Blue Wins
3.5 to 2.5
KASPAROV vs DEEP BLUE
the rematch
With a dramatic victory in Game 6, Deep Blue won its six-game rematch with Champion Garry Kasparov
OVERVIEW
EVENT COVERAGE
MATCH NEWS
MAIN STORIES
Commentary
George Plimpton on chess, Kasparov, and the limitations of computers
Read the article
Commentary
Vishwanathan Anand on the legacy of Kasparov vs. Deep Blue
Read the article
Club Kasparov
Visit the virtual home of the world's greatest chess player.
Guest essays
Thoughts on chess, computers, and what it all means
Read the essays...
Community
During the rematch, more than 20,000 people from 120 countries joined the community to talk about the match.
Clips from the rematch
Video footage from the games
Highlights from the games
Press room
Chess reference
Feedback
Site guide
IBM
Document: Done

Chess Grand Master Garry Kasparov, left, comtemplates his next move against IBM's Deep Blue chess computer while Chung-Jen Tan, manager of the Deep Blue project looks on iduring the first game of a six-game rematch between Kasparov and Deep Blue in this file photo from 1997. The computer program made history by becoming the first to beat a world chess champion, Kasparov, at a serious game. Photo: Adam Nadel/Associated Press

Othello: Murakami vs. Logistello

Takeshi Murakami
World Othello Champion



open sourced

- 1997: The Logistello software crushed Murakami, 6 to 0
- Humans can not win against it
- Othello, with 10^{28} states, is still not solved

Go: Goemate vs. a young player

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:

Go has too high a branching factor for existing search techniques

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

thereby winning $15,000

Typical simple case for a game

- **2-person** game
- Players alternate moves
- **Zero-sum**: one player's loss is the other's gain
- **Perfect information**: both players have access to complete information about the state of the game. No information is 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 serch?
- Heuristic Search?
- Local Search?
- Constraint based search?

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** is used to evaluate the "goodness" of a game position
  - Contrast with heuristic search where evaluation function is a non-negative estimate of cost from the start node to a goal and passing through given node
- Zero-sum assumption permits a single evaluation function to describe goodness of board w.r.t. both players
  - **f(n) >> 0**: position n good for me and bad for you
  - **f(n) << 0**: position n bad for me and 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

- Example evaluation function for Tic-Tac-Toe

  f(n) = [# 3-lengths open for me] - [# 3-lengths open for other]

  where a 3-length is a complete row, column, or diagonal

- Alan Turing's function for chess
  - **f(n) = w(n)/b(n)** where w(n) = sum of the point value of white's pieces and b(n) = sum of black's
  - Traditional piece values are
    - Pawn:1
    - Knight, bishop: 3
    - Rook: 5
    - Queen: 9

Evaluation function examples

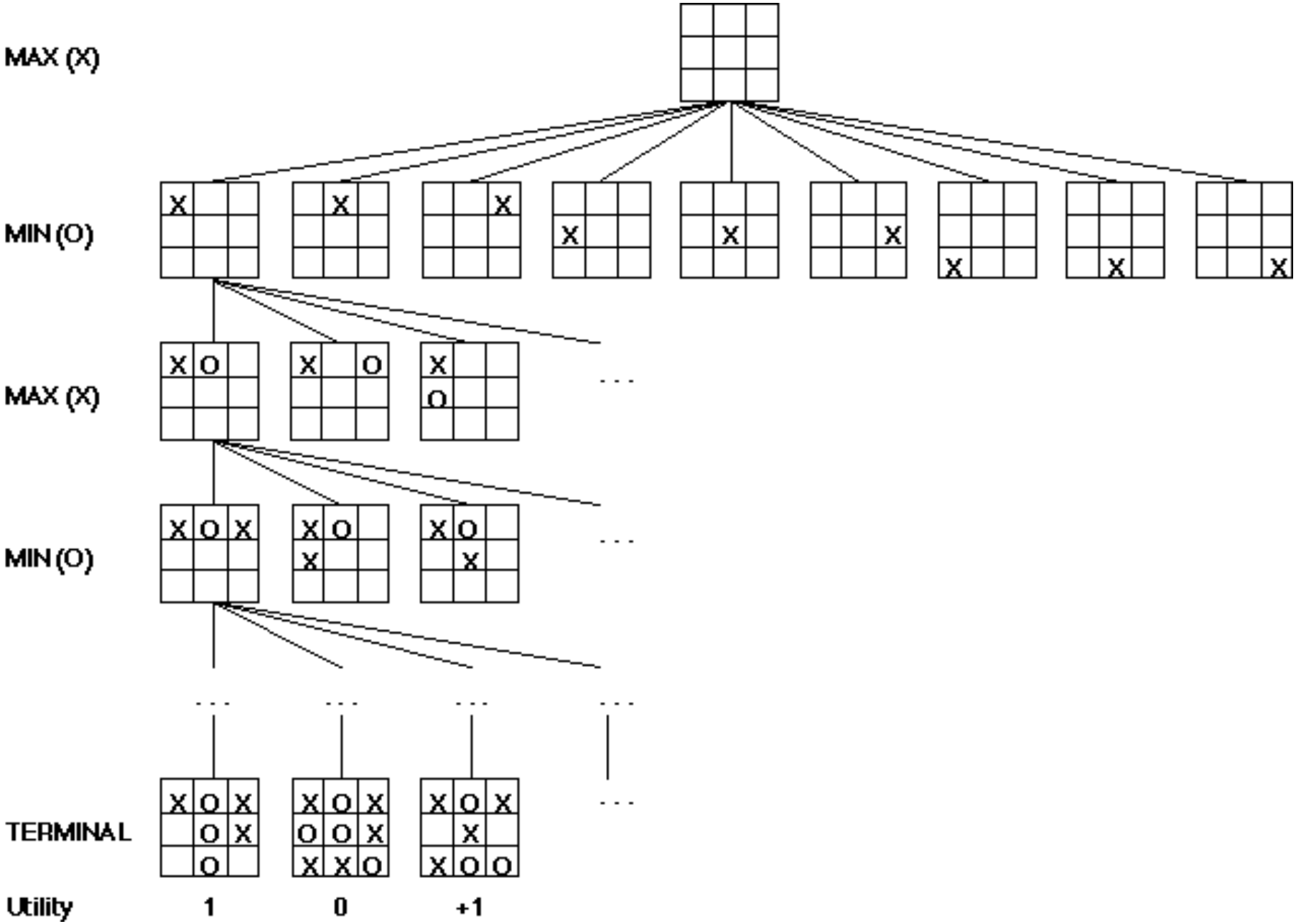
- Most evaluation functions specified as a weighted sum of positive features

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

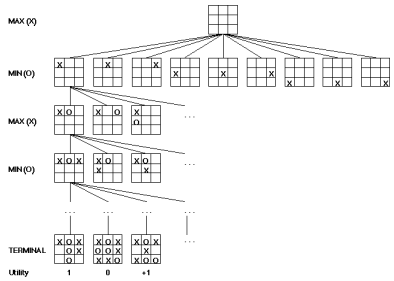
- Example features for chess are piece count, piece values, piece placement, squares controlled, etc.
- Deep Blue had >8K features in its evaluation function

That's not how people play

- People 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?

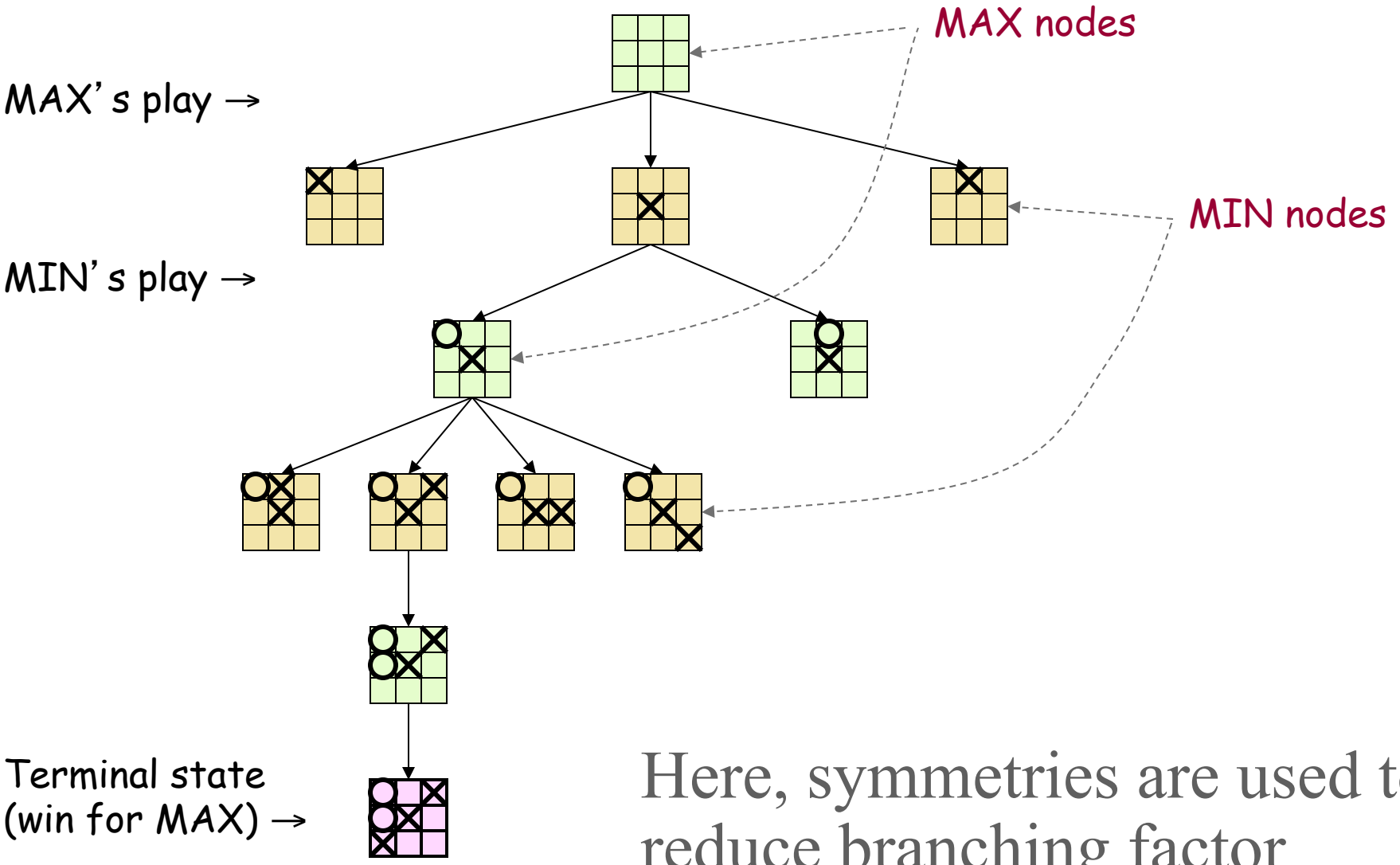

MAX (X)
MIN (O)
MAX (X)
MIN (O)
TERMINAL
Utility
1
0
+1

Game trees



- Problem spaces for typical games are trees
- Root node represents current board configuration; player must decide best single move to make next
- **Static evaluator function** rates a board position **f(board):**real, >0 for me; <0 for opponent
- Arcs represent the possible legal moves for a player
- If it is **my turn** to move, then the root is labeled a "**MAX**" node; otherwise it is labeled a "**MIN**" node, indicating **my opponent's turn**.
- Each tree level has nodes that 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

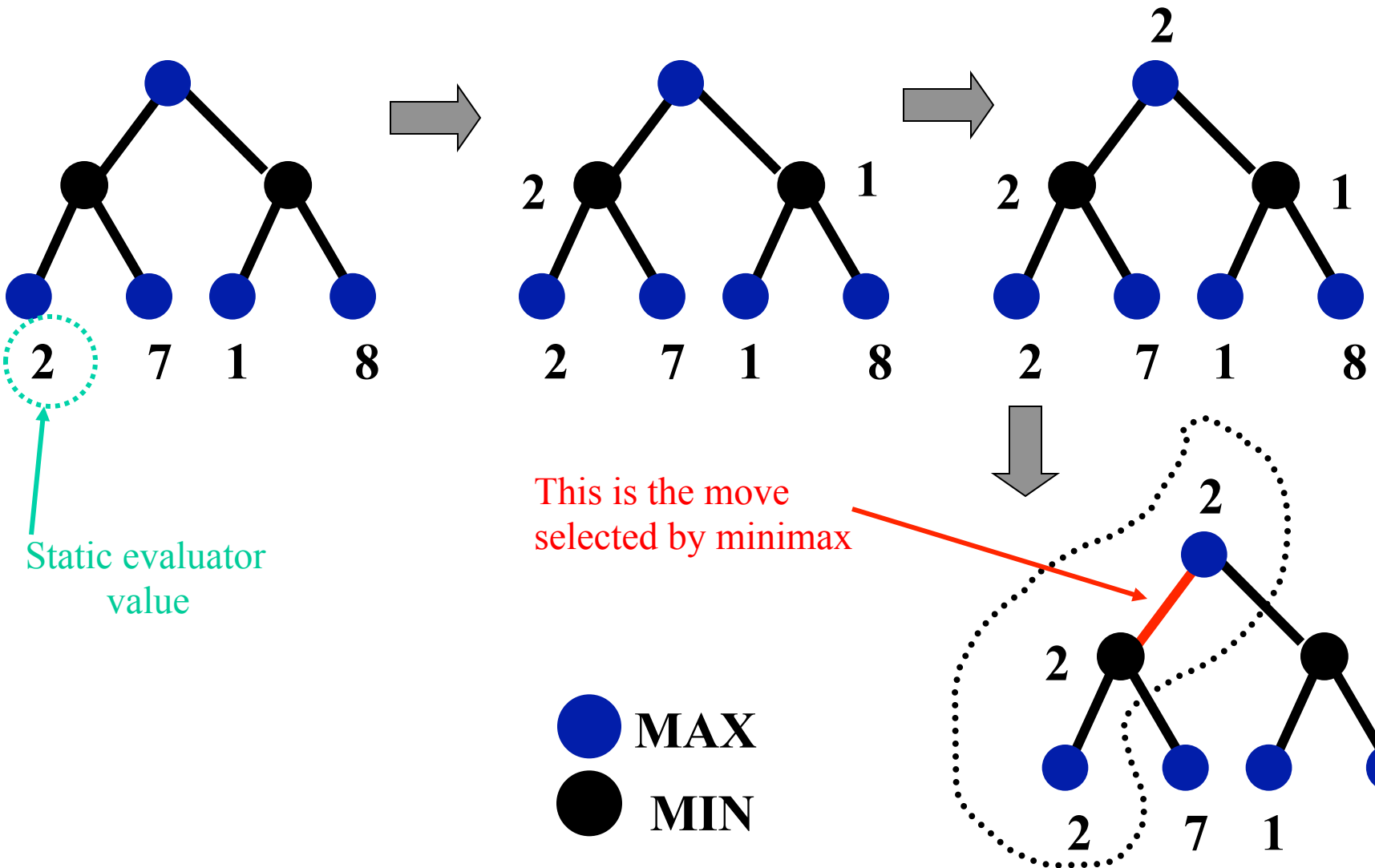
- Create start (MAX) node with current board configuration
- Expand nodes down to some **depth** (a.k.a. **ply**) of lookahead in the game
- Apply evaluation function at each leaf node
- "Back up" values for each of the non-leaf nodes until value is computed for the root node
  - At MIN nodes, backed-up value is **minimum** of values associated with its children.
  - At MAX nodes, backed-up value is **maximum** of values associated with its children.
- Pick operator associated with child node whose backed-up value determined value at the root

Minimax theorem

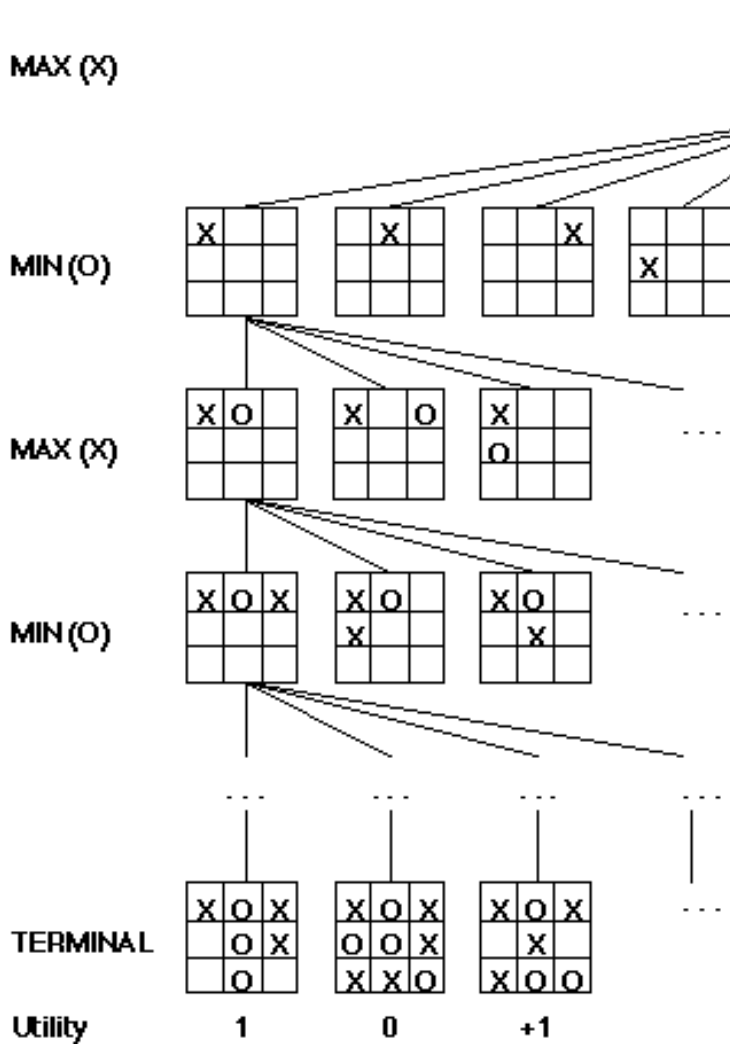
- Intuition: assume your opponent is at least as smart as you and play accordingly
  - If she's not, you can only do better!
- Von Neumann, J: *Zur Theorie der Gesellschafts-spiele* Math. Annalen. **100** (1928) 295-320

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

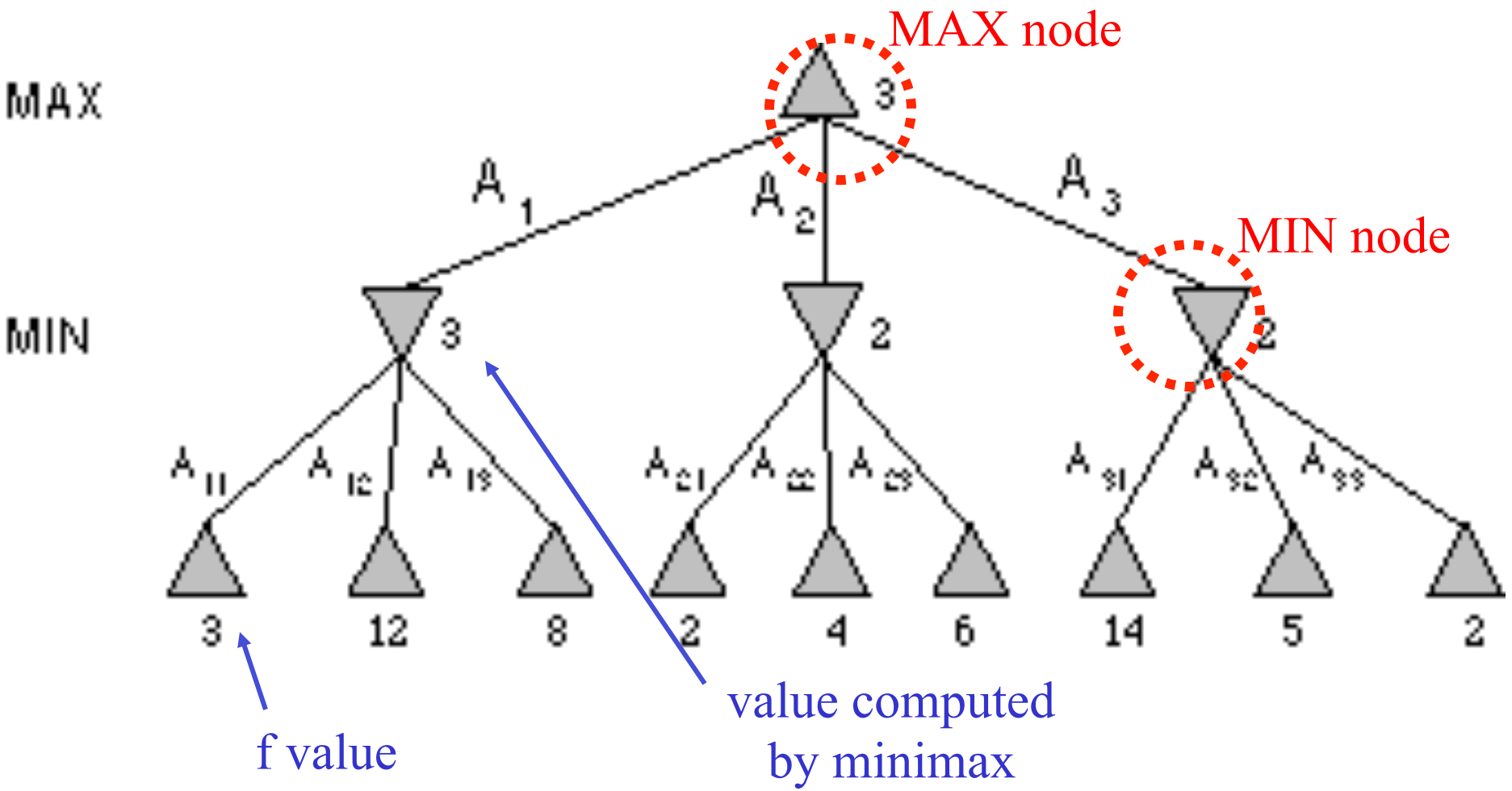


- f(n) = +1 if the position is a win for X.
- f(n) = -1 if the position is a win for O.
- f(n) = 0 if the position is a draw.

Why use backed-up values?

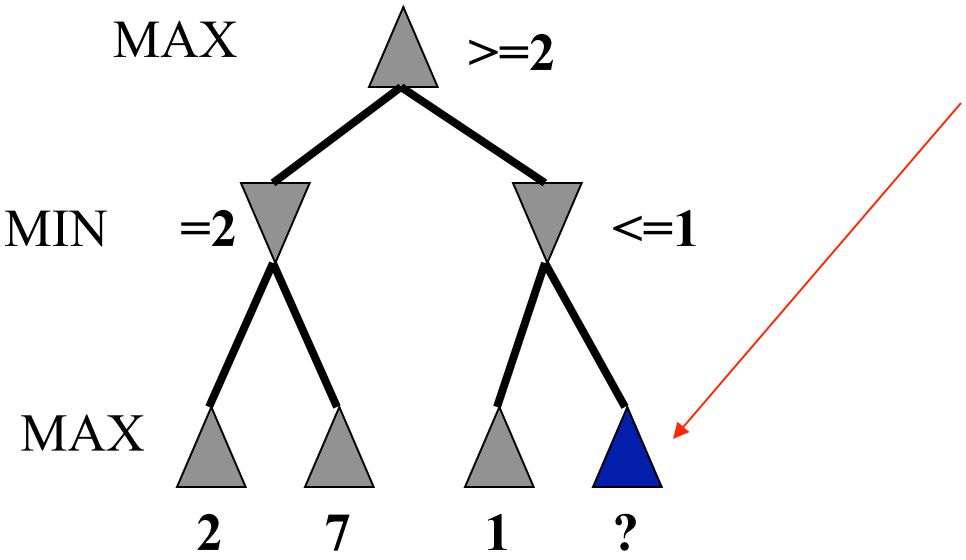
- Intuition: if evaluation function is good, doing look ahead and backing up values with Minimax should be better
- A non-leaf node N's backed-up value is value of best state that MAX can reach at depth **h** if MIN plays well
  - "well" : same criterion as MAX applies to itself
- If e is trustworthy, then backed-up value is better estimate of how favorable STATE(N) is than e(STATE(N))
- We use a horizon **h** because our time to compute a move is limited

Minimax Tree



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

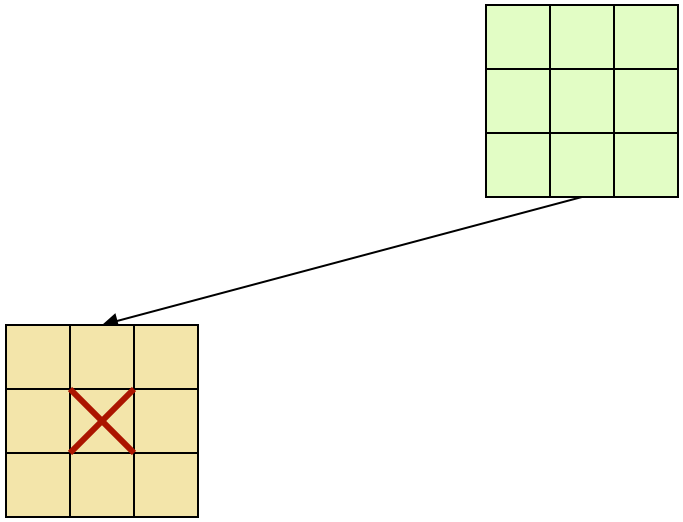


- We don't need to compute the value at this node
- No matter what it is, it can't affect value of the root node

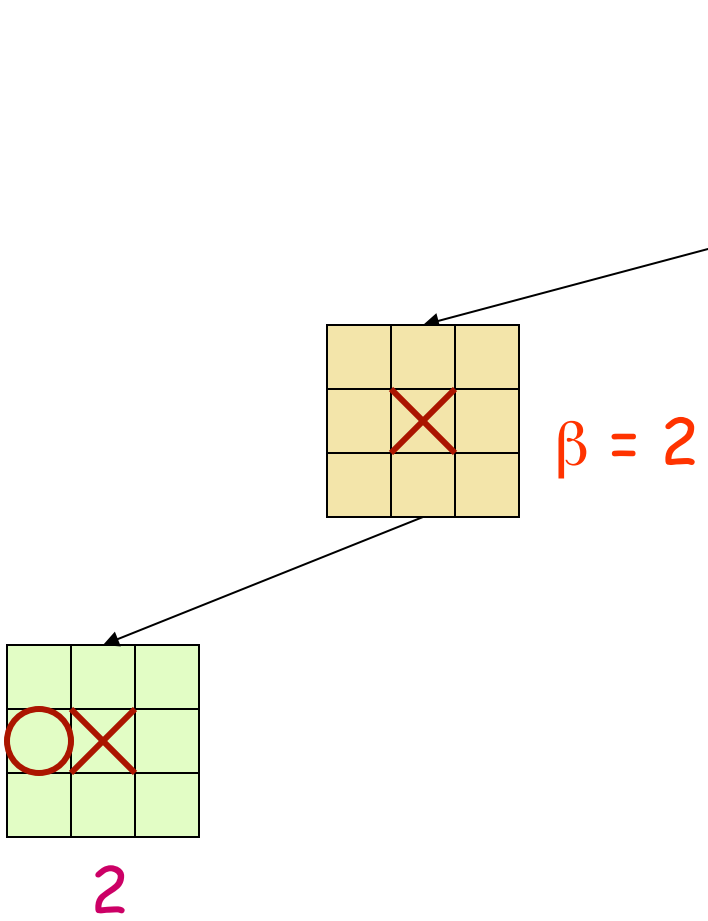
Alpha-beta pruning

- Traverse search tree in depth-first order
- At **MAX** node n, **alpha(n)** = max value found so far
- At **MIN** node n, **beta(n)** = min value found so far
  - Alpha values start at -∞ and only increase, while beta values start at +∞ 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 Tic-Tac-Toe Example

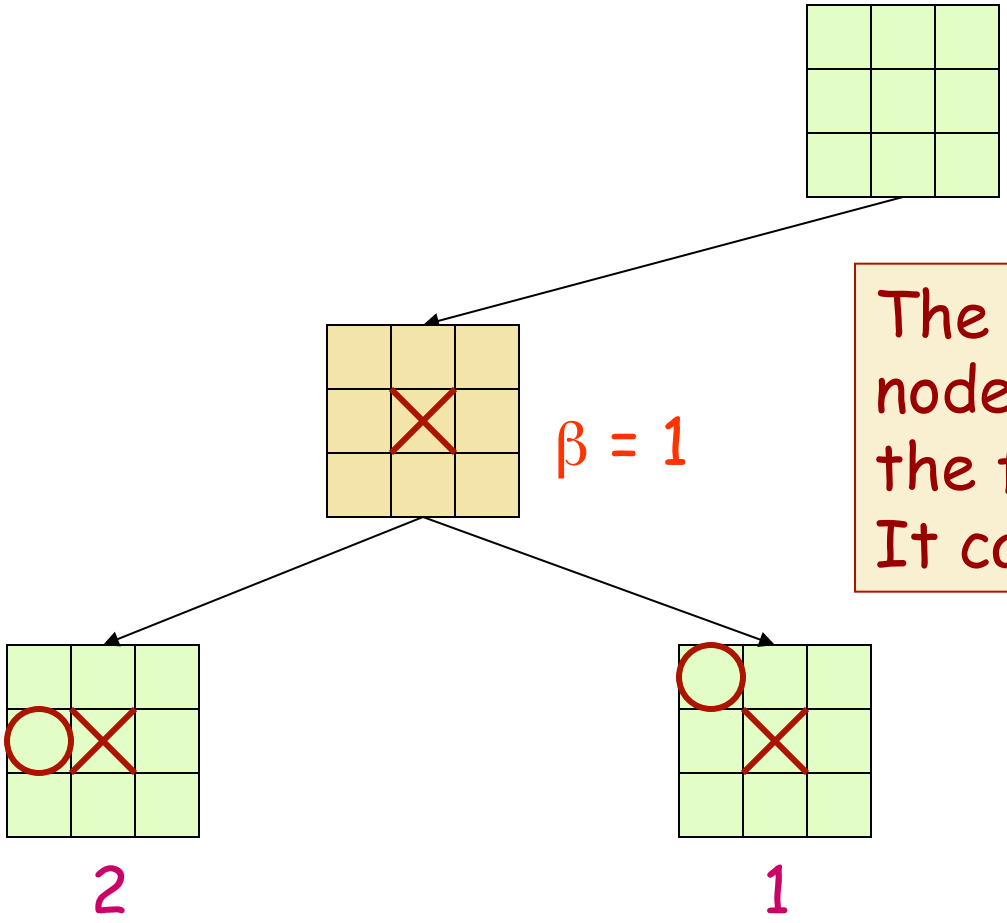


Alpha-Beta Tic-Tac-Toe Example



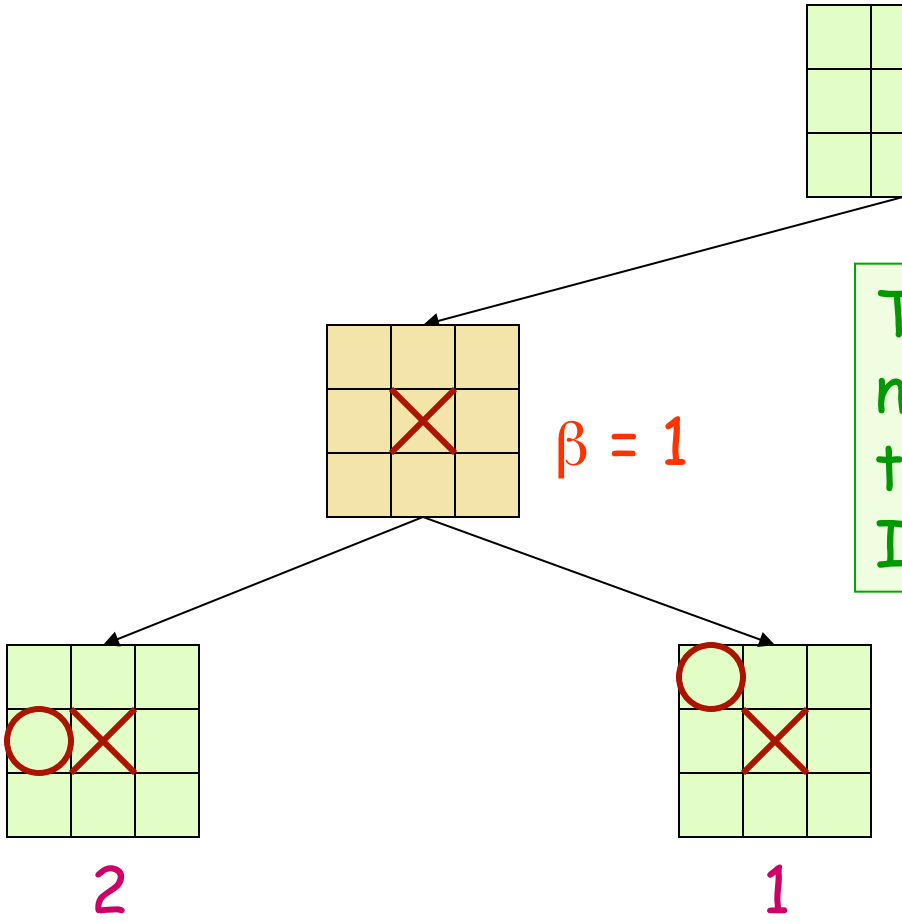
The beta value of a MIN node is an upper bound on the final backed-up value. It can never increase

Alpha-Beta Tic-Tac-Toe Example



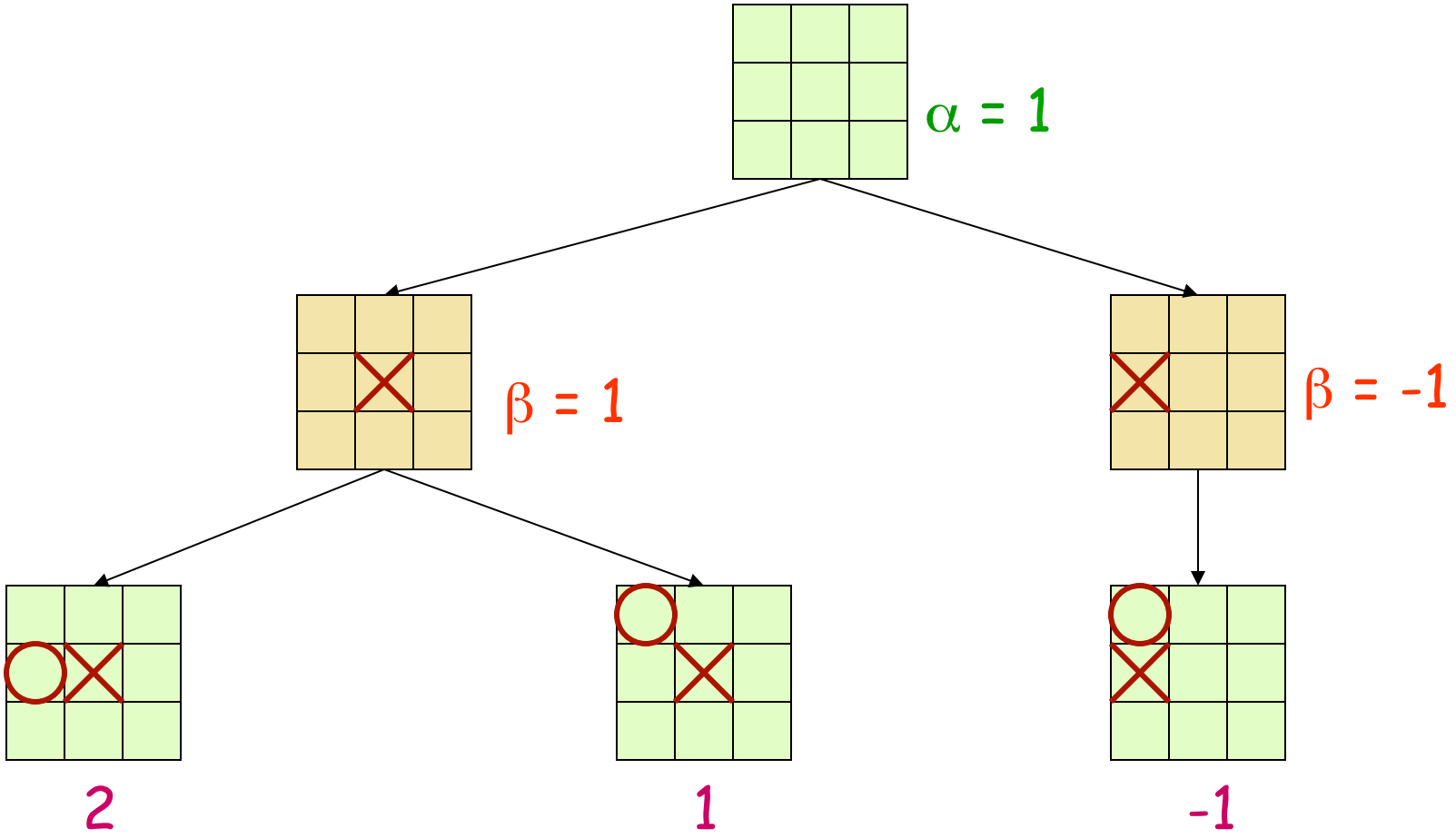
The beta value of a MIN node is an upper bound on the final backed-up value. It can never increase

Alpha-Beta Tic-Tac-Toe Example

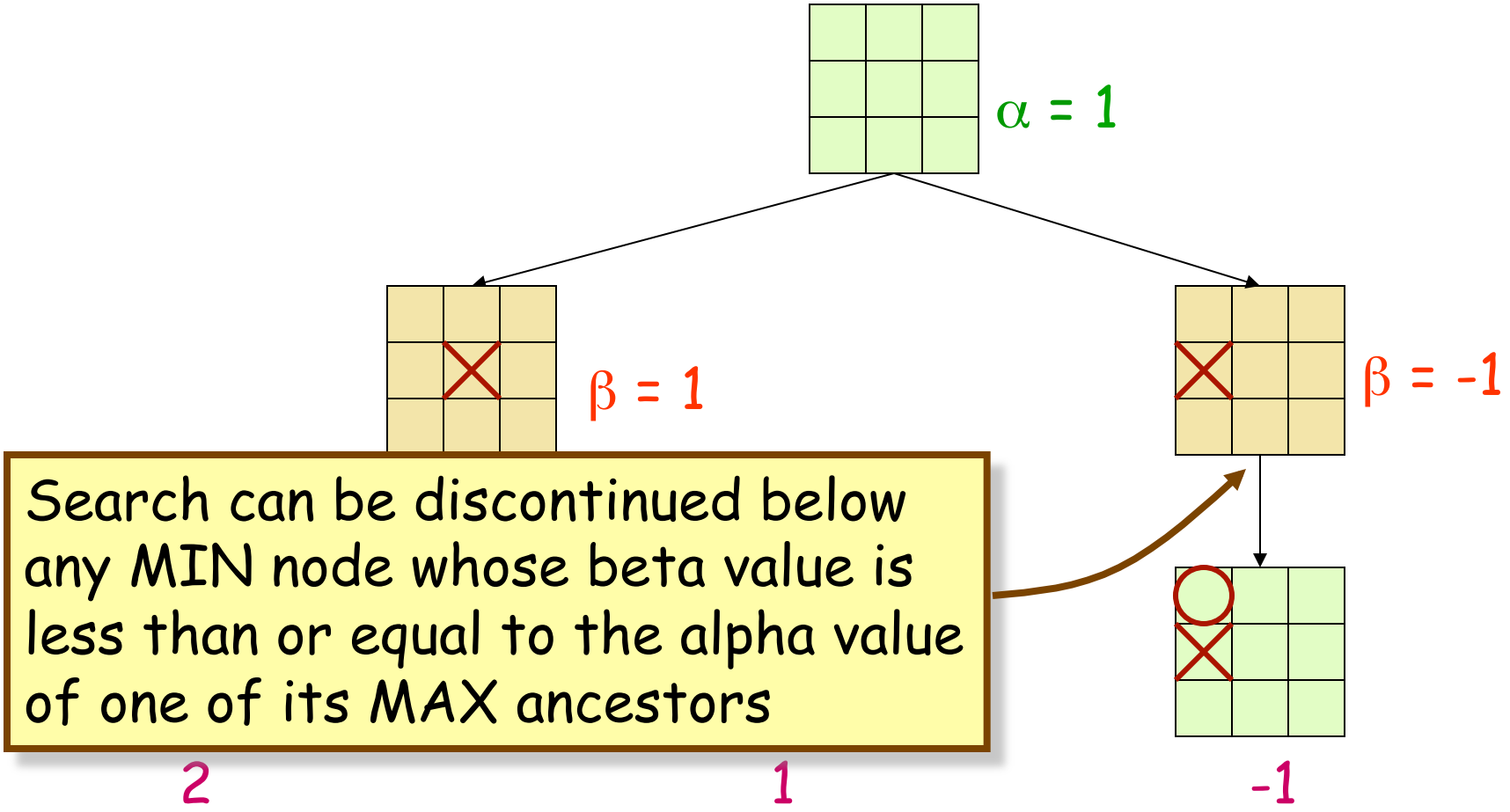


The alpha value of a MAX node is a lower bound on the final backed-up value. It can never decrease

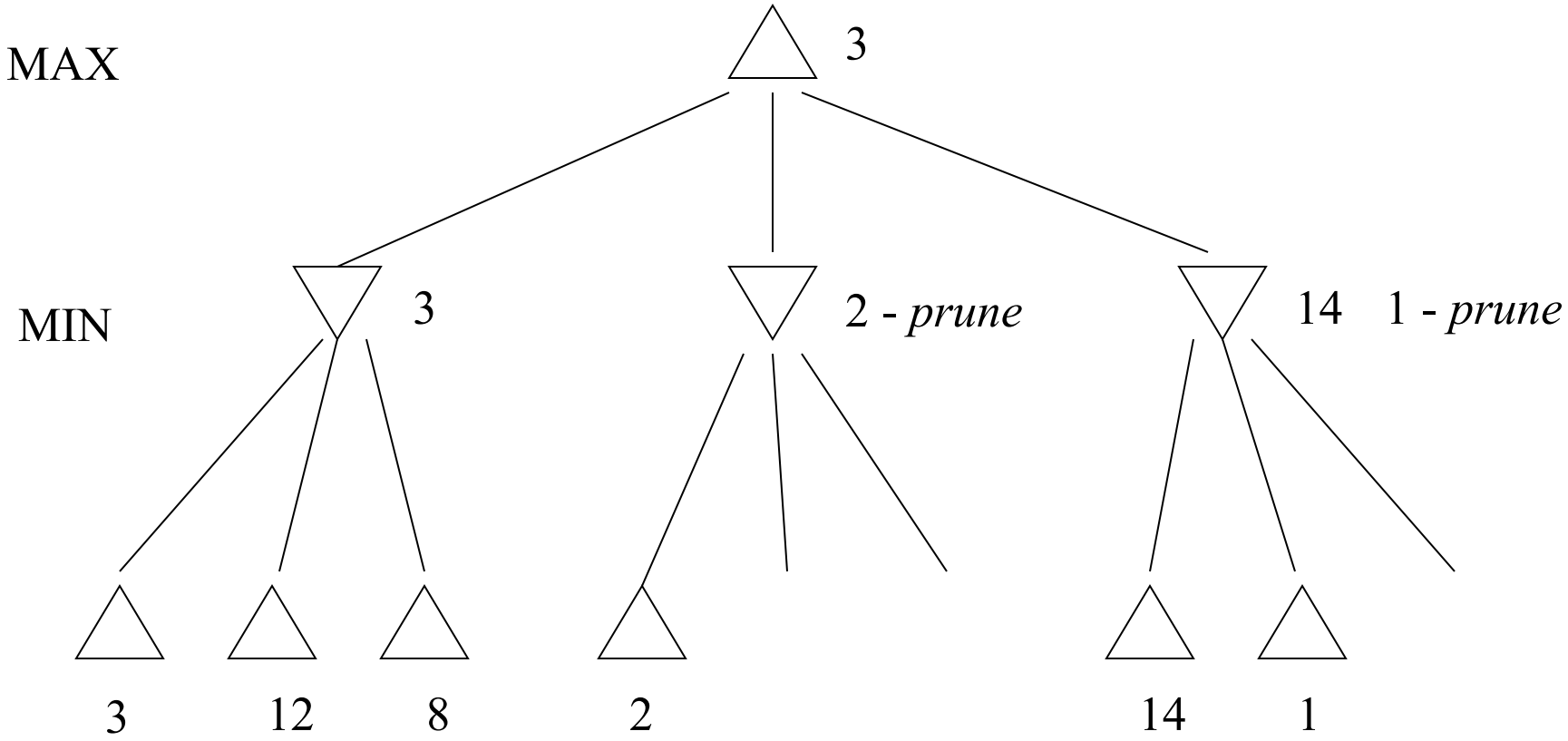
Alpha-Beta Tic-Tac-Toe Example



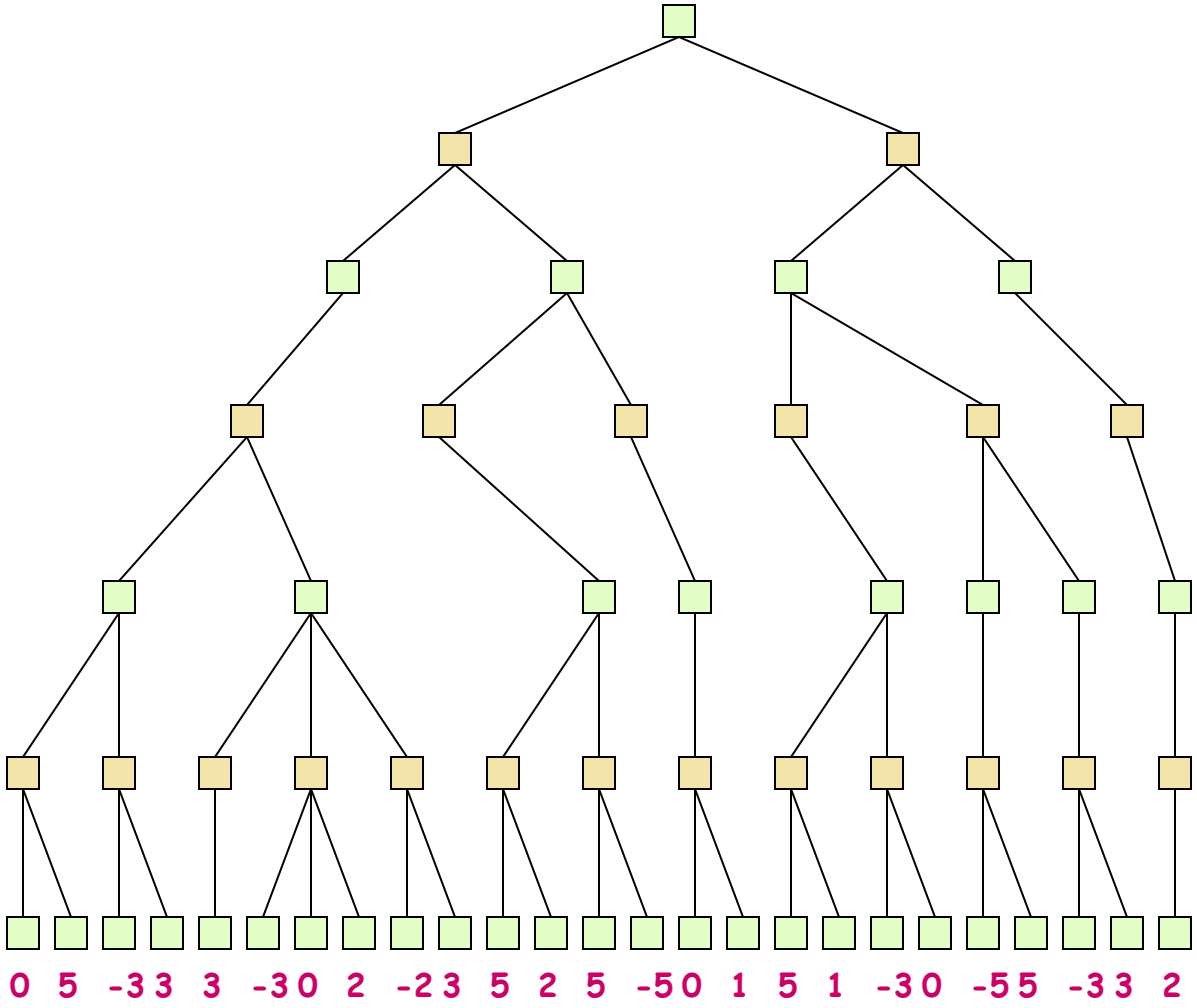
Alpha-Beta Tic-Tac-Toe Example

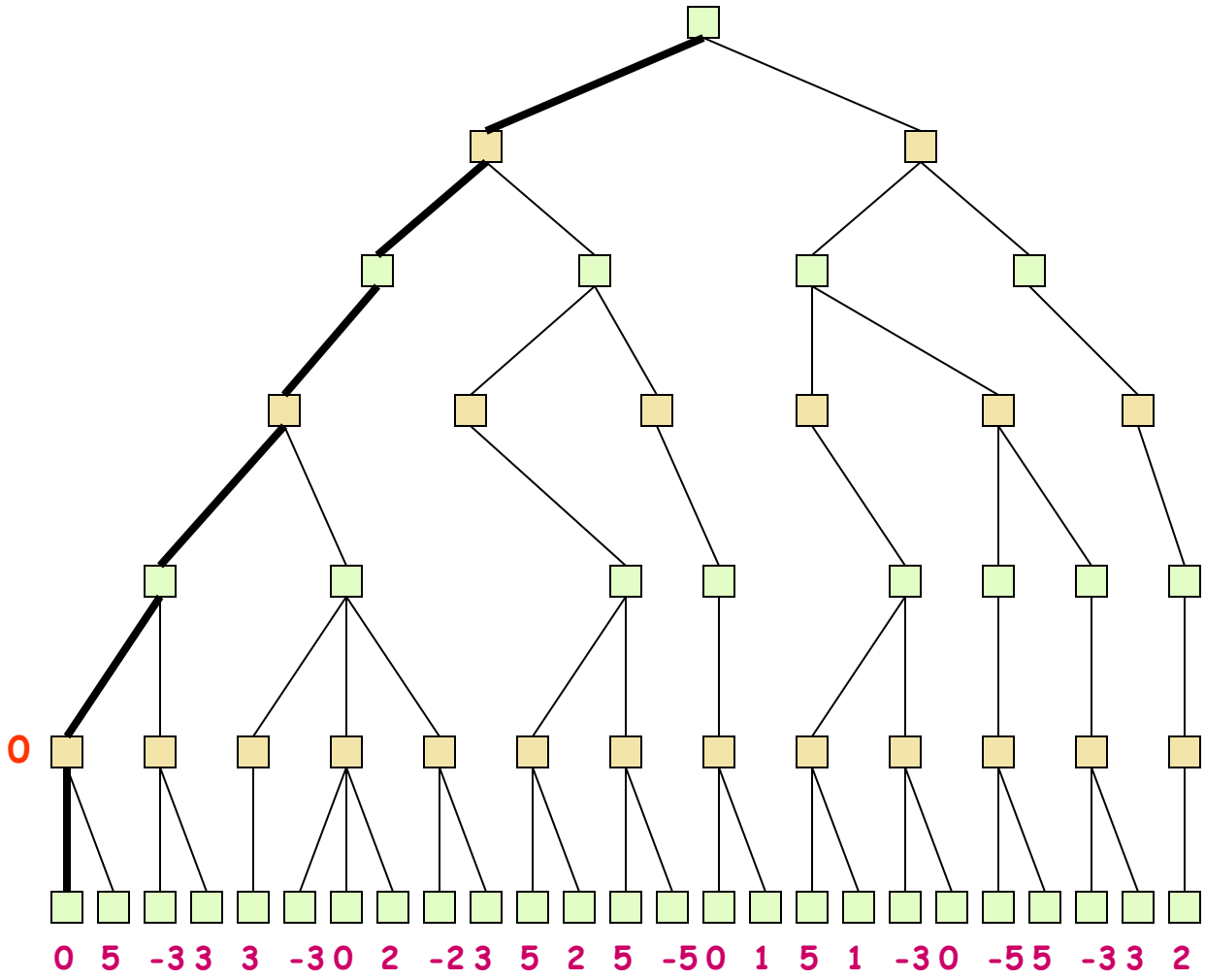


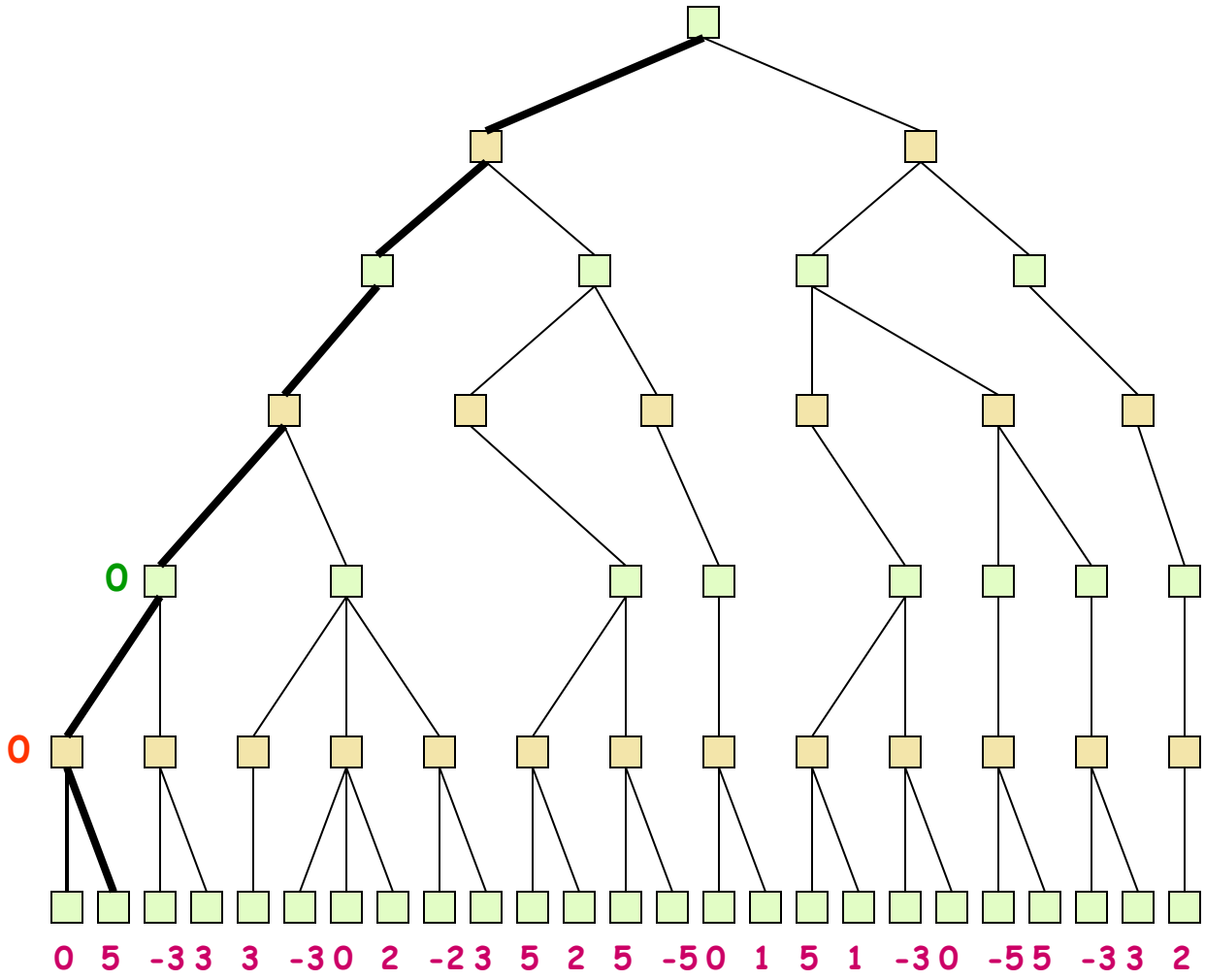
Alpha-beta general example

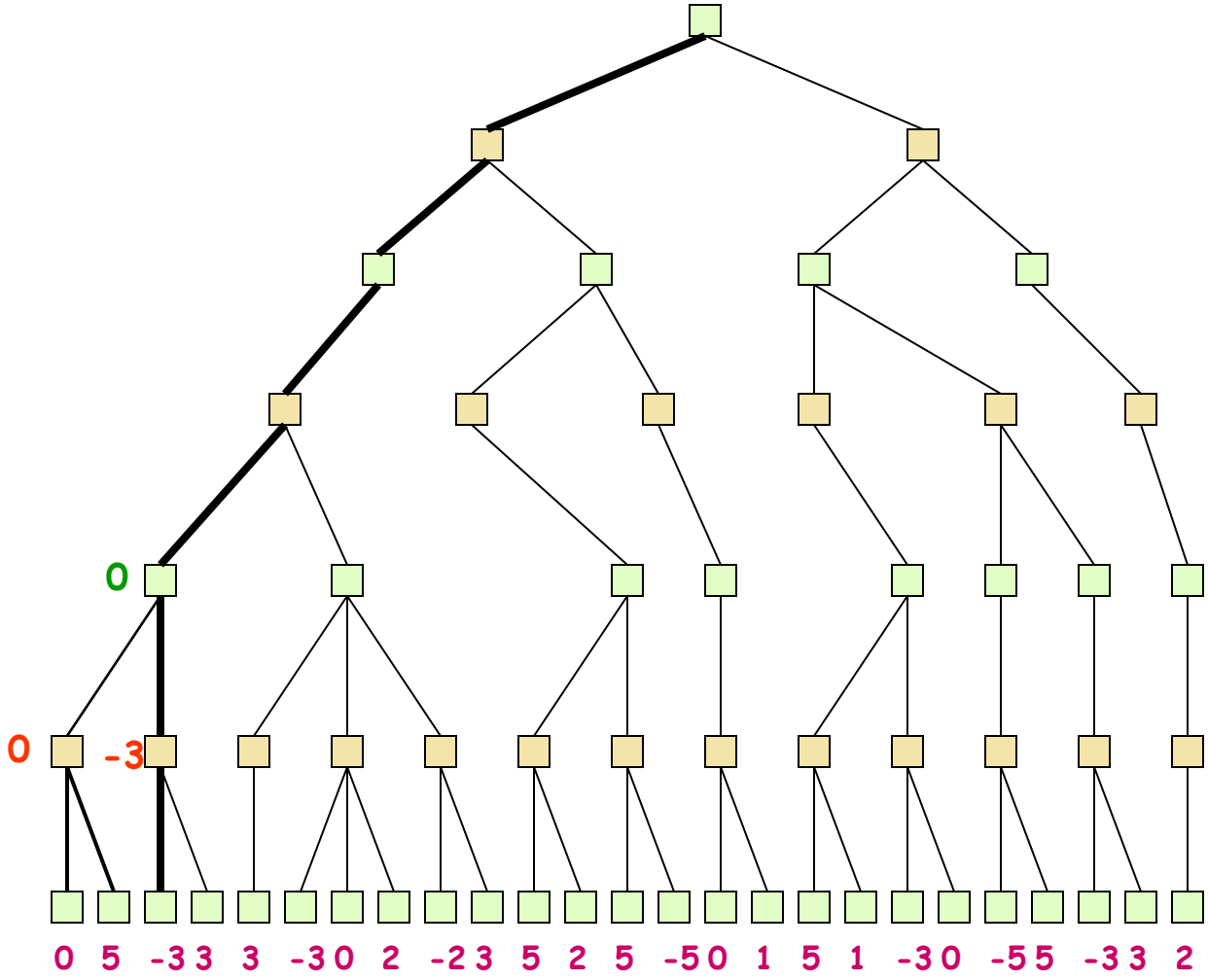


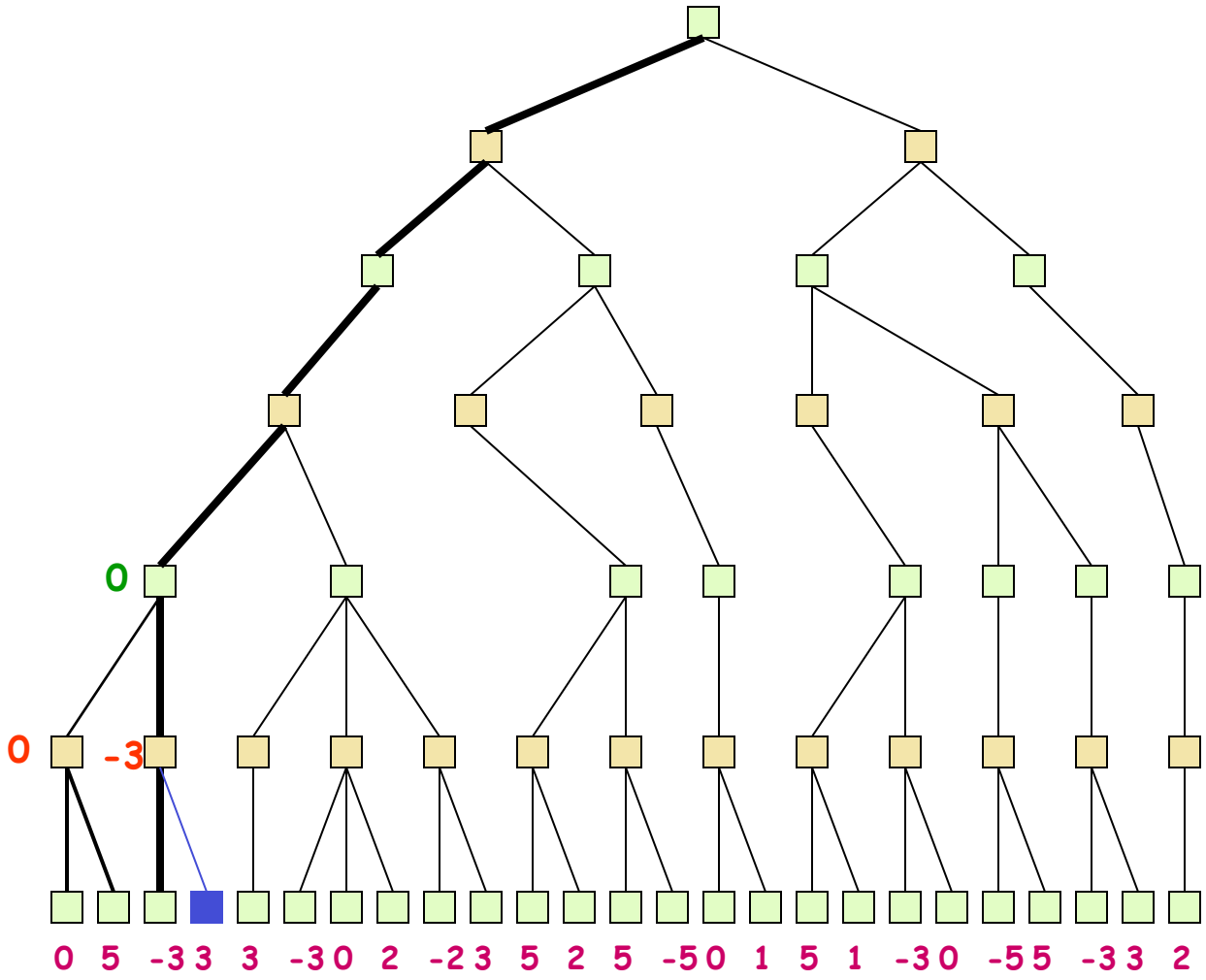
Alpha-Beta Tic-Tac-Toe Example 2

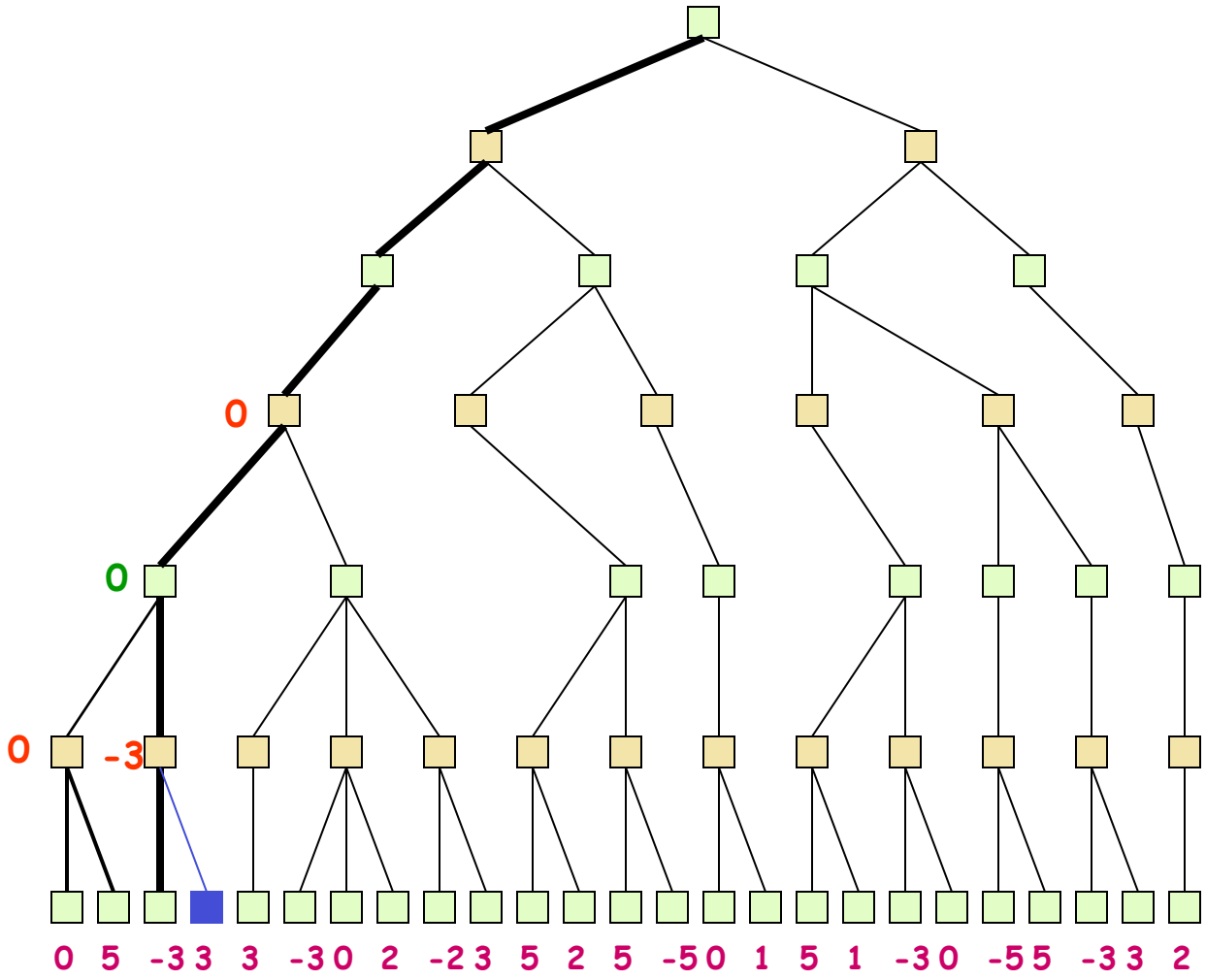


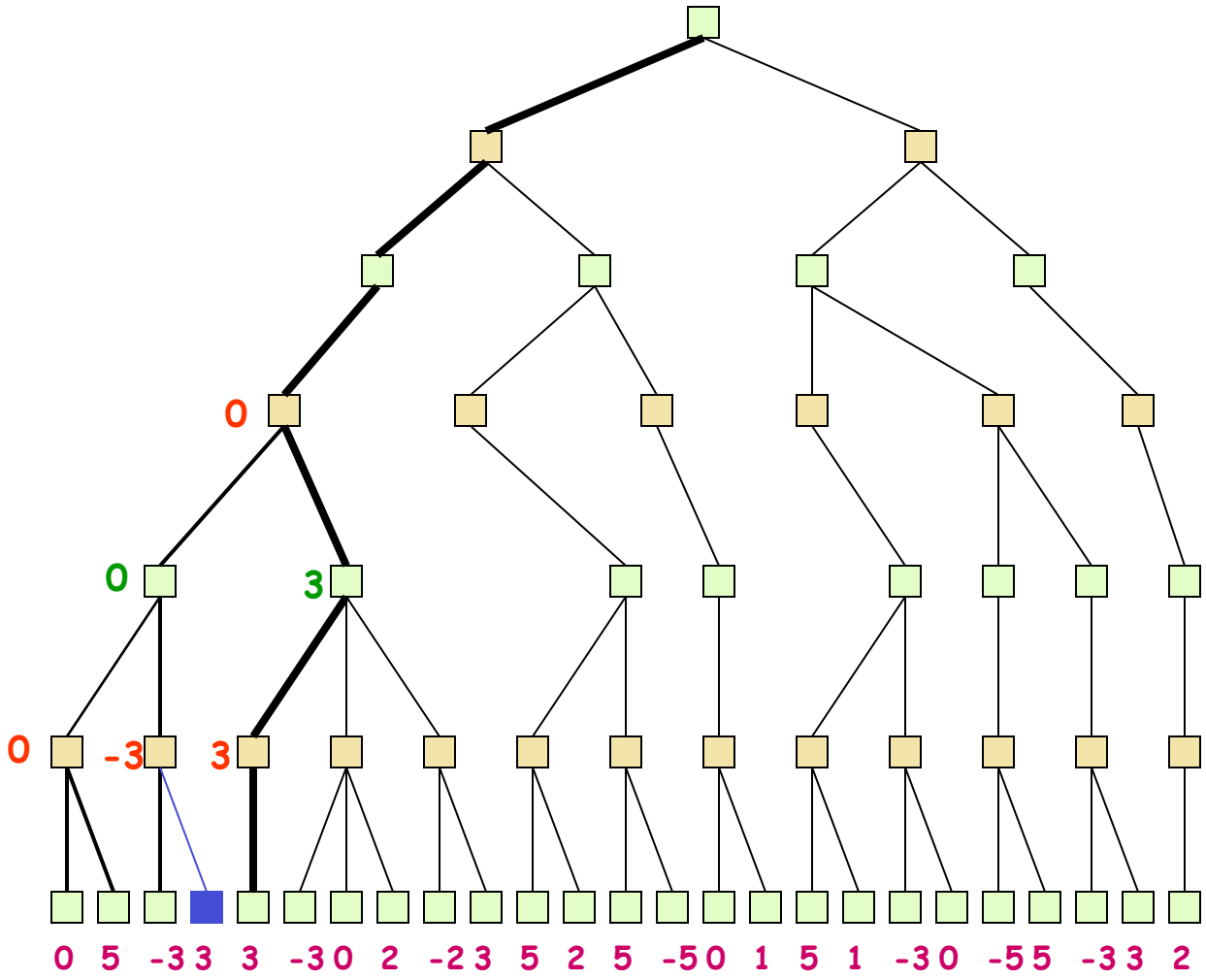

0
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

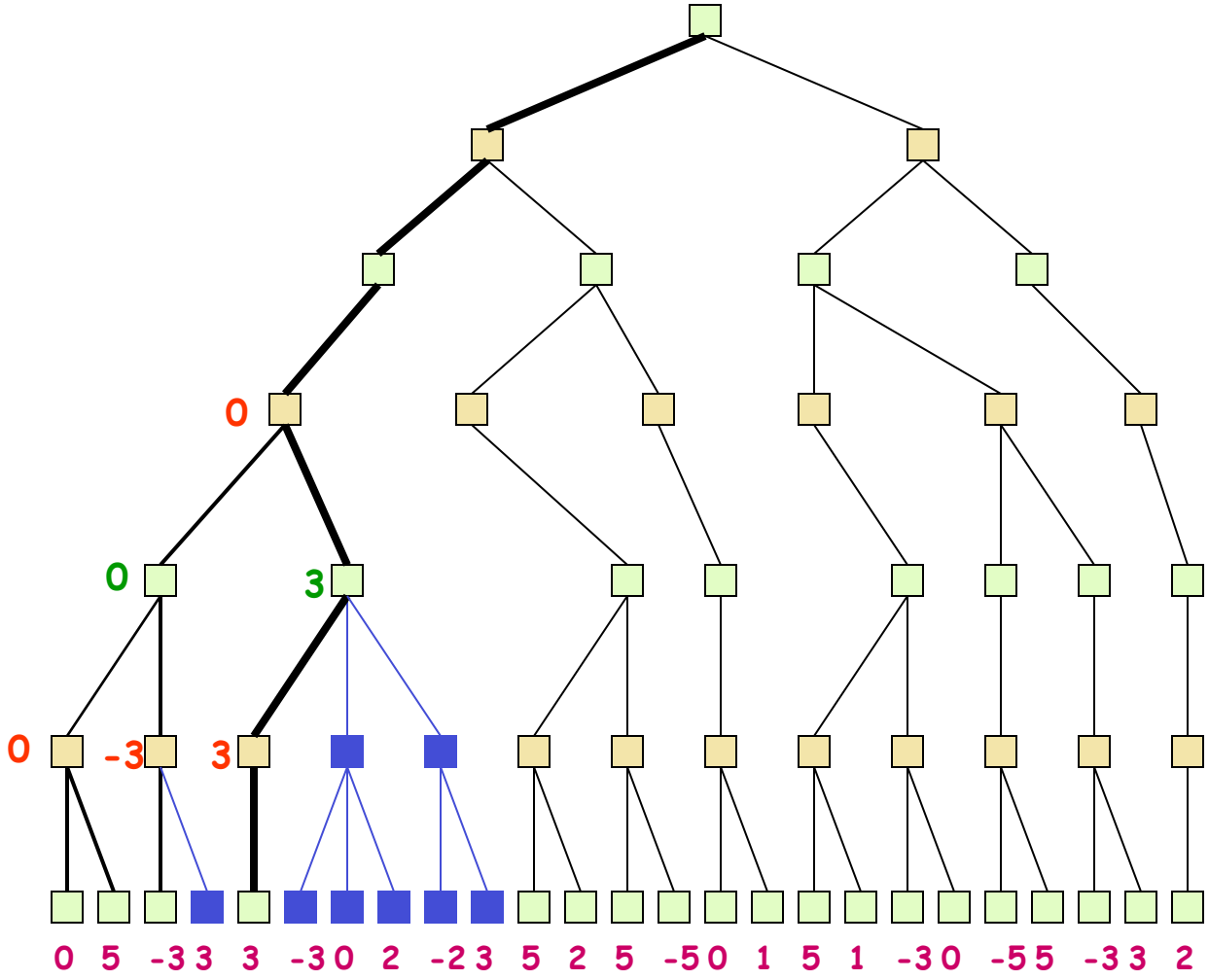

0
0
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

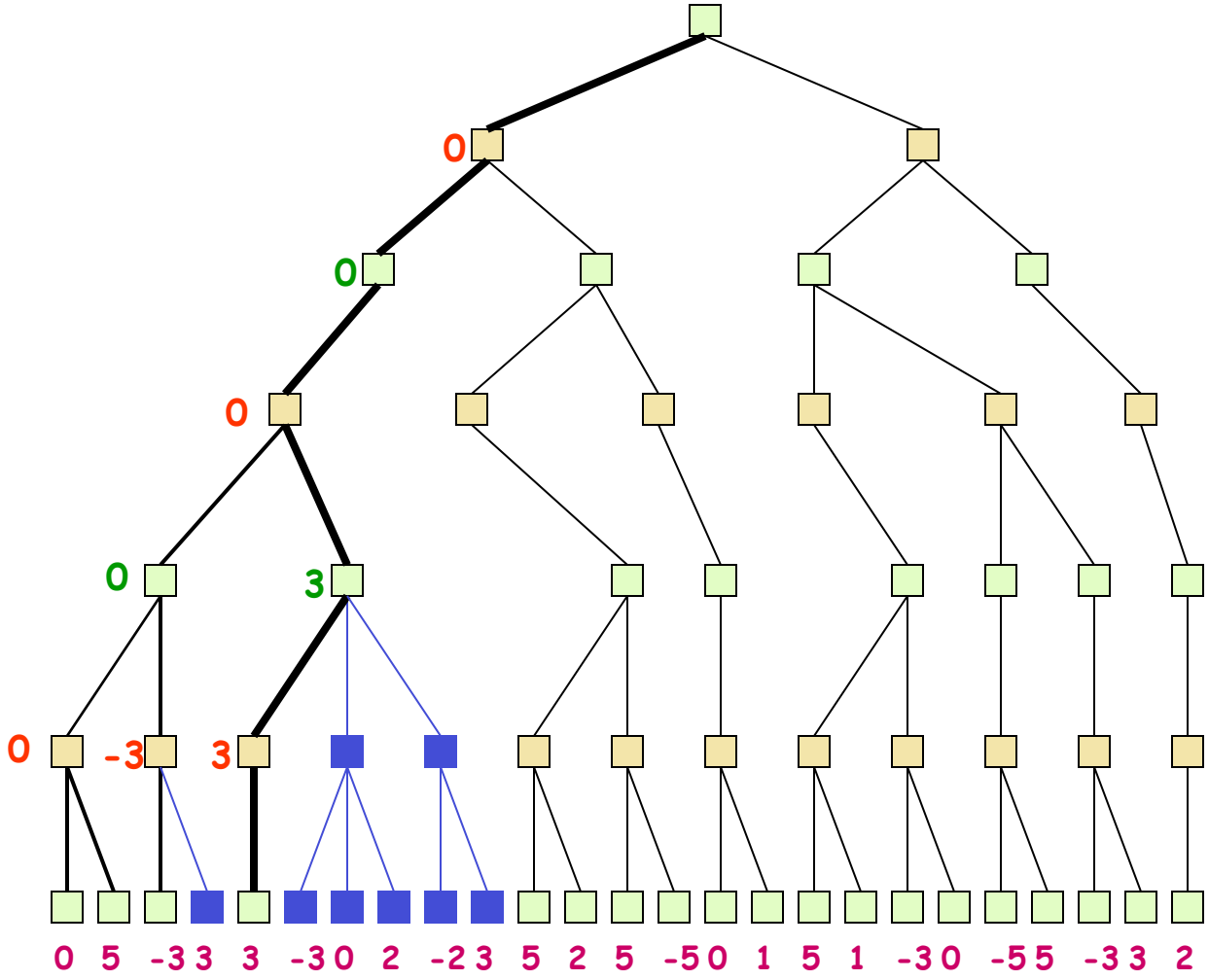

0
0
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

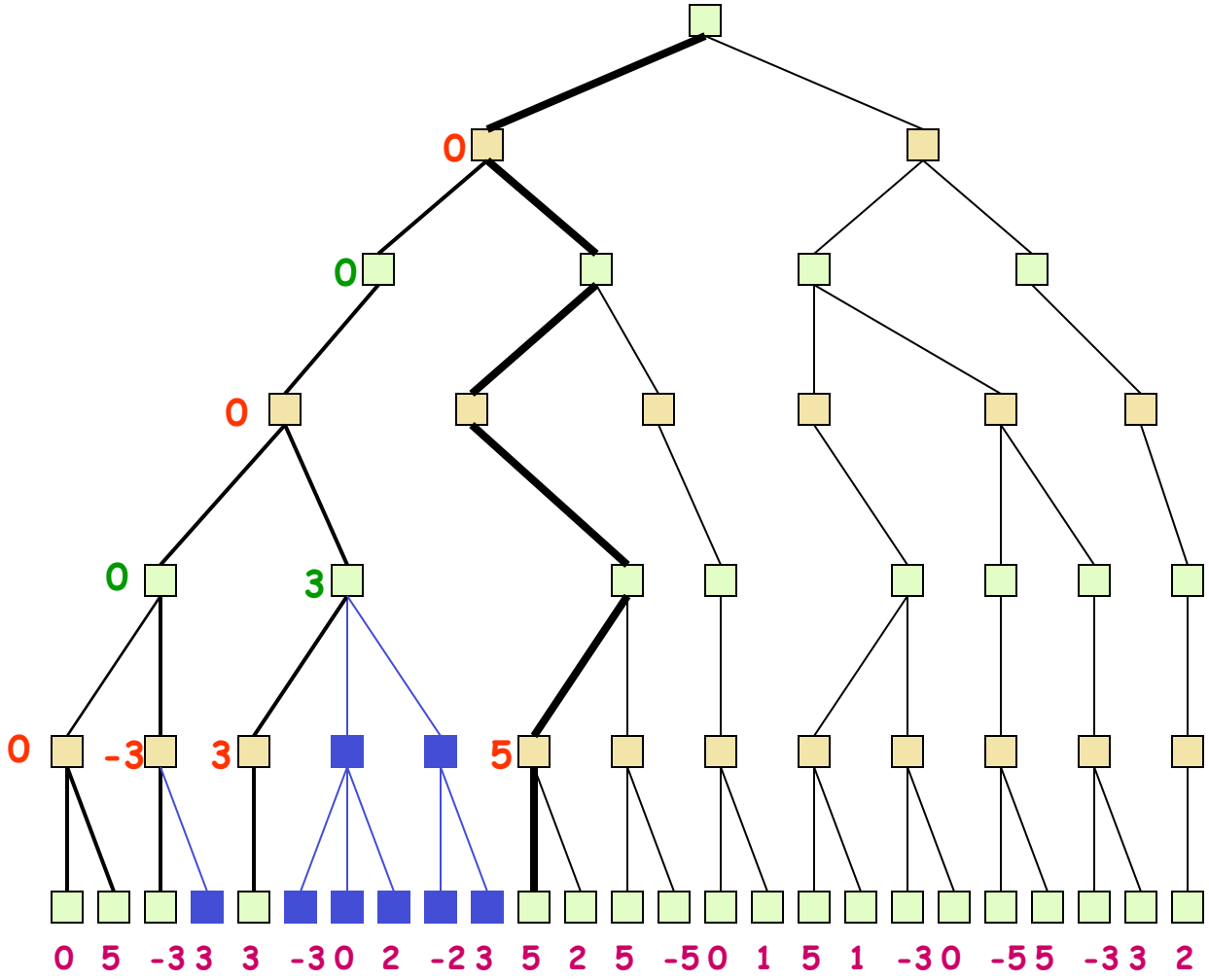

0
0
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

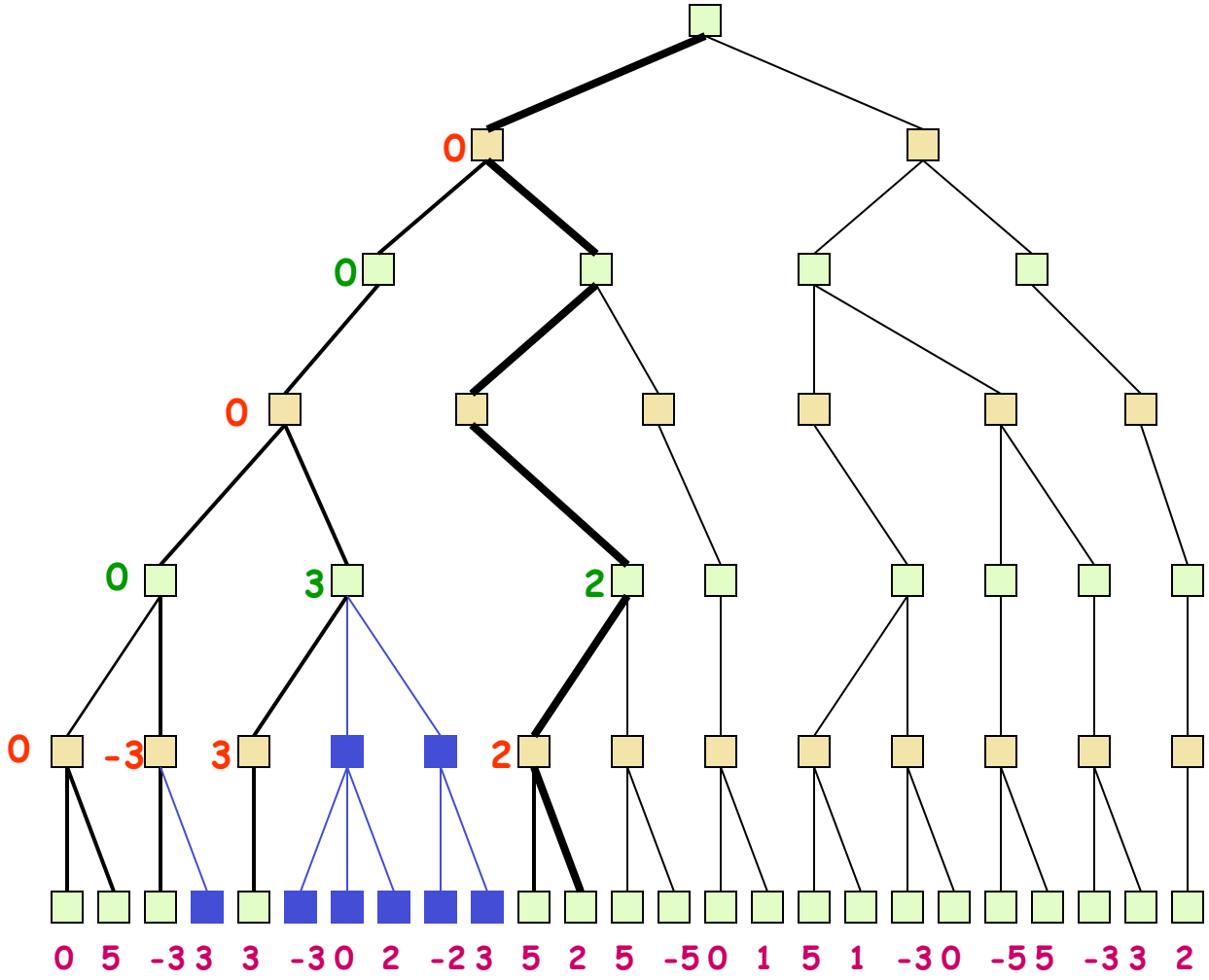

0
0
0
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

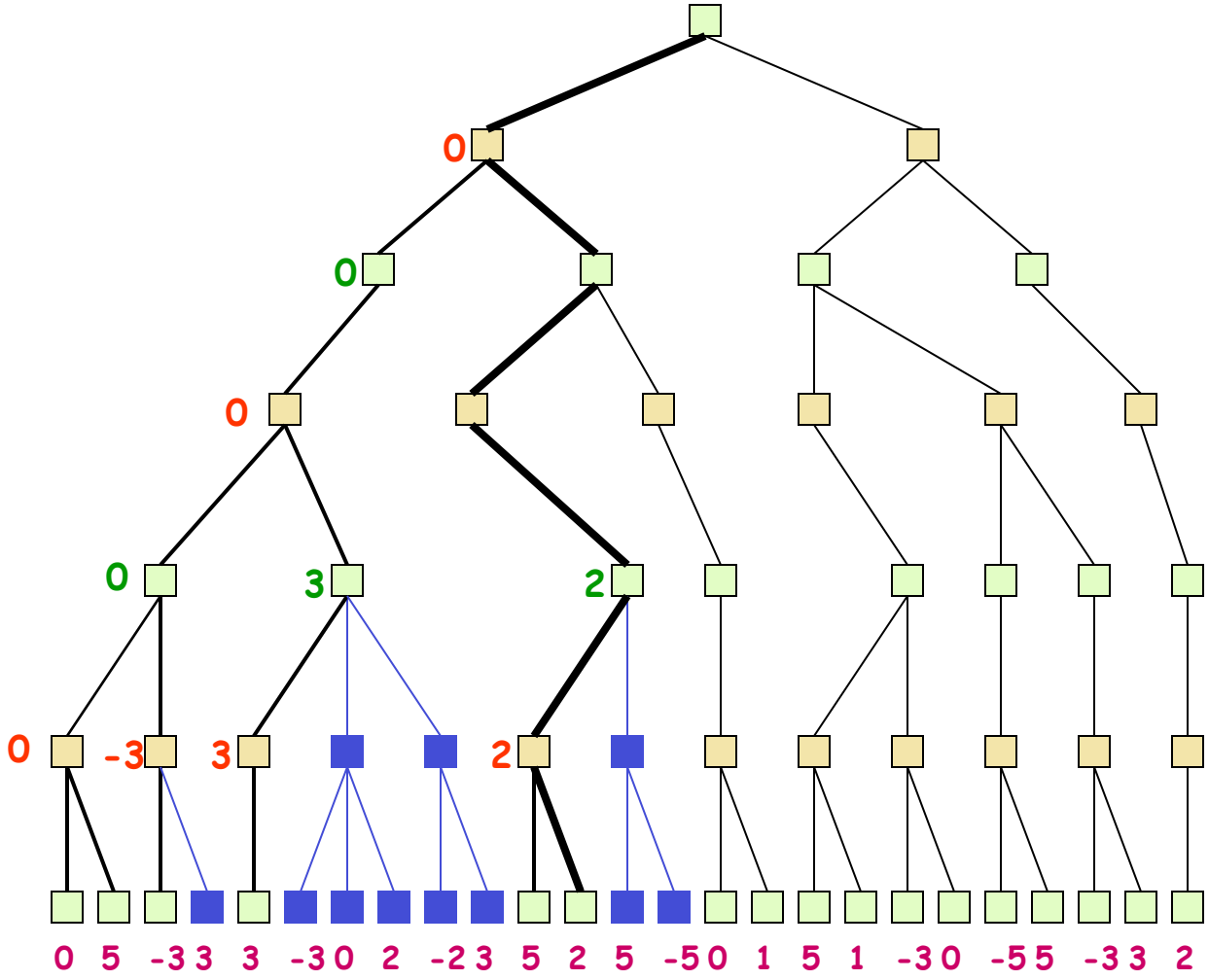

0
0
3
0
-3
3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

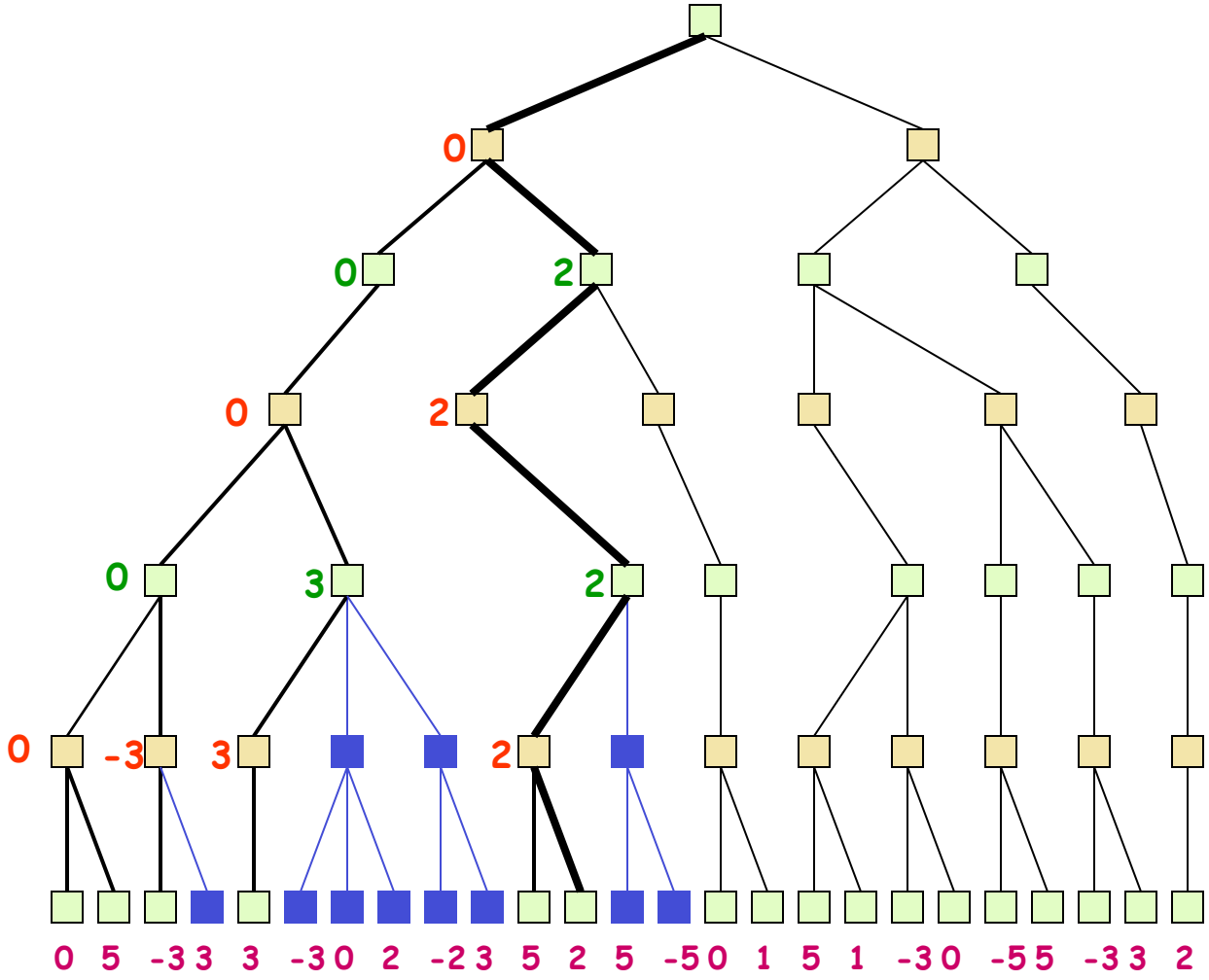

0
0
3
0
-3
3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

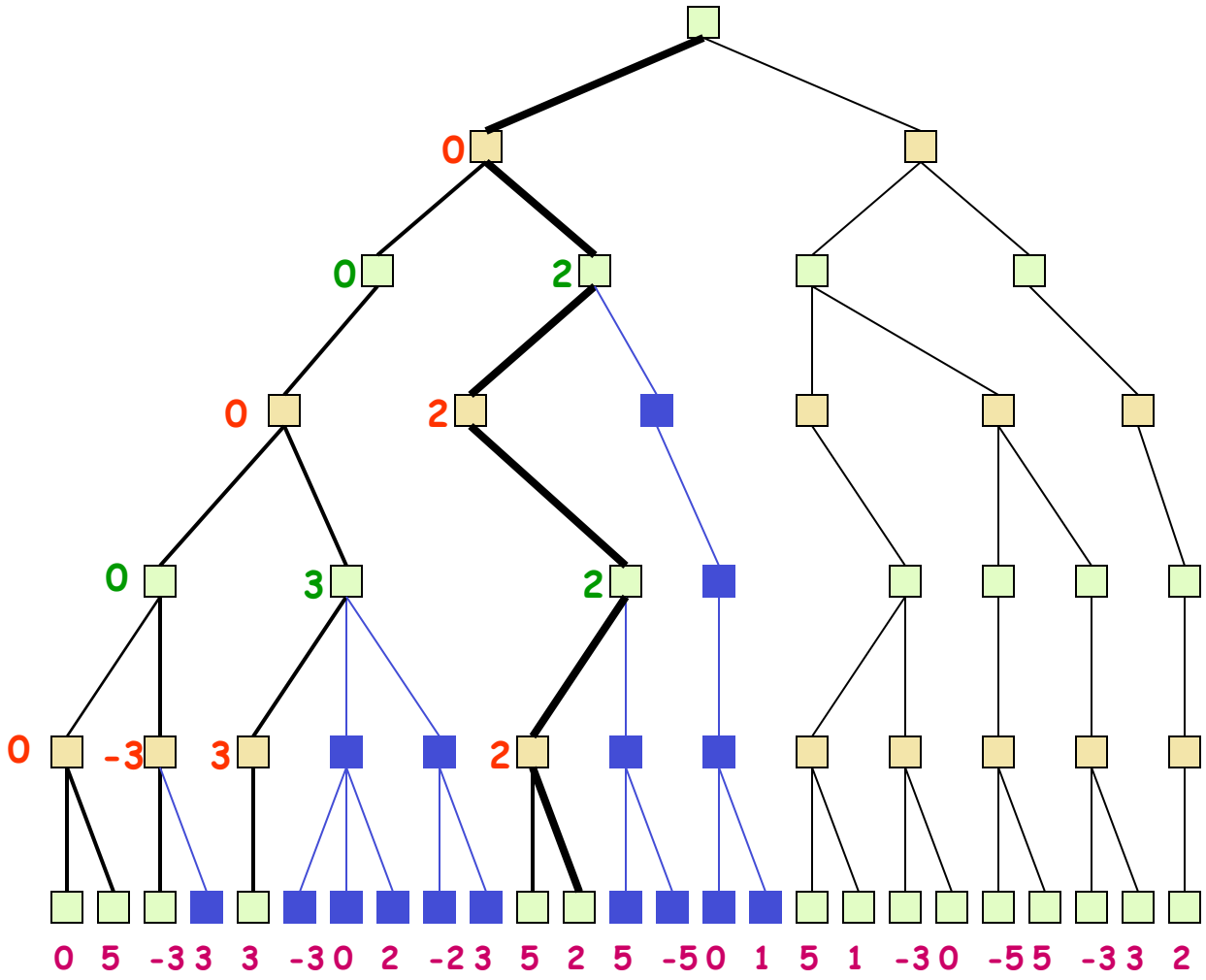

0
0
0
0
3
0
-3
3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

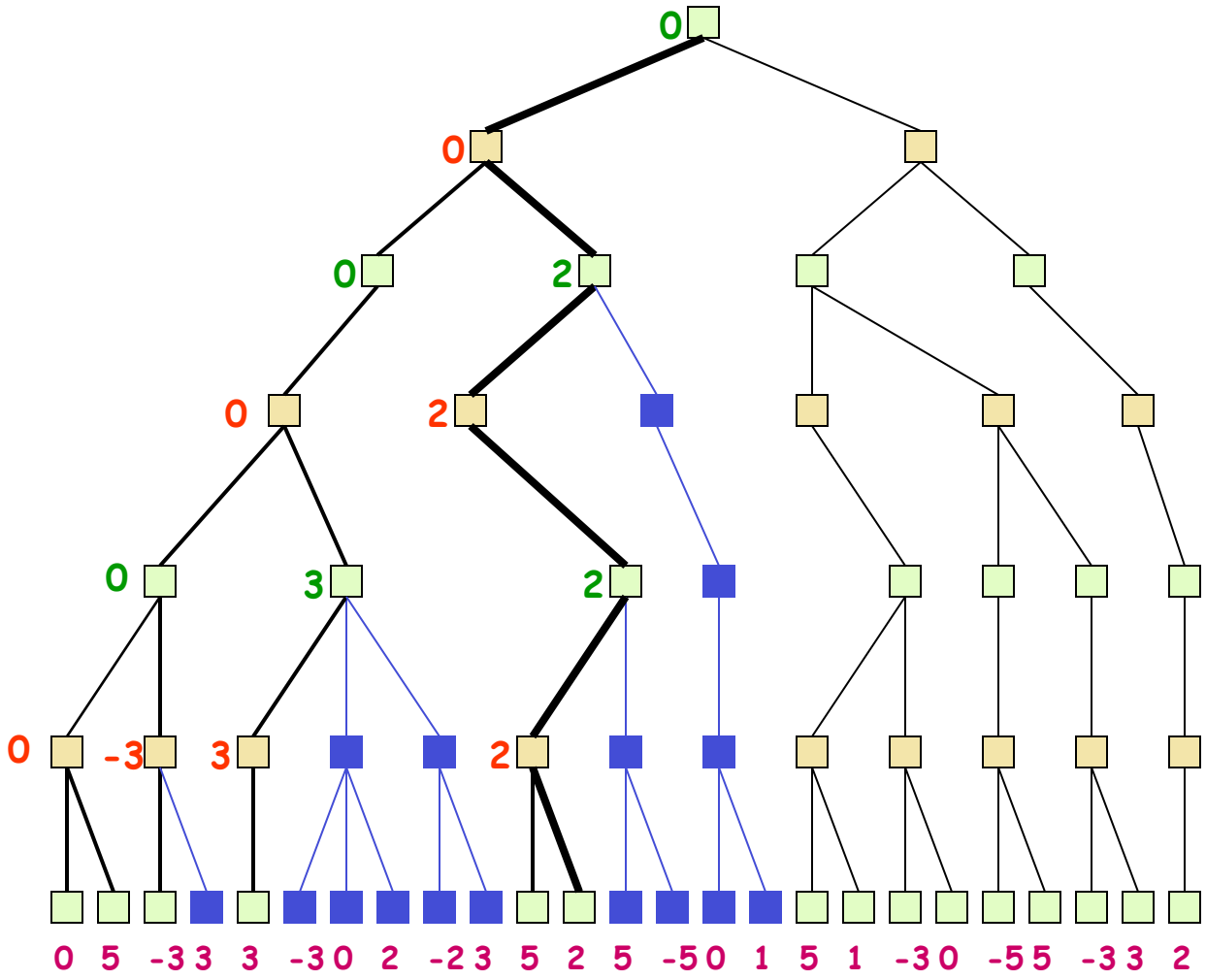

0
0
0
0
3
0
-3
3
5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

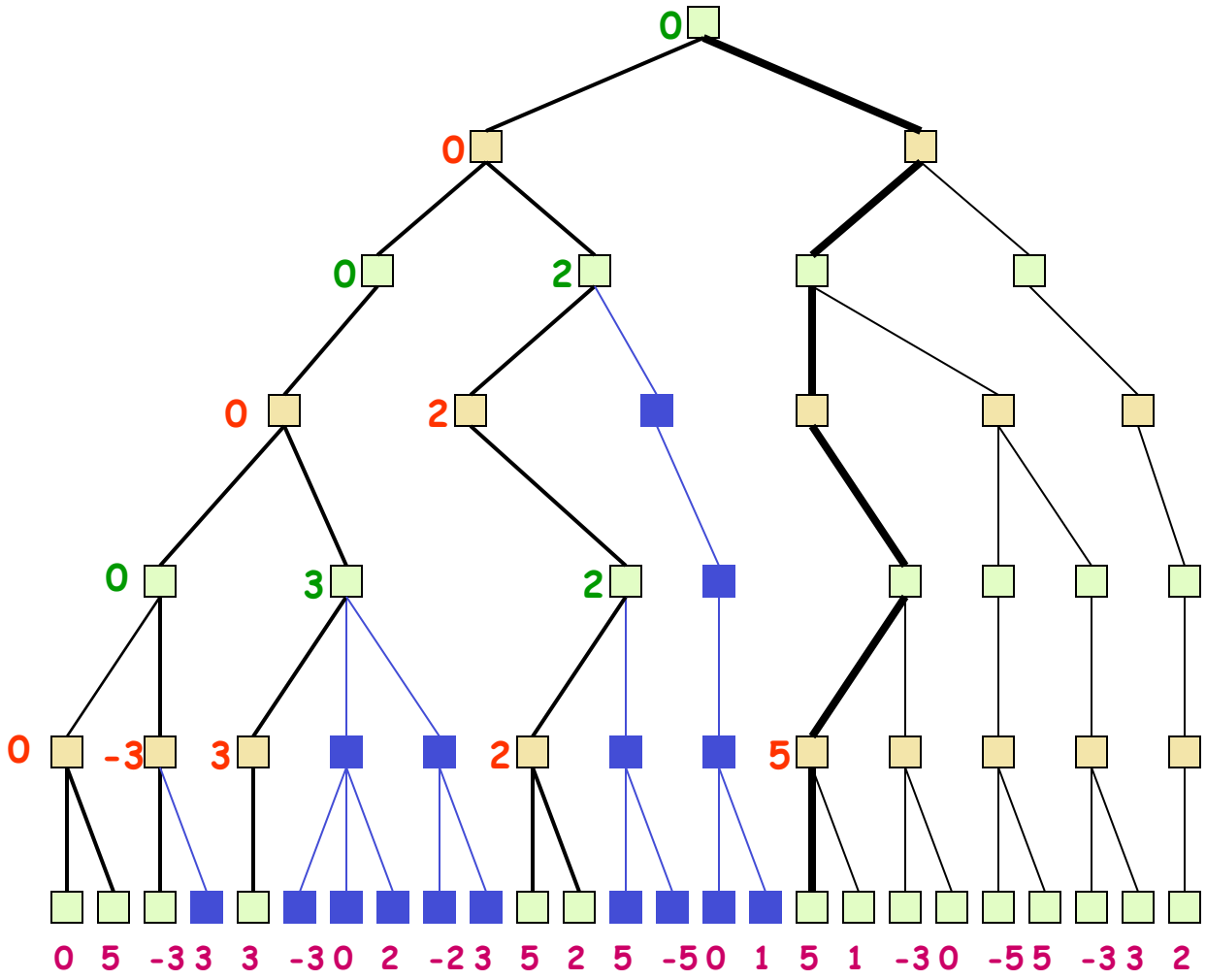

0
0
0
0
3
2
0
-3
3
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

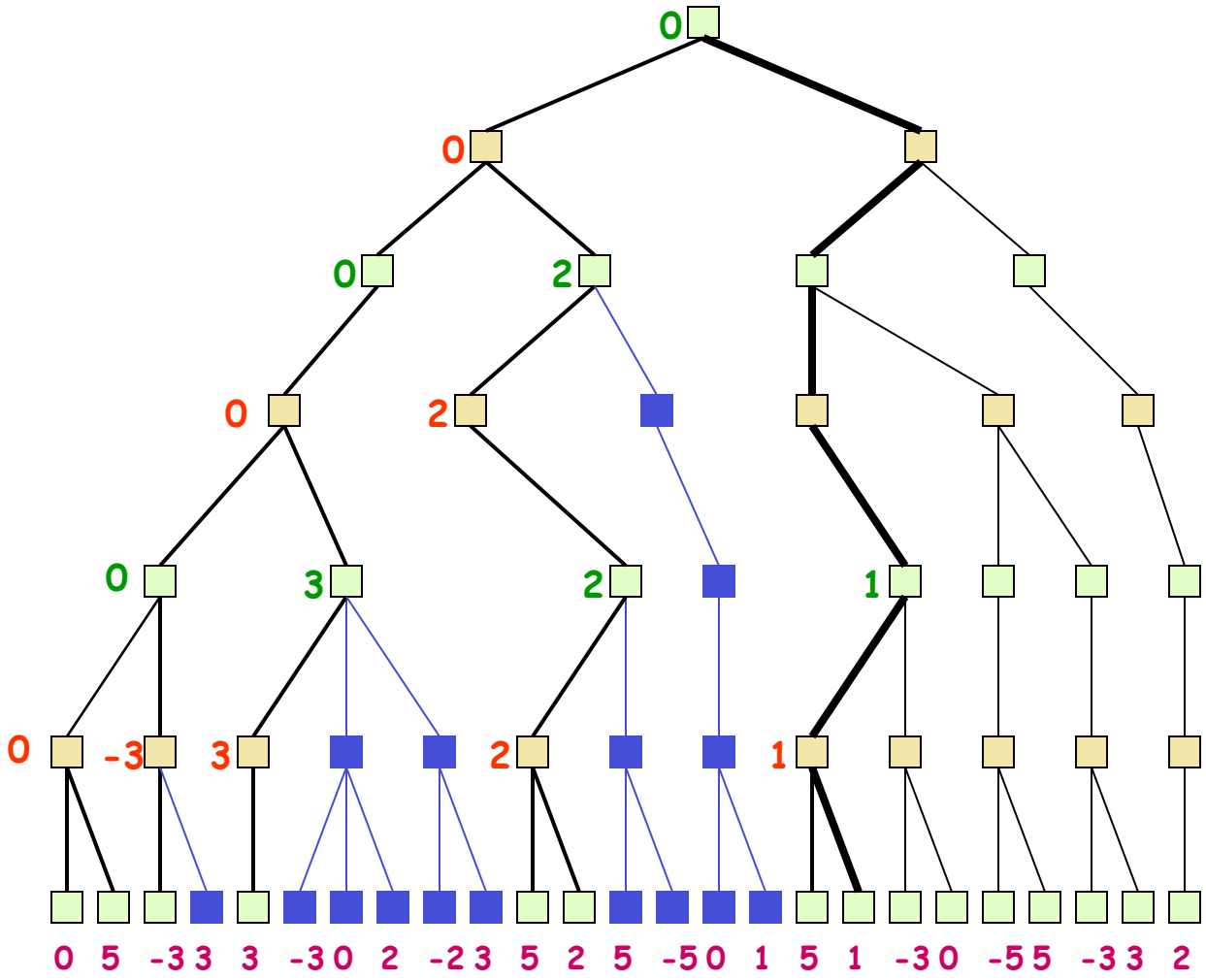

0
0
0
0
3
2
0
-3
3
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

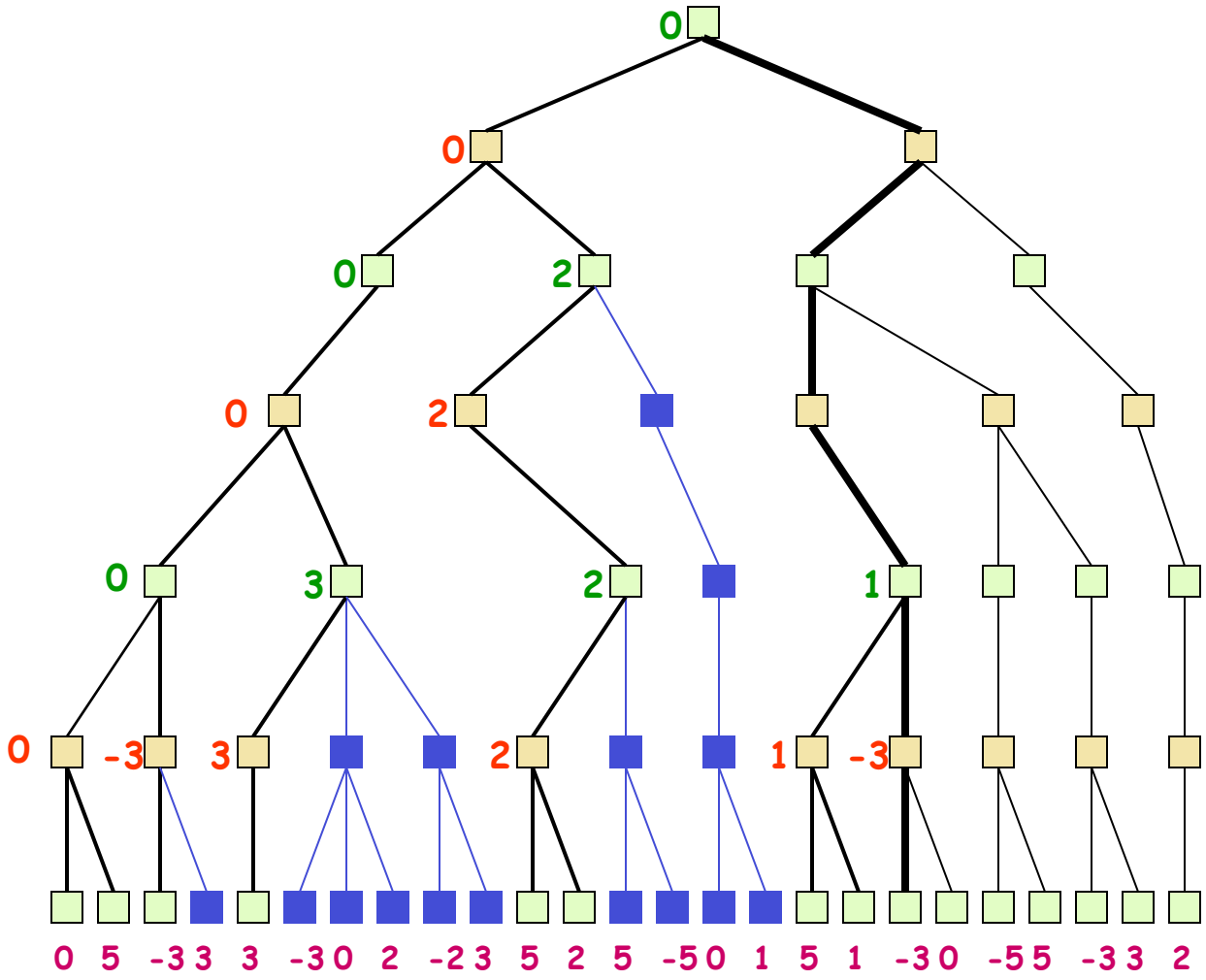

0
0
2
0
2
0
3
2
0
-3
3
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

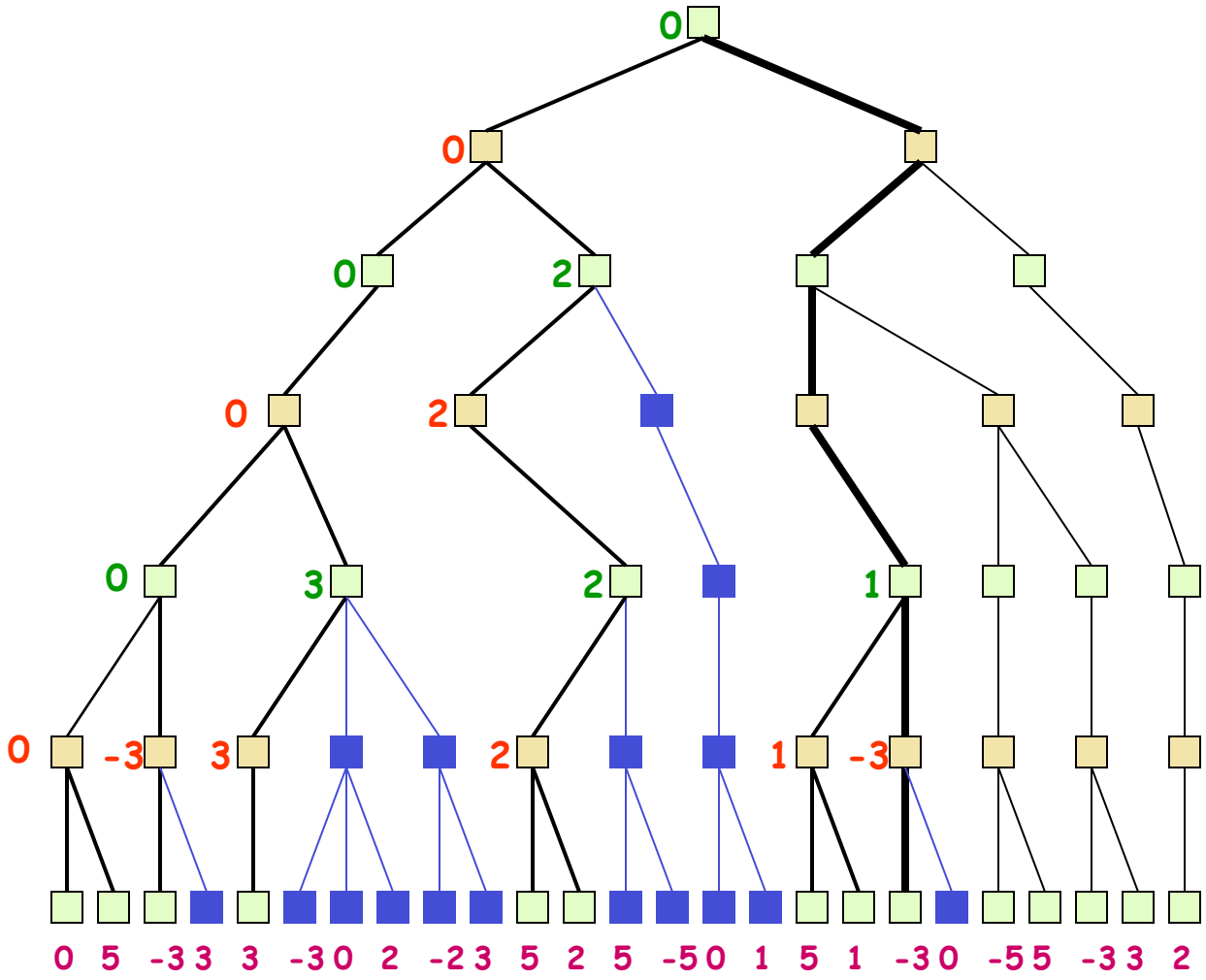

0
0
2
0
2
0
3
2
0
-3
3
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

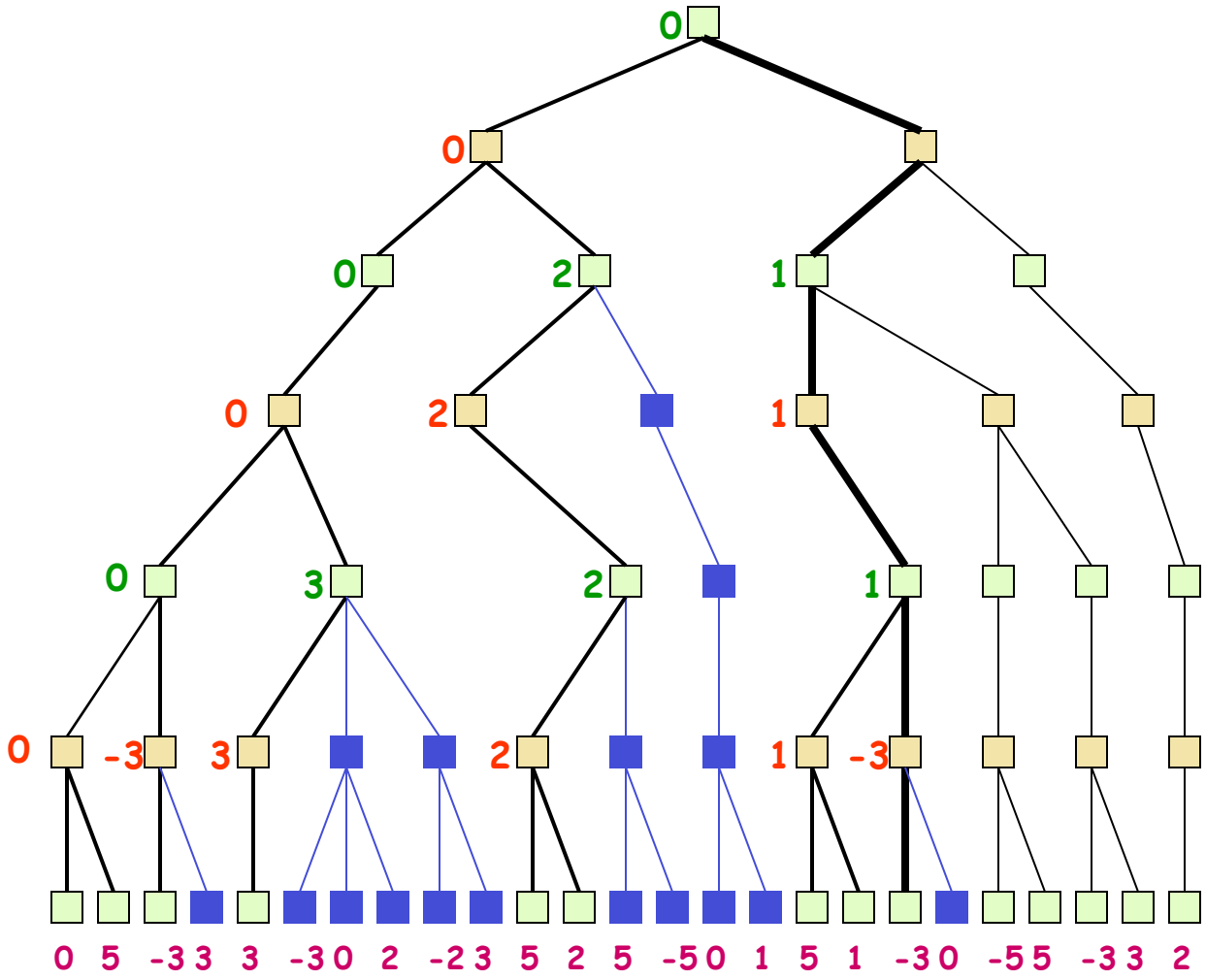

0
0
0
2
0
2
0
3
2
0
-3
3
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

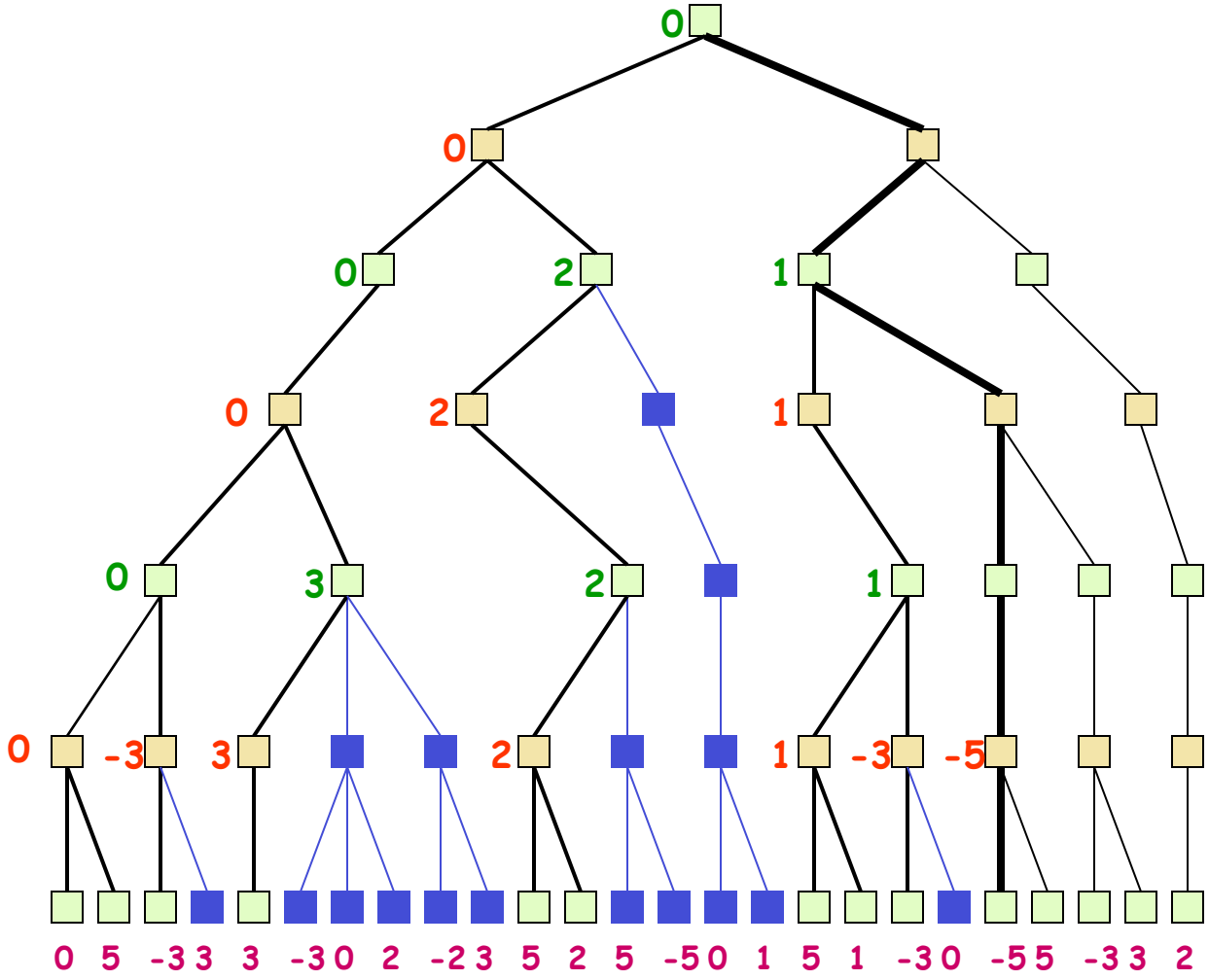

0
0
0
2
0
2
0
3
2
0
-3
3
2
5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

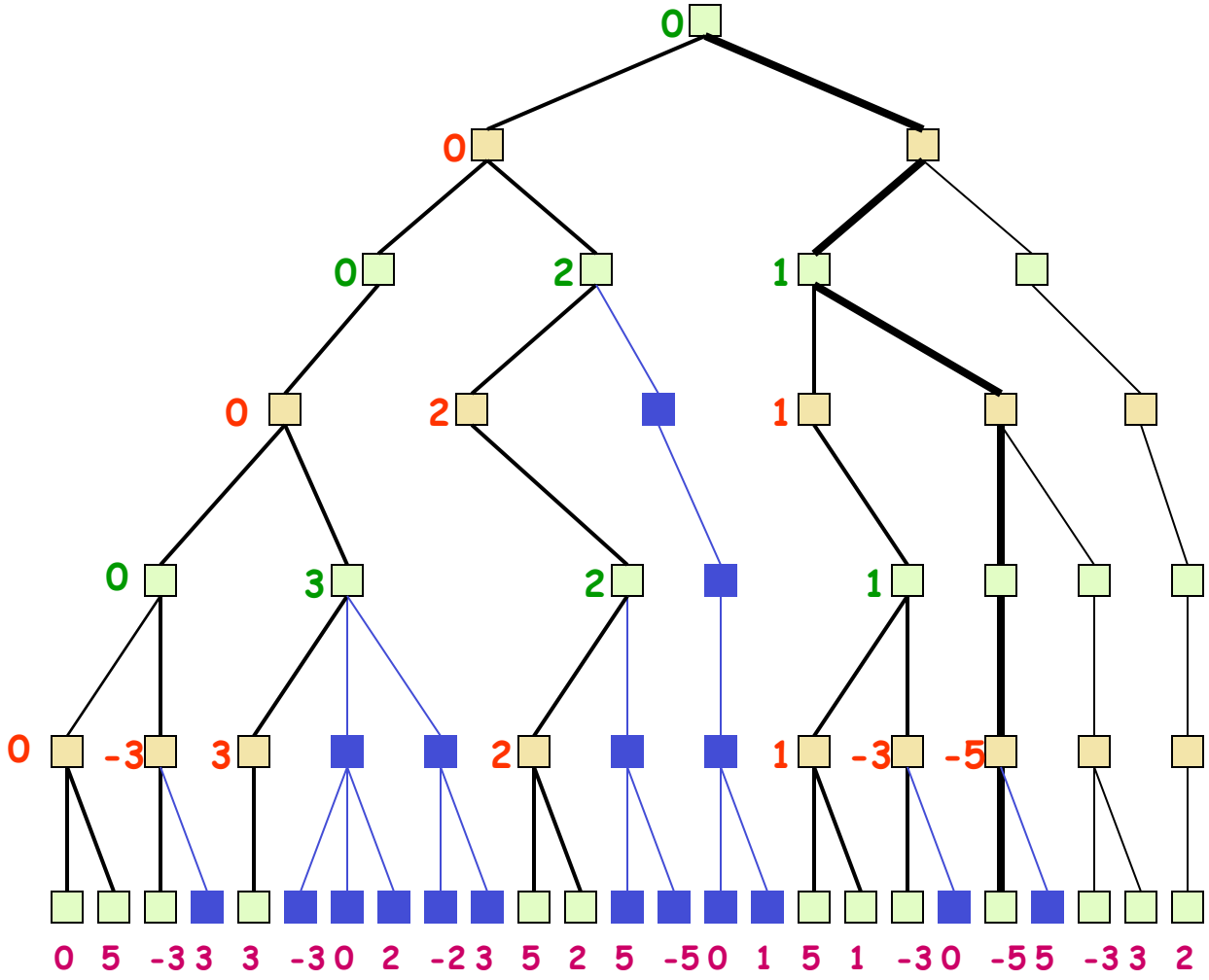

0
0
0
2
0
2
0
3
2
1
0
-3
3
2
1
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

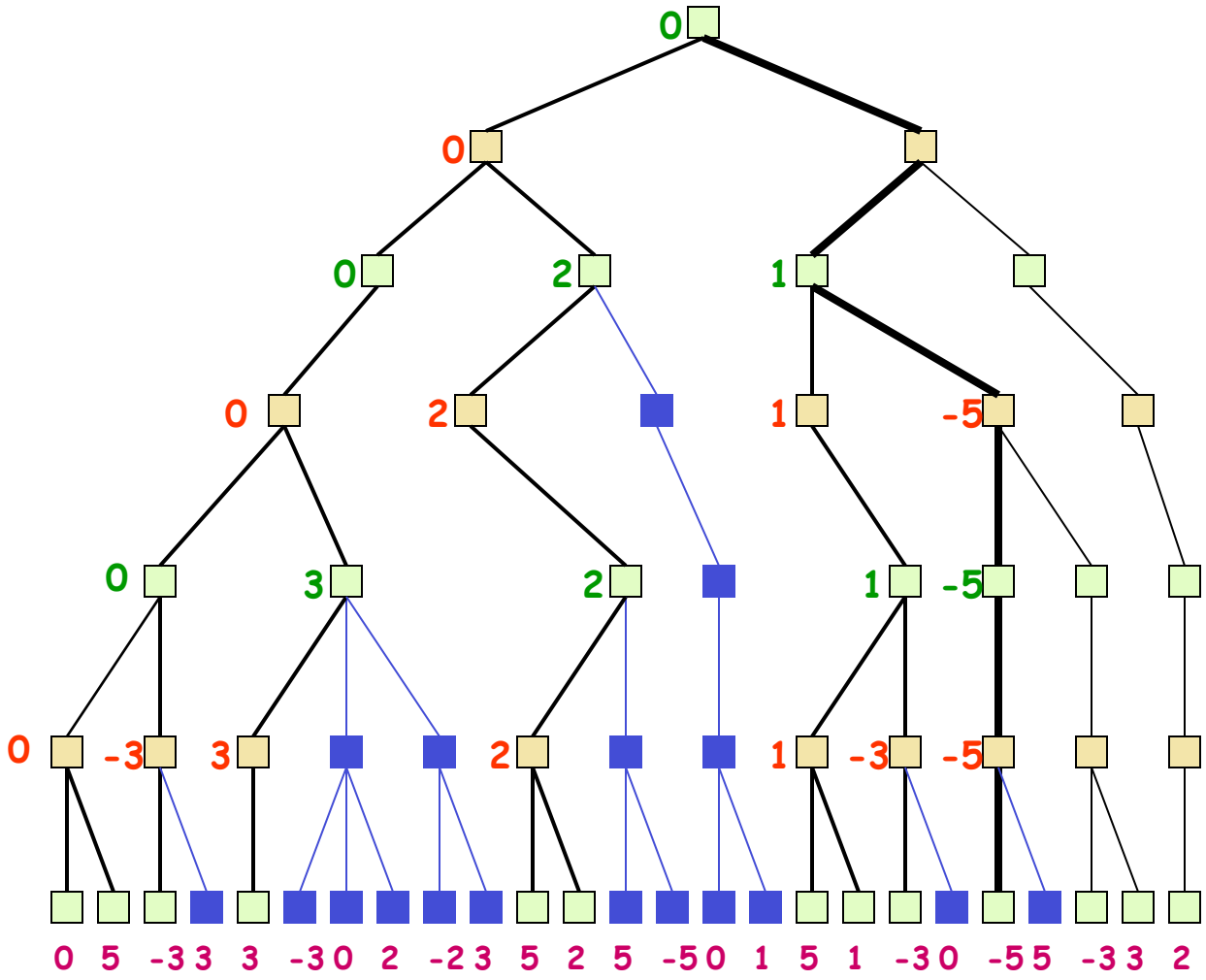

0
0
0
2
0
2
0
3
2
1
0
-3
3
2
1
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

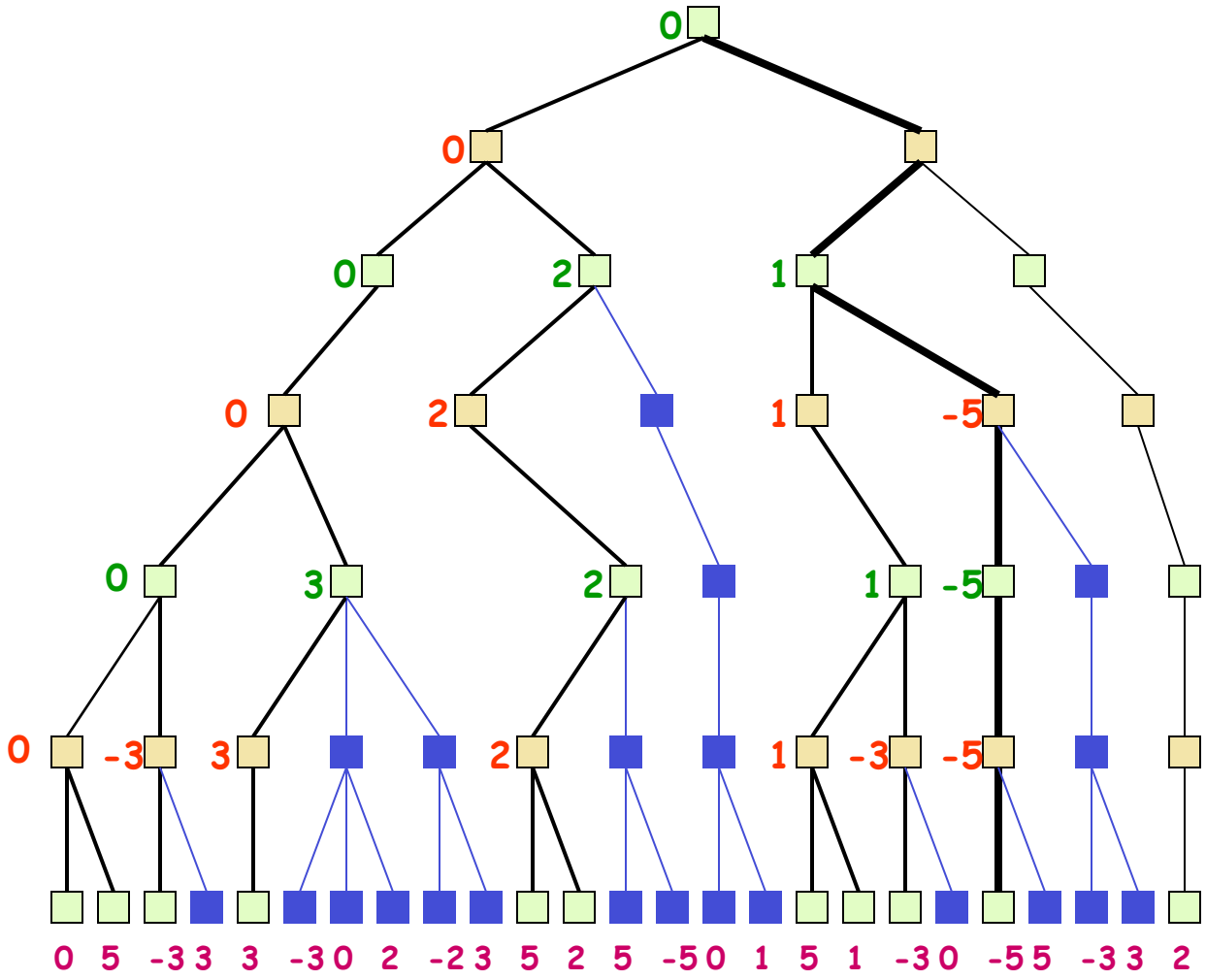

0
0
0
2
0
2
0
3
2
1
0
-3
3
2
1
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

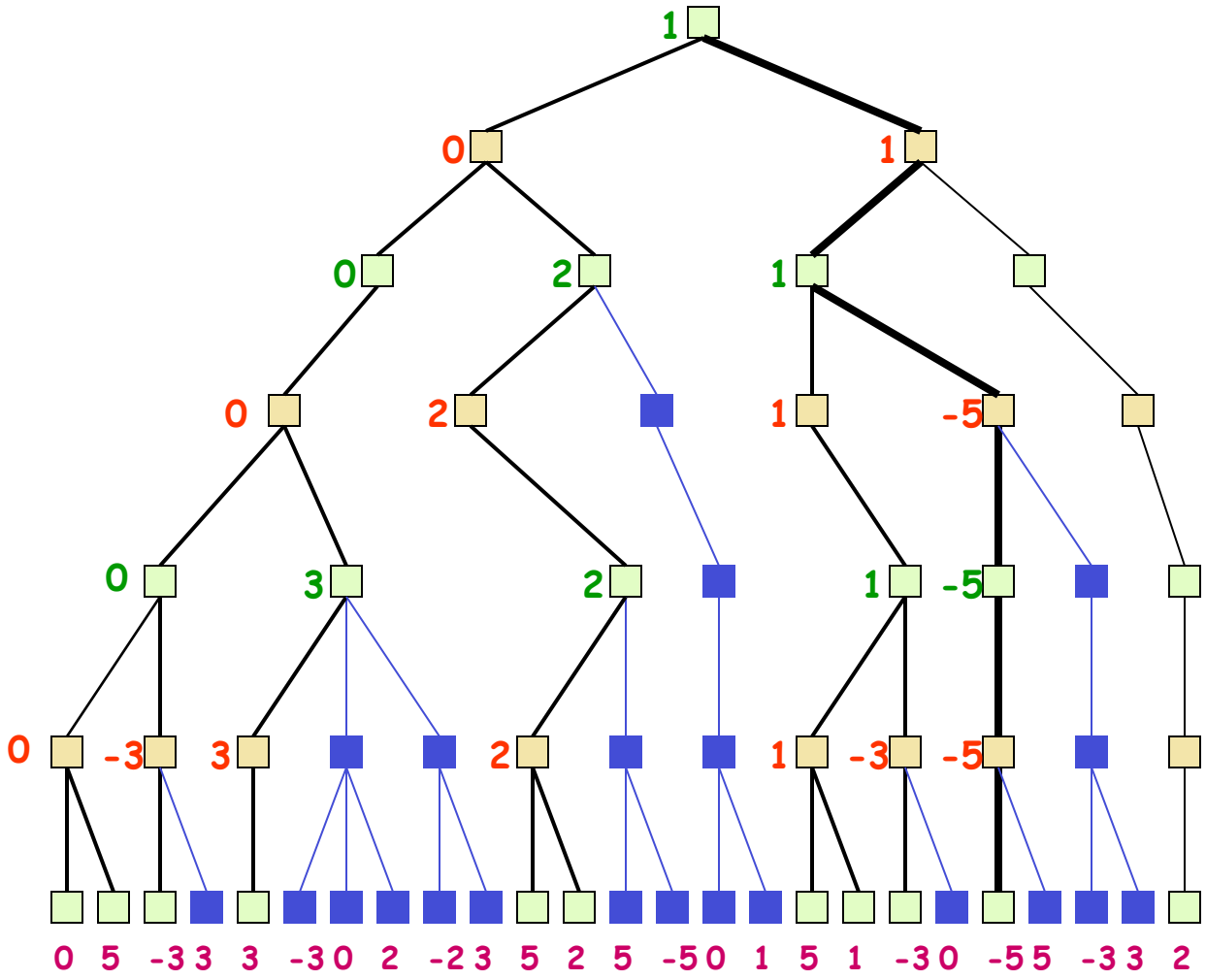

0
0
0
2
1
0
2
1
0
3
2
1
0
-3
3
2
1
-3
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

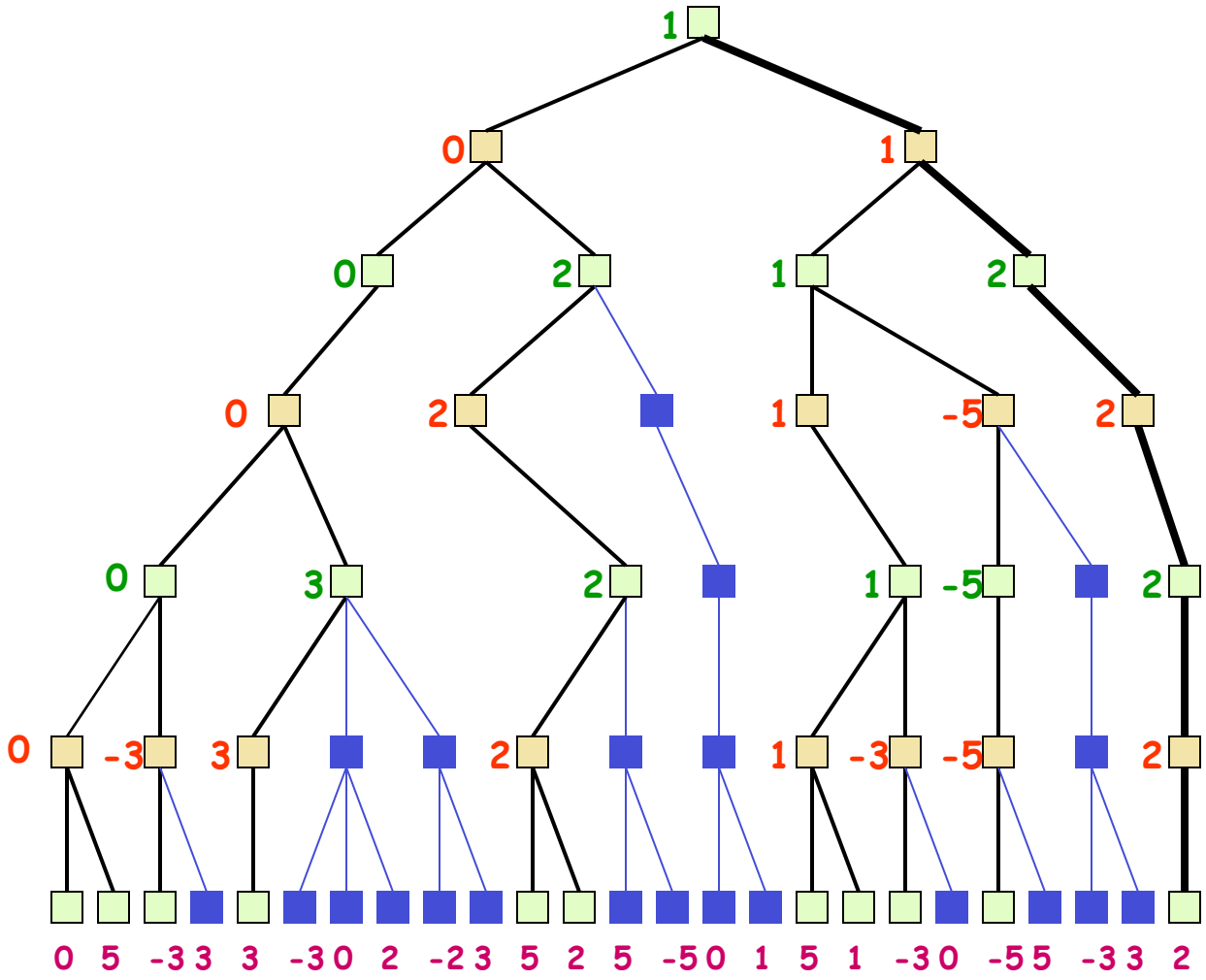

0
0
0
2
1
0
2
1
0
3
2
1
0
-3
3
2
1
-3
-5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

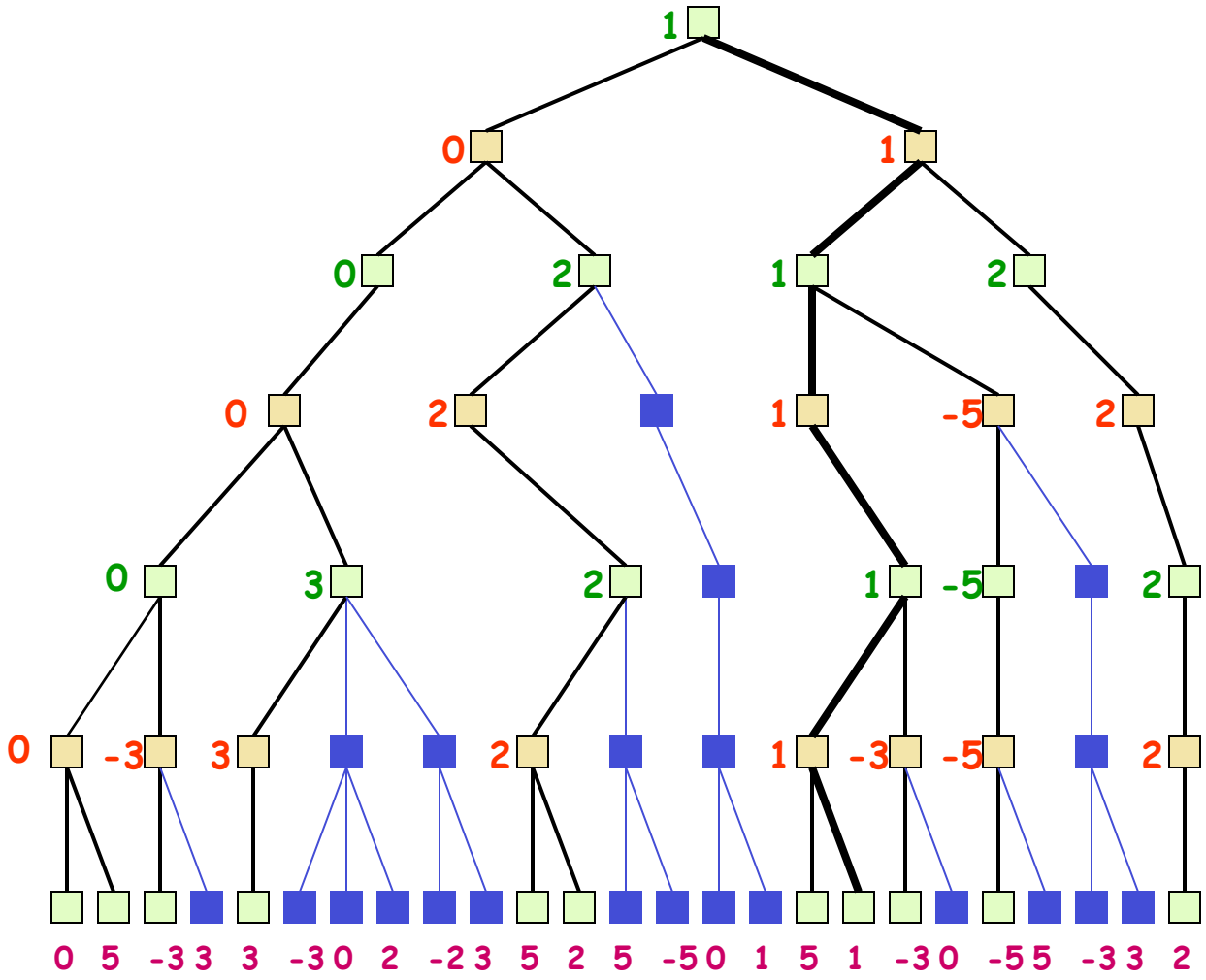

0
0
0 2
1
0 2 1
0 3 2 1
0 -3 3 2 1 -3 -5
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2


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


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

```
function MAX-VALUE (state, α, β)
;; α = best MAX so far; β = best MIN
if TERMINAL-TEST (state) then return
  UTILITY(state)
v := -∞
for each s in SUCCESSORS (state) do
    v := MAX (v, MIN-VALUE (s, α, β))
    if v >= β then return v
    α := MAX (α, v)
end
return v

function MIN-VALUE (state, α, β)
if TERMINAL-TEST (state) then return
  UTILITY(state)
v := ∞
for each s in SUCCESSORS (state) do
    v := MIN (v, MAX-VALUE (s, α, β))
    if v <= α then return v
    β := MIN (β, v)
end
return v
```

Alpha-beta algorithm

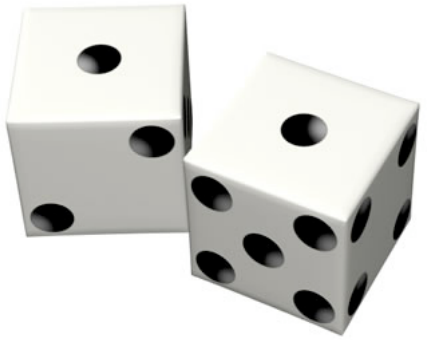
Effectiveness of alpha-beta

- Alpha-beta guaranteed to compute same value for root node as minimax, but with $\leq$ 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 generated
- In Deep Blue, alpha-beta pruning meant that average branching factor at each node was about six instead of about 35!

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 the "horizon effect"
- **Singular extension**: If a move is obviously better than the others in a node at horizon h, expand it
- Use **transposition tables** to deal with repeated states
- **Null-move** search: assume player forfeits move; do a shallow analysis of tree; result must surely be worse than if player had moved. Can be used to recognize moves that should be explored fully.

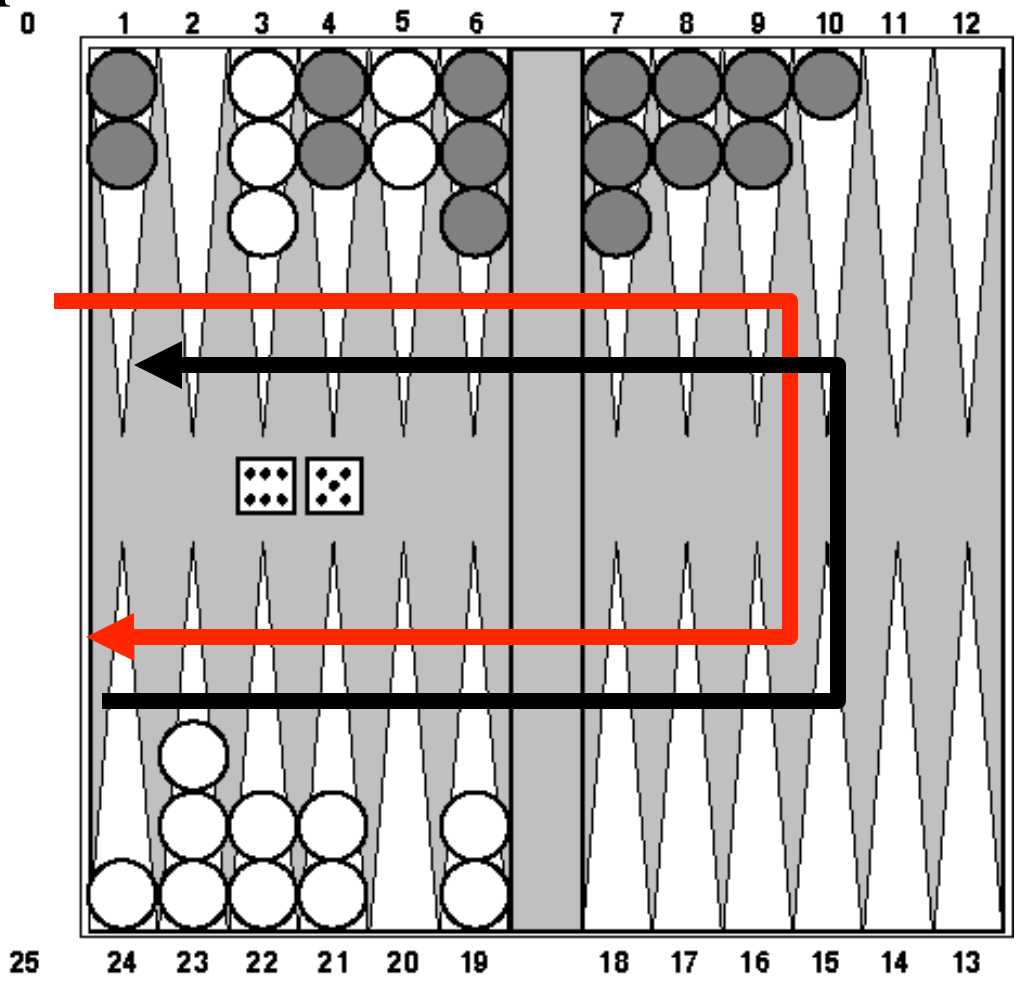
Stochastic Games



- 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

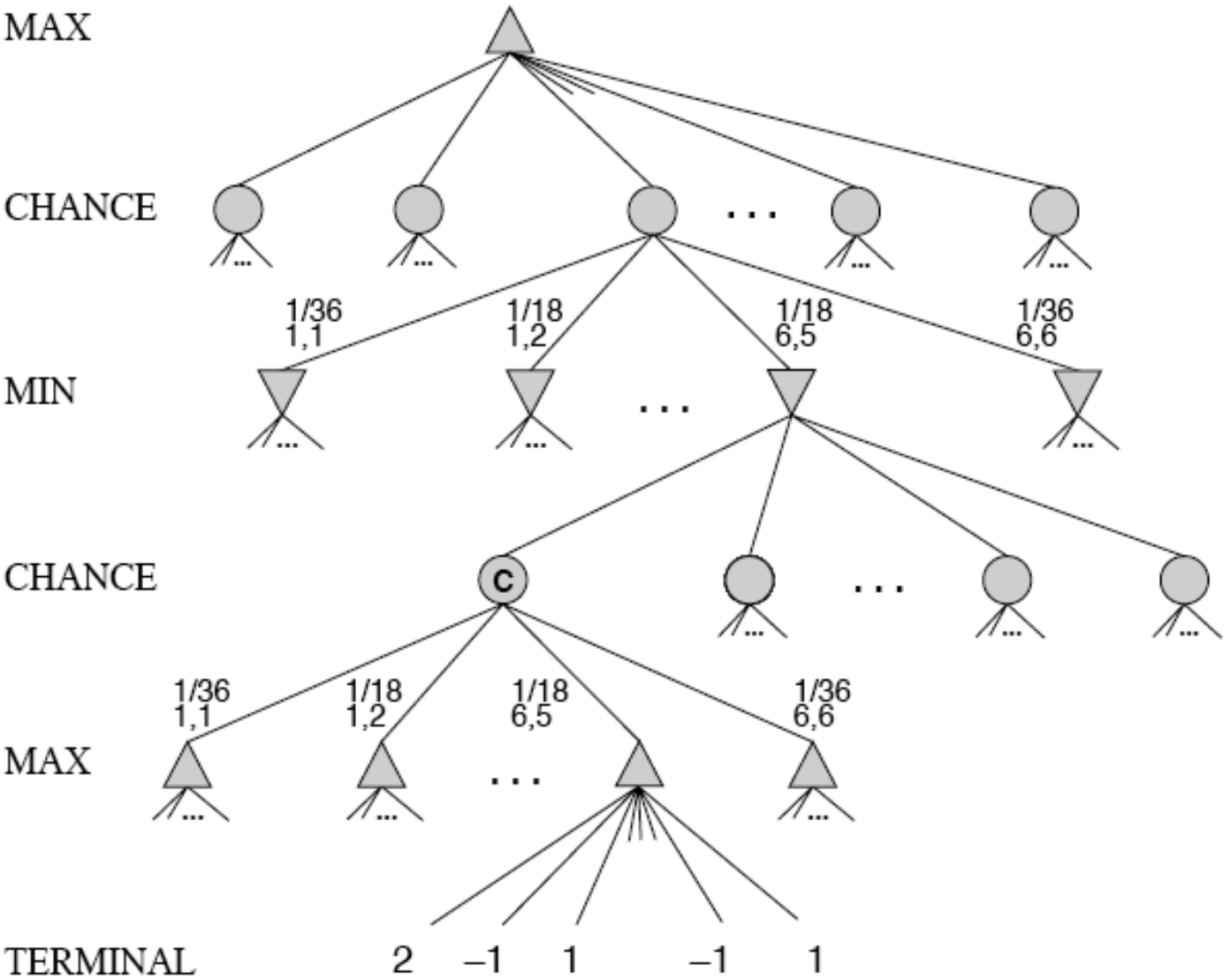
- Backgammon is a two-player game with **uncertainty**.
- Players roll dice to determine what moves to make.
- White has just rolled *5 and 6* and has four legal moves:
  - 5-10, 5-11
  - 5-11, 19-24
  - 5-10, 10-16
  - 5-11, 11-16
- Such games are good for exploring decision making in adversarial problems involving skill and luck



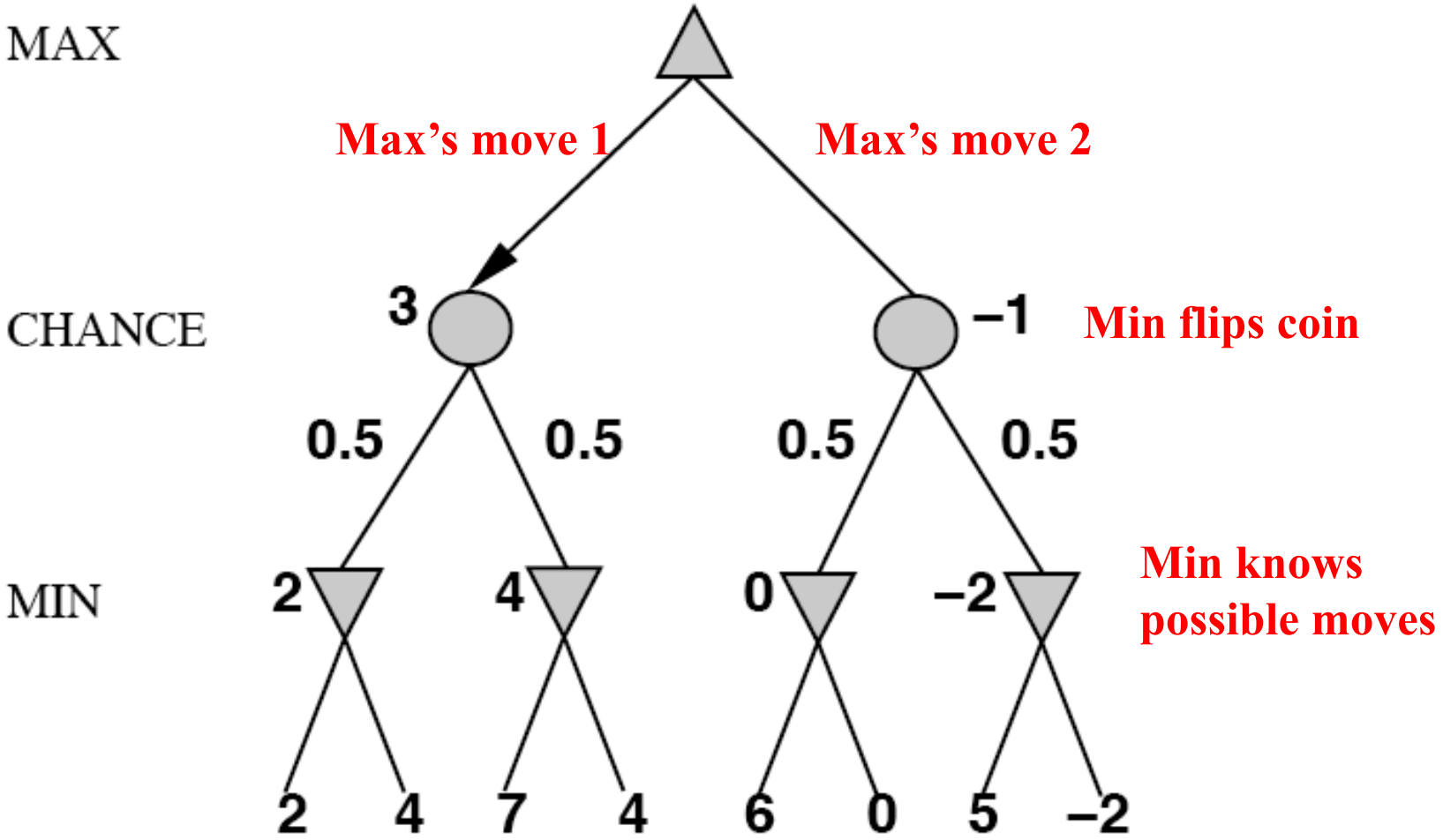
Why can’t we use MiniMax?

- Before a player chooses her move, she rolls dice and then knows exactly what they are
- And the immediate outcome of each move is also known
- But she does not know what moves her opponent will have available to choose from
- We need to adapt MiniMax to handle this

MiniMax trees with Chance Nodes



Understanding the notation



Board state includes chance outcome determining what moves are available

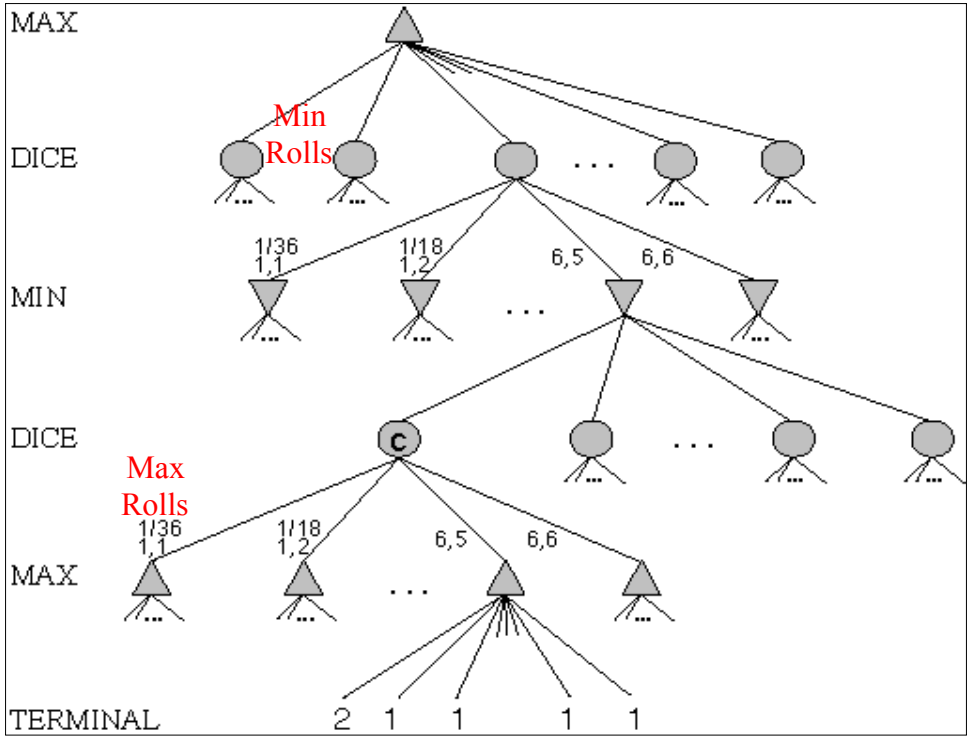
Game trees with chance nodes

- **Chance nodes** (circles) represent random events
- For a random event with N outcomes, chance node has N children; probability is associated with each
- 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) = $\sum_i(P(d_i)*maxval(i))$

- Chance nodes over min node:

expectimin(C) = $\sum_i(P(d_i)*minval(i))$

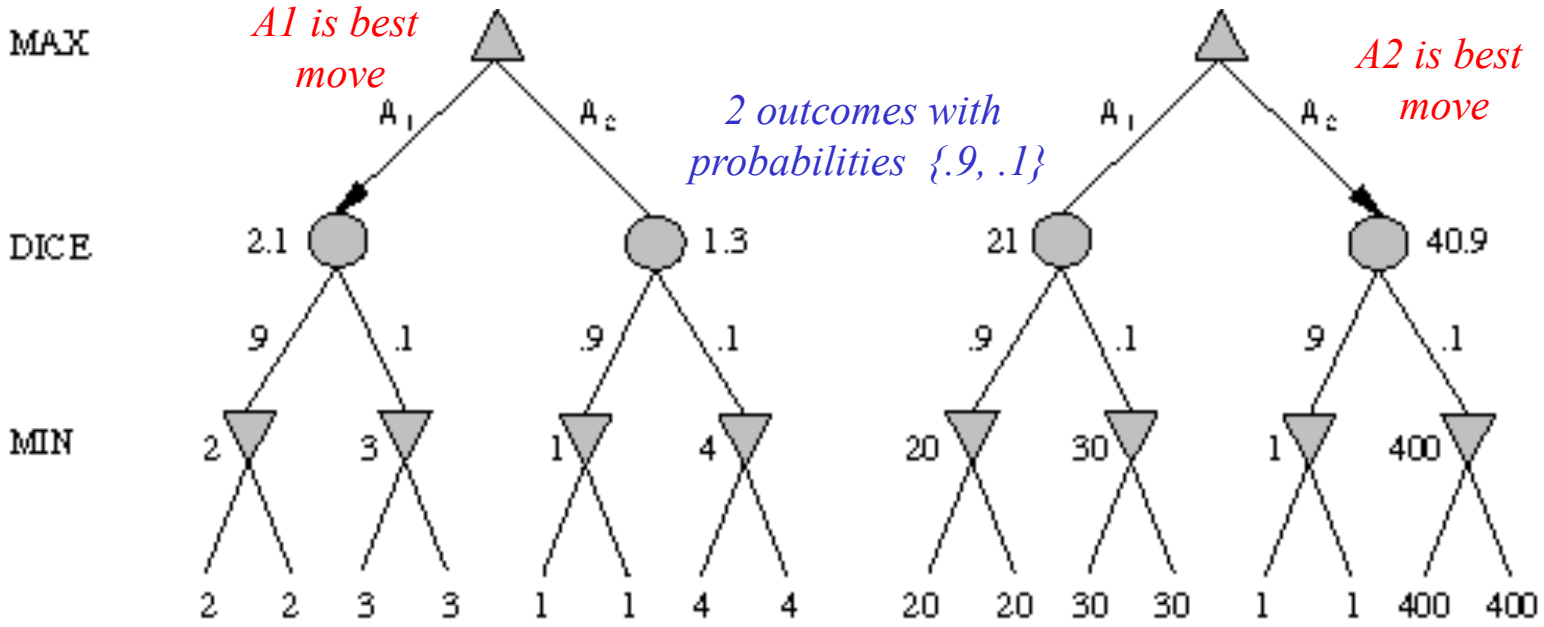


Impact on Lookahead

- Dice rolls increase branching factor
  - 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
  - value of lookahead 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 and expected values we must be careful about the *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

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

  1) generating 100 deals consistent with bidding information

  2) picking the action that wins most tricks on average

High-Performance Game Programs

- 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 endgame database of 444 billion board configurations
- Methods are general, but implementation is improved by many specifically tuned-up enhancements (e.g., the evaluation functions)

Other Issues

- Multi-player games
  - E.g., many card games like Hearts
- Multiplayer games with alliances
  - E.g., Risk
  - More on this when we discuss game theory
  - Good model for a social animal like humans, where we are always balancing cooperation and competition

AI and Games II

- AI also of interest to the video game industry
- Many games include ‘agents’ controlled by the game program that could be
  - Adversaries, in *first person shooter* games
  - Collaborators, in a virtual reality game
- Some game environments used as AI challenges
  - 2009 Mario AI Competition
  - Unreal Tournament bots

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

General Game Playing

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

General Game Playing

- Stanford's GGP is a Web-based system
- Complete, logical specification of many different games in terms of:
  - relational descriptions of states
  - legal moves and their effects
  - goal relations and their payoffs
- Management of matches between automated players and of competitions involving many players and games

GGP

- Input: logical description of a game in a custom game description language
- Game bots must
  - Learn how to play legally from description
  - Play well using general problem solving strategies
  - Improve using general machine learning techniques
- Yearly completions since 2005, $10K prize
- Java General Game Playing Base Package

GGP Peg Jumping Game

; http://games.stanford.edu/gamemaster/games-debug/peg.kif

(init (hole a c3 peg))

(init (hole a c4 peg))

…

(init (hole d c4 empty))

…

(<= (next (pegs ?x)) (does jumper (jump ?sr ?sc ?dr ?dc)) (true (pegs ?y))
(succ ?x ?y)) (<= (next (hole ?sr ?sc empty)) (does jumper (jump ?sr ?sc ?dr ?dc)))

…

(<= (legal jumper (jump ?sr ?sc ?dr ?dc)) (true (hole ?sr ?sc peg))
(true (hole ?dr ?dc empty)) (middle ?sr ?sc ?or ?oc ?dr ?dc) (true (hole ?or ?oc peg)))

…

(<= (goal jumper 100) (true (hole a c3 empty)) (true (hole a c4 empty))
(true (hole a c5 empty))

…

(succ s1 s2)

(succ s2 s3)

…

Tic-Tac-Toe in GDL

```
(role xplayer)
(role oplayer)
;; Initial State
(init (cell 1 1 b))
(init (cell 1 2 b))
...
(init (control xplayer))
;; Dynamic Components
(<= (next (cell ?m ?n x))
  (does xplayer (mark ?m ?n))
  (true (cell ?m ?n b)))

(<= (next (cell ?m ?n o))
  (does oplayer (mark ?m ?n))
  (true (cell ?m ?n b)))
```

```
…
...
(<= (next (control xplayer))
  (true (control oplayer)))
(<= (legal ?w (mark ?x ?y))
  (true (cell ?x ?y b))
  (true (control ?w)))
(<= (legal xplayer noop)
  (true (control oplayer)))
(<= (legal oplayer noop)
  (true (control xplayer)))
...
(<= (goal xplayer 100) (line x))
...
(<= (goal oplayer 0) (line x))
...
```

See ggp-base repository: http://bit.ly/RB49q5

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