Adversarial Search

Chapter 5

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.

What Kinds of Games?

Mainly games of strategy with the following characteristics:

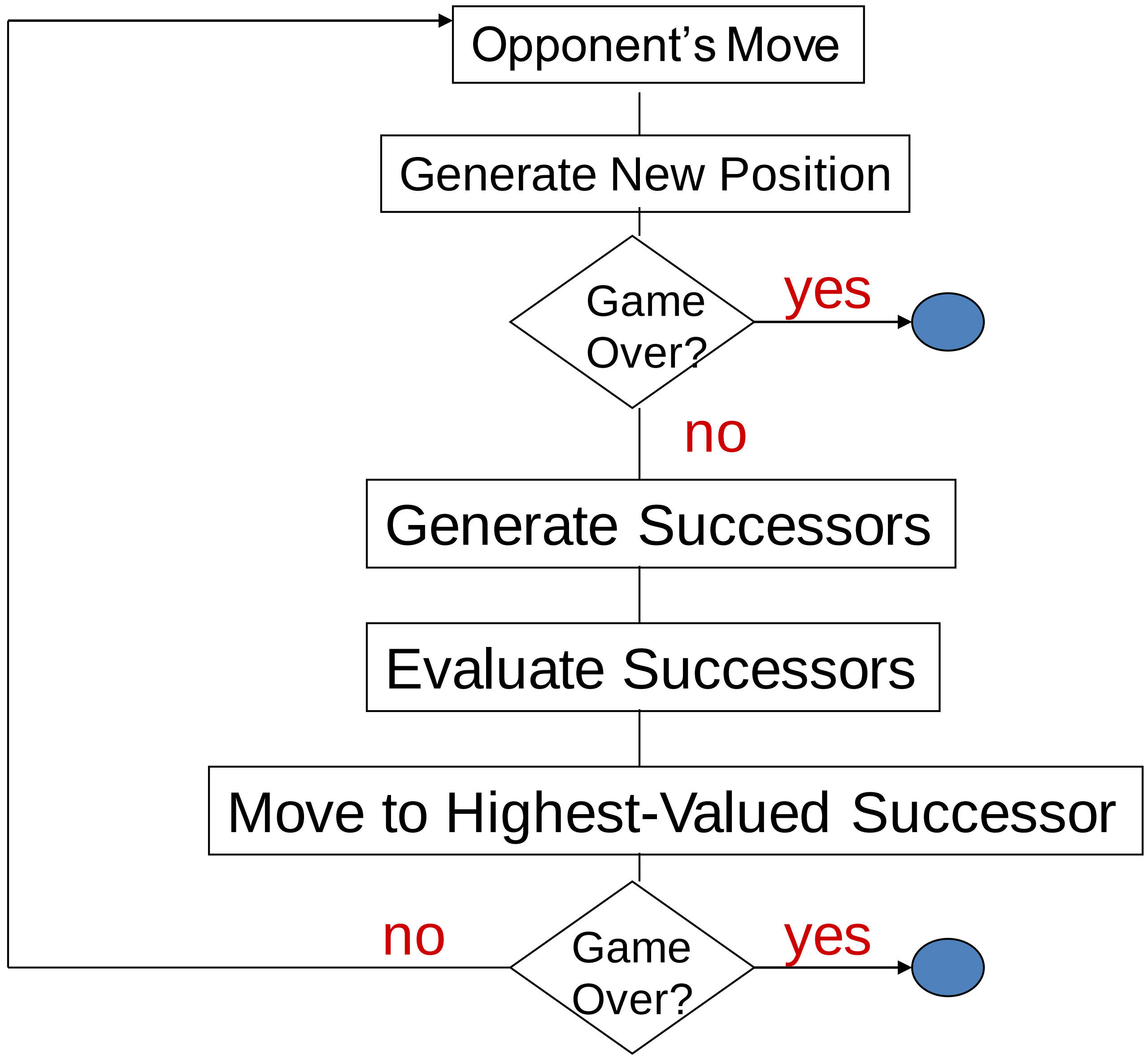
1. Sequence of moves to play
2. Rules that specify possible moves
3. Rules that specify a payment for each move
4. Objective is to maximize your payment

Games vs. Search Problems

- Unpredictable opponent → specifying a move for every possible opponent reply

- Time limits → unlikely to find goal, must approximate

Two-Player Game



Games as Adversarial Search

- States:
  - board configurations
- Initial state:
  - the board position and which player will move
- Successor function:
  - returns list of (move, state) pairs, each indicating a legal move and the resulting state
- Terminal test:
  - determines when the game is over
- Utility function:
  - gives a numeric value in terminal states

  (e.g., -1, 0, +1 for loss, tie, win)

Game Tree (2-player, Deterministic, Turns)

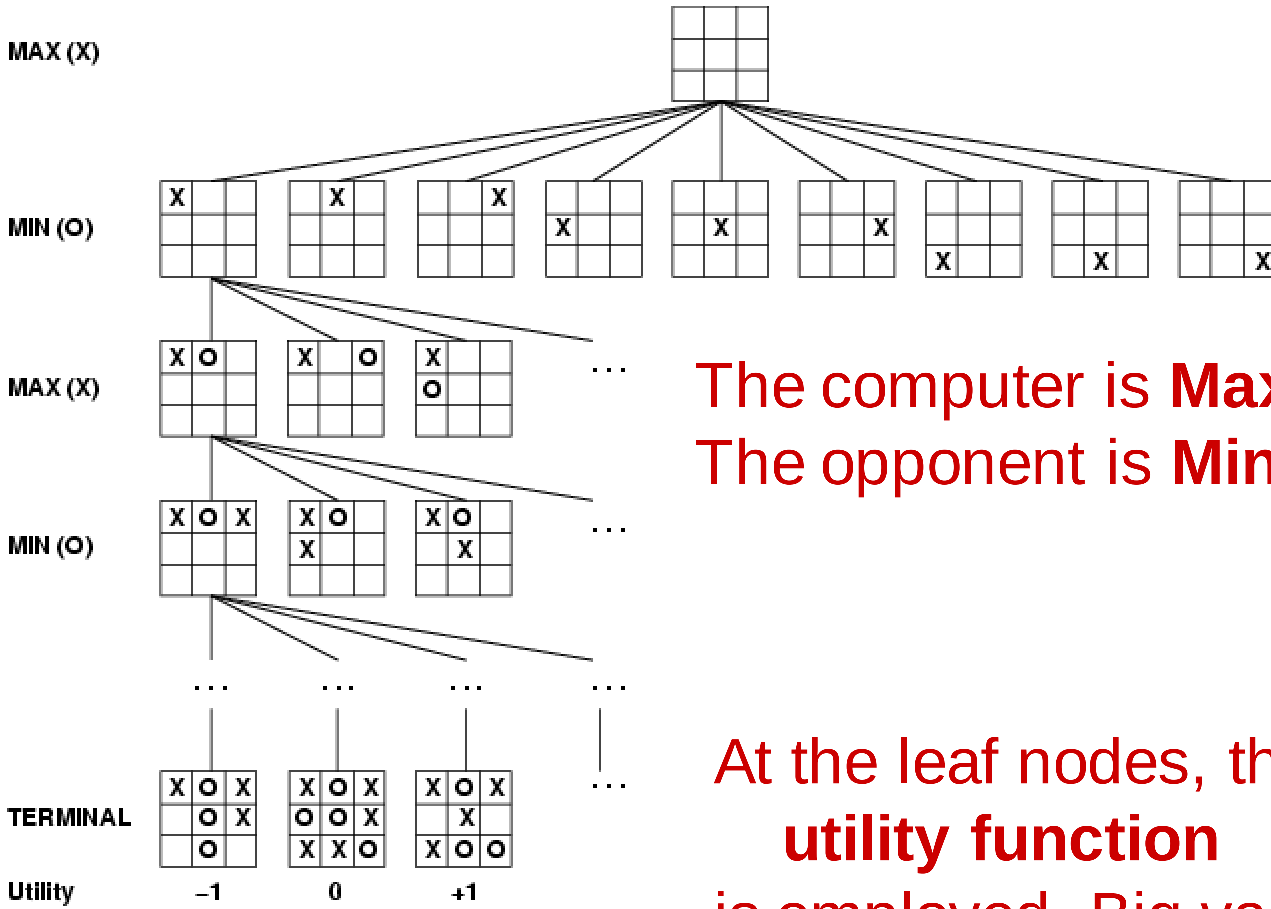
computer's turn

opponent's turn

computer's turn

opponent's turn

leaf nodes are evaluated



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

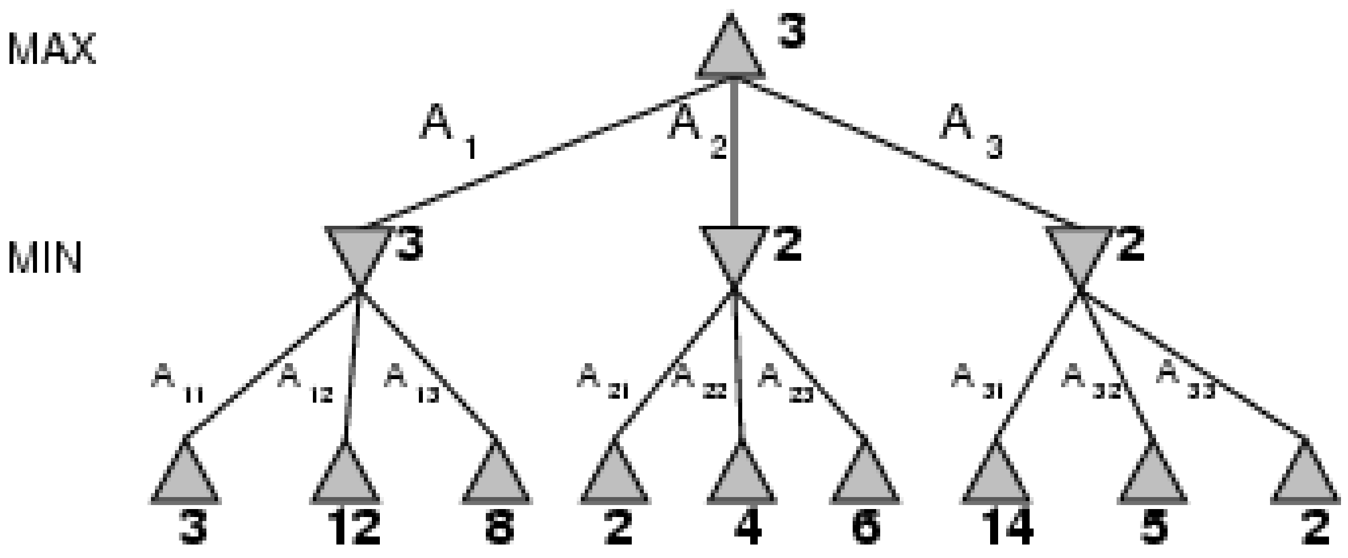
At the leaf nodes, the **utility function** is employed. Big value means good, small is bad.

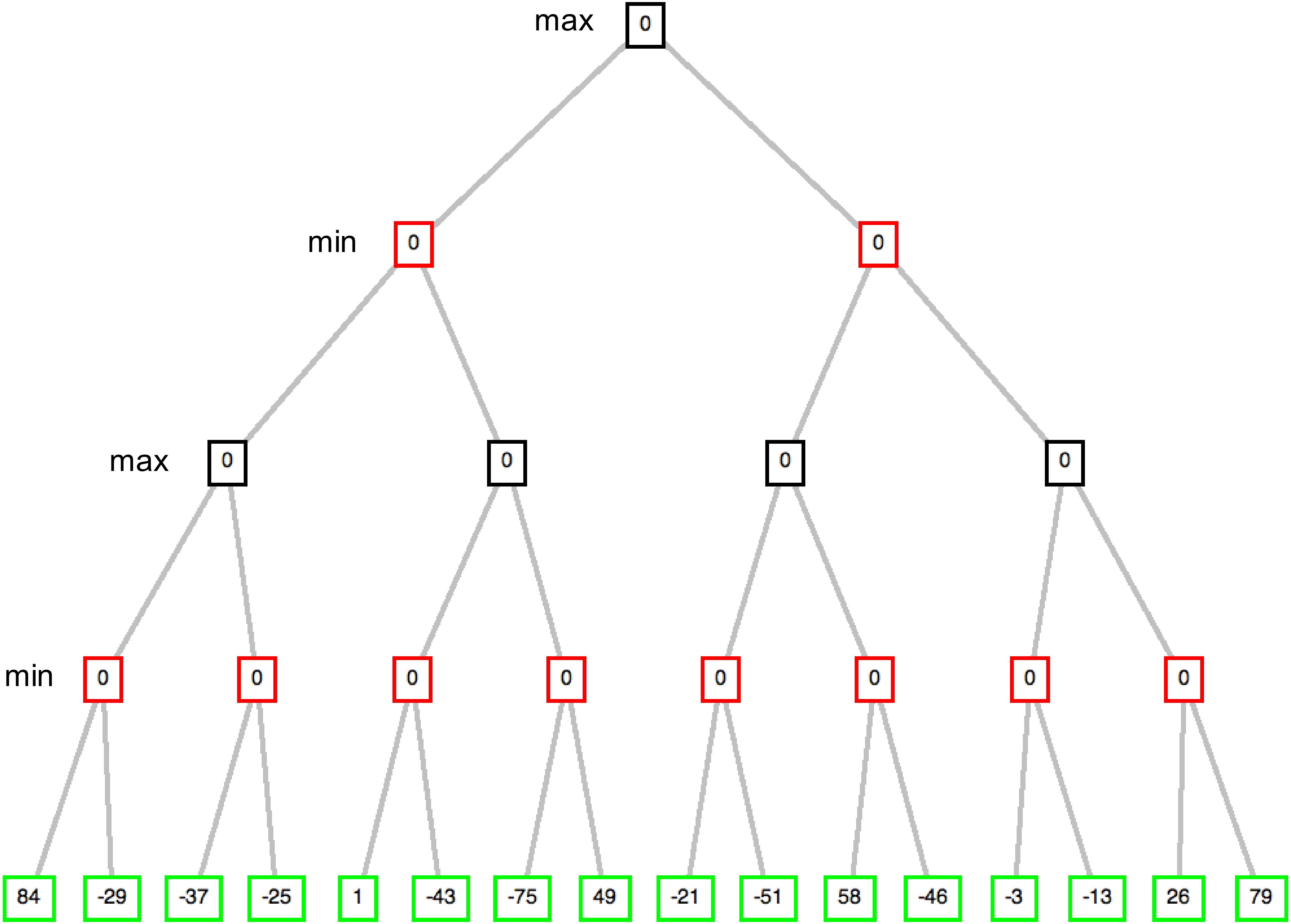
Mini-Max Terminology

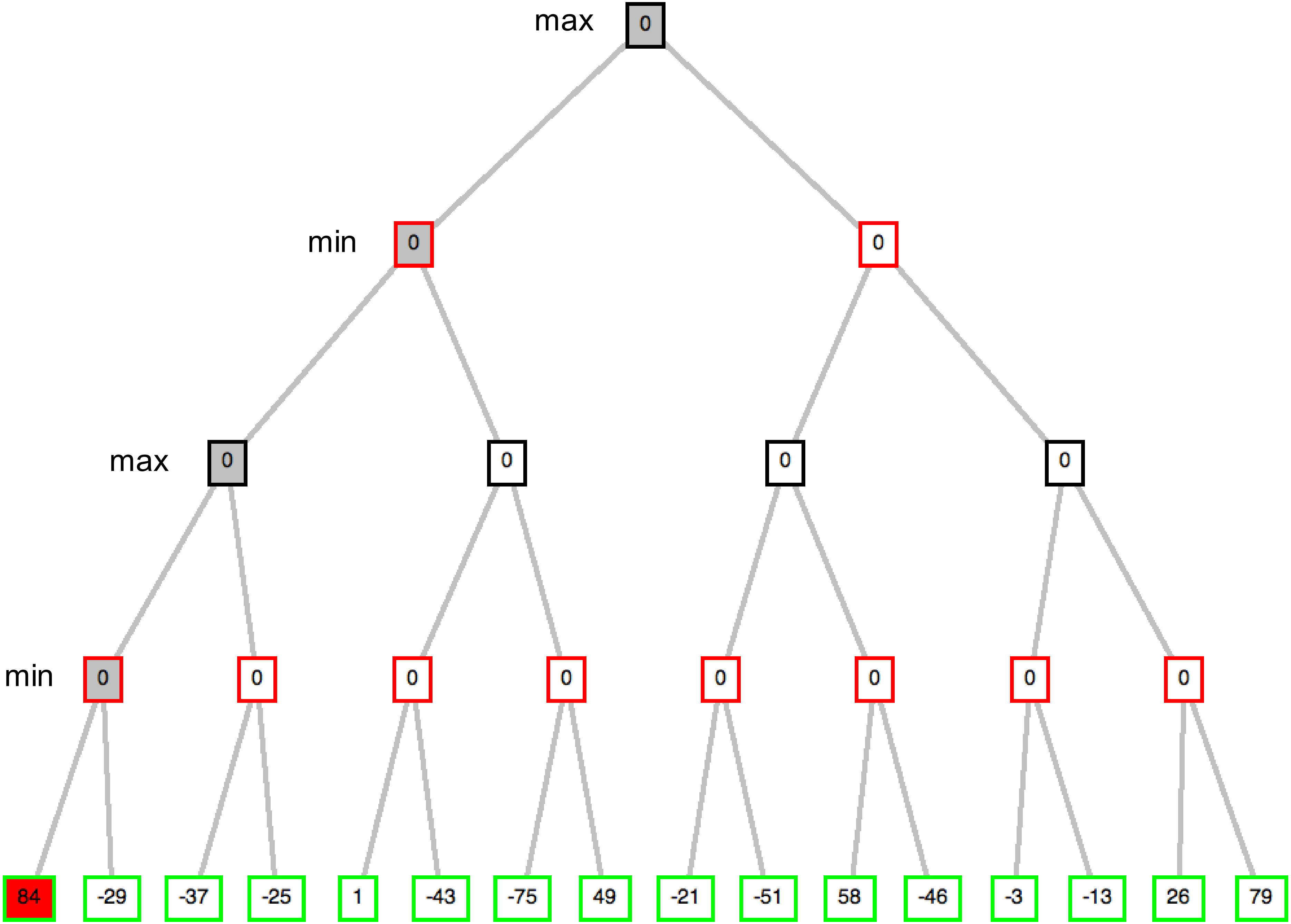
- move: a move by both players
- ply: a half-move
- utility function: the function applied to leaf nodes
- backed-up value
  - of a max-position: the value of its largest successor
  - of a min-position: the value of its smallest successor
- minimax procedure: search down several levels; at the bottom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

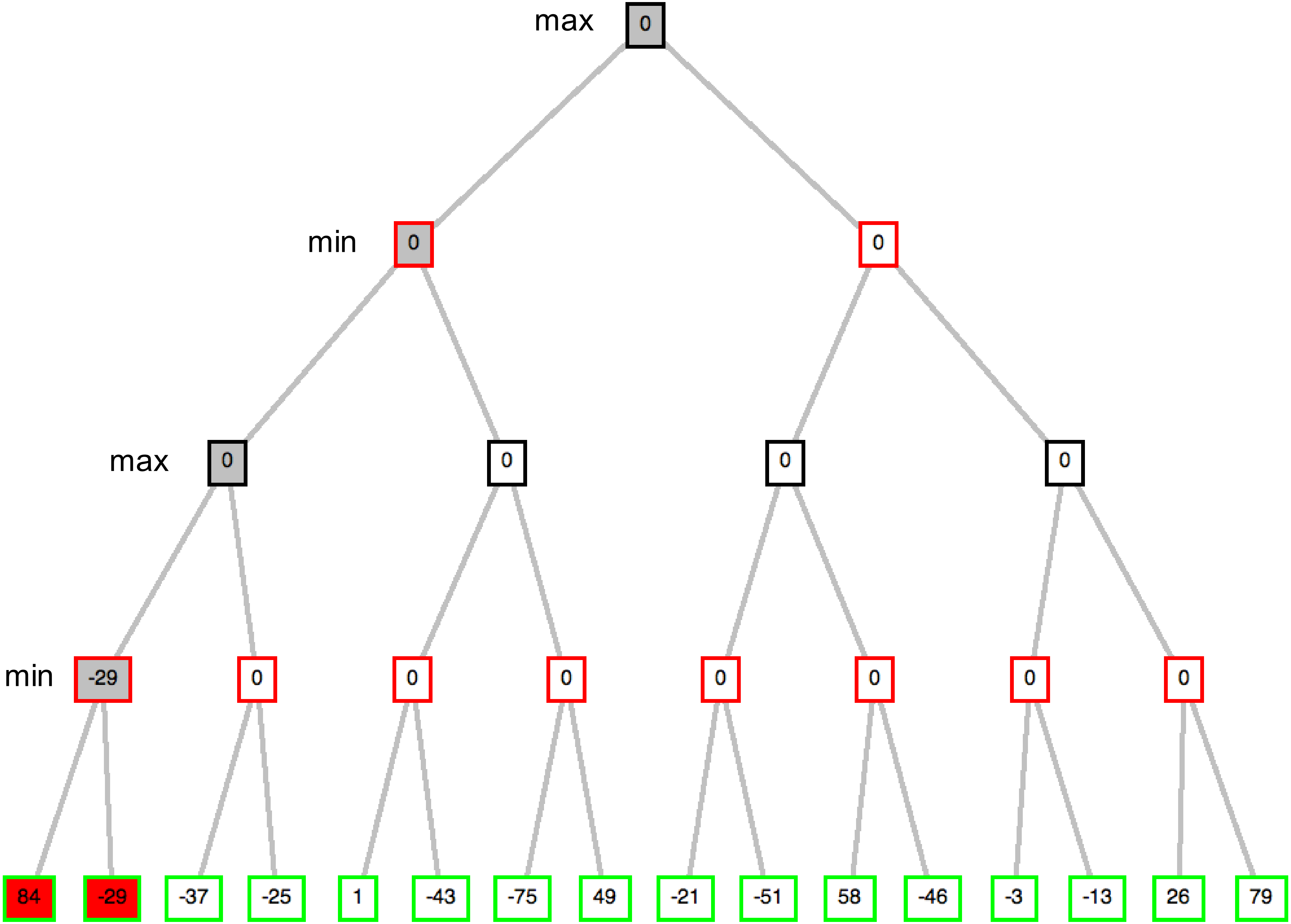
Minimax

- 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:

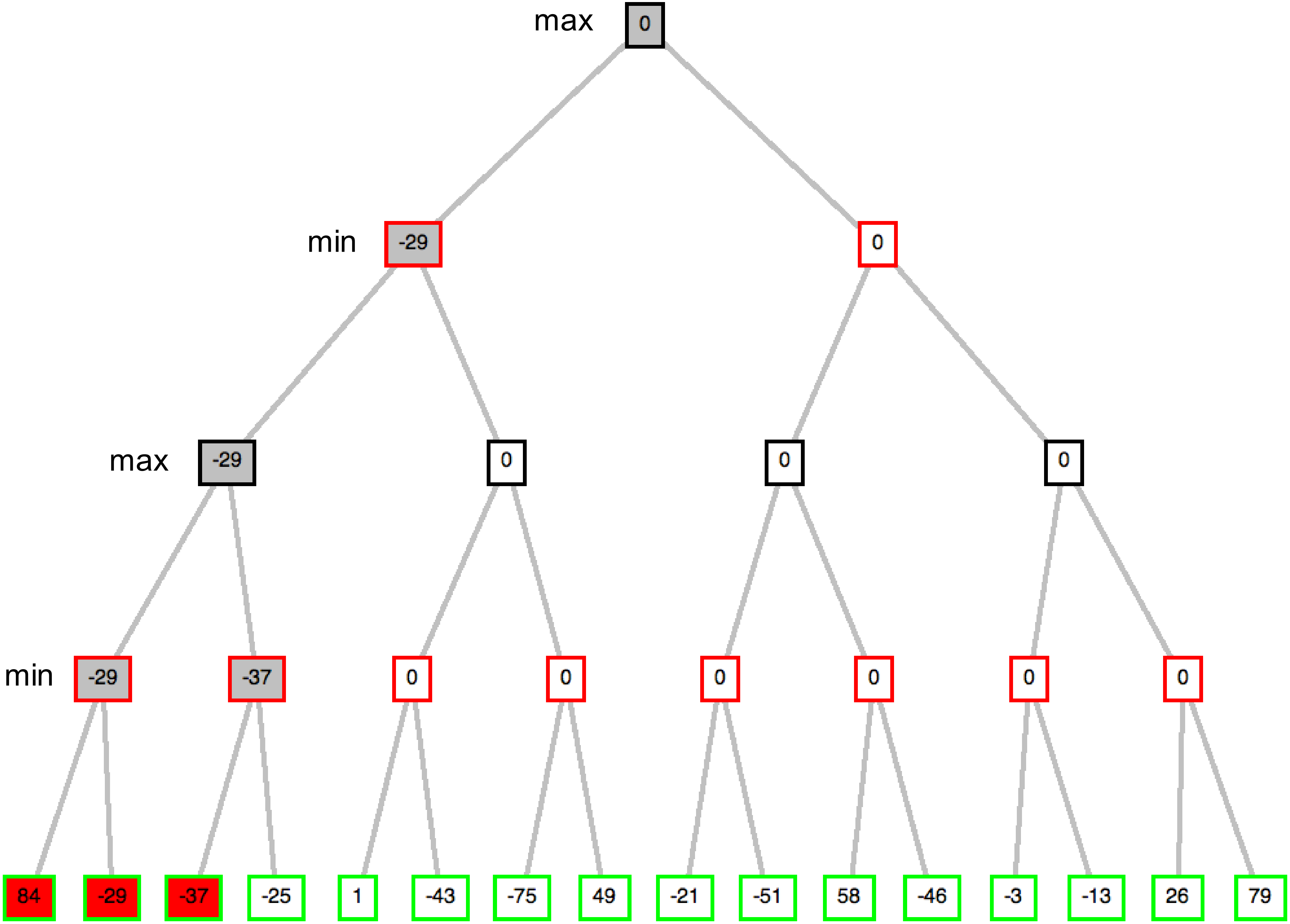


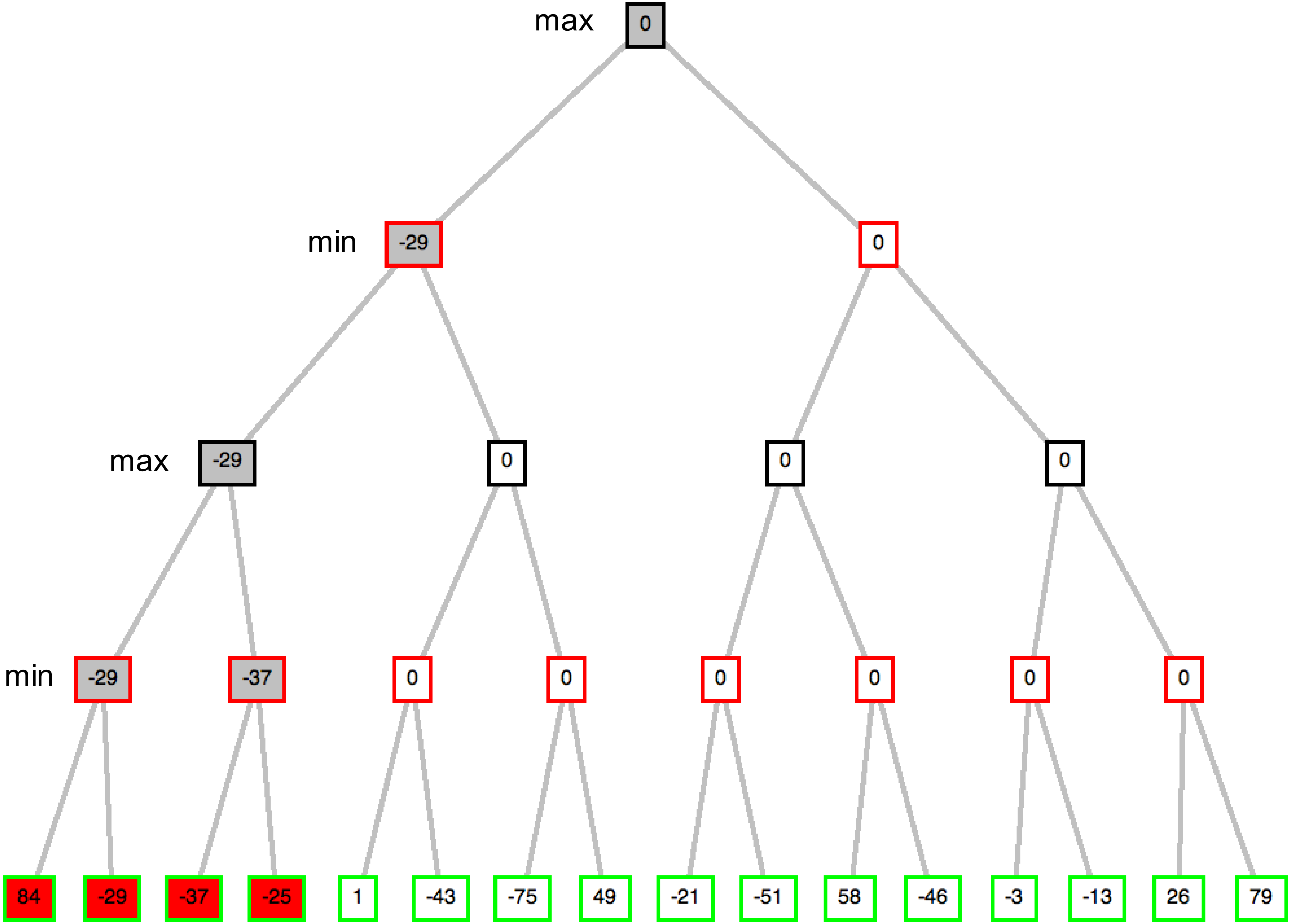

max
0
min
0
0
max
0
0
0
0
min
0
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

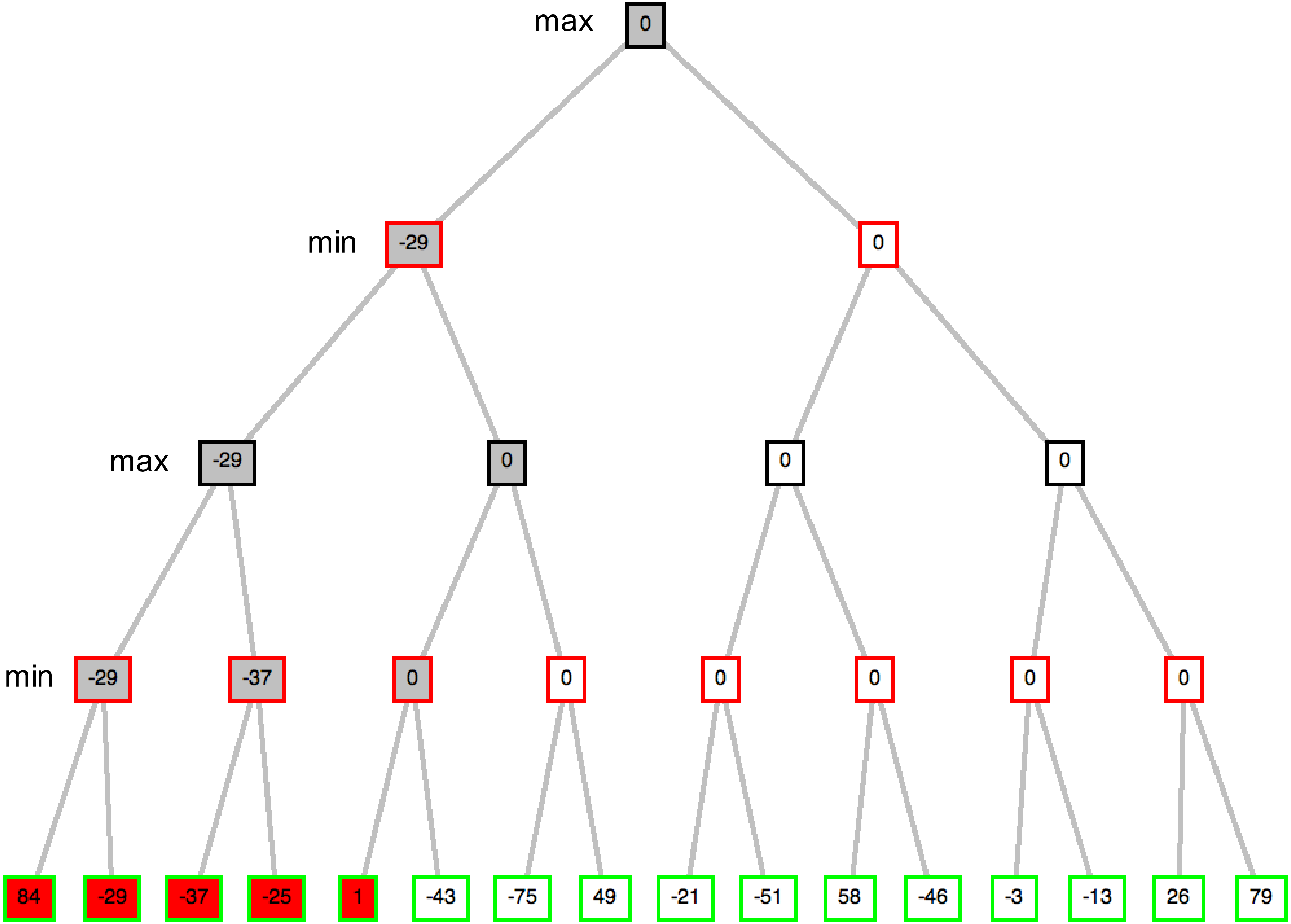

max
0
min
0
0
max
0
0
0
0
min
0
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

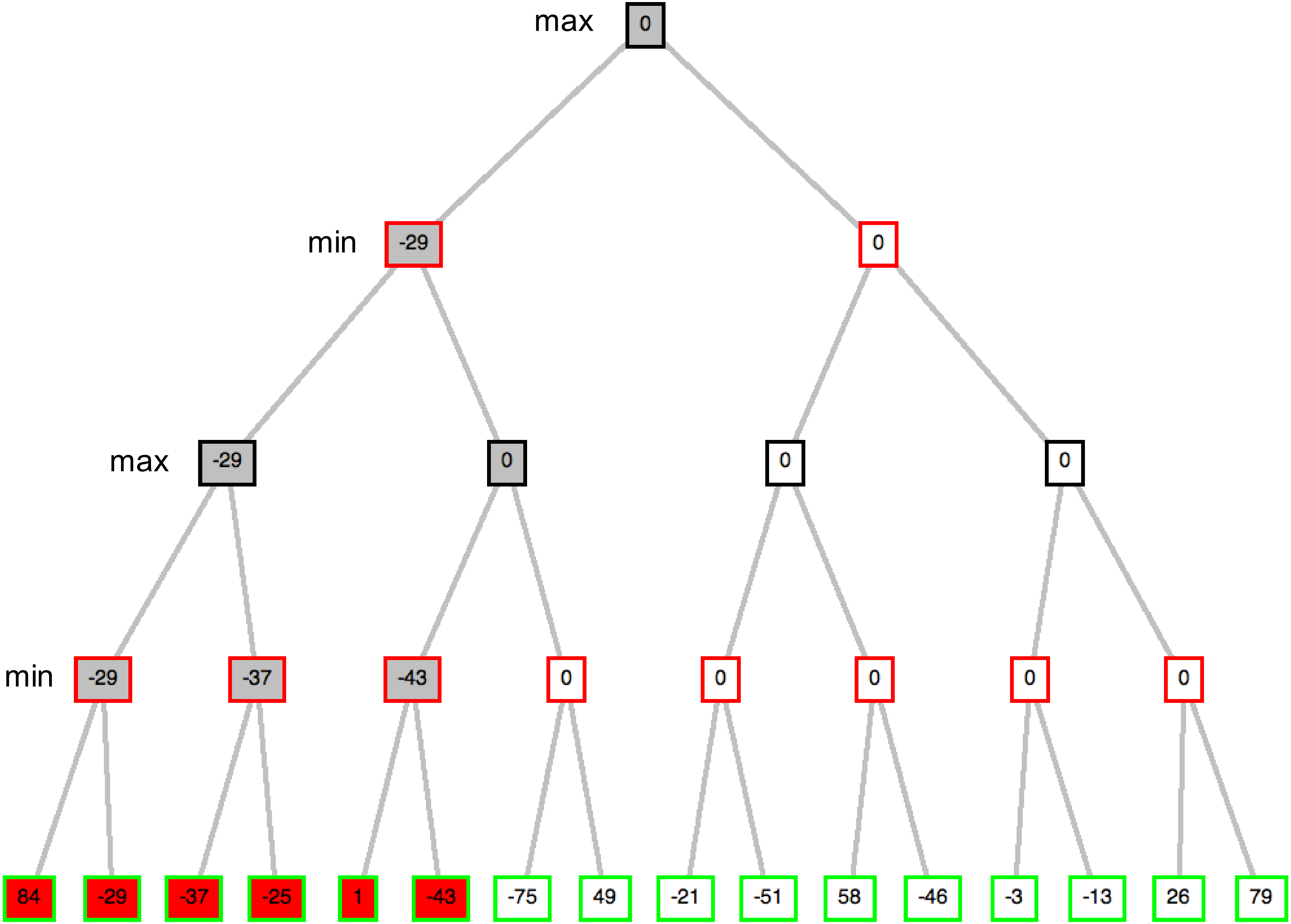


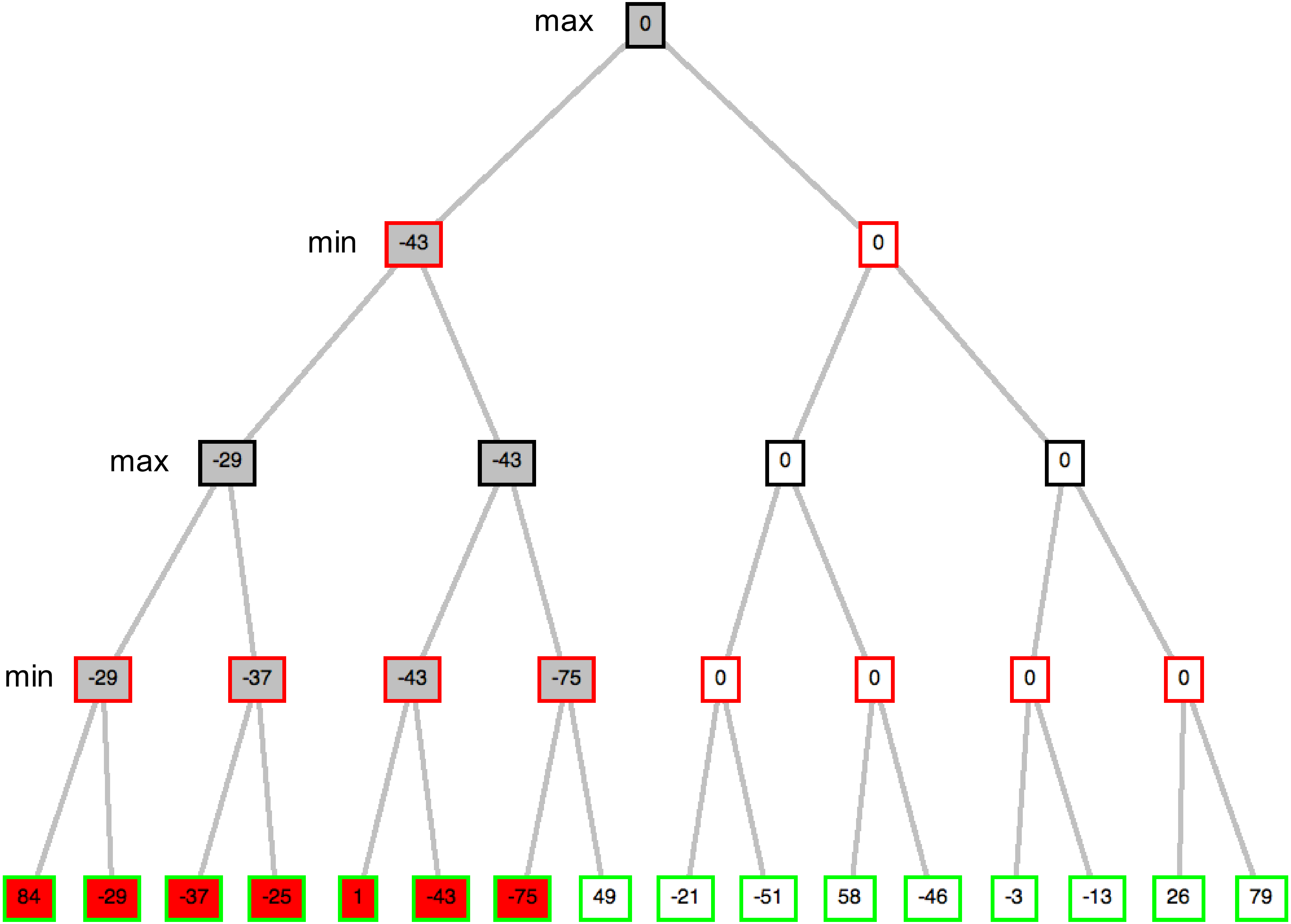
max
0
min
0
0
max
0
0
0
0
min
-29
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

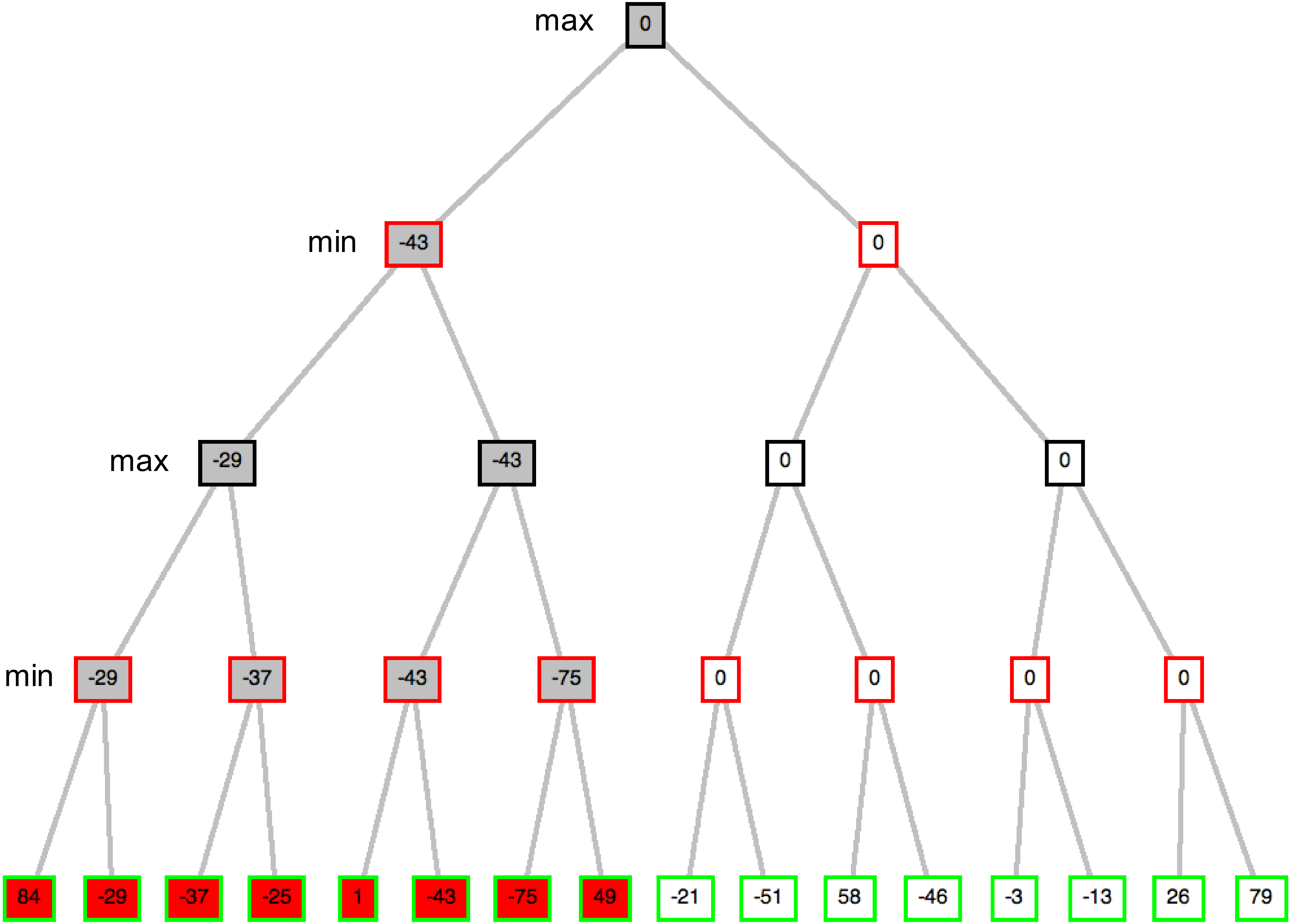

max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

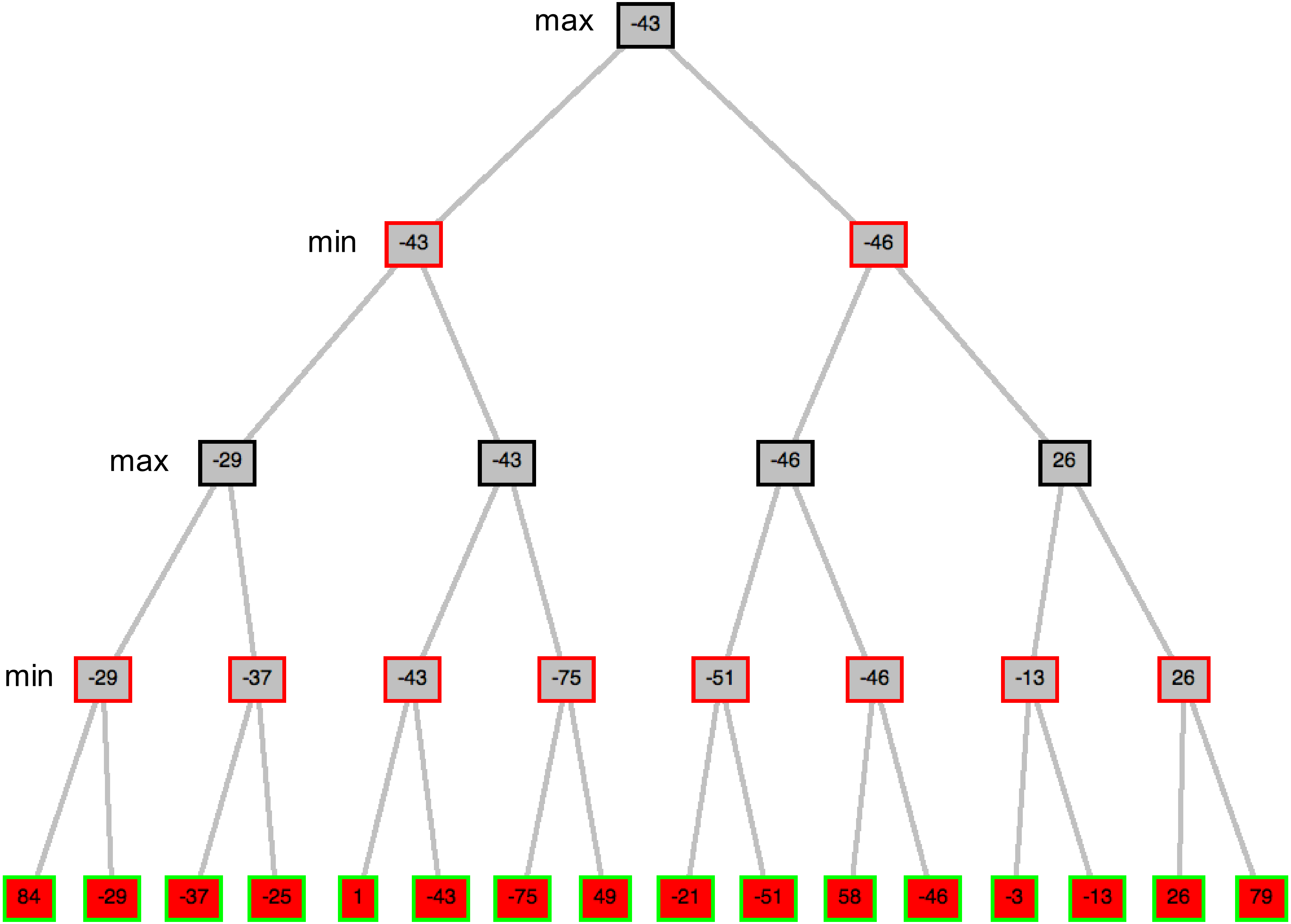

max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79


max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

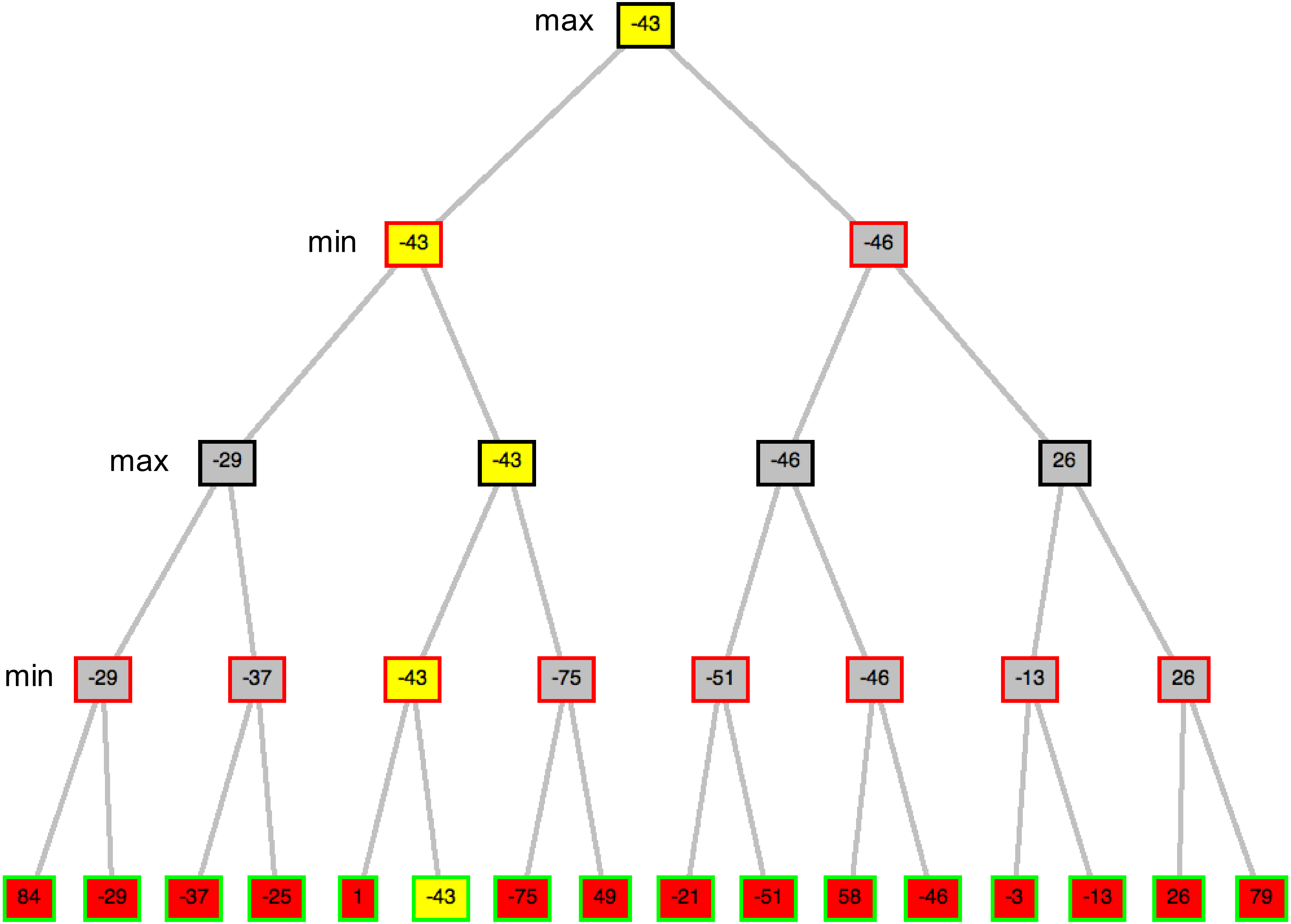

max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
-43
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79


max
0
min
-43
0
max
-29
-43
0
0
min
-29
-37
-43
-75
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79


max
0
min
-43
0
max
-29
-43
0
0
min
-29
-37
-43
-75
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79



max
-43
min
-43
-46
max
-29
-43
-46
26
min
-29
-37
-43
-75
-51
-46
-13
26
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79


max
-43
min
-43
-46
max
-29
-43
-46
26
min
-29
-37
-43
-75
-51
-46
-13
26
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

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 algorithm
Adversarial analogue of DFS

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)
  - No (does not exploit opponent weakness against suboptimal opponent)
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - O(bm) (depth-first exploration)

Good Enough?

- Chess:
  - branching factor $b \approx 35$
  - game length $m \approx 100$
  - search space $b^m \approx 35^{100} \approx 10^{154}$
- The Universe:
  - number of atoms $\approx 10^{78}$
  - age $\approx 10^{18}$ seconds
  - 10^8 moves/sec x 10^{78} x $10^{18} = 10^{104}$
- Exact solution completely infeasible

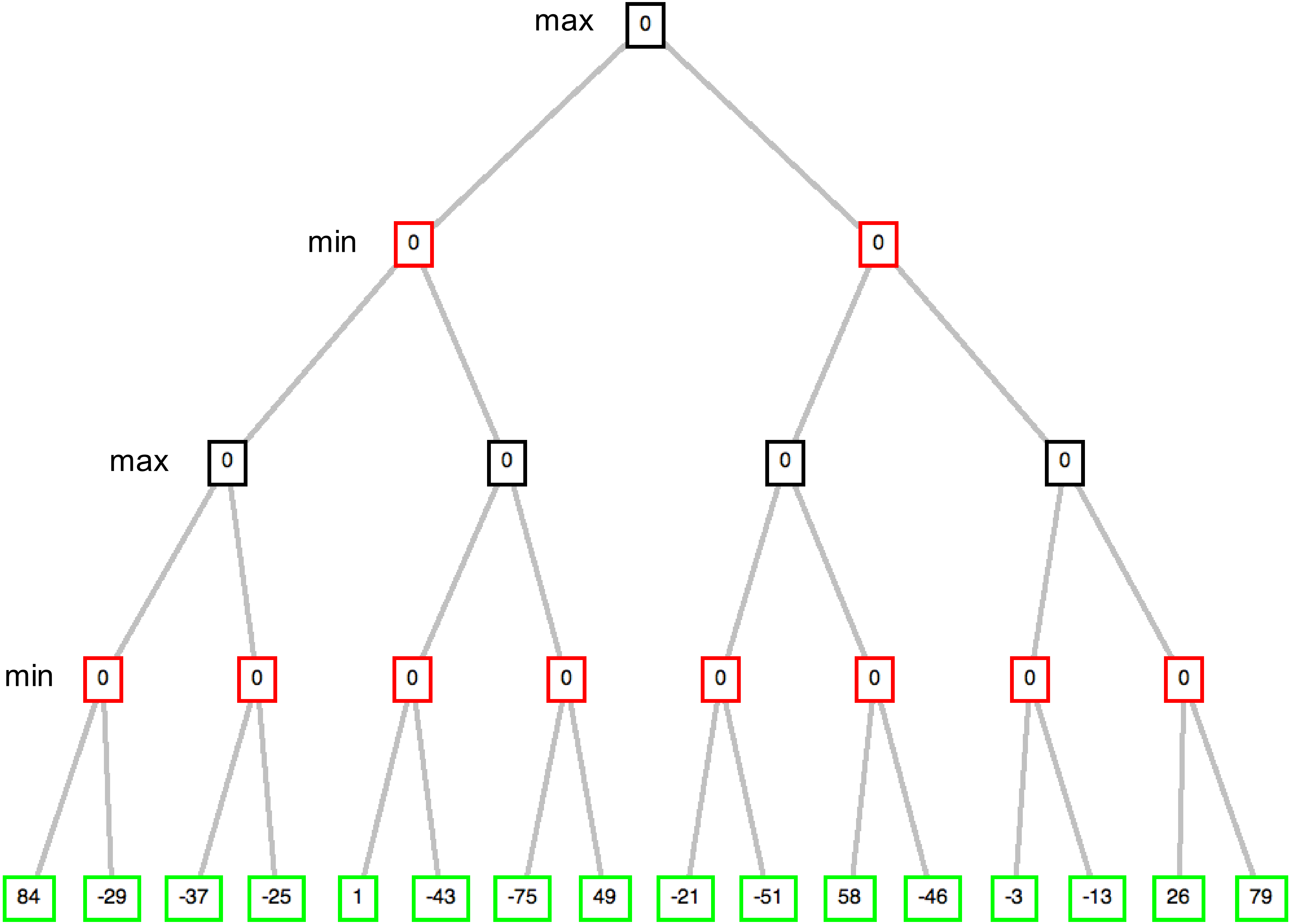
Alpha-Beta Procedure

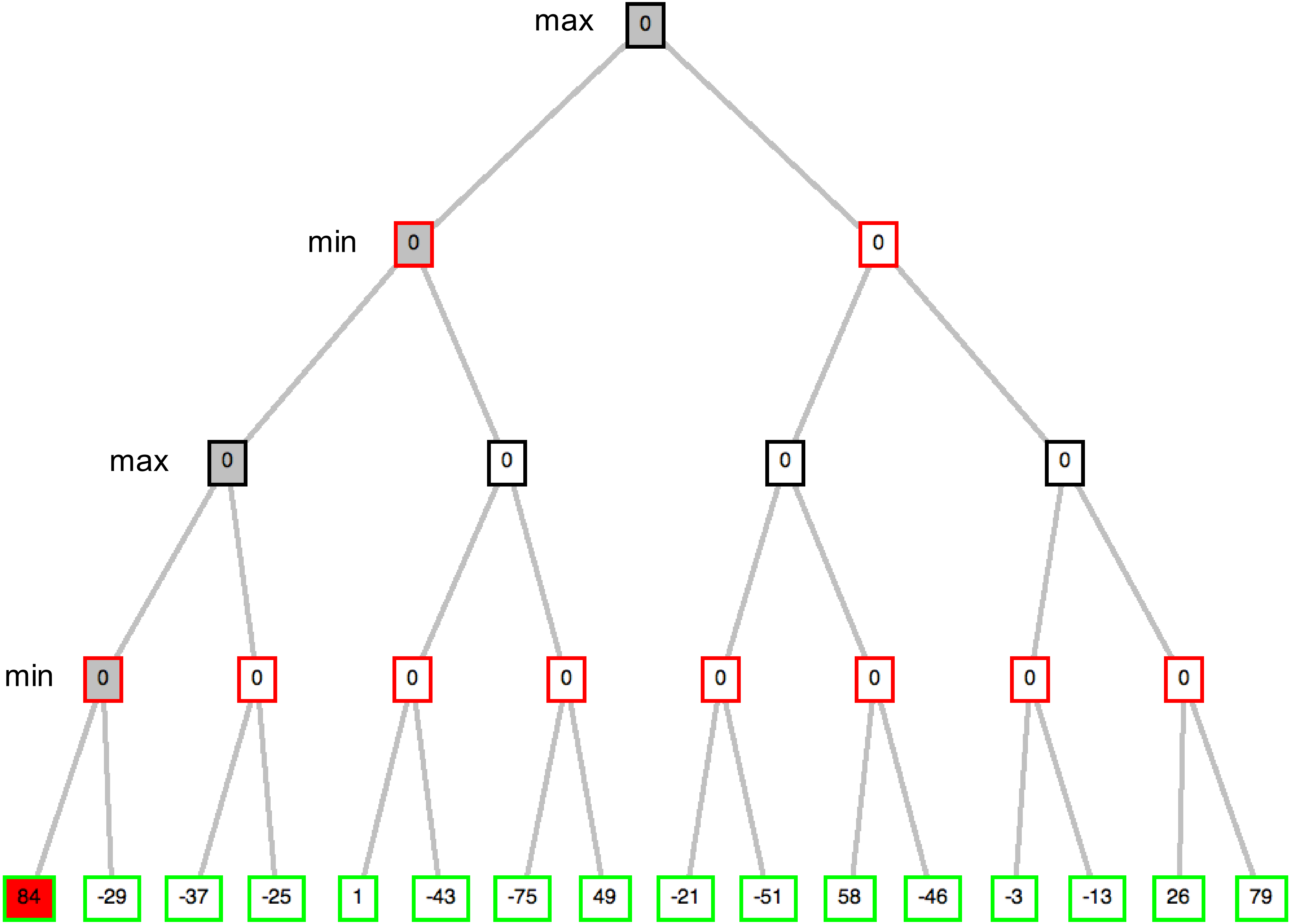
- The alpha-beta procedure can speed up a depth-first minimax search.
- Alpha: a lower bound on the value that a max node may ultimately be assigned

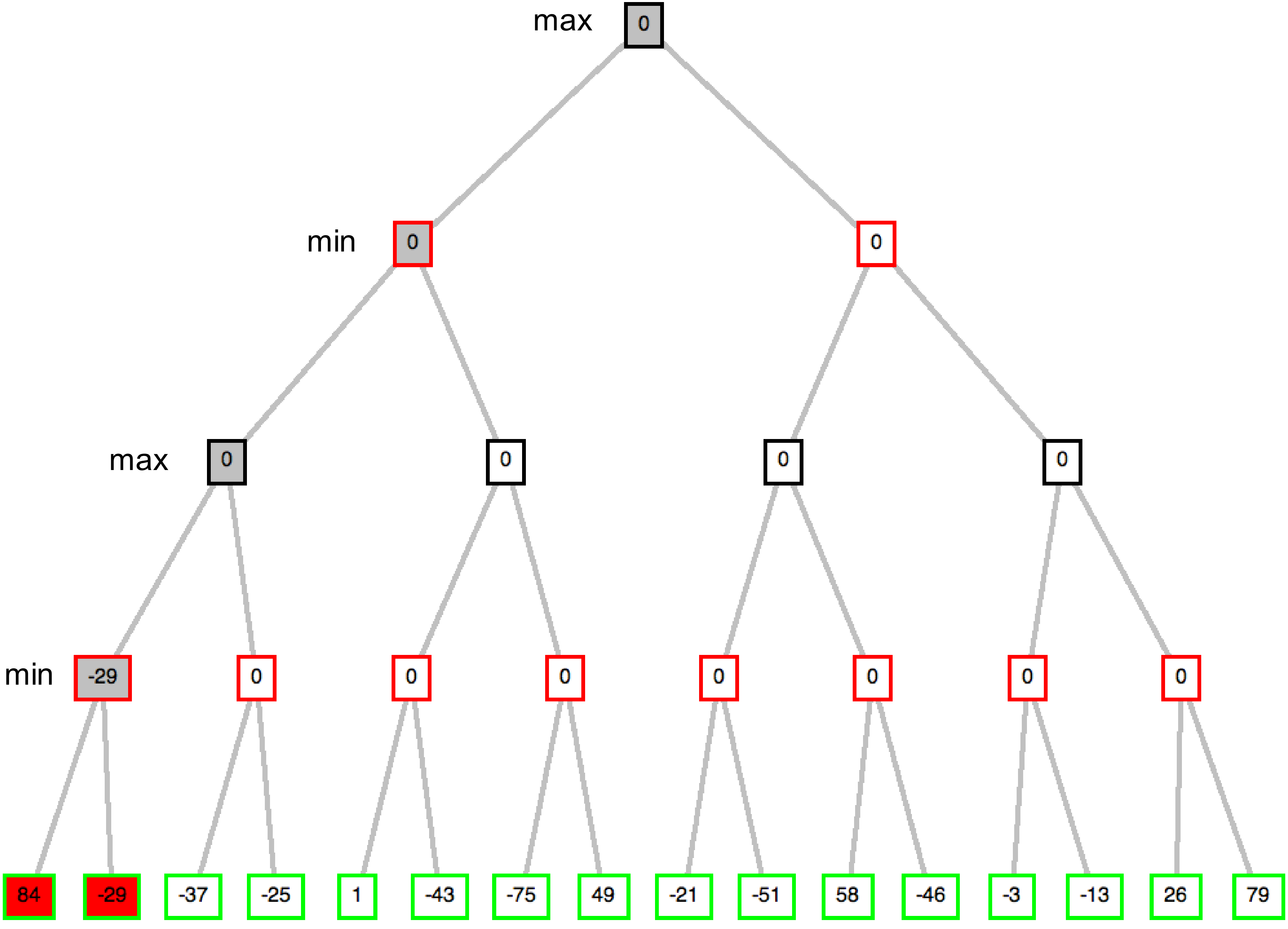
$$v \geq \alpha$$

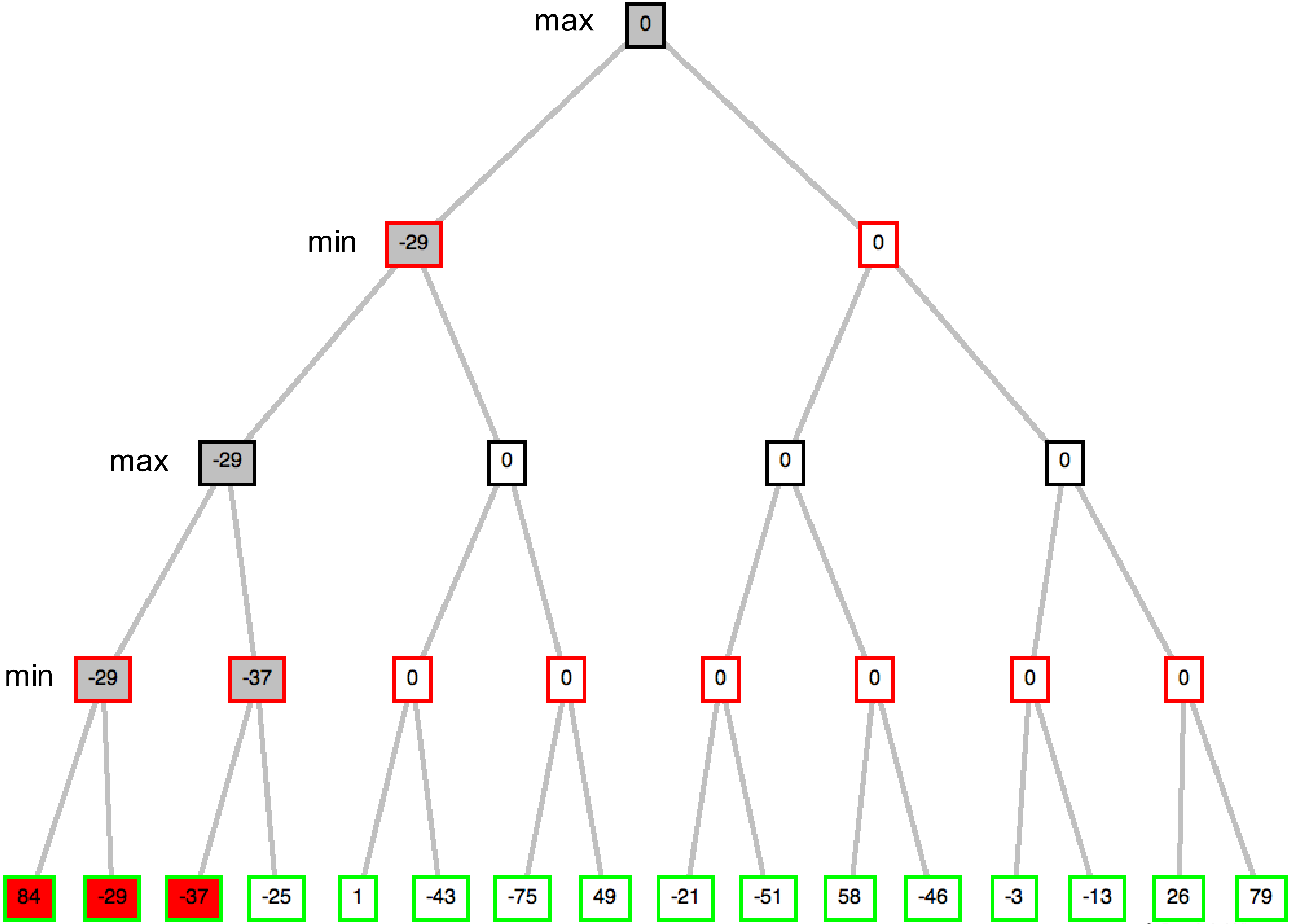
- Beta: an upper bound on the value that a minimizing node may ultimately be assigned

$$v \leq \beta$$

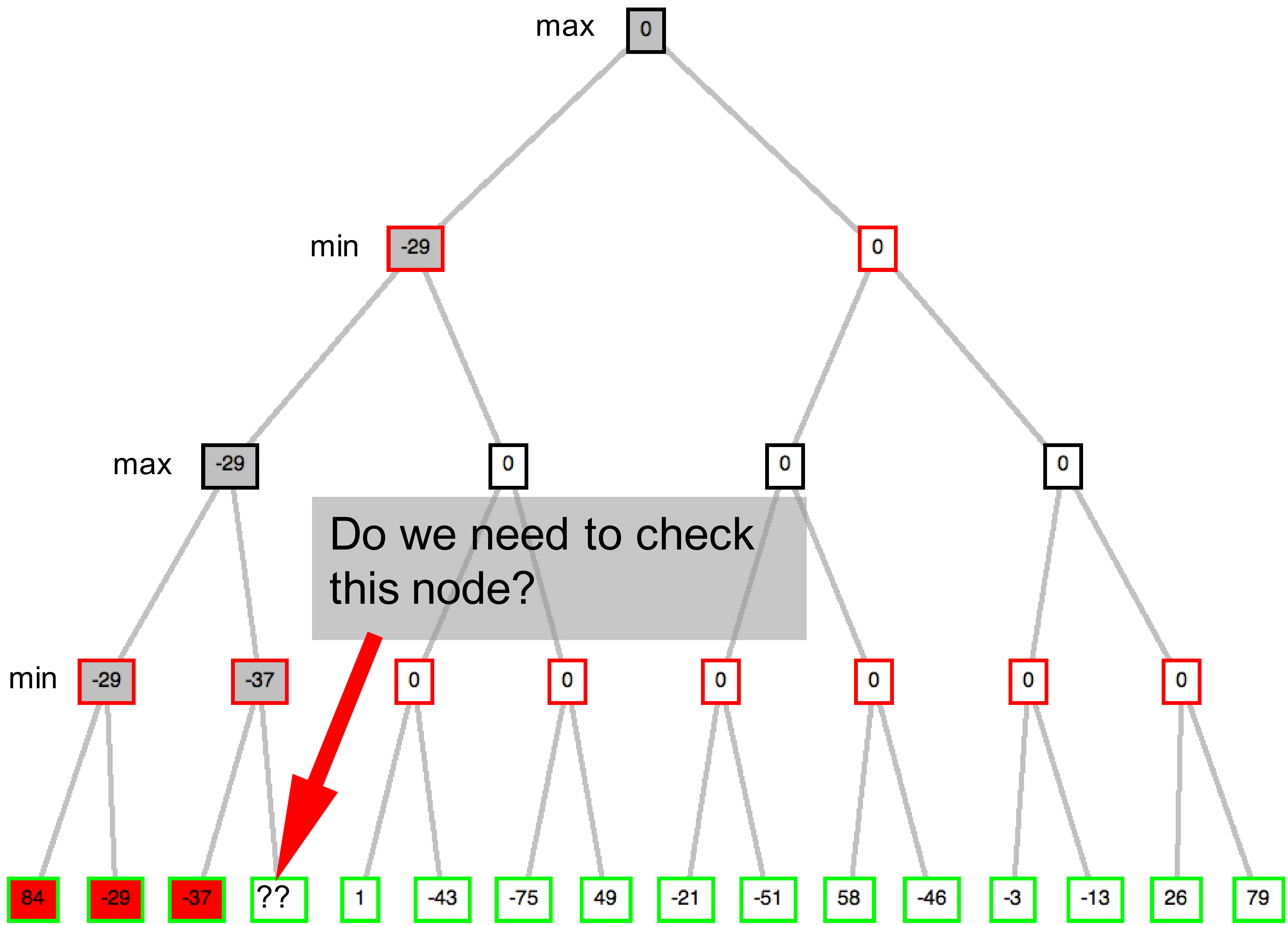

max
0
min
0
0
max
0
0
0
0
min
0
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

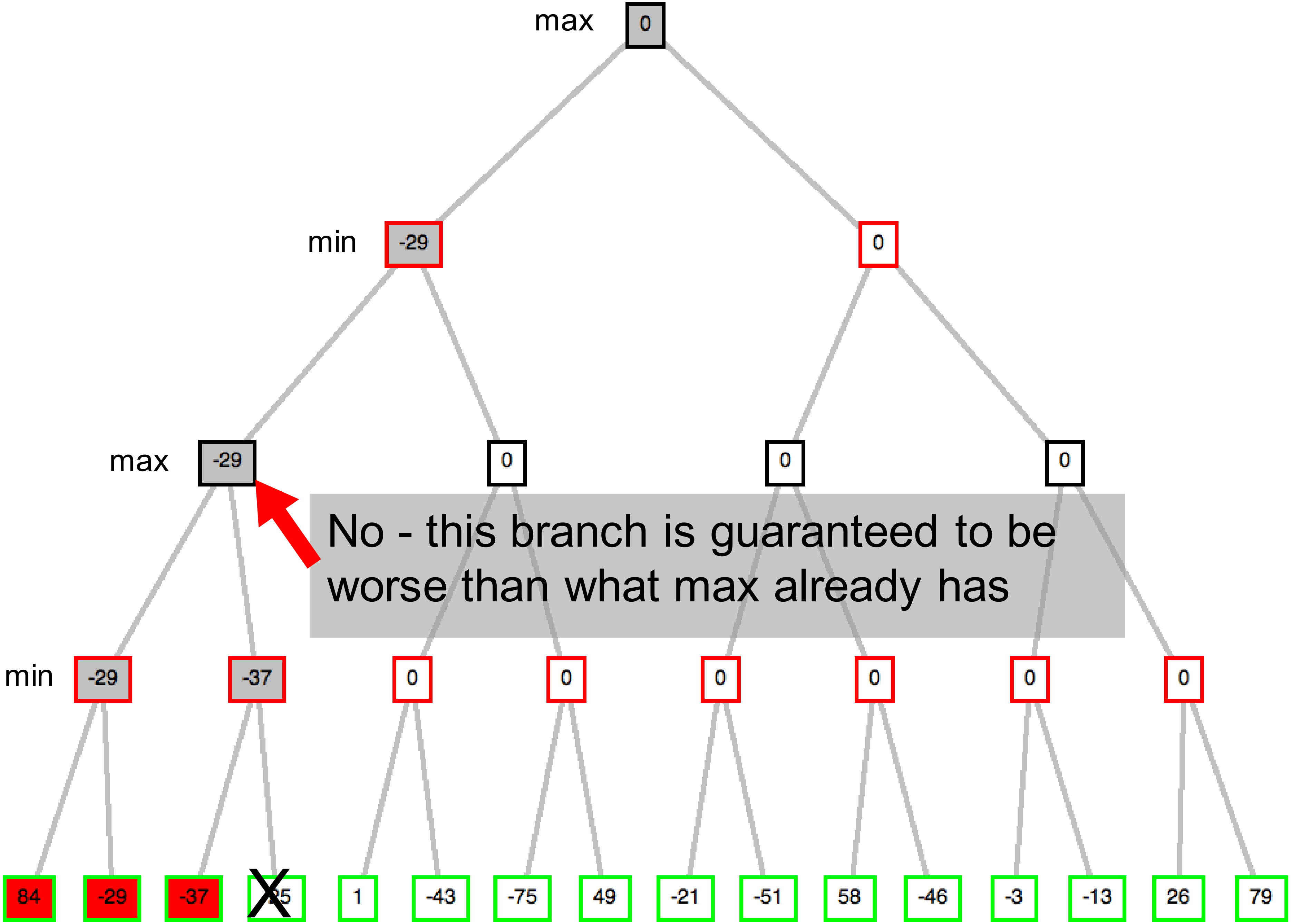

max
0
min
0
0
max
0
0
0
0
min
0
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79


max
0
min
0
0
max
0
0
0
0
min
-29
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79



max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79


max
0
min
-29
0
max
-29
0
0
0
Do we need to check this node?
min
-29
-37
0
0
0
0
0
0
84
-29
-37
??
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79


max
0
min
-29
0
max
-29
0
0
0
No - this branch is guaranteed to be worse than what max already has
min
-29
-37
0
0
0
0
0
0
84
-29
-37
25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

Alpha-Beta

```
MinVal(state, alpha, beta){
  if (terminal(state))
          return utility(state);
  for (s in children(state)){
     child = MaxVal(s,alpha,beta);
     beta = min(beta,child);
     if (alpha>=beta) return child;
  }
  return beta; }
```

alpha = the **highest** value for **MAX** along the path

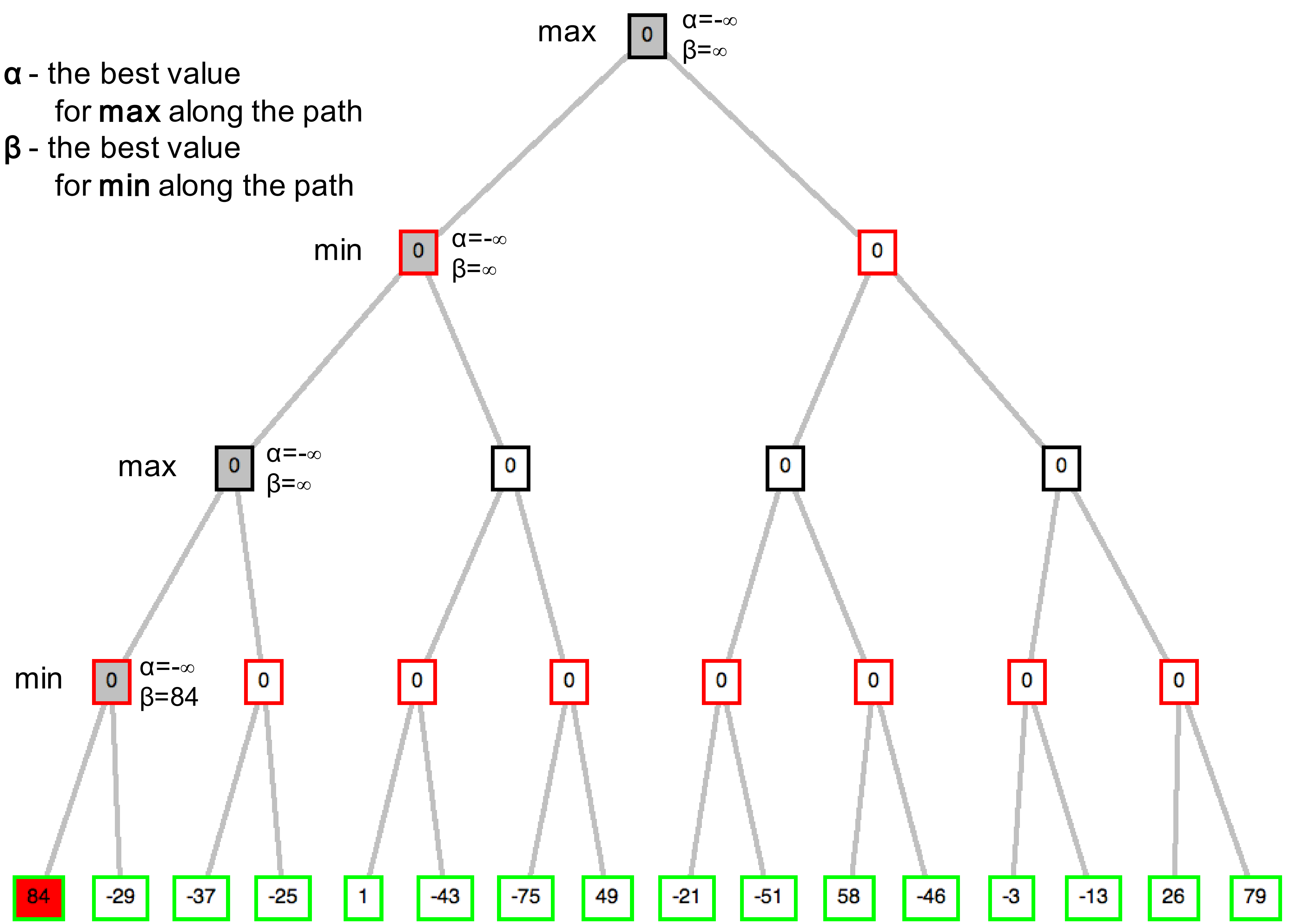
beta = the **lowest** value for **MIN** along the path

Alpha-Beta

```
MaxVal(state, alpha, beta){
  if (terminal(state))
           return utility(state);
  for (s in children(state)){
     child = MinVal(s,alpha,beta);
     alpha = max(alpha,child);
     if (alpha>=beta) return child;
  }
  return beta; }
```

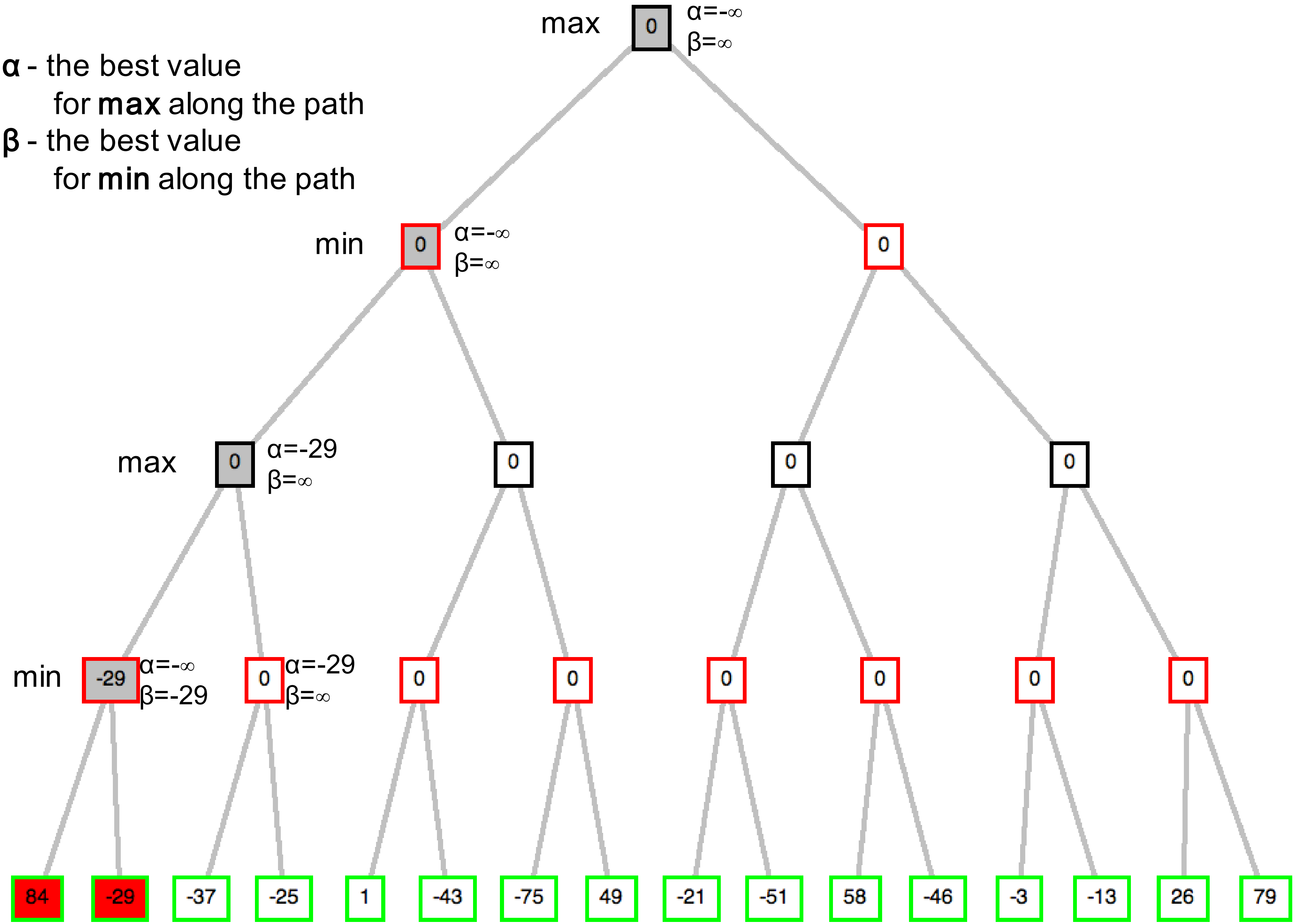
alpha = the **highest** value for **MAX** along the path

beta = the **lowest** value for **MIN** along the path



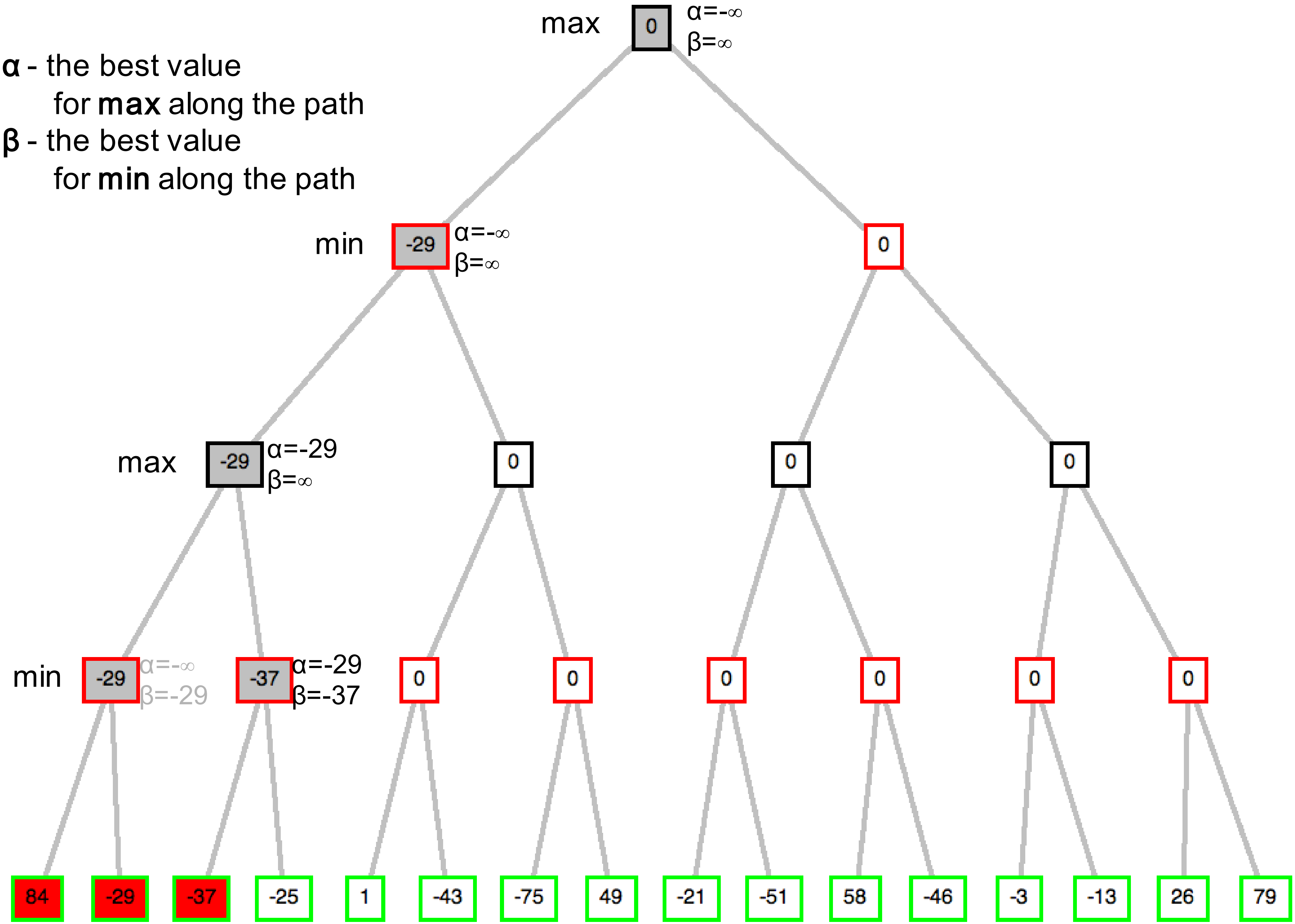
max
α=-∞
β=∞
α - the best value
for max along the path
β - the best value
for min along the path
min
α=-∞
β=∞
max
α=-∞
β=∞
min
α=-∞
β=84
0
0
0
0
0
0
0
0
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

α - the best value
for **max** along the path
β - the best value
for **min** along the path



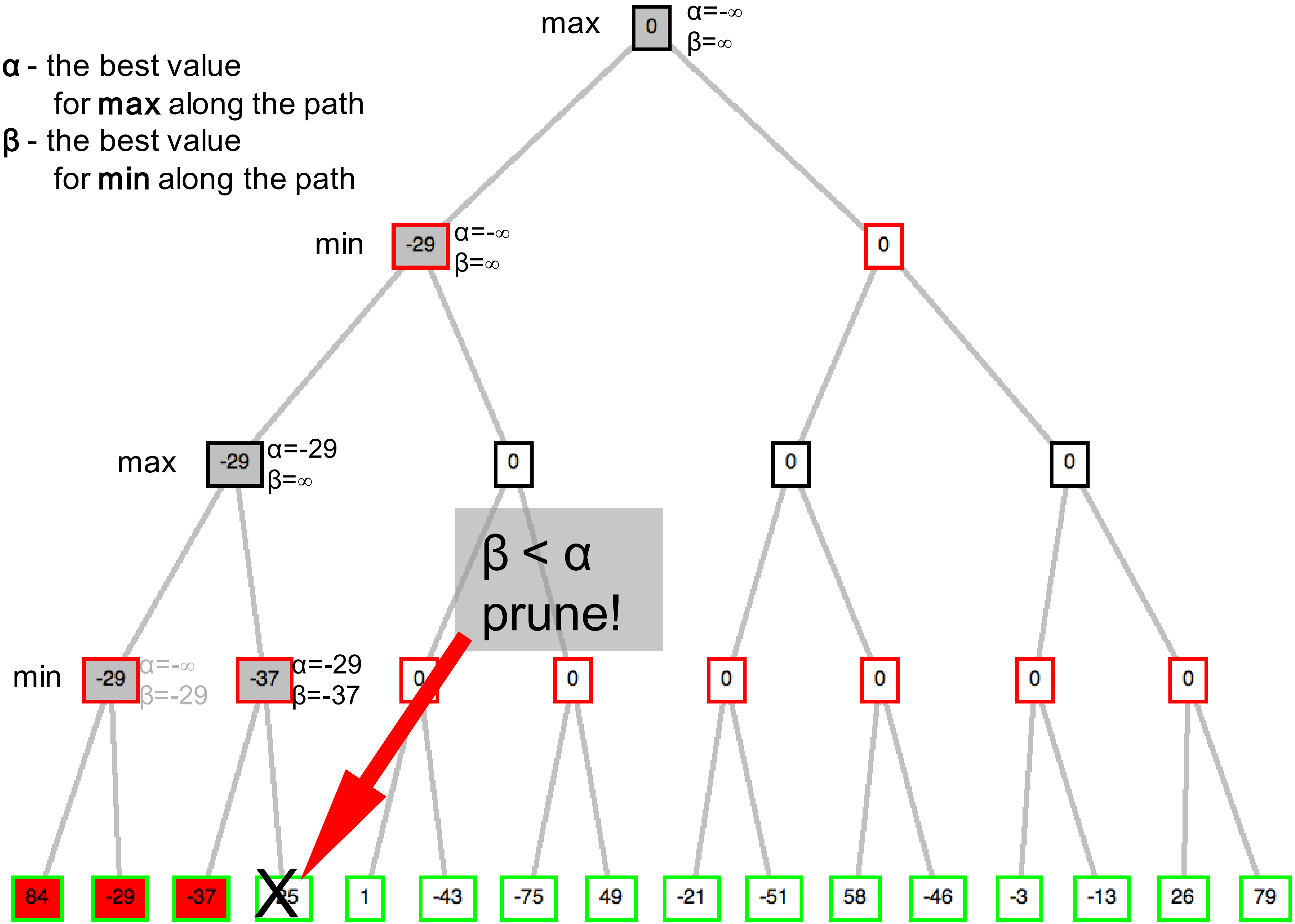
α - the best value for **max** along the path

β - the best value for **min** along the path



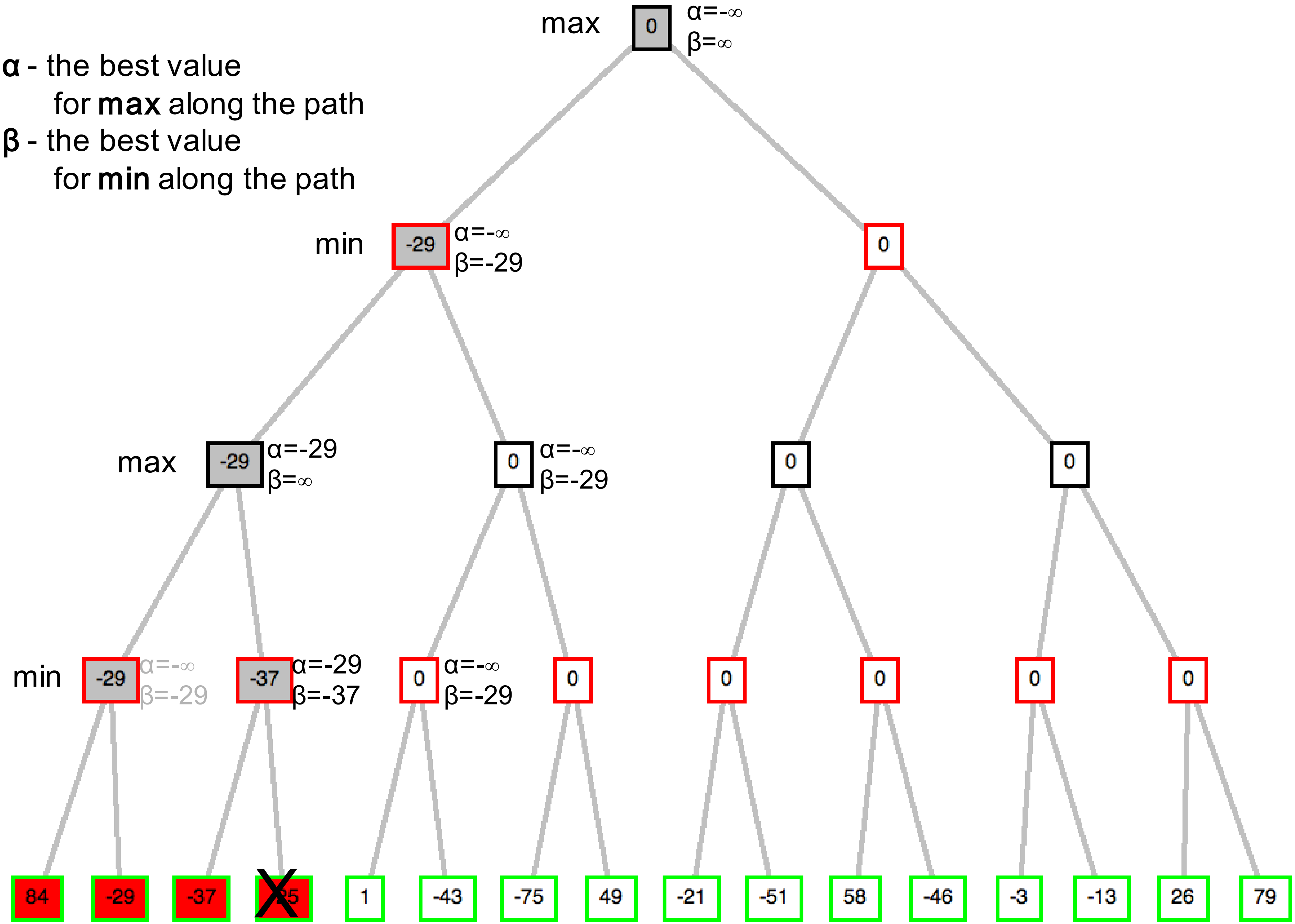
α - the best value for **max** along the path

β - the best value for **min** along the path

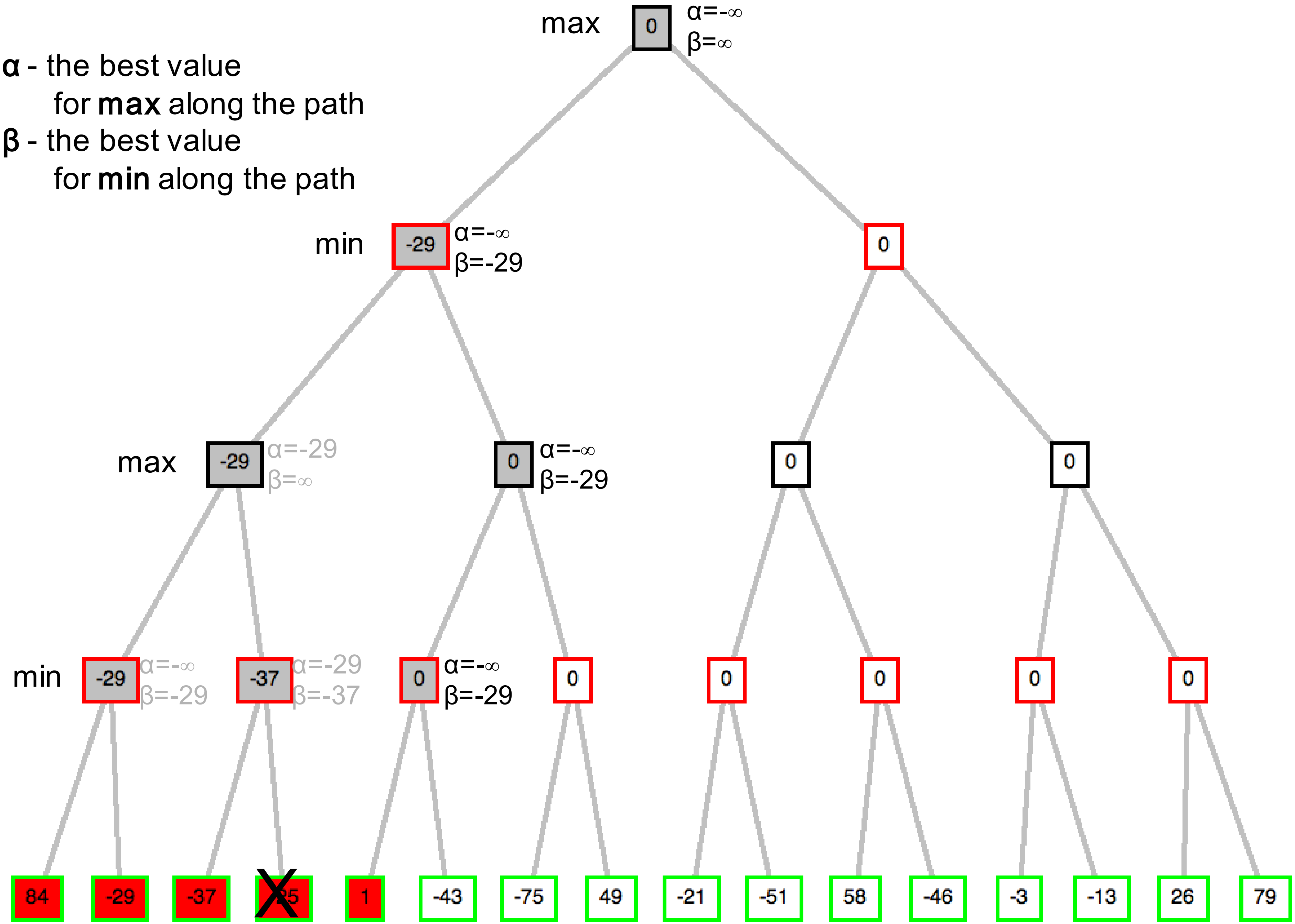


α - the best value
for **max** along the path

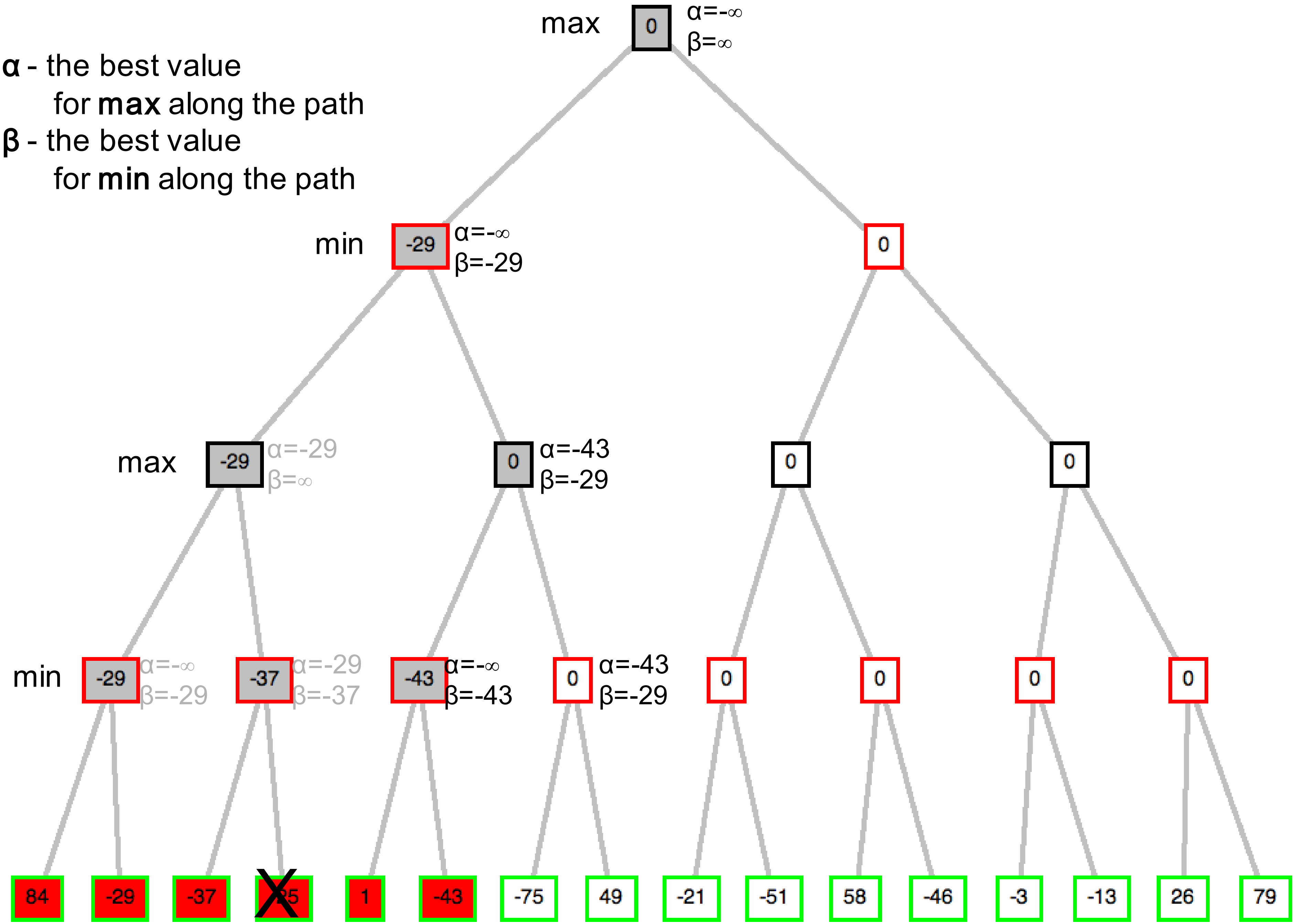
β - the best value
for **min** along the path



α - the best value
for **max** along the path
β - the best value
for **min** along the path

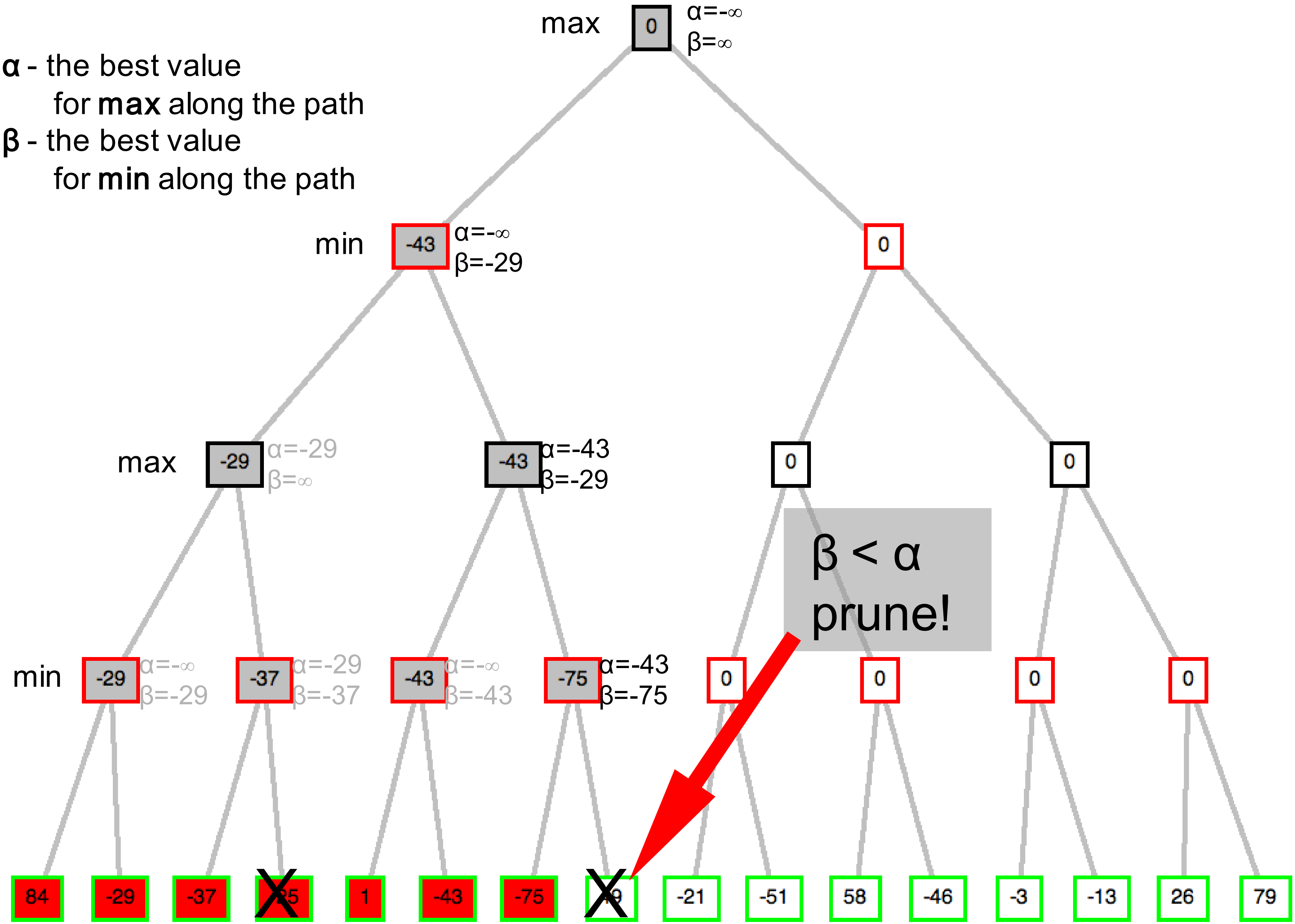


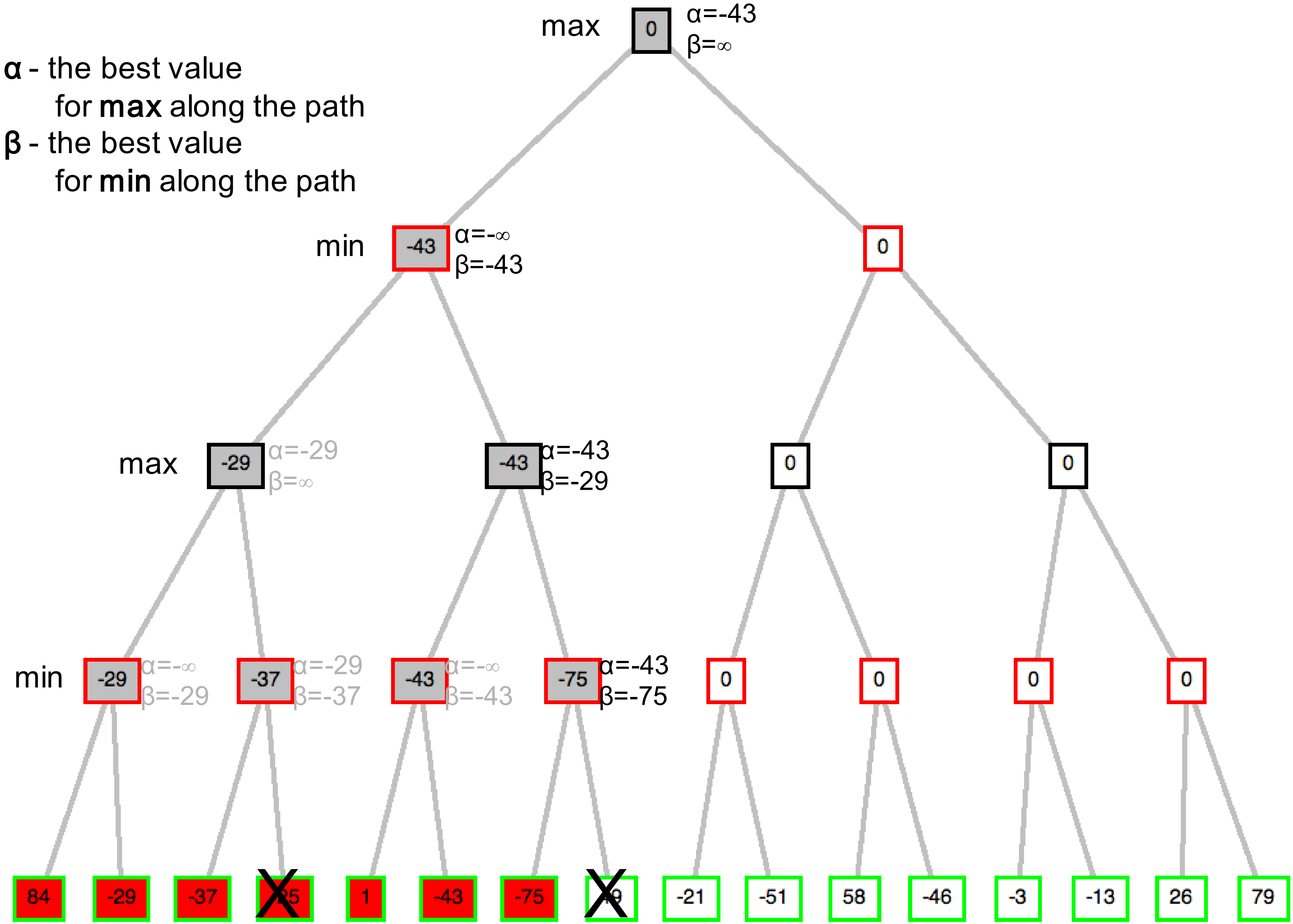
α - the best value for **max** along the path
β - the best value for **min** along the path



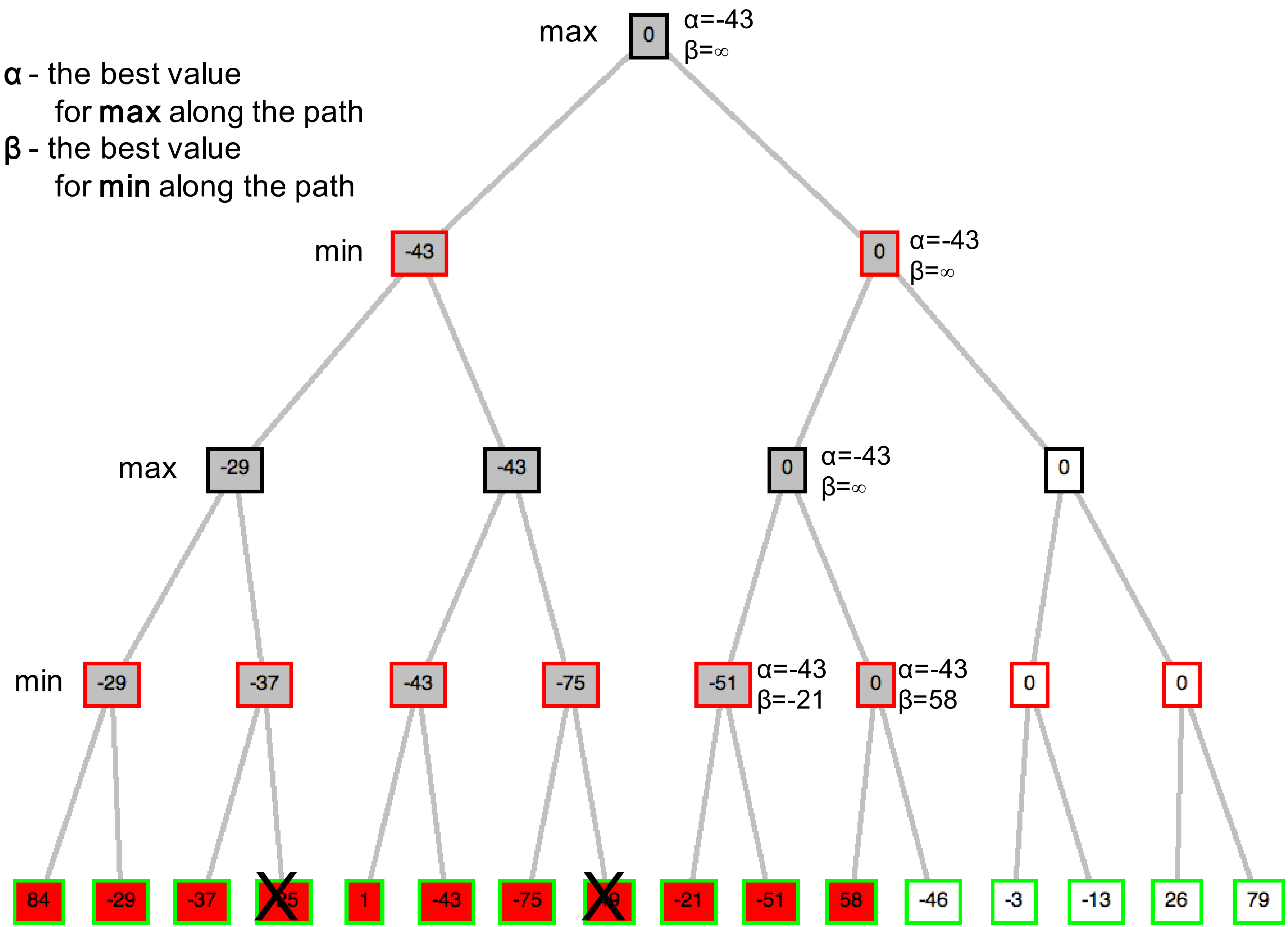
α - the best value for **max** along the path

β - the best value for **min** along the path



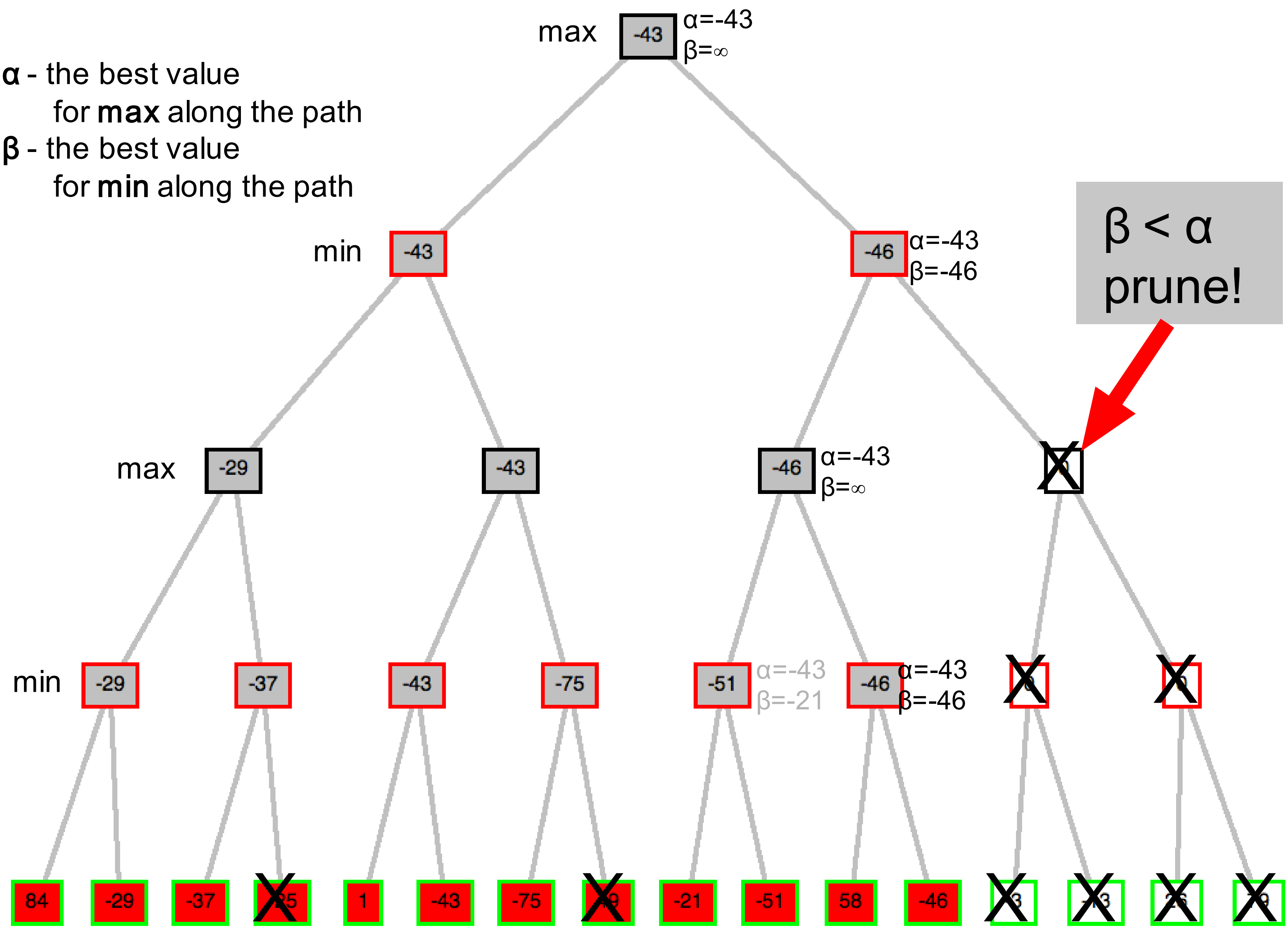


α - the best value for **max** along the path
β - the best value for **min** along the path
max
min
max
min
0
α=-43
β=∞
-43
α=-∞
β=-43
0
-29
α=-29
β=∞
-43
α=-43
β=-29
0
0
-29
α=-∞
β=-29
-37
α=-29
β=-37
-43
α=-∞
β=-43
-75
α=-43
β=-75
0
0
0
0
84
-29
-37
1
-43
-75
-21
-51
58
-46
-3
-13
26
79


α - the best value
for **max** along the path
β - the best value
for **min** along the path
max
0
α=-43
β=∞
min
-43
0
α=-43
β=∞
max
-29
-43
0
α=-43
β=∞
0
min
-29
-37
-43
-75
-51
α=-43
β=-21
0
α=-43
β=58
0
0
84
-29
-37
1
-43
-75
-21
-51
58
-46
-3
-13
26
79

α - the best value for **max** along the path

β - the best value for **min** along the path



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

Shallow Search Techniques

1. limited search for a few levels

2. reorder the level-1 sucessors

3. proceed with α-β minimax search

Good Enough?

- Chess:
  - branching factor $b \approx 35$
  - game length $m \approx 100$
  - search space $b^{m/2} \approx 35^{50} \approx 10^{77}$

The universe can play chess - can we?

- The Universe:
  - number of atoms $\approx 10^{78}$
  - age $\approx 10^{18}$ seconds
  - 10^8 moves/sec x 10^{78} x 10^{18} = 10^{104}

Cutting off Search

MinimaxCutoff is identical to *MinimaxValue* except

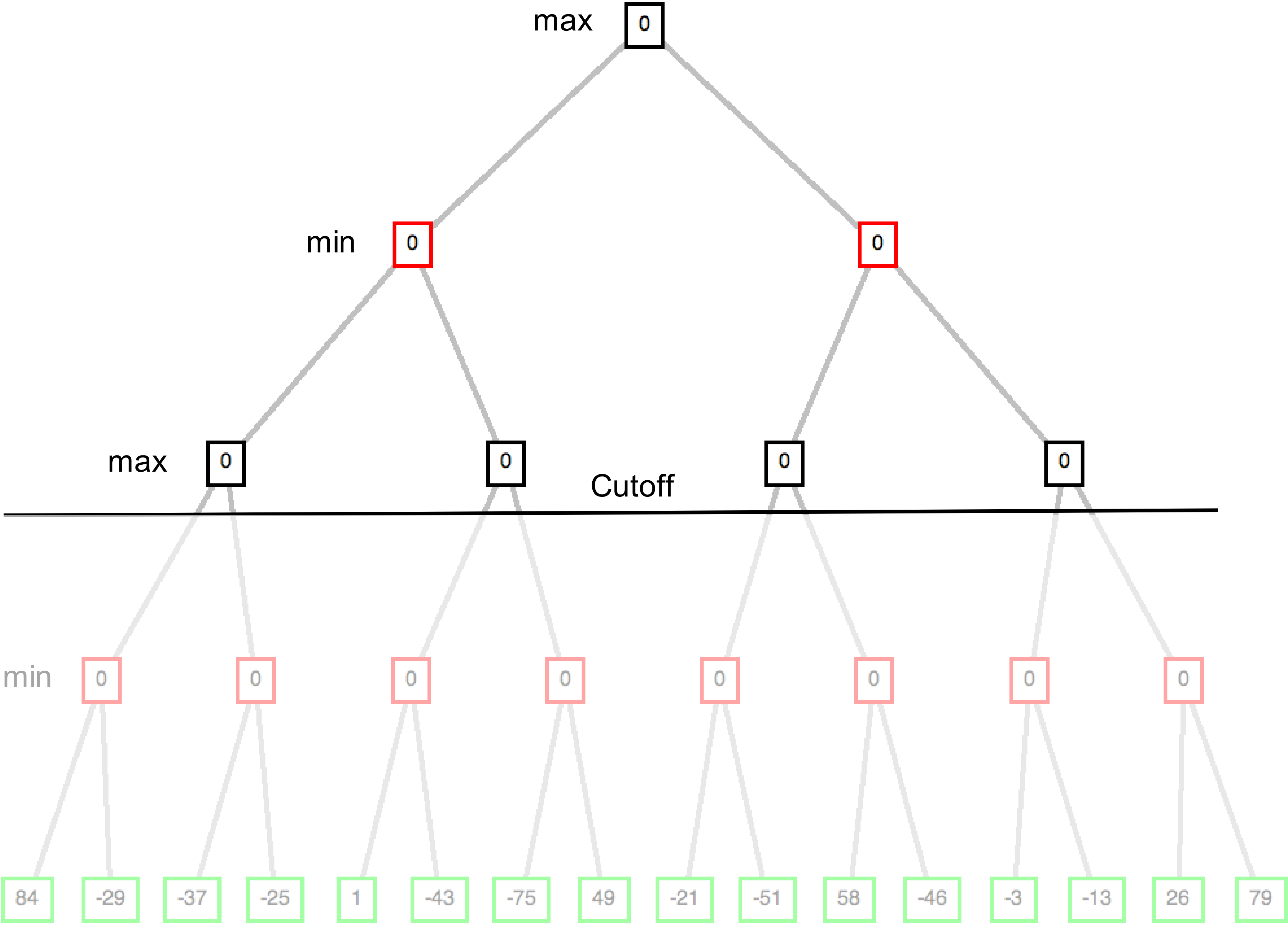
1. *Terminal?* is replaced by *Cutoff?*
2. *Utility* is replaced by *Eval*

Does it work in practice?

$b^m = 10^6$, $b=35 \rightarrow m=4$

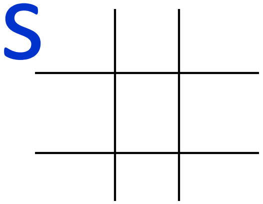
4-ply lookahead is a hopeless chess player!

- 4-ply ≈ human novice
- 8-ply ≈ typical PC, human master
- 12-ply ≈ Deep Blue, Kasparov



max
0
min
0
0
max
0
0
0
0
Cutoff
min
0
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

Evaluation Functions
Tic Tac Toe



- Let p be a position in the game
- Define the utility function f(p) by
  - f(p) =
    - largest positive number if p is a win for computer
    - smallest negative number if p is a win for opponent
    - RCDC – RCDO
  - where RCDC is number of rows, columns and diagonals in which computer could still win
  - and RCDO is number of rows, columns and diagonals in which opponent could still win.

Sample Evaluations

- X = Computer; O = Opponent

	O	
	X	

O	O	X
X	X	

	X	O
rows		
cols		
diags		

	X	O
rows		
cols		
diags		

Evaluation functions

- For chess/checkers, typically linear weighted sum of features

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + ... + w_n f_n(s)$$

e.g., $w_1 = 9$ with

$f_1(s)$ = (number of white queens) – (number of black queens), etc.

Example: Samuel's Checker-Playing Program

- It uses a linear evaluation function

  $f(n) = a_1x_1(n) + a_2x_2(n) + \ldots + a_mx_m(n)$

  For example: $f = 6K + 4M + U$

  - K = King Advantage
  - M = Man Advantage
  - U = Undenied Mobility Advantage (number of moves that Max where Min has no jump moves)

Samuel's Checker Player

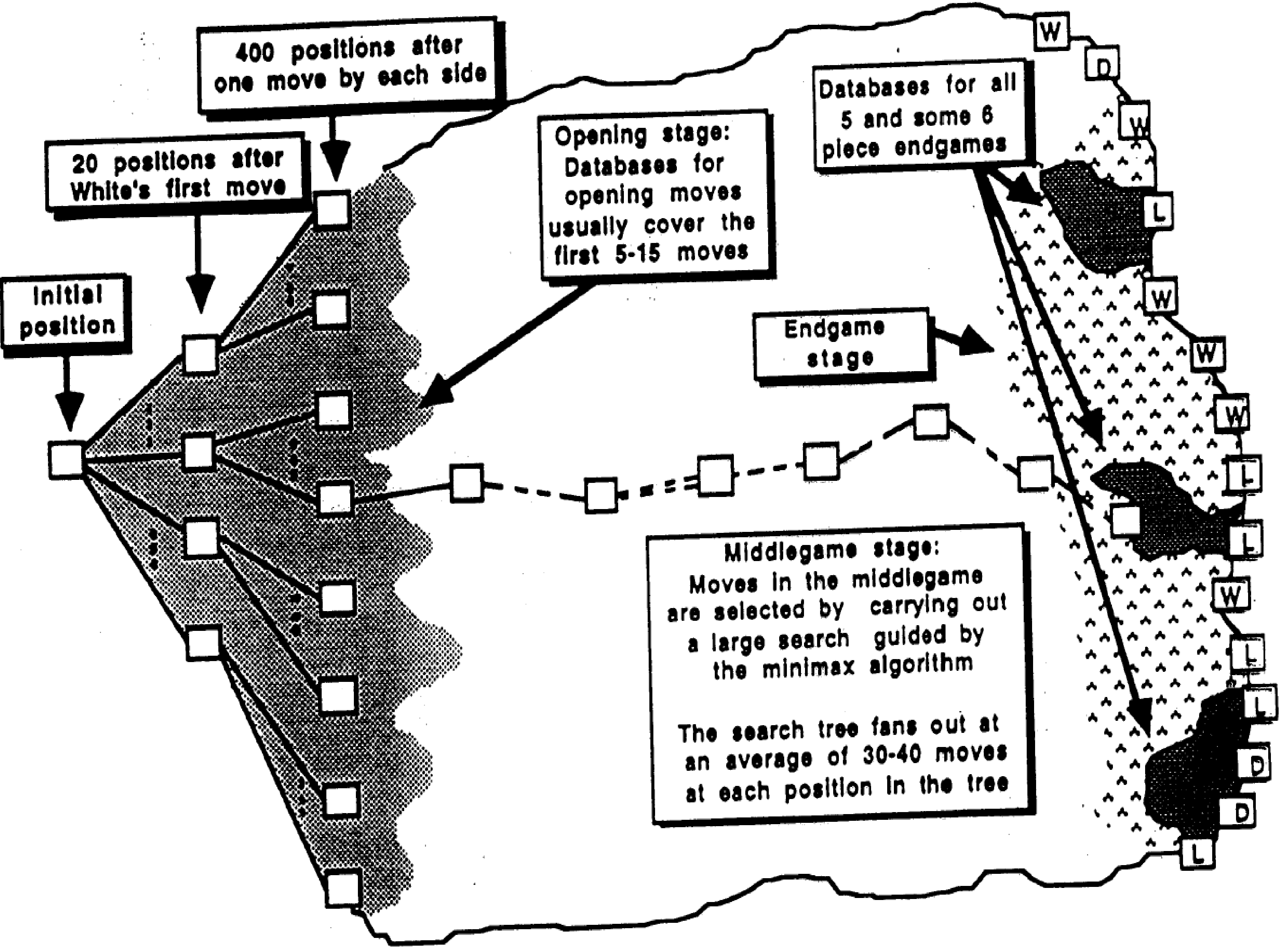
- In learning mode
  - Computer acts as 2 players: A and B
  - A adjusts its coefficients after every move
  - B uses the static utility function
  - If A wins, its function is given to B

Additional Refinements

- Waiting for Quiescence: continue the search until no drastic change occurs from one level to the next.
- Secondary Search: after choosing a move, search a few more levels beneath it to be sure it still looks good.
- Openings/Endgames: for some parts of the game (especially initial and end moves), keep a catalog of best moves to make.

Chess: Rich history of cumulative ideas

- Minimax search, evaluation function learning (1950).
- Alpha-Beta search (1966).
- Transposition Tables (1967).
- Iterative deepening DFS (1975).
- End game data bases ,singular extensions(1977, 1980)
- Parallel search and evaluation(1983 ,1985)
- Circuitry (1987)

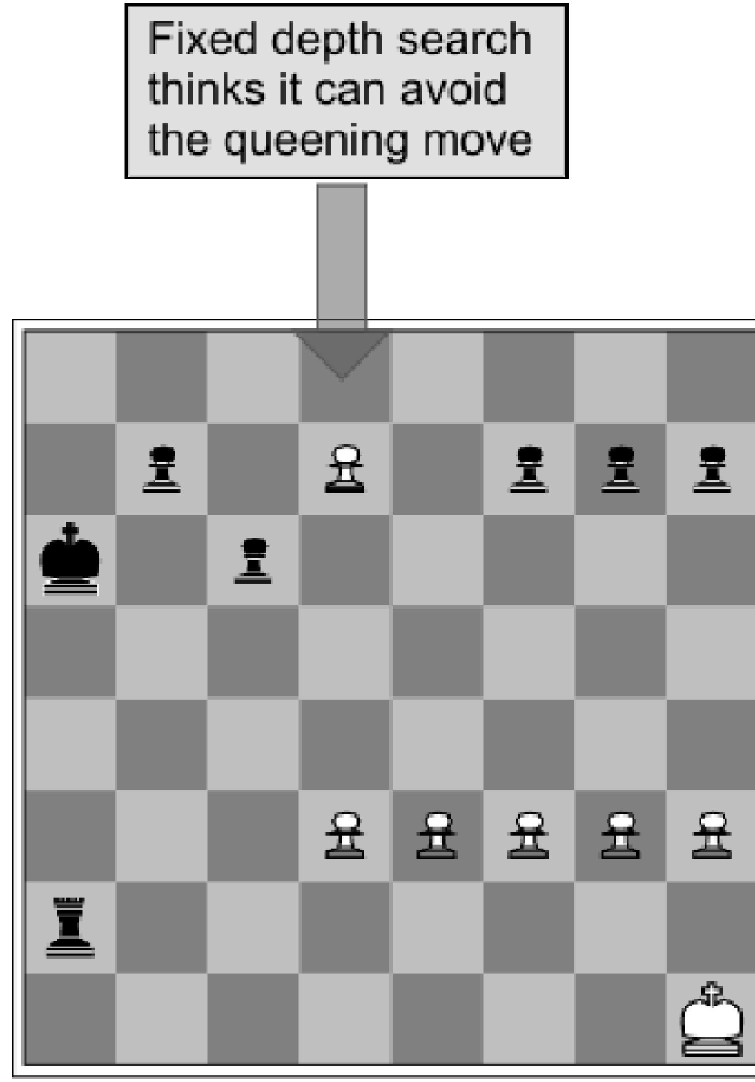


400 positions after one move by each side
20 positions after White's first move
Initial position
Opening stage: Databases for opening moves usually cover the first 5-15 moves
Databases for all 5 and some 6 piece endgames
Endgame stage
Middlegame stage: Moves in the middlegame are selected by carrying out a large search guided by the minimax algorithm
The search tree fans out at an average of 30-40 moves at each position in the tree
W
D
W
L
W
W
W
L
L
W
L
L
D
D
L

Problem with fixed depth Searches

if we only search n moves ahead, it may be possible that the catastrophy can be delayed by a sequence of moves that do not make any progress

also works in other direction (good moves may not be found)



Black to move

Quiescence Search

This involves searching past the terminal search nodes (depth of 0) and testing all the non-quiescent or 'violent' moves until the situation becomes calm, and only then apply the evaluator.

Enables programs to detect long capture sequences and calculate whether or not they are worth initiating.

Expand searches to avoid evaluating a position where tactical disruption is in progress.

Deterministic Games in Practice

- Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions. Checkers is now solved!

- Chess: Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic!

- Othello: human champions refuse to compete against computers, who are too good.

- Go: human champions refuse to compete against computers, who are too bad. In Go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves, along with aggressive pruning.

Game of Go

human champions refuse to compete against computers, because software is <u>too bad</u>.

	Chess	Go
Size of board	8 x 8	19 x 19
Average no. of moves per game	100	300
Avg branching factor per turn	35	235
Additional complexity		Players can pass

The story of AlphaGo so far

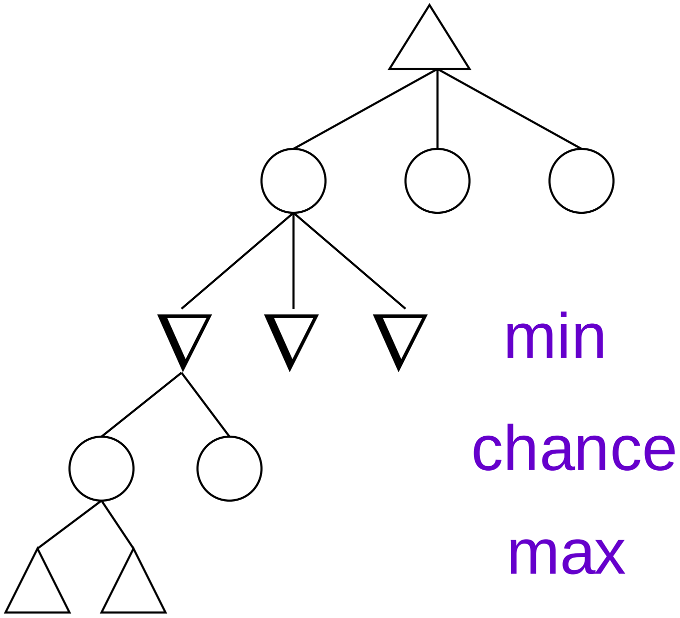
AlphaGo is the first computer program to defeat a professional human Go player, the first program to defeat a Go world champion, and arguably the strongest Go player in history.

AlphaGo’s first formal match was against the reigning 3-times European Champion, Mr Fan Hui, in October 2015.Its 5-0 win was the first ever against a Go professional, and the results were published in full technical detail in the international journal, **Nature**. AlphaGo then went on to compete against legendary player Mr Lee Sedol, winner of18 world titles and widely considered to be the greatest player of the past decade.

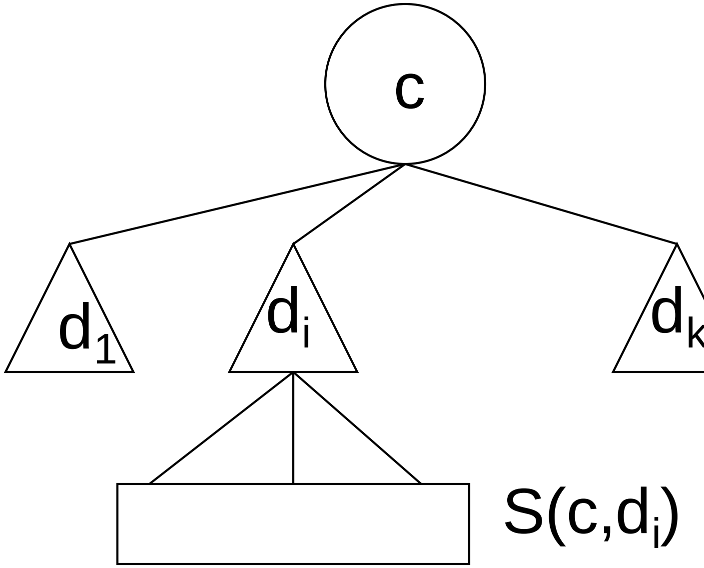
AlphaGo's 4-1 victory in Seoul, South Korea, in March 2016was watched by over 200 million people worldwide. It was a landmarkachievement that experts agreed was a decade ahead of its time,and earned AlphaGo a 9 dan professional ranking (the highestcertification) - the first time a computer Go player had everreceived the accolade.

Games of Chance

- What about games that involve chance, such as
  - rolling dice
  - picking a card
- Use three kinds of nodes:
  - max nodes
  - min nodes
  - chance nodes



Games of Chance
Expectiminimax

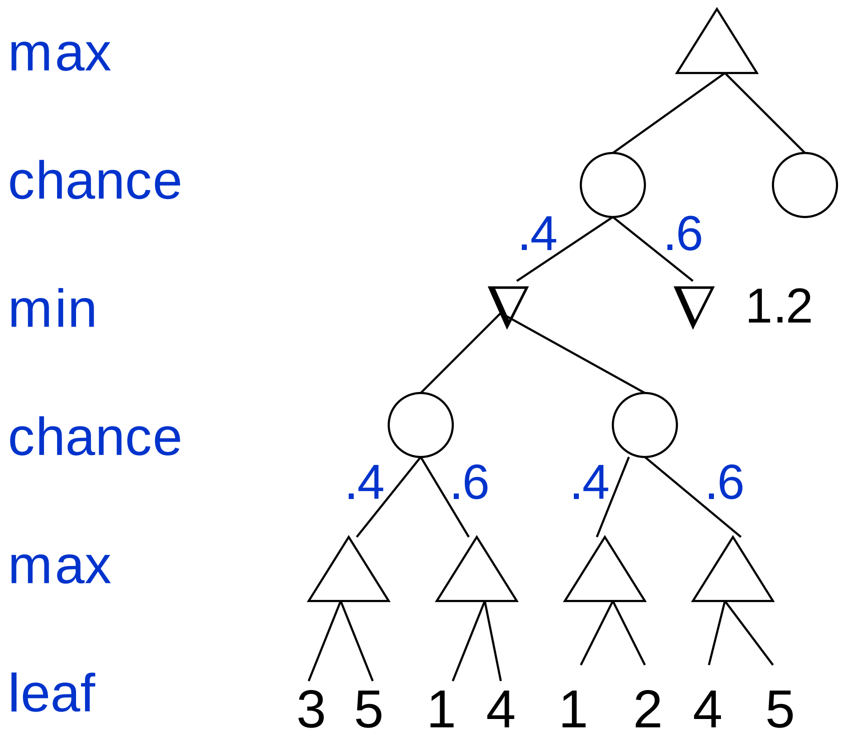


chance node with max children

$$\text{expectimax}(c) = \sum_i P(d_i) \max_{s \text{ in } S(c,d_i)}(\text{backed-up-value}(s))$$

$$\text{expectimin}(c') = \sum_i P(d_i) \min_{s \text{ in } S(c,d_i)}(\text{backed-up-value}(s))$$

Example Tree with Chance



Complexity

- Instead of $O(b^m)$, it is $O(b^m n^m)$ where n is the number of chance outcomes.
- Since the complexity is higher (both time and space), we cannot search as deeply.
- Pruning algorithms may be applied.

Imperfect Information

- E.g. card games, where opponents' initial cards unknown
- Idea: For all deals consistent with what you can see
  – compute the minimax value of available actions for each of possible deals
  – compute the expected value over all deals

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.