ARTIFICIAL INTELLIGENCE
UNIT-III

COLLEGE OF ENGINEERING
(AUTONOMOUS)

Dr. R. Seeta Sireesha
Associate Professor
CSE Department
GVPCE(A)

Unit-3

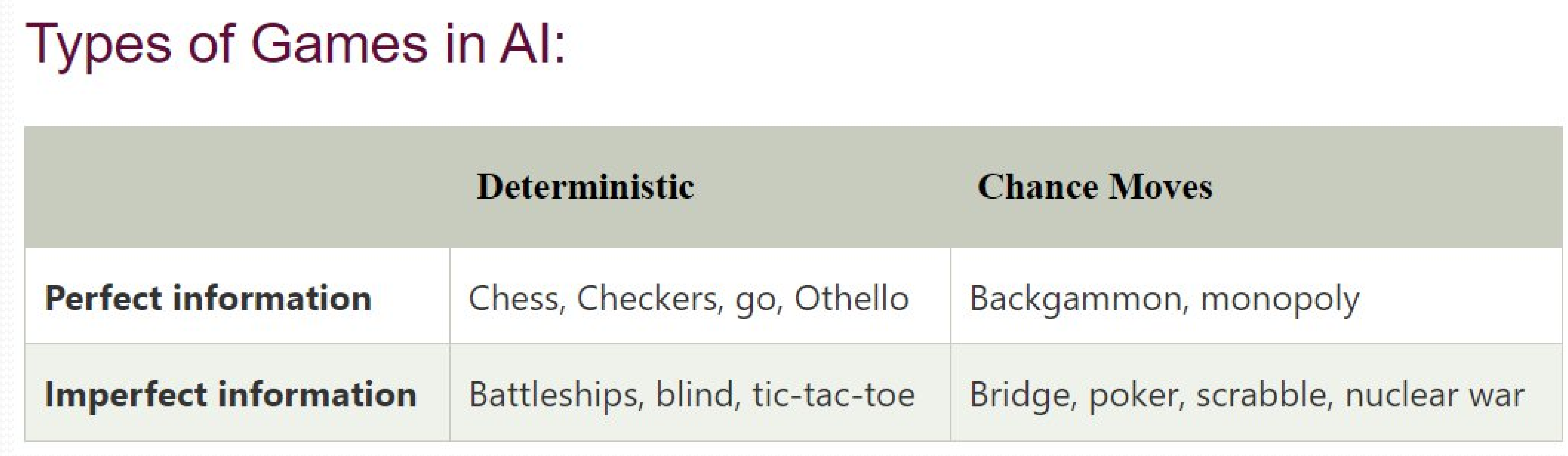
Adversarial search strategies: Introduction, Optimal strategies, The minmax algorithm, Alpha-Beta pruning, Constraint Satisfaction problem, Cryptarithmetic problem.

Knowledge and Reasoning: Knowledge representation issues, predicate logic-Resolution, Unification, Representation knowledge using Rules-Inference in First – order logic , forward and backward reasoning.

Introduction

- There might be some situations where more than one agent is searching for the solution in the same search space, and this situation usually occurs in game playing.
- The environment with more than one agent is termed as **multi-agent environment**, in which each agent is an opponent of other agent and playing against each other. Each agent needs to consider the action of other agent and effect of that action on their performance.
- So, **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games**.

- Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI.



Types of Games in AI:

	Deterministic	Chance Moves
Perfect information	Chess, Checkers, go, Othello	Backgammon, monopoly
Imperfect information	Battleships, blind, tic-tac-toe	Bridge, poker, scrabble, nuclear war

- **Perfect information:** Agents have all the information about the game, and they can see each other moves also. Examples are Chess, Checkers, Go, etc.
- **Imperfect information:** If in a game, agents do not have all information about the game and not aware with what's going on, such type of games are called the game with imperfect information, such as tic-tac-toe.
- **Deterministic games:** Deterministic games are those games which follow a strict pattern and set of rules for the games, and there is no randomness associated with them. Examples are chess, Checkers, Go, tic-tac-toe, etc.
- **Non-deterministic games:** Non-deterministic are those games which have various unpredictable events and has a factor of chance or luck. This factor of chance or luck is introduced by either dice or cards. These are random, and each action response is not fixed. games. Example: Backgammon, Monopoly, Poker, etc.

Formalization of the problem:

A game can be defined as a type of search in AI which can be formalized of the following elements:

- **Initial state:** It specifies how the game is set up at the start.
- **Player(s):** It specifies which player has moved in the state space.
- **Action(s):** It returns the set of legal moves in state space.
- **Result(s, a):** It is the transition model, which specifies the result of moves in the state space.
- **Terminal-Test(s):** The state where the game ends is called terminal states.
- **Utility(s, p):** A utility function gives the final numeric value for a game that ends in terminal states s for player p. It is also called payoff function. For Chess, the outcomes are a win, loss, or draw and its payoff values are +1, 0, ½. And for tic-tac-toe, utility values are +1, -1, and 0.

Optimal strategies

Characteristics of Games of strategy are as follows:

- Sequence of the moves to play.
- Rules that specify possible moves.
- Rules that specify a payment for each move.
- Objective is to maximize the payment.

Optimal strategies (cont..)

Game tree:

A game tree is a tree where *nodes* of the tree are the *game states* and *Edges* of the tree are the *moves by players*. Game tree involves initial state, actions function, and result Function.

Example: Tic-Tac-Toe game tree:

The following figure is showing part of the game-tree for tic-tac-toe game. Following are some key points of the game:

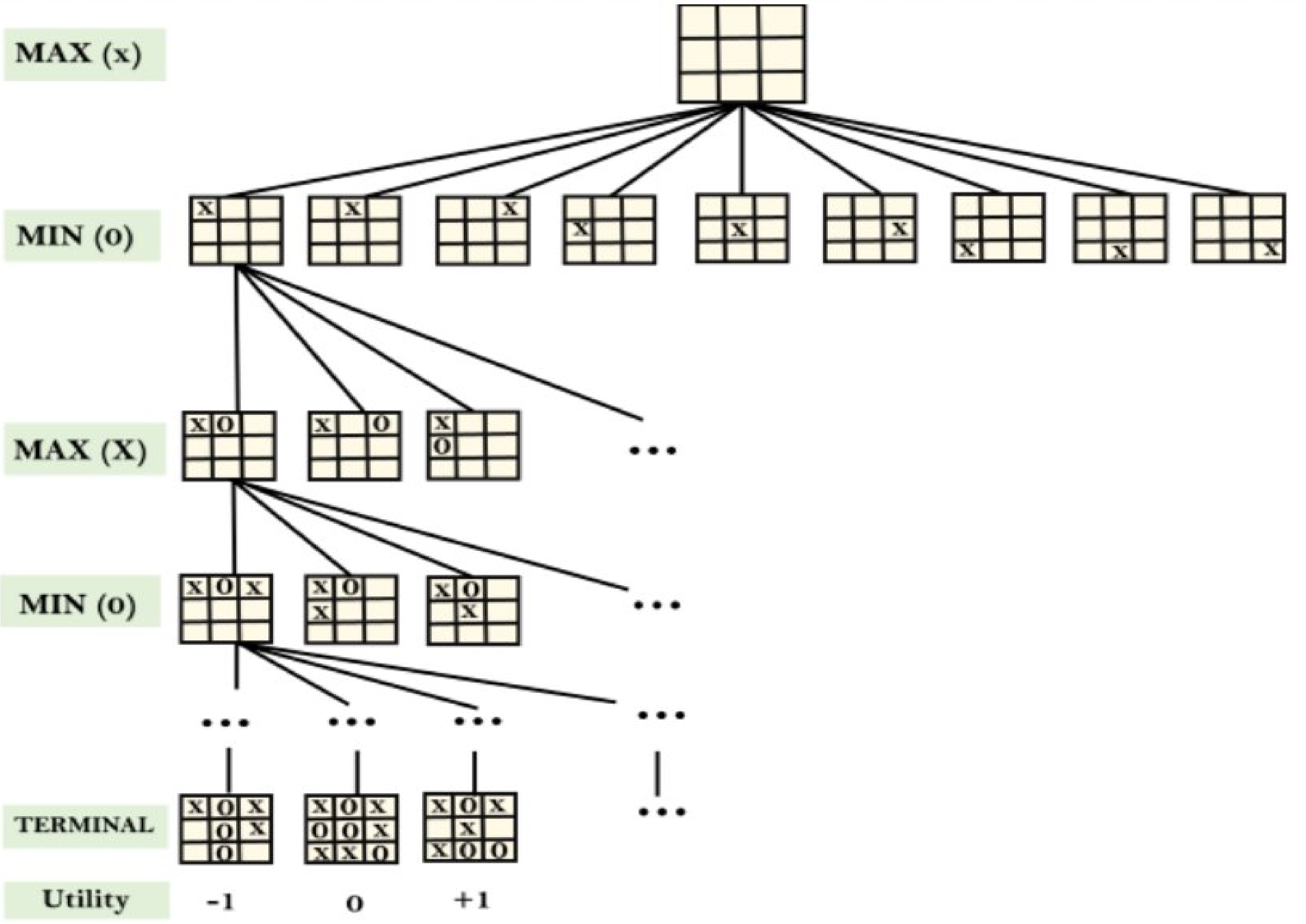
- There are two players MAX and MIN.
- Players have an alternate turn and start with MAX.
- MAX maximizes the result of the game tree.
- MIN minimizes the result.

Optimal strategies (cont..)

A game with two players (MAX and MIN, MAX moves first, taking turn) can be defined as a search problem with the following:

1. *Initial state*: Initial state consists of the position of the board and shows whose move it is.
2. *Successor function*: Successor function defines of the legal moves a player can make.
3. *Terminal state*: Terminal state is the position of the board when the game is over.
4. *Goal test*: Whether the game is over – Terminal states.
5. *Utility function*: Utility function is the function that assigns a numeric value for the outcome of a game.
6. *Game tree* = Initial state + Legal moves.

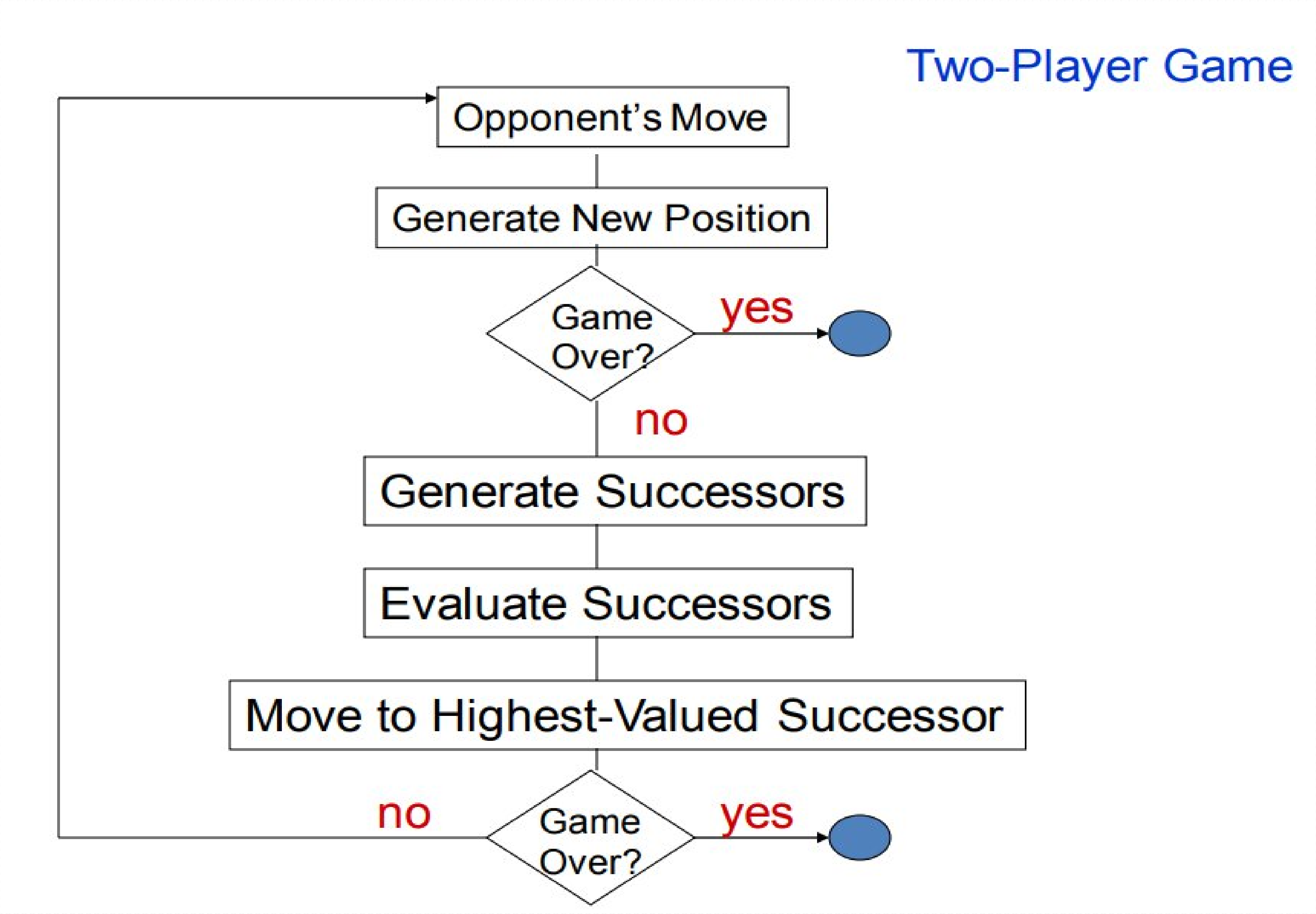
Optimal strategies



Example Explanation

- From the initial state, MAX has 9 possible moves as he starts first. MAX place x and MIN place o, and both player plays alternatively until we reach a leaf node where one player has three in a row or all squares are filled.
- Both players will compute each node, minimax, the minimax value which is the best achievable utility against an optimal adversary.
- Suppose both the players are well aware of the tic-tac-toe and playing the best play. Each player is doing his best to prevent another one from winning. MIN is acting against Max in the game.
- So in the game tree, we have a layer of MAX, a layer of MIN, and each layer is called as **Ply**. MAX place x, then MIN puts o to prevent MAX from winning, and this game continues until the terminal node.
- In this either MIN wins, MAX wins, or it's a draw. This game-tree is the whole search space of possibilities that MIN and MAX are playing tic-tac-toe and taking turns alternately.

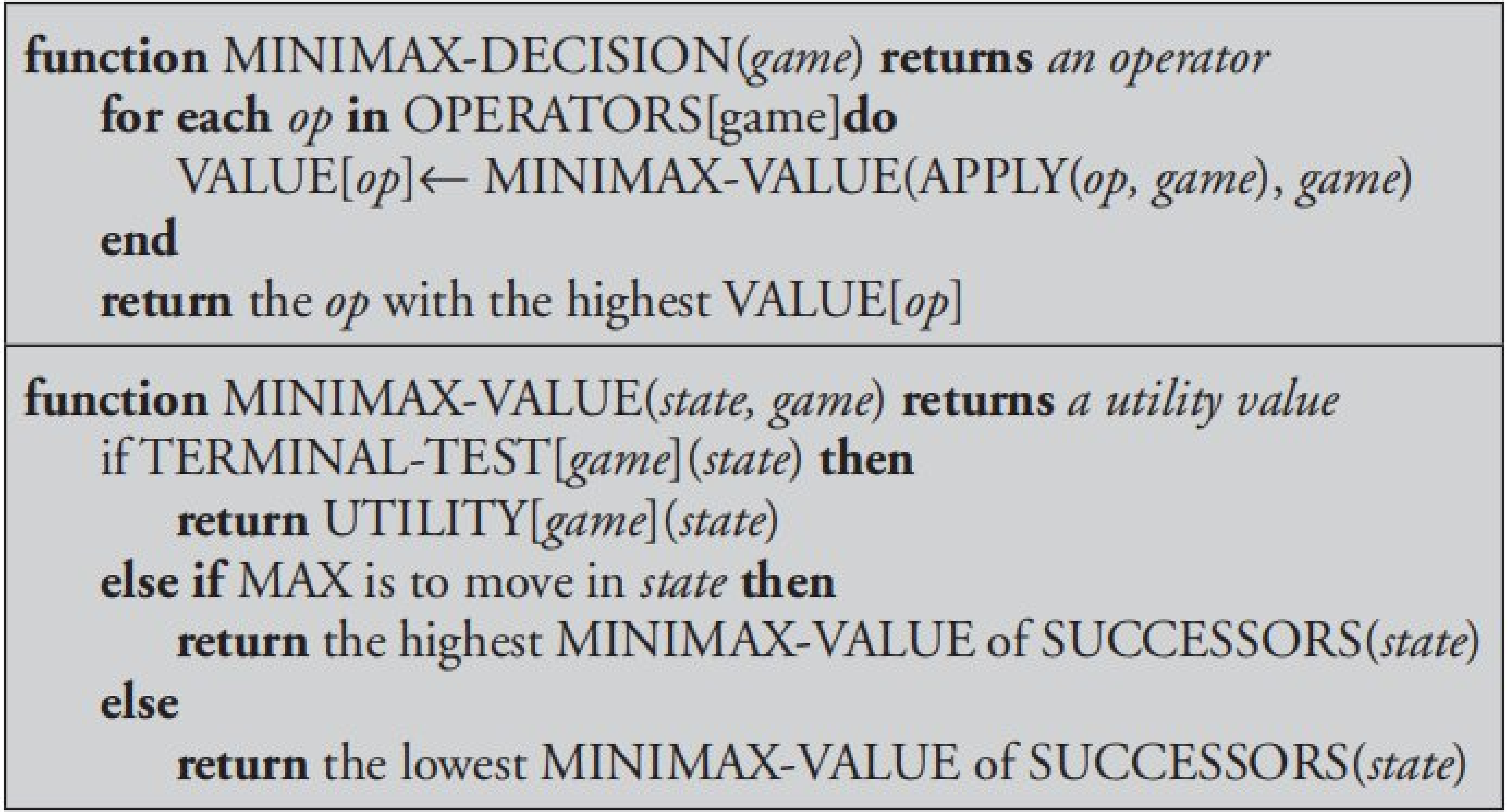
Optimal strategies



The Mini-Max algorithm

- Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory.
- It provides an optimal move for the player assuming that opponent is also playing optimally.
- Min-Max algorithm is mostly used for game playing in AI, This Algorithm computes the minimax decision for the current state.
- **Max** will try to *maximize its utility* and **min** will try to *minimize its utility*.

Pseudo code



```
function MINIMAX-DECISION(game) returns an operator
    for each op in OPERATORS[game]do
        VALUE[op]← MINIMAX-VALUE(APPLY(op, game), game)
    end
    return the op with the highest VALUE[op]
```

```
function MINIMAX-VALUE(state, game) returns a utility value
    if TERMINAL-TEST[game](state) then
        return UTILITY[game](state)
    else if MAX is to move in state then
        return the highest MINIMAX-VALUE of SUCCESSORS(state)
    else
        return the lowest MINIMAX-VALUE of SUCCESSORS(state)
```

Algorithm

Step 1: In case of search reaching its limit, the static value of the current position relative to the appropriate player is calculated. The result is reported.

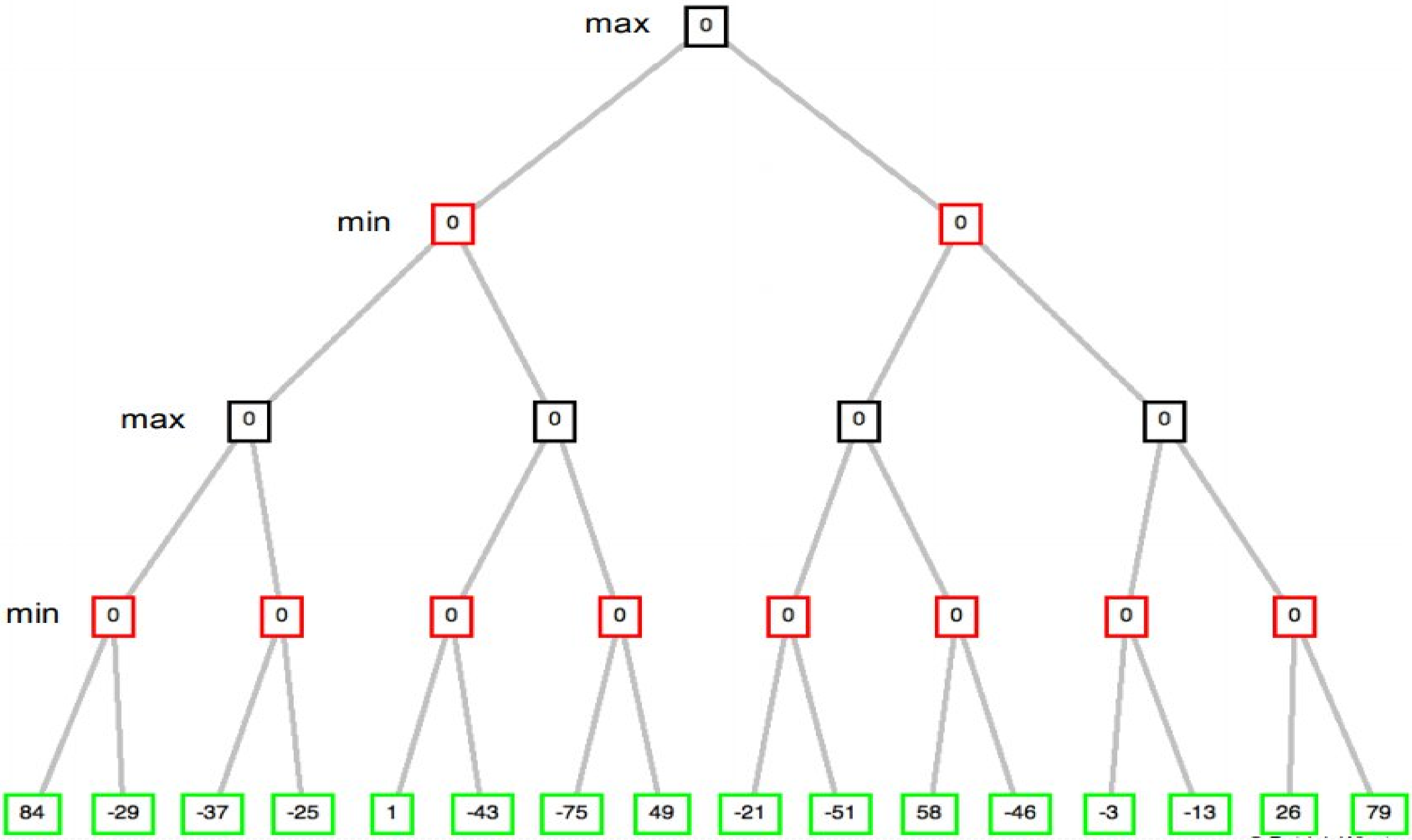
Step 2: Otherwise, if the level is a minimizing level, use the minimax on the children of the current position. The minimum value of the results is reported here

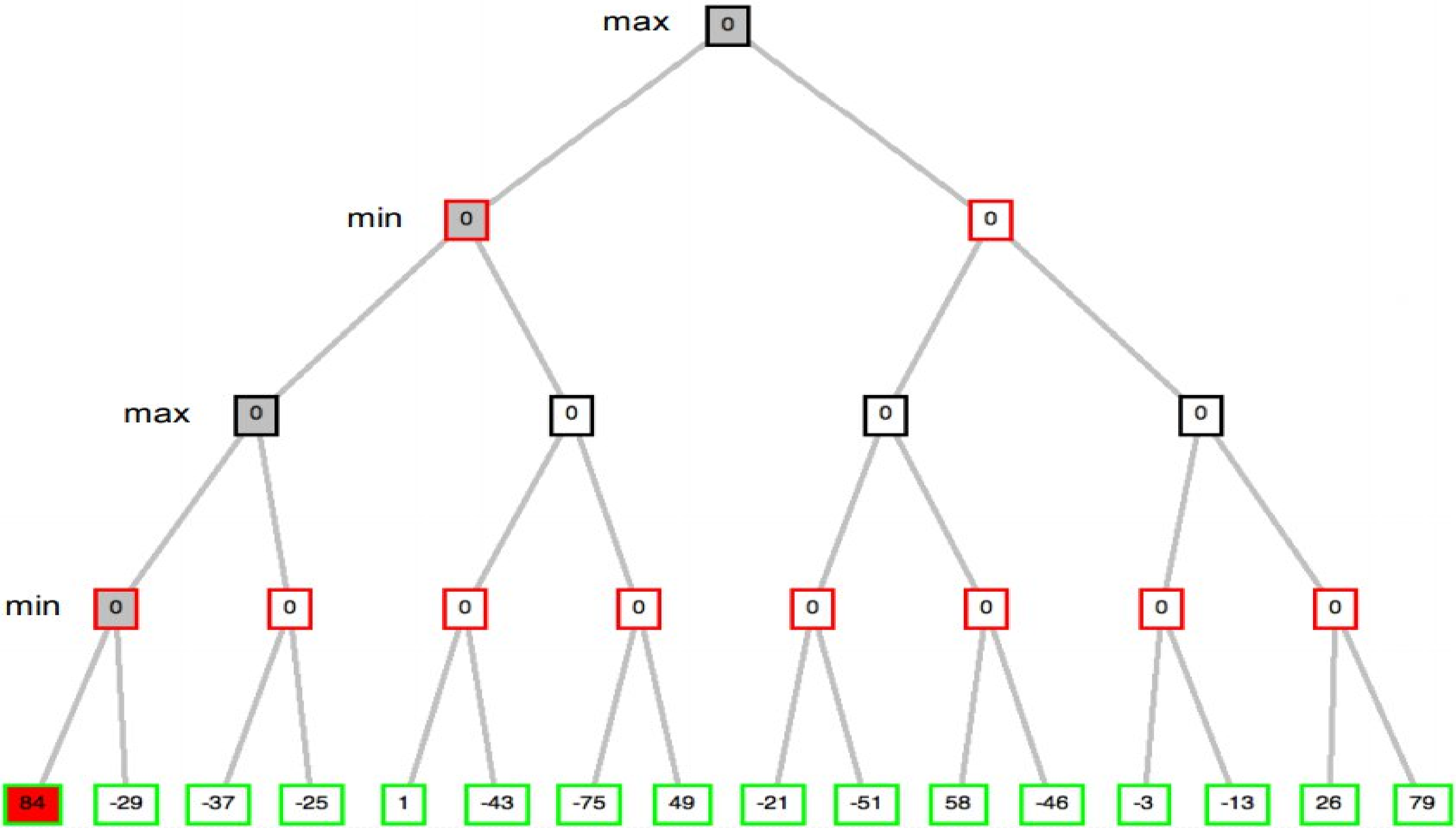
Step 3: Also, if the level is a maximizing level, use the minimax on the children of the current position. The maximum of the results is reported here.

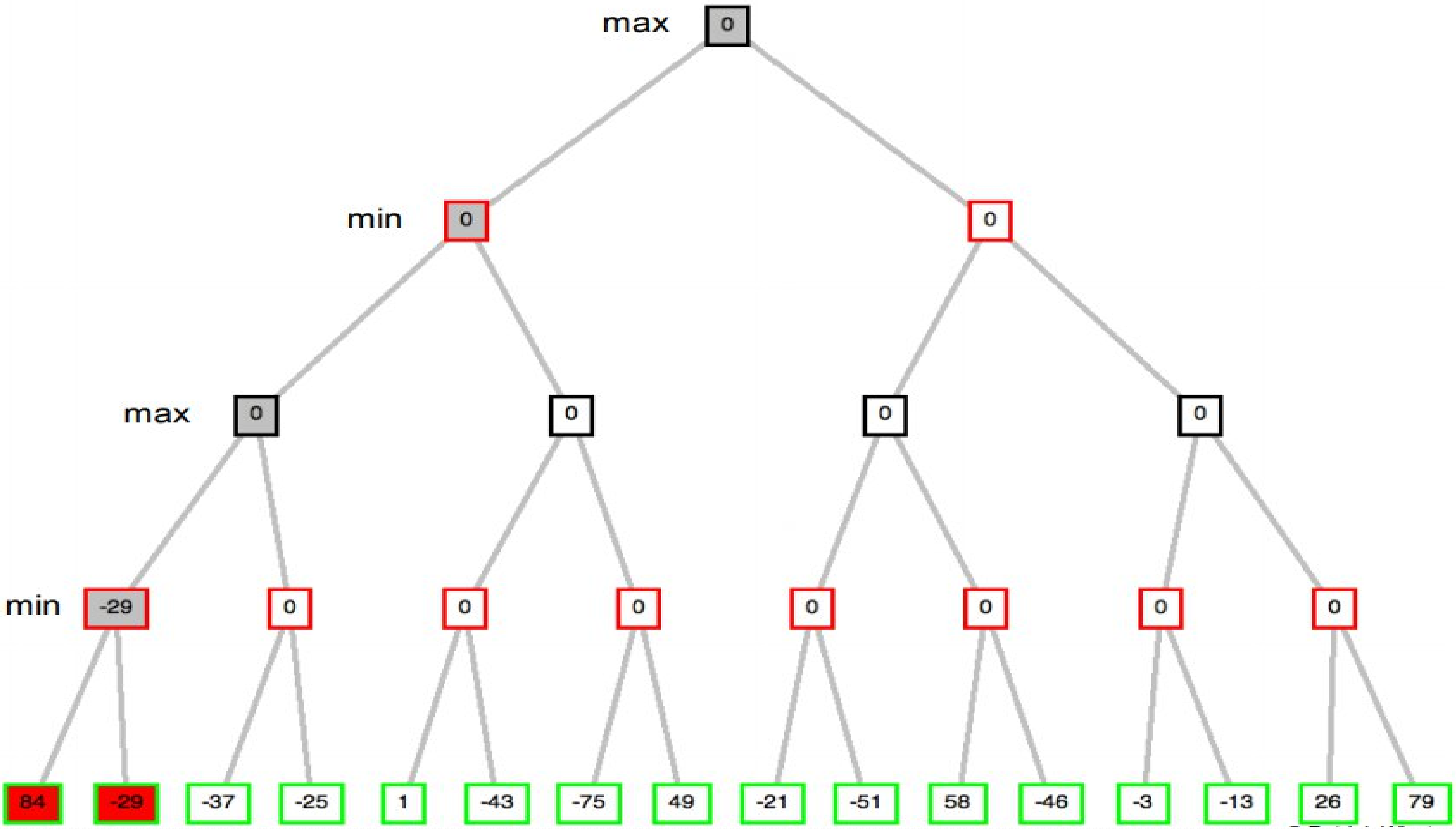
Step 4: The utility values is calculated with the help of leaves considering one layer at a time until the root of the tree.

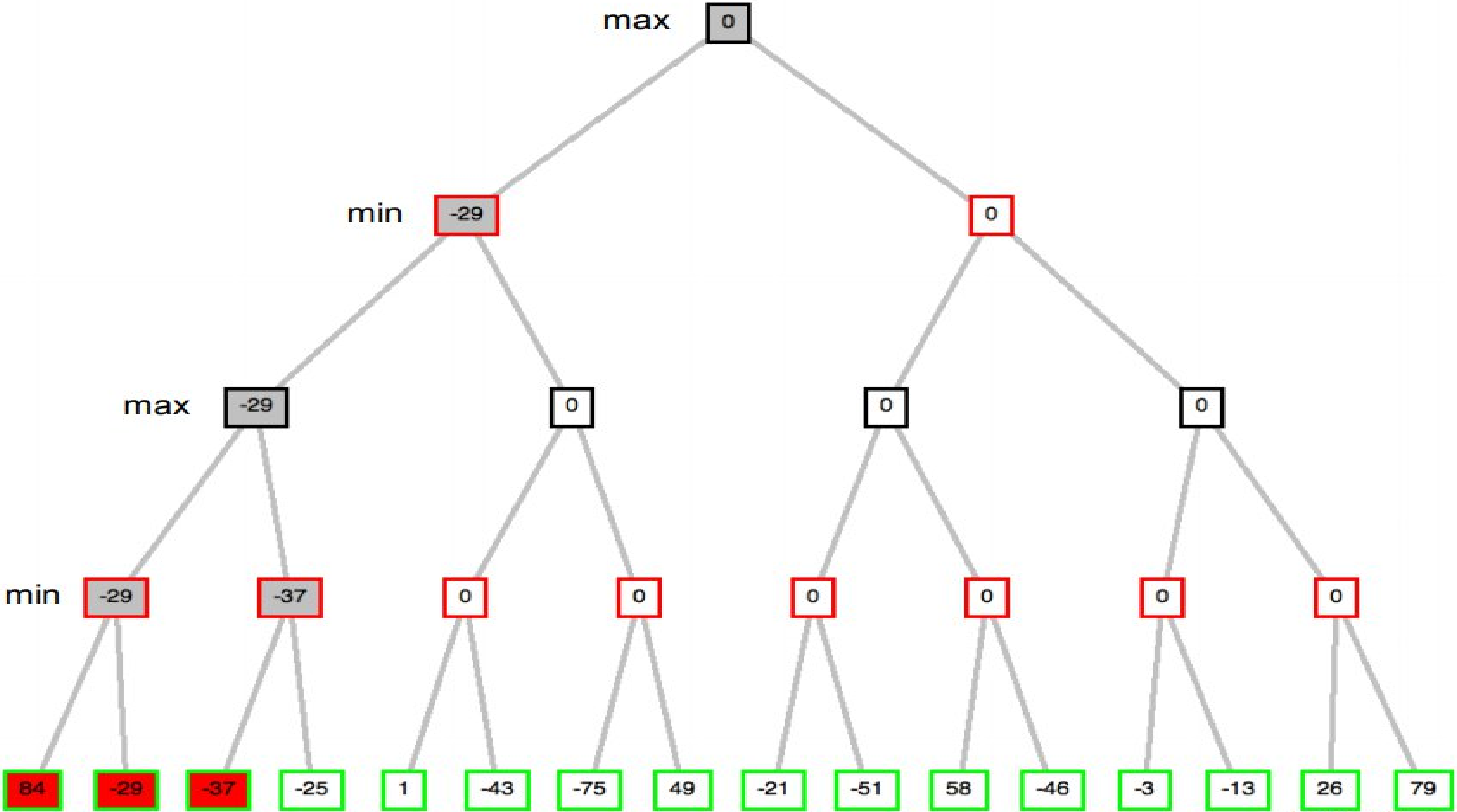
Step 5: Eventually, all the backed-up values reach to the root of the tree, that is, the topmost point. At that point, MAX is responsible for choosing the highest value.

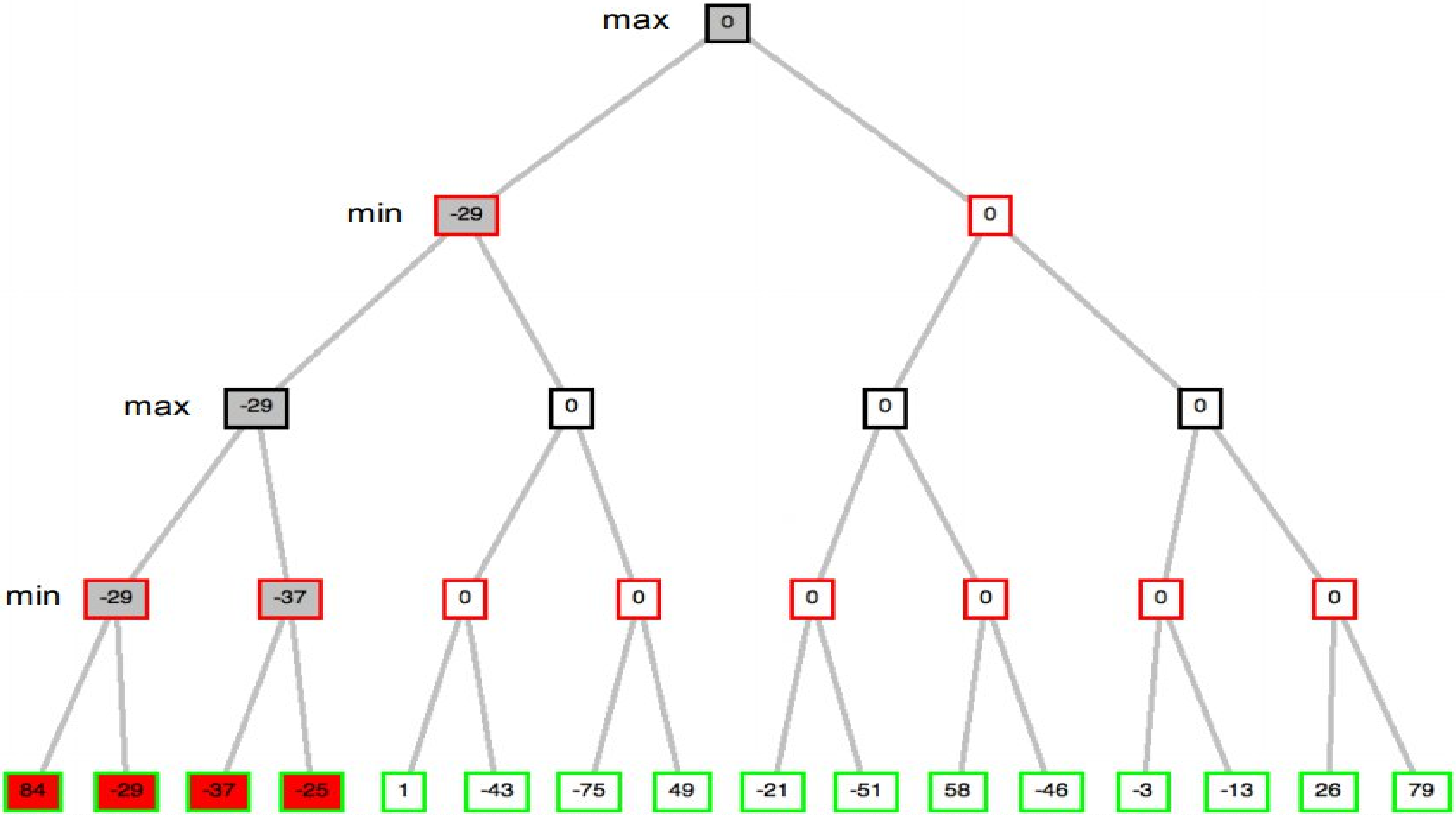
Example

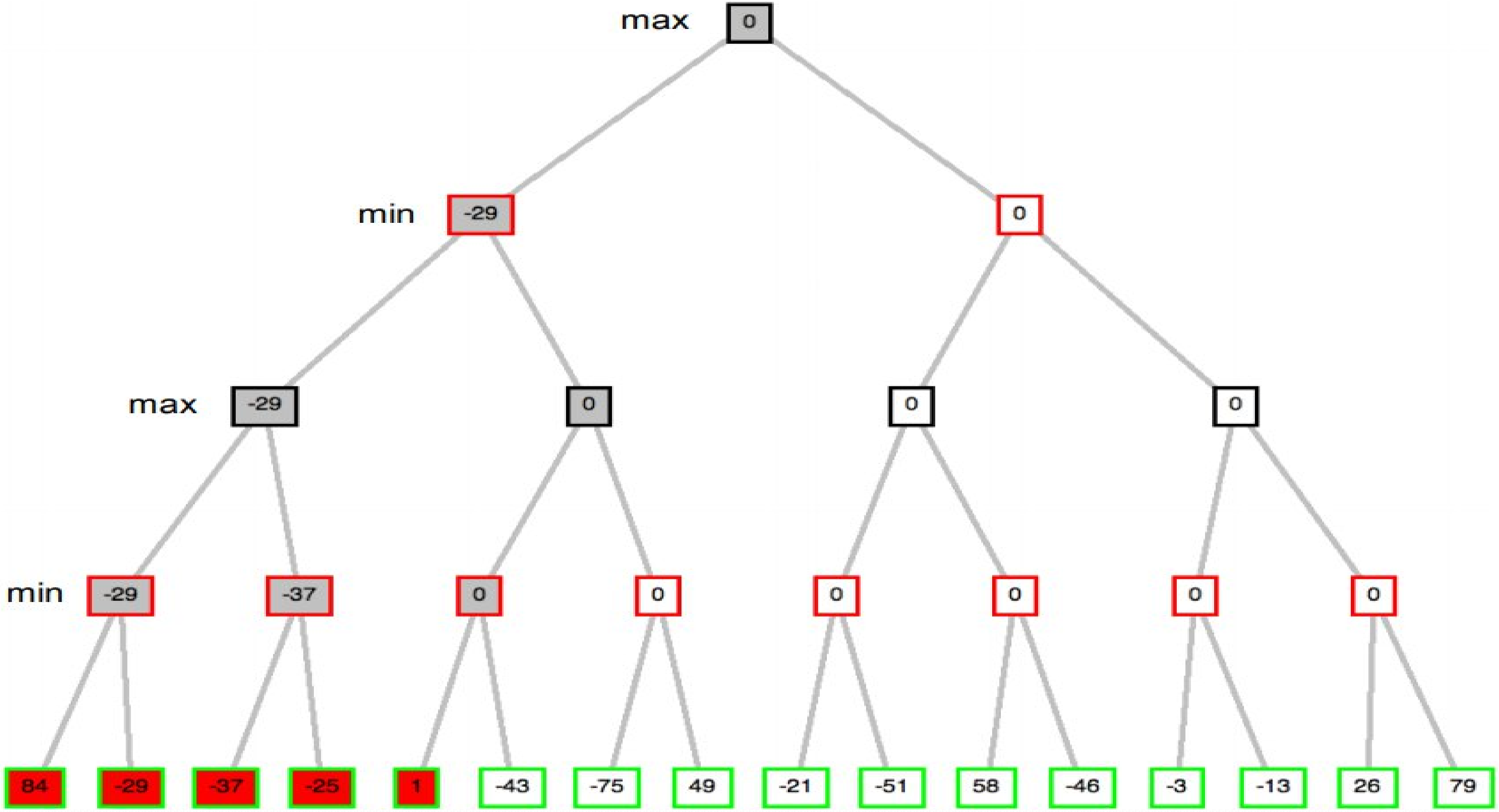

max
0
min
0
0
max
0
0
0
0
min
0
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

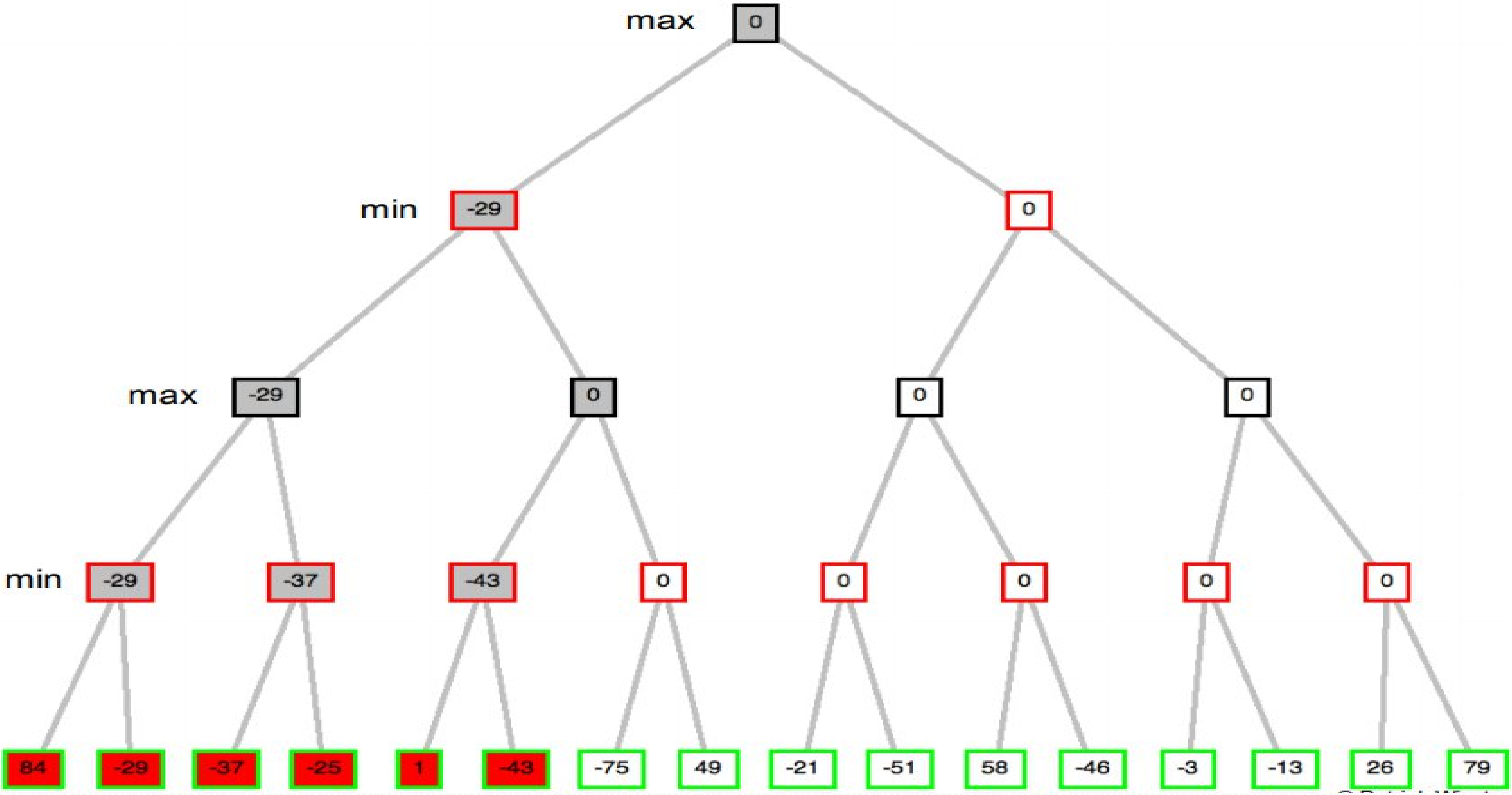

max
0
min
0
0
max
0
0
0
0
min
0
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

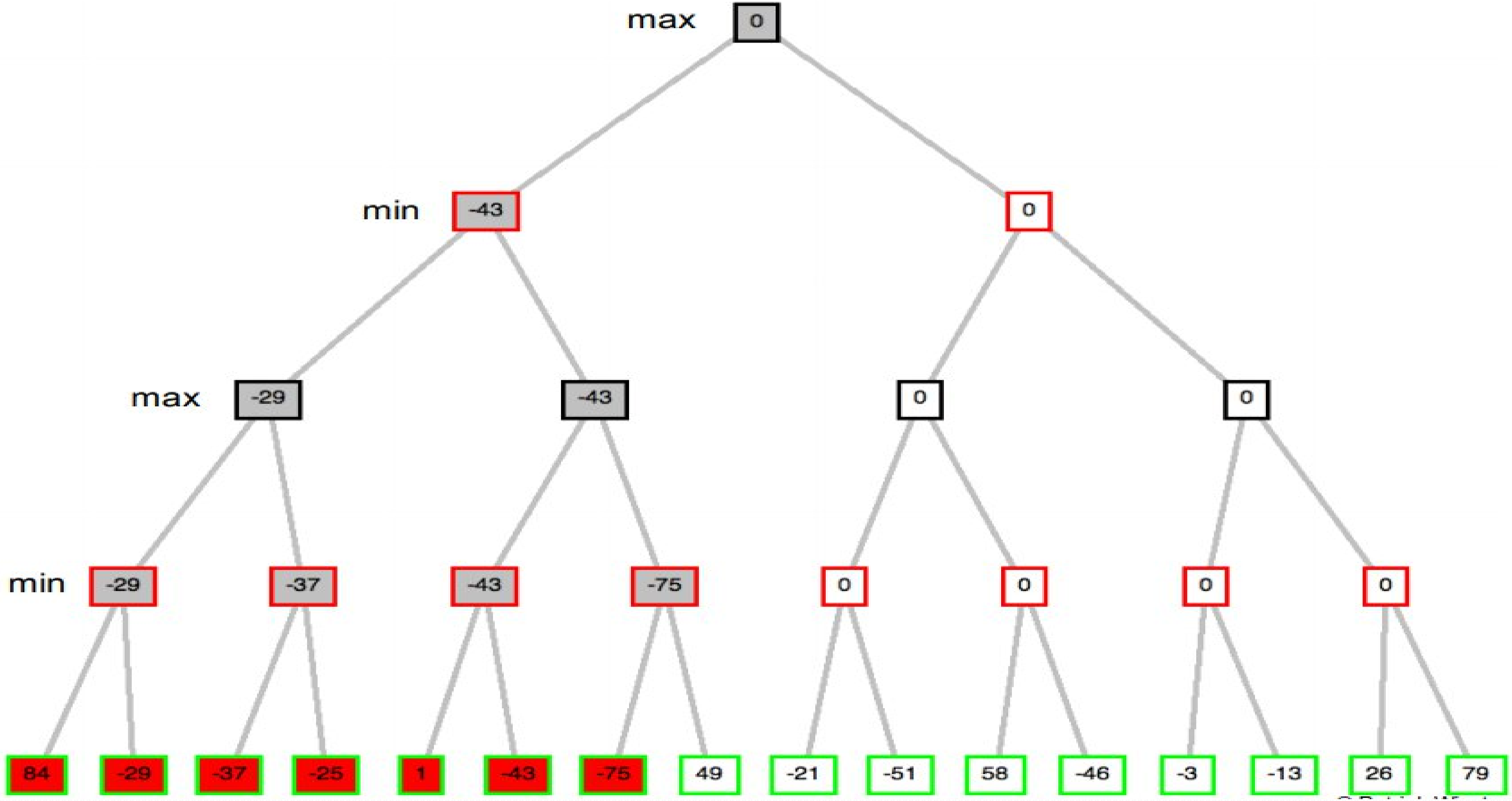

max
0
min
0
0
max
0
0
0
0
min
-29
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

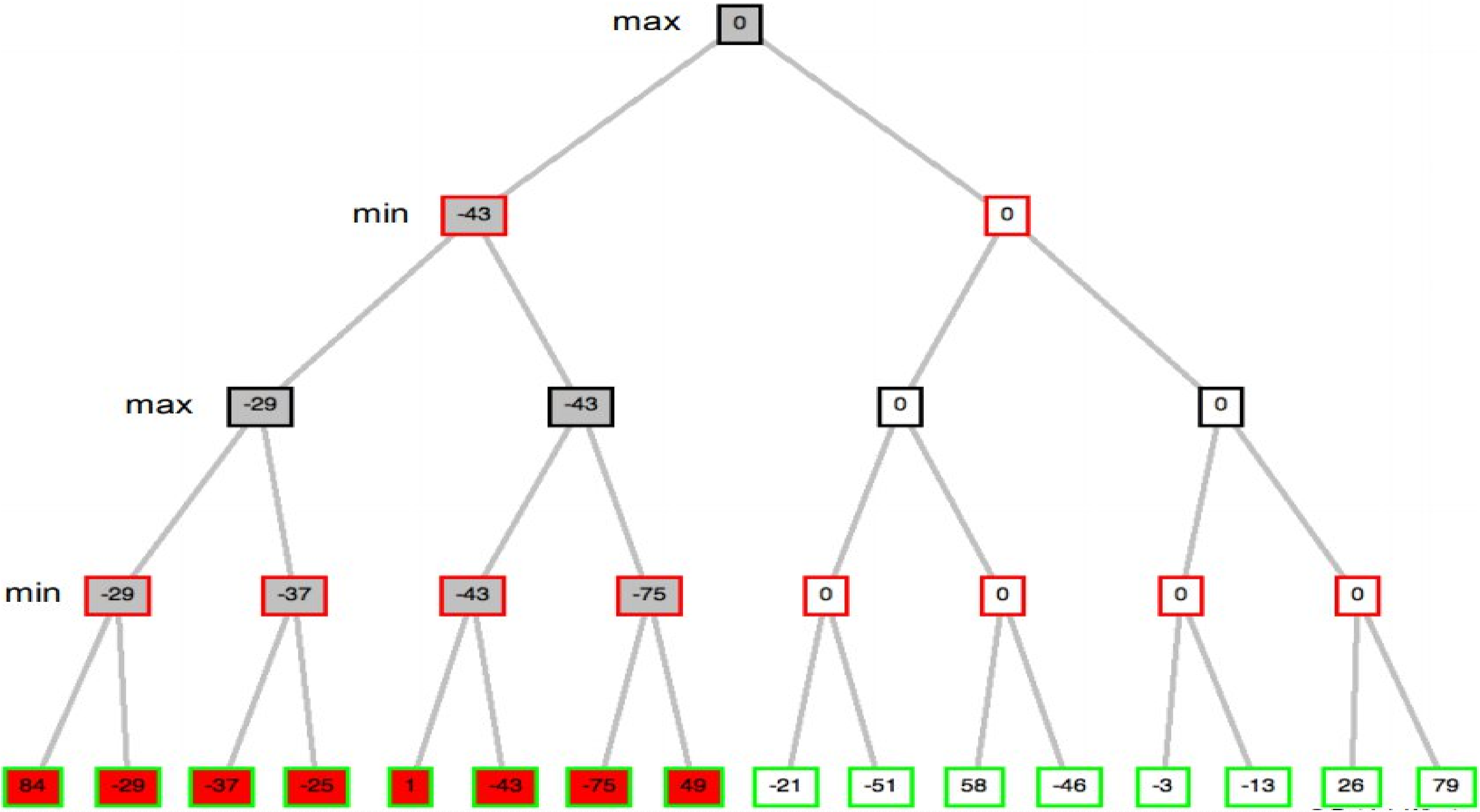

max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

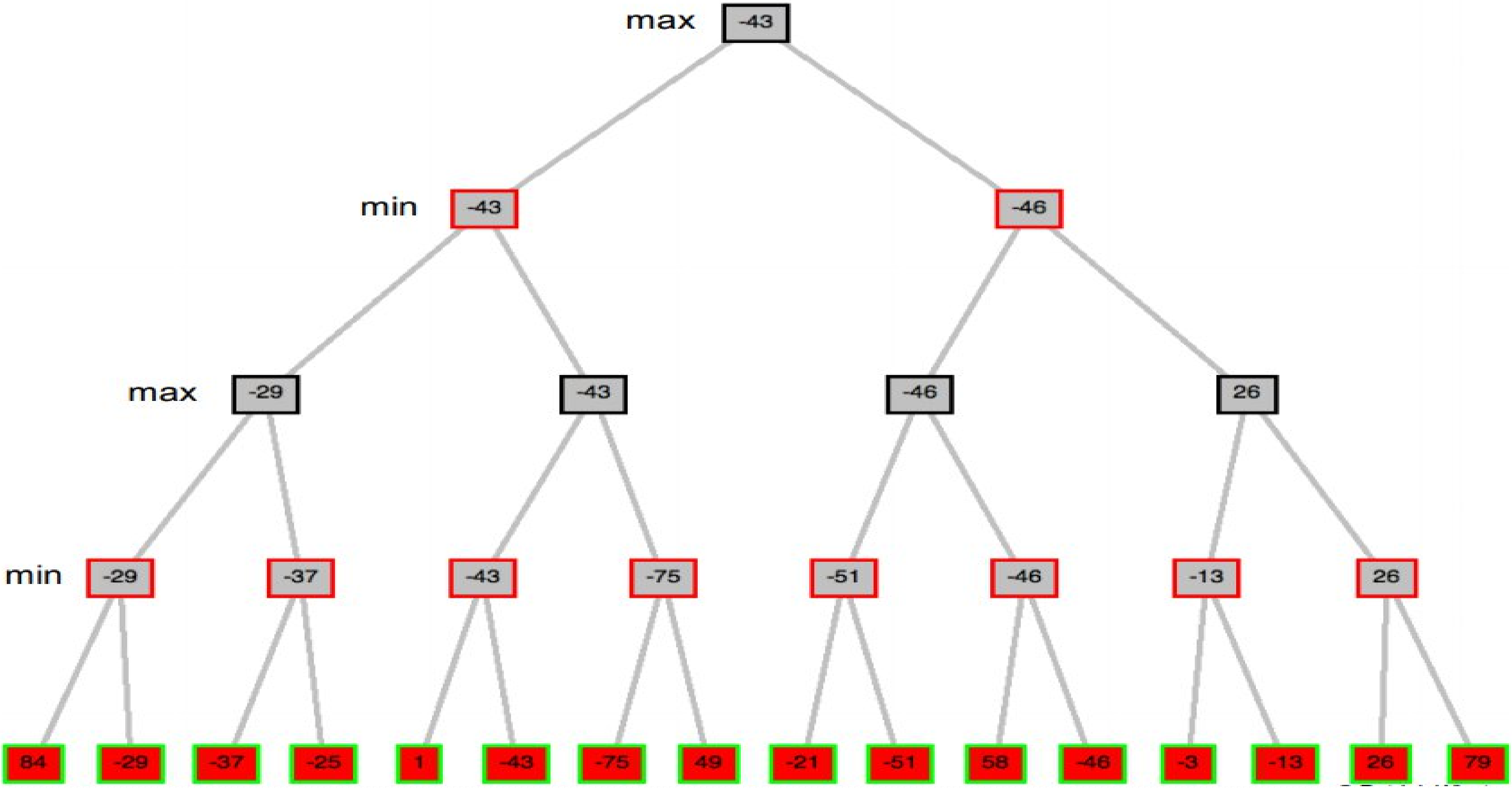

max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

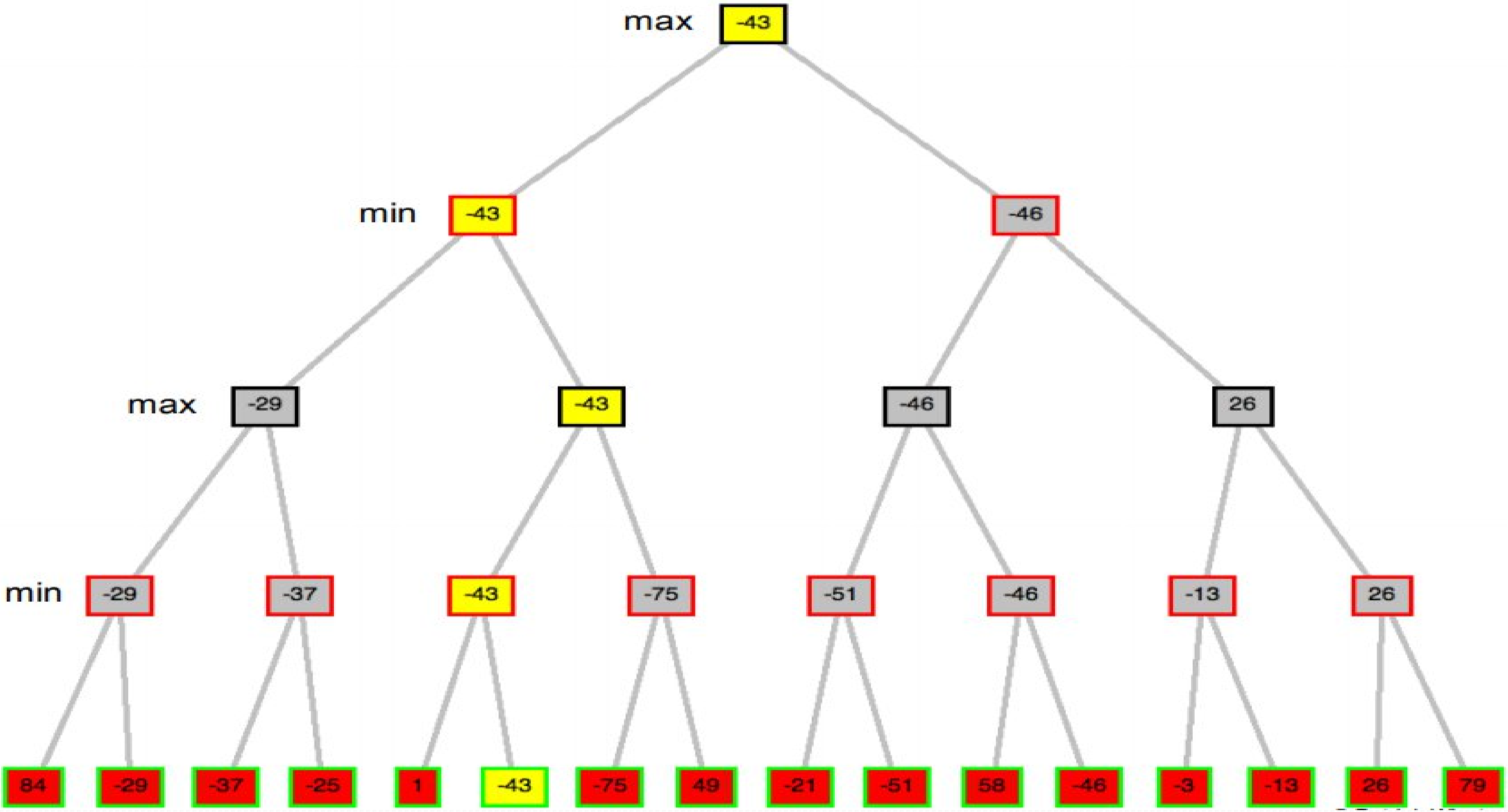

max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79


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

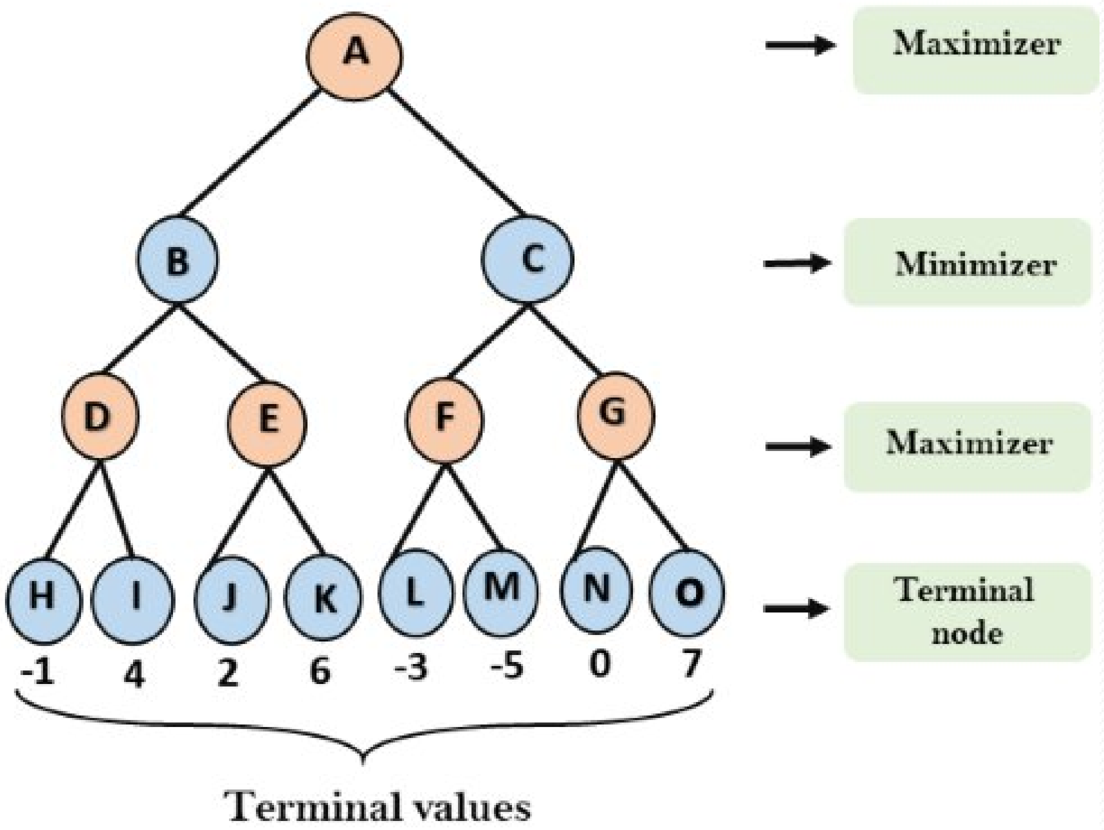

max
0
min
-43
0
max
-29
-43
0
0
min
-29
-37
-43
-75
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

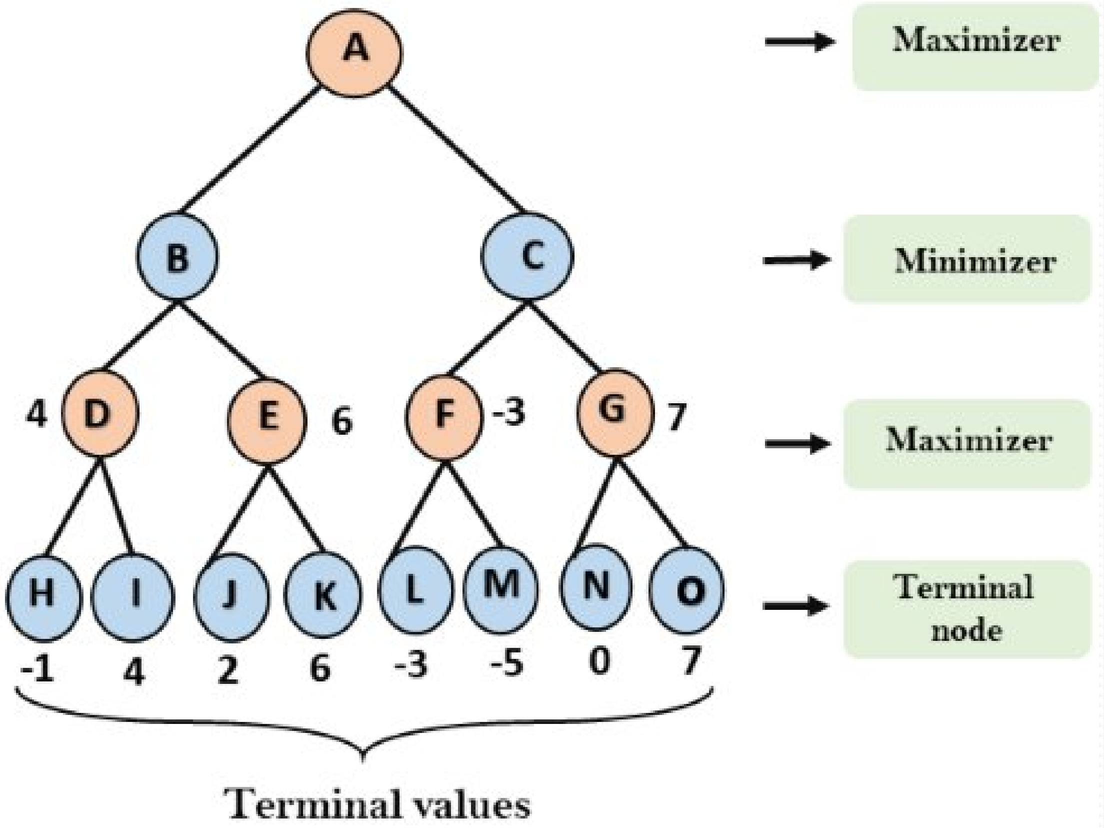

max
0
min
-43
0
max
-29
-43
0
0
min
-29
-37
-43
-75
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

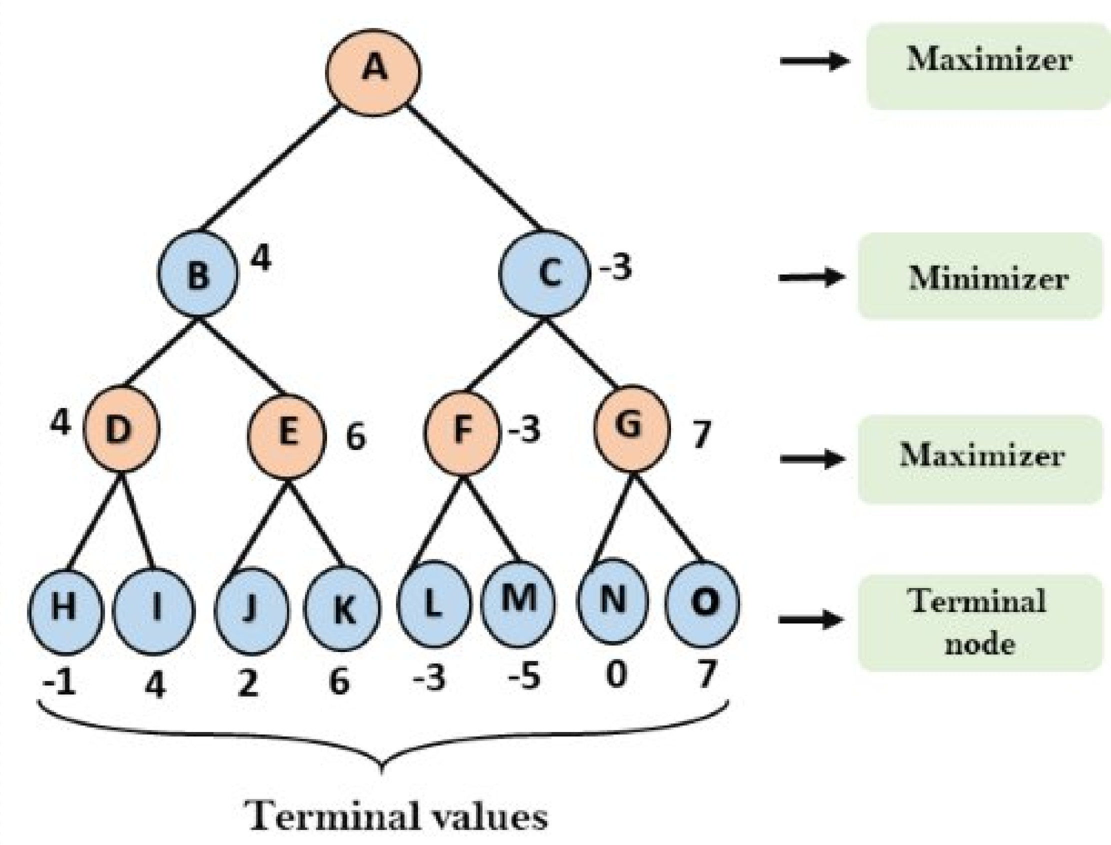

max
-43
min
-43
-46
max
-29
-43
-46
26
min
-29
-37
-43
-75
-51
-46
-13
26
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79


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

Example


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node


A
B
C
4
D
E
6
F
-3
G
7
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node



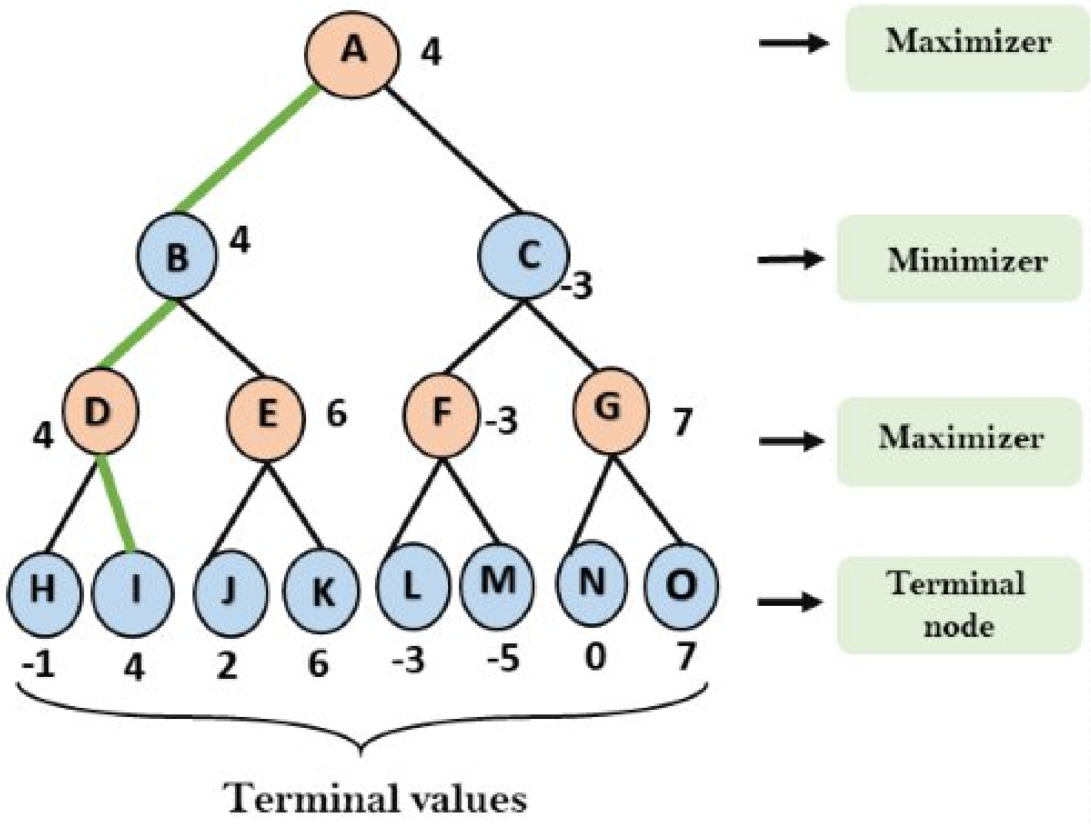
→ Maximizer

→ Minimizer

→ Maximizer

→ Terminal node

Terminal values



→ Maximizer

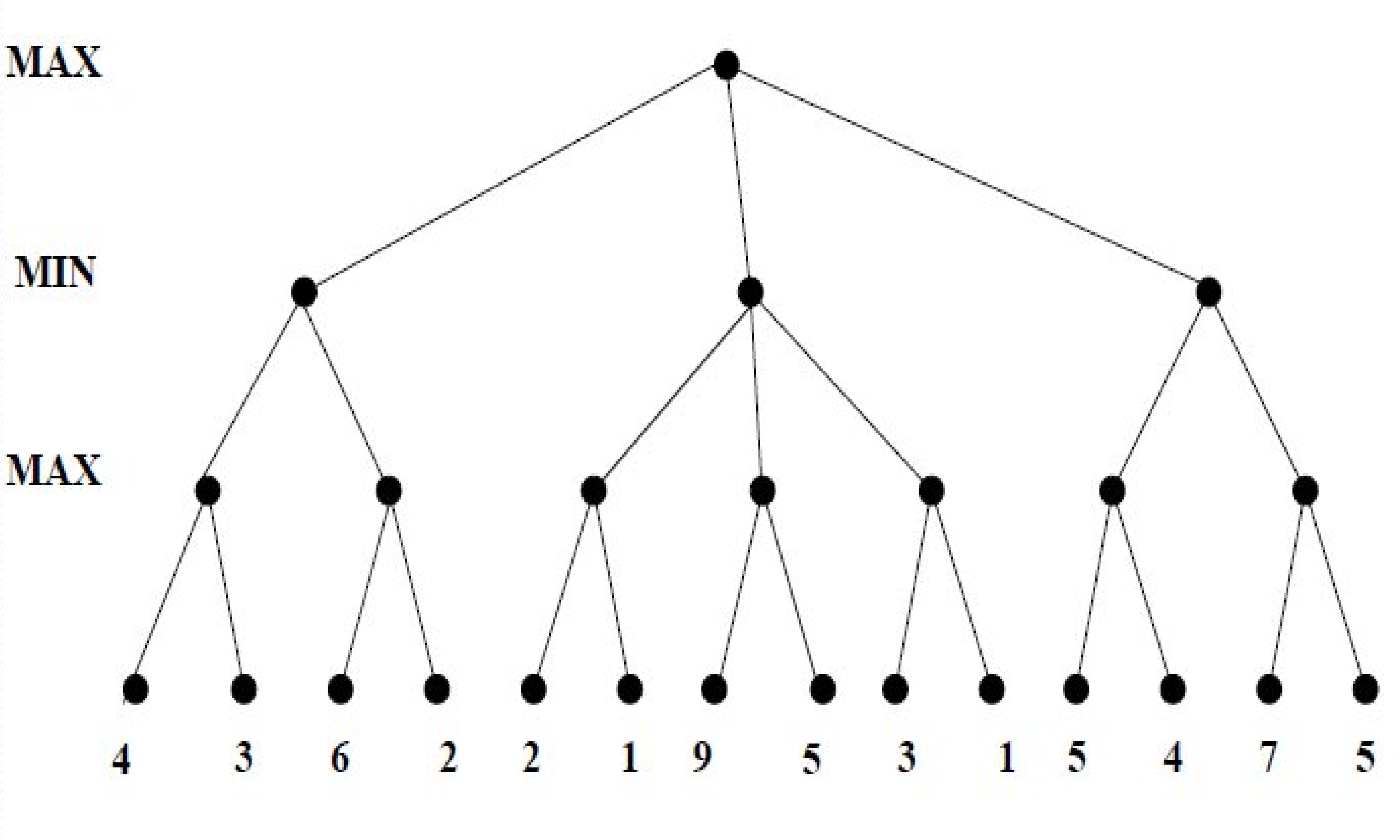
→ Minimizer

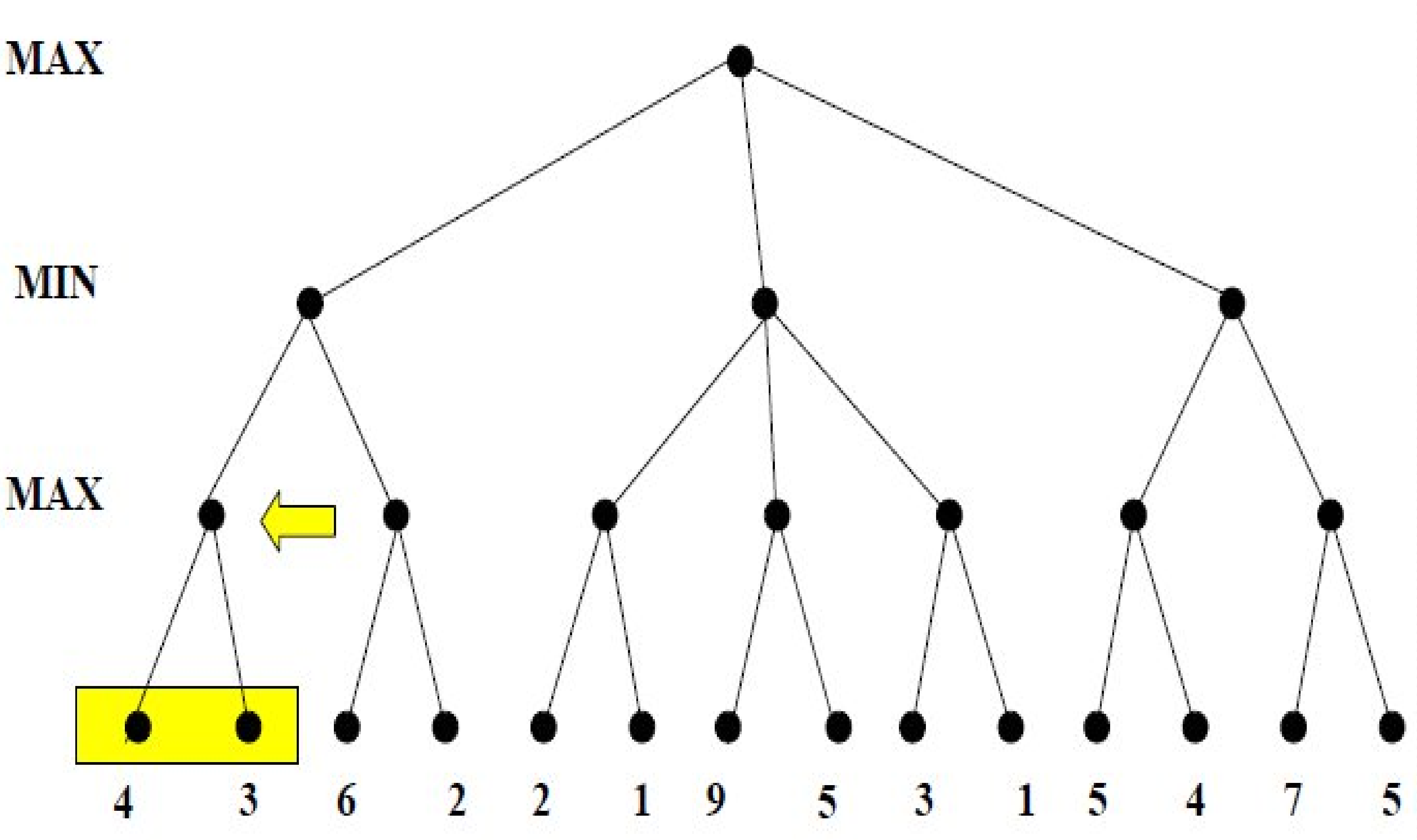
→ Maximizer

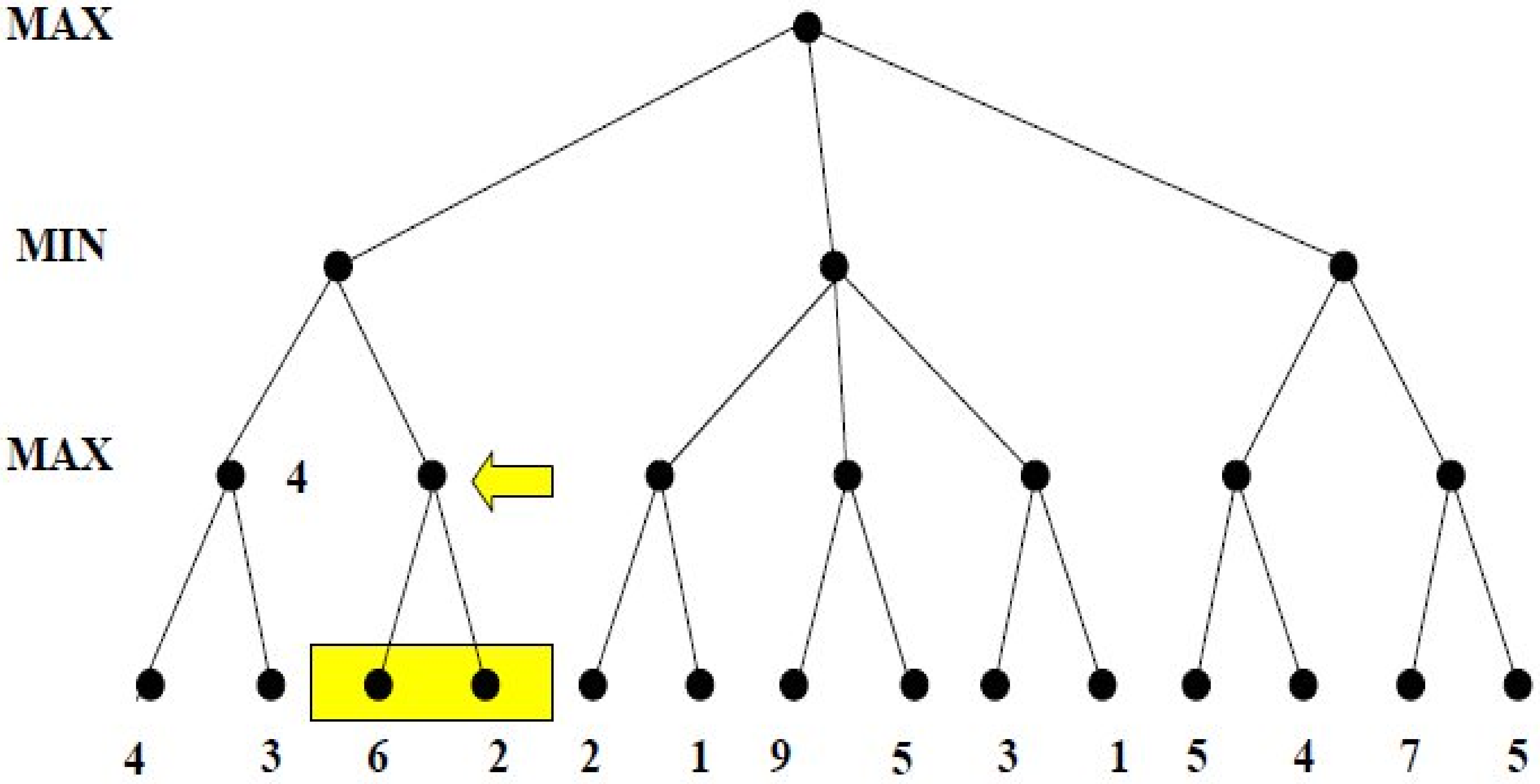
→ Terminal node

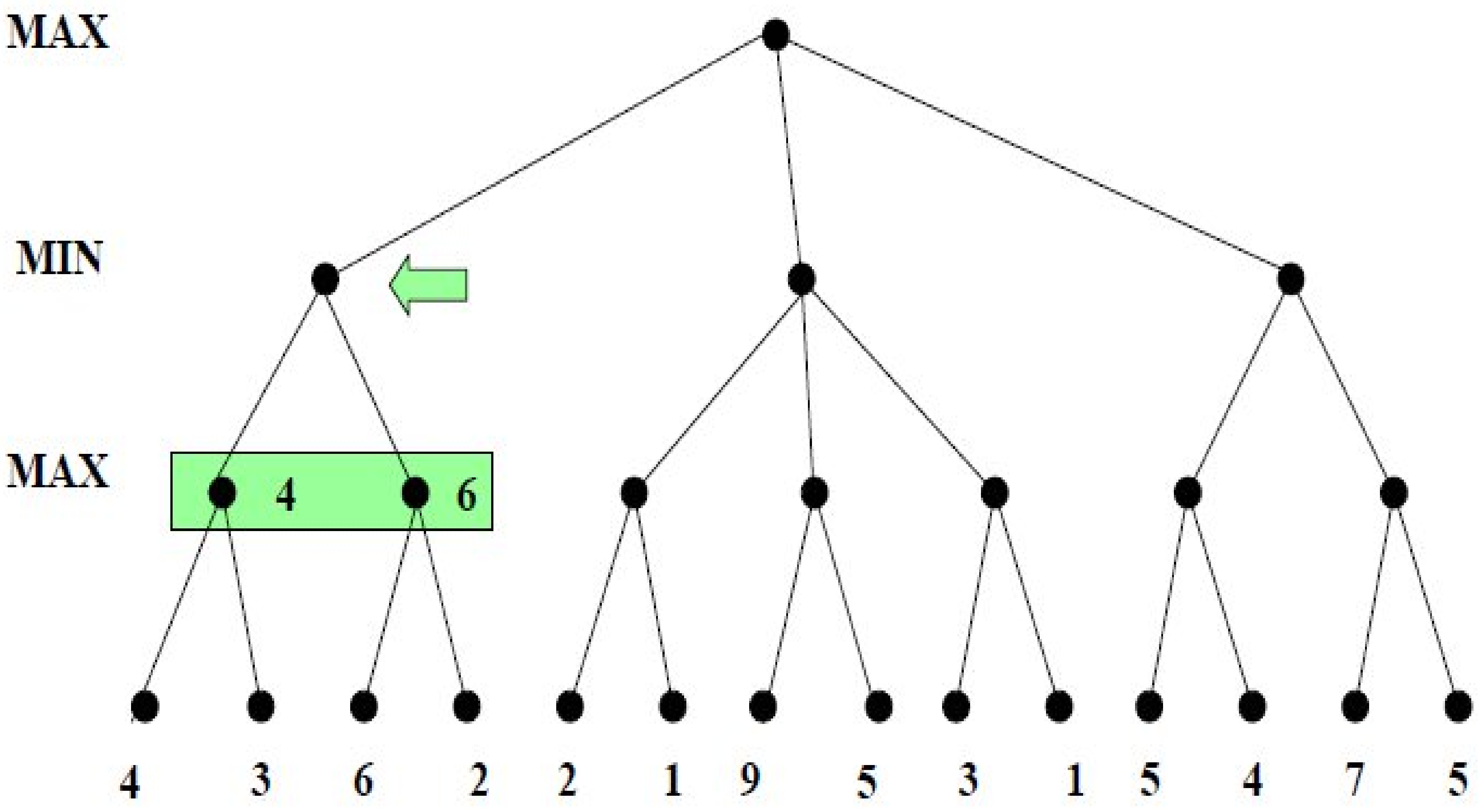
Terminal values

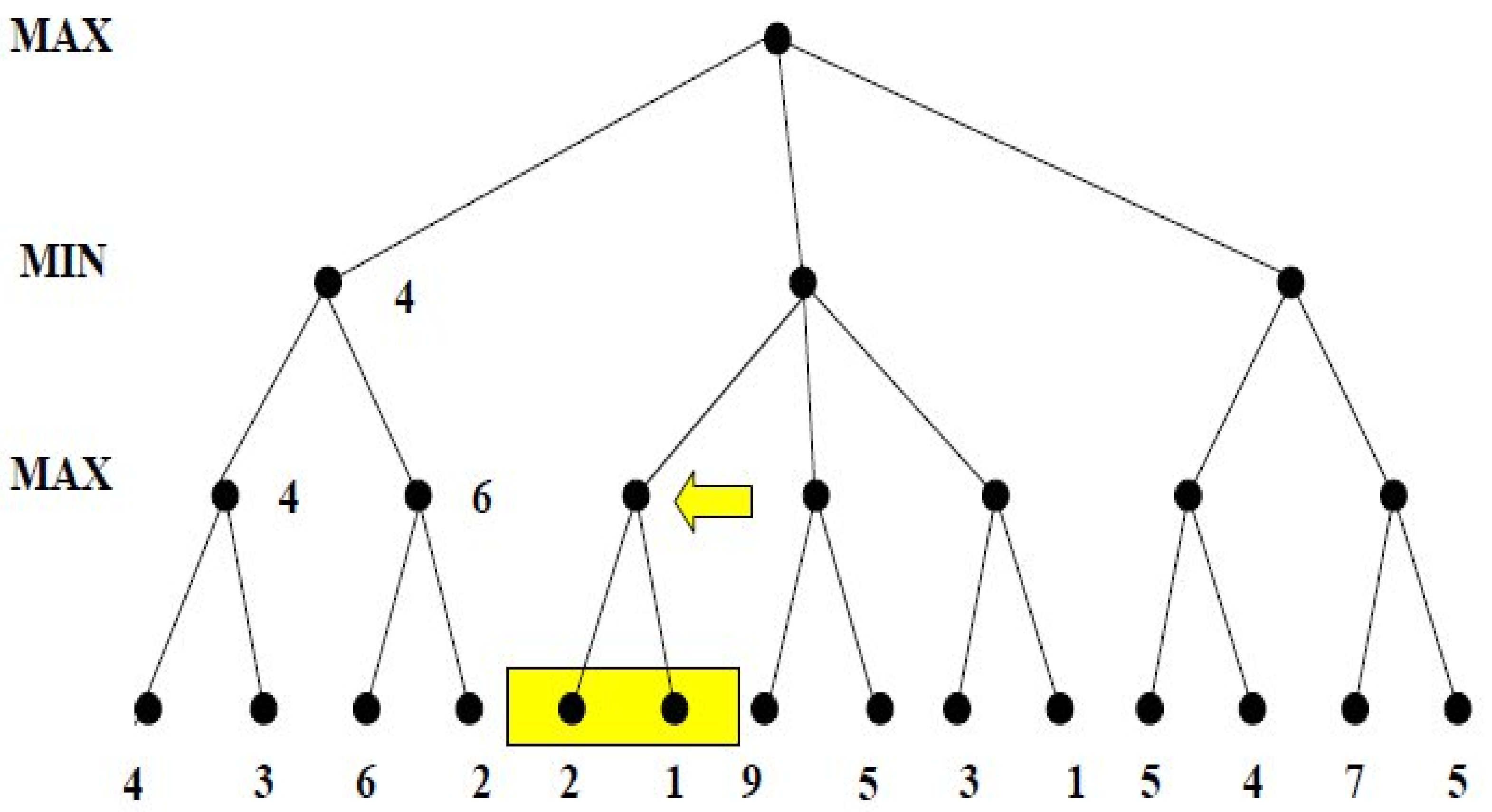
Example

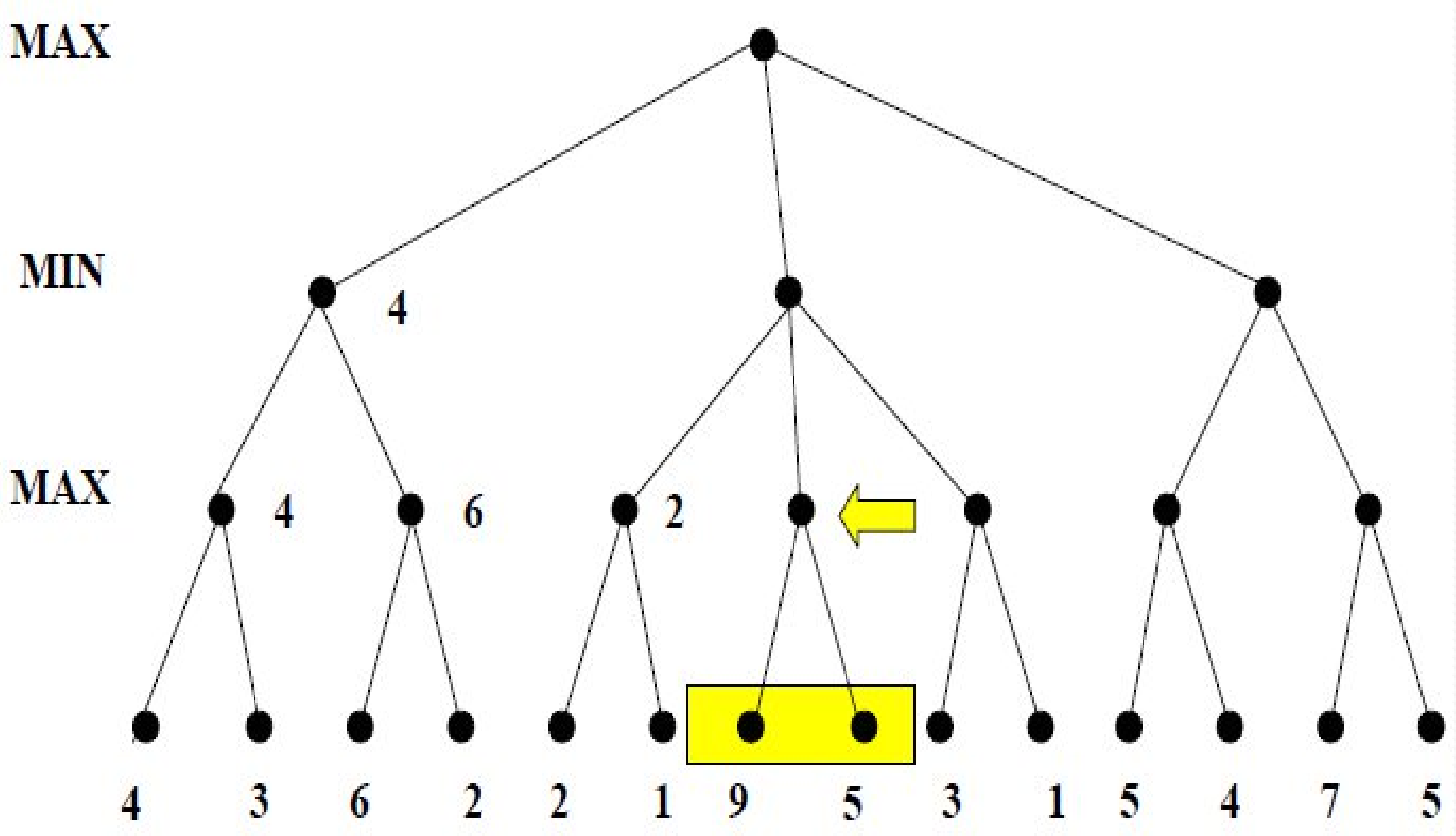


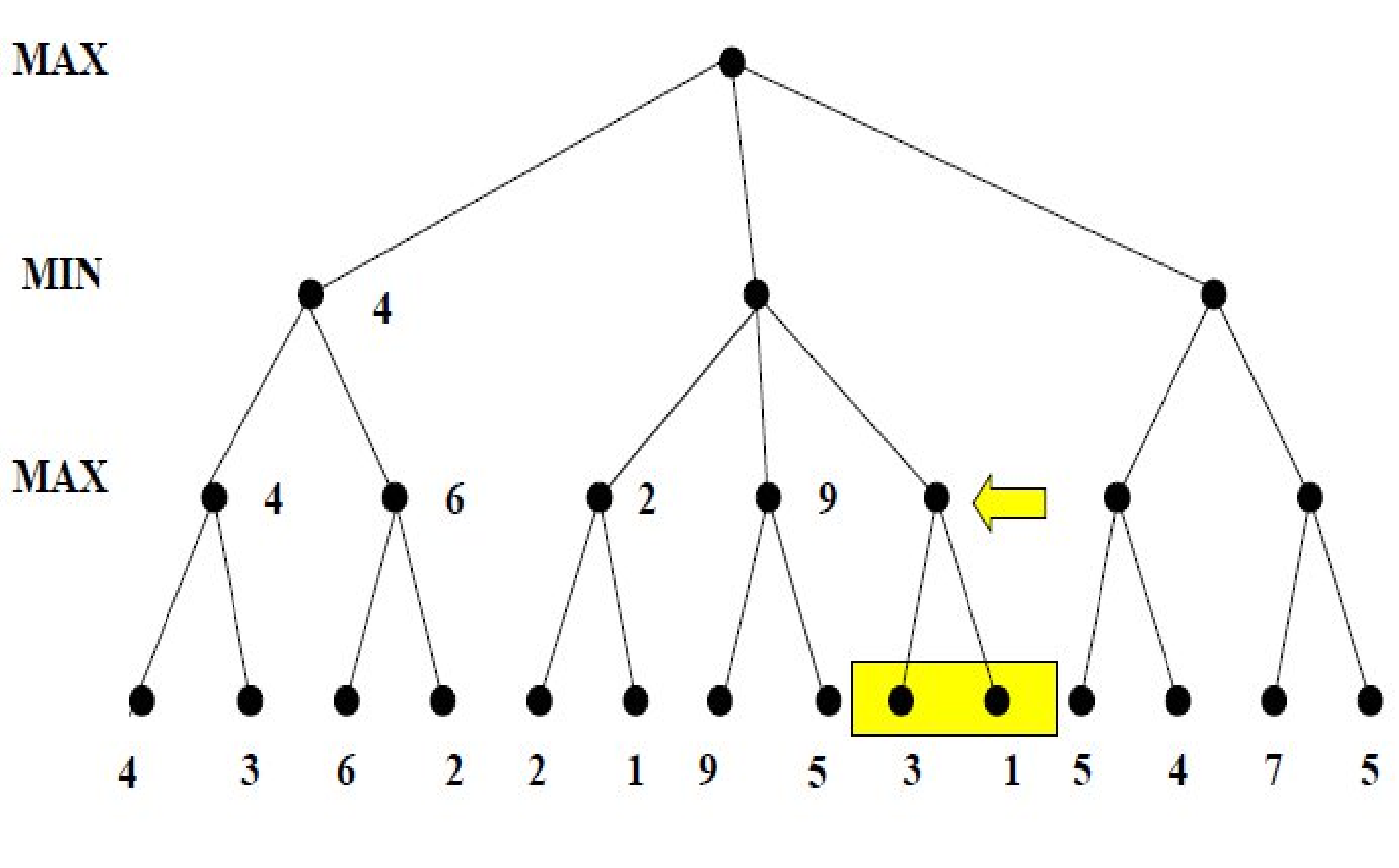

MAX
MIN
MAX
4
3
6
2
2
1
9
5
3
1
5
4
7
5

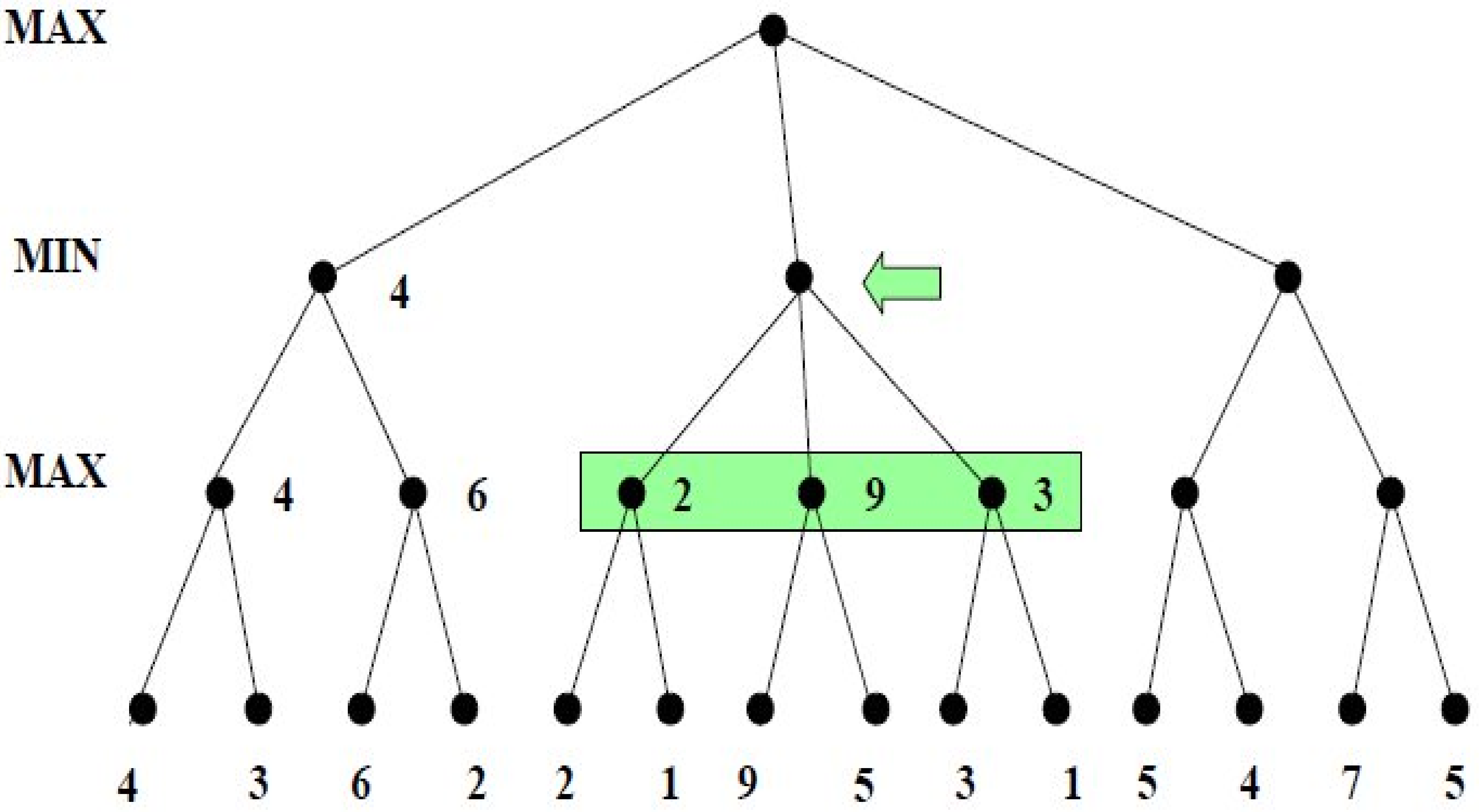

MAX
MIN
MAX
4
4
3
6
2
2
1
9
5
3
1
5
4
7
5

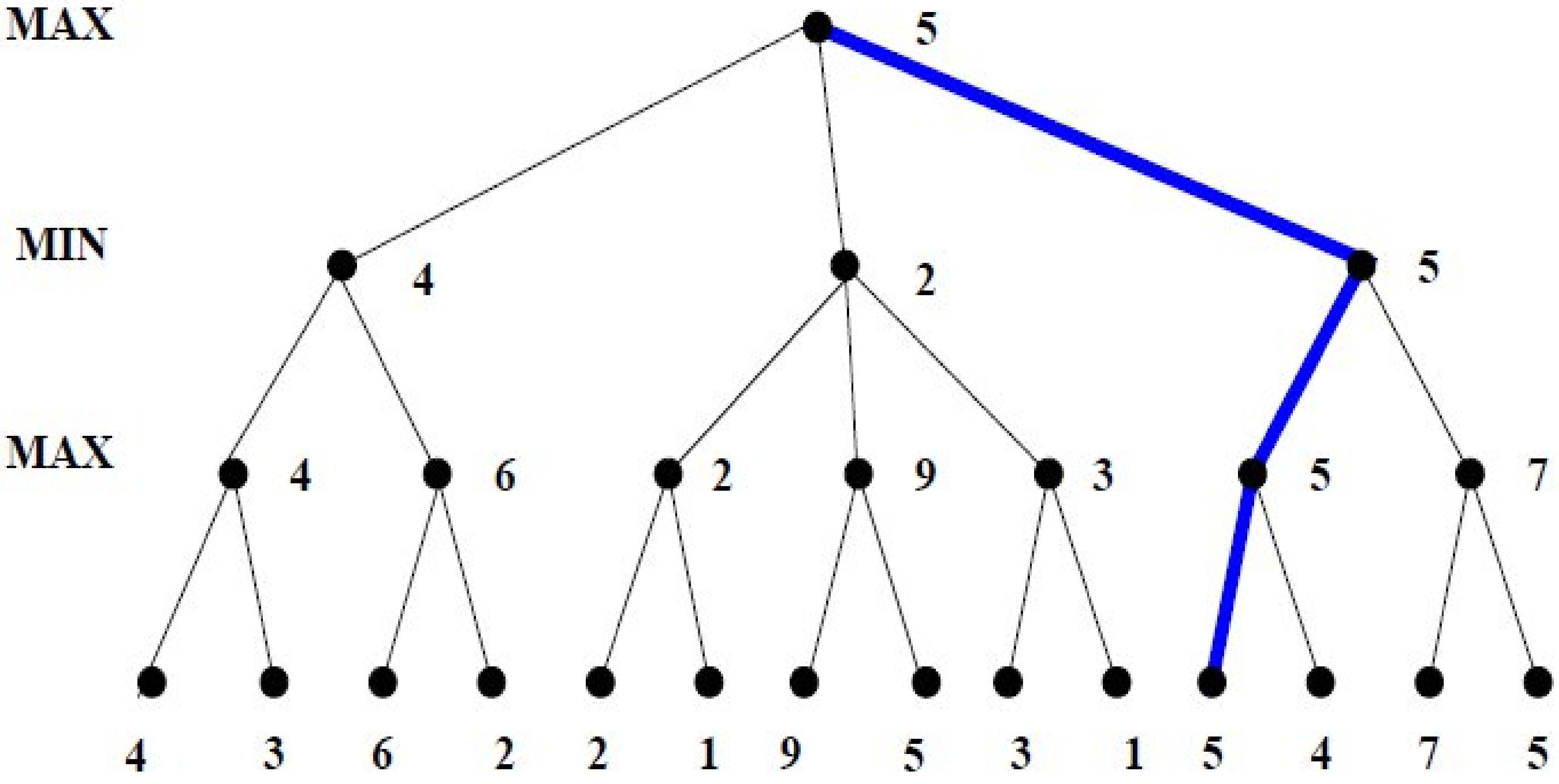

MAX
MIN
MAX
4
6
4
3
6
2
2
1
9
5
3
1
5
4
7
5


MAX
MIN
MAX
4
4
6
4
3
6
2
2
1
9
5
3
1
5
4
7
5


MAX
MIN
MAX
4
4
6
2
4
3
6
2
2
1
9
5
3
1
5
4
7
5


MAX
MIN
MAX
4
4
6
2
9
4
3
6
2
2
1
9
5
3
1
5
4
7
5


MAX
MIN
MAX
4
4
6
2
9
3
4
3
6
2
2
1
9
5
3
1
5
4
7
5


MAX
5
MIN
4
2
5
MAX
4
6
2
9
3
5
7
4
3
6
2
2
1
9
5
3
1
5
4
7
5

Alpha – beta pruning

- The idea behind alpha-beta pruning is to search only the most promising branches of the game tree, while ignoring those that are unlikely to lead to a better move.
- It is an advanced version of Minimax algorithm.
- It works by keeping track of two values for each player: ***alpha and beta.***

1. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is **-**∞.
2. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞.

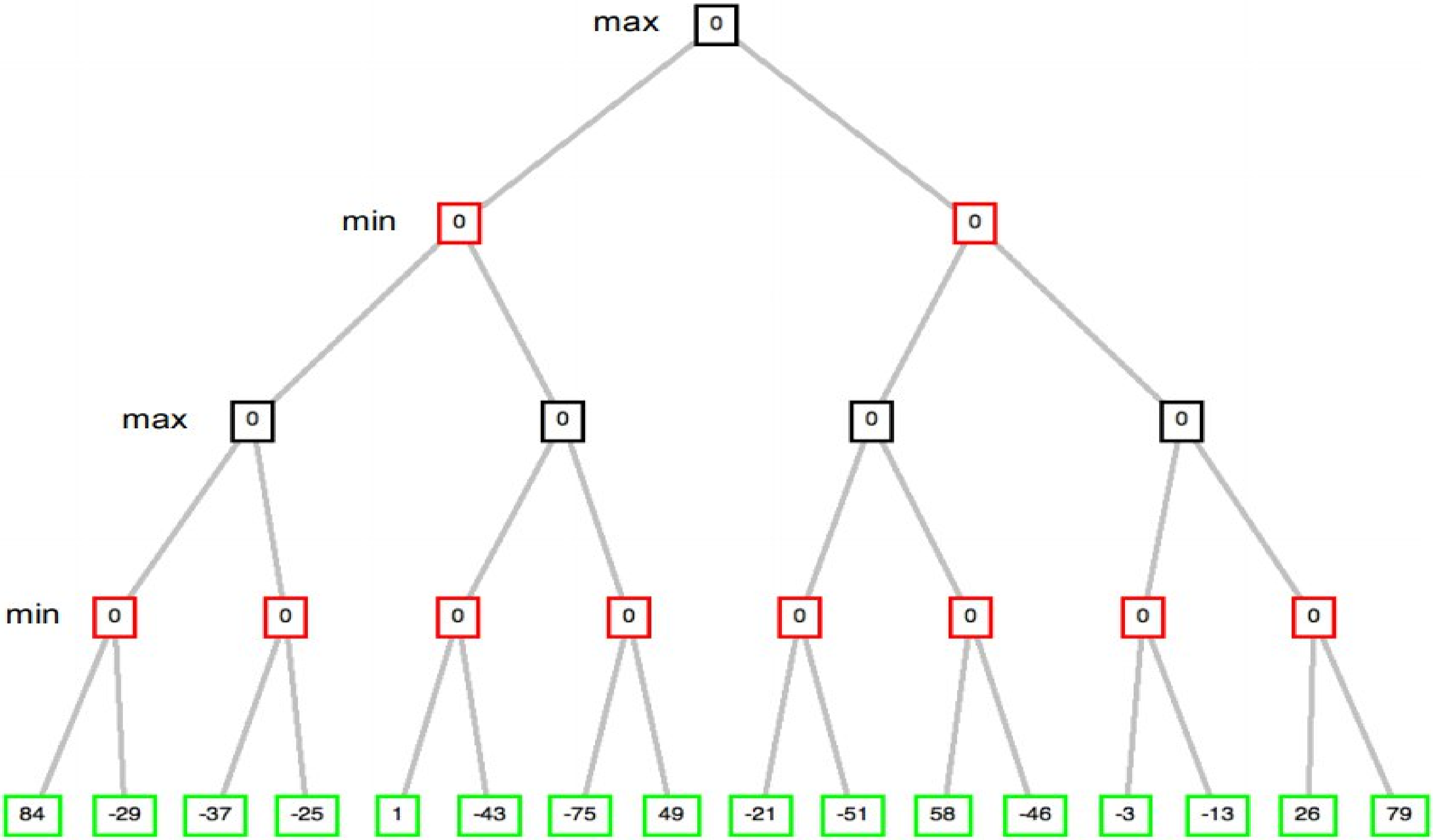
The main condition required for alpha – beta pruning is:

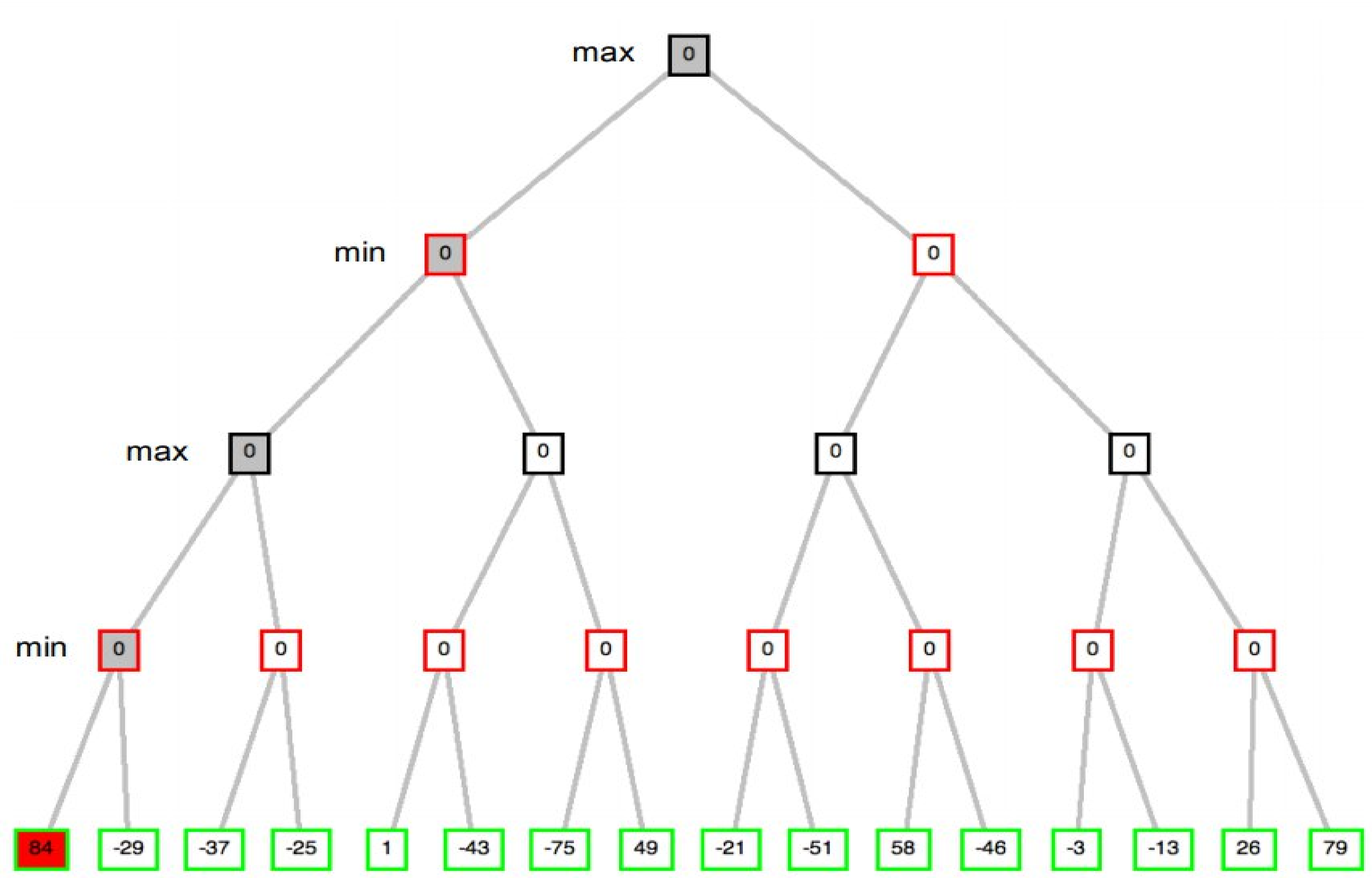
$\alpha >= \beta$

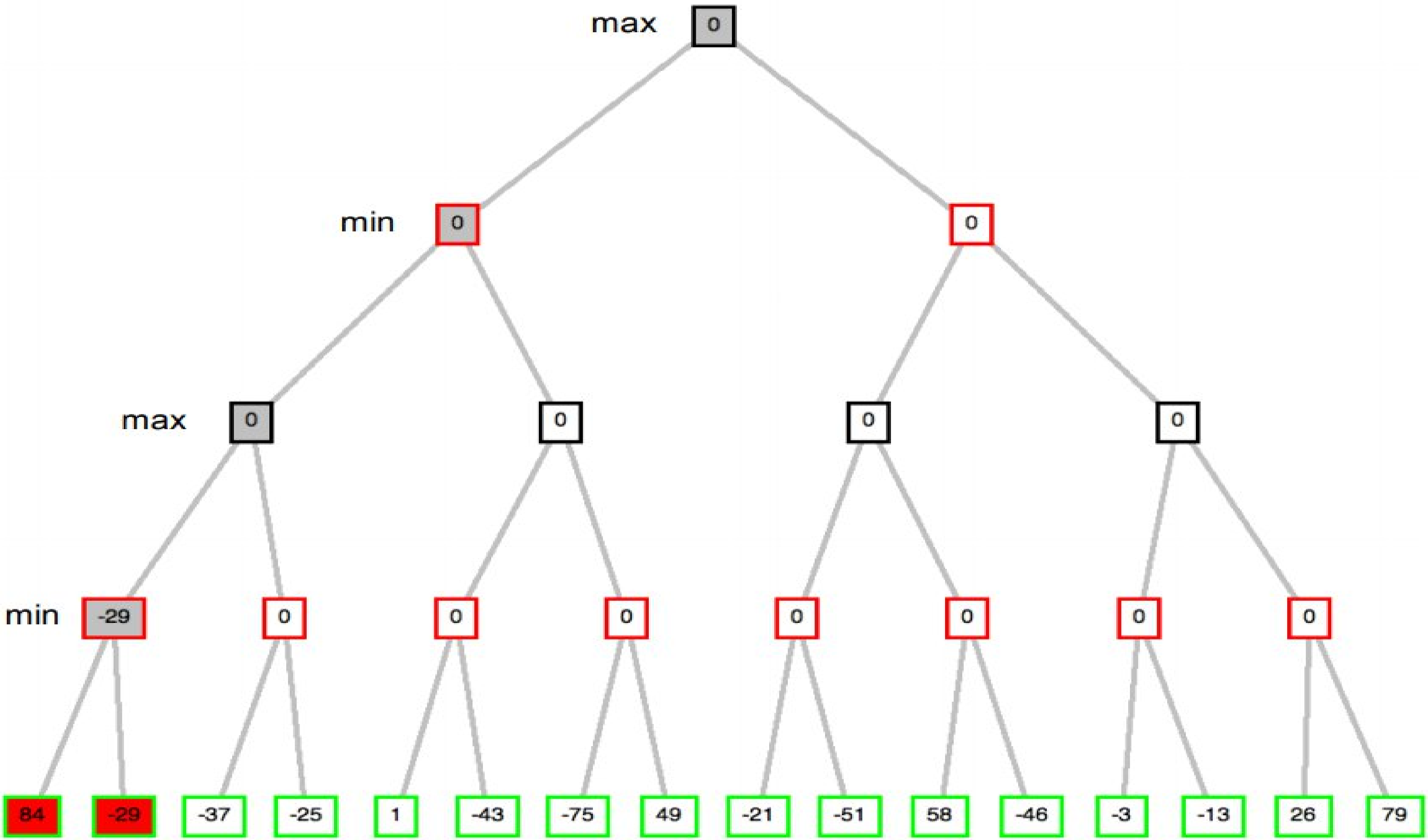
ALGORITHM

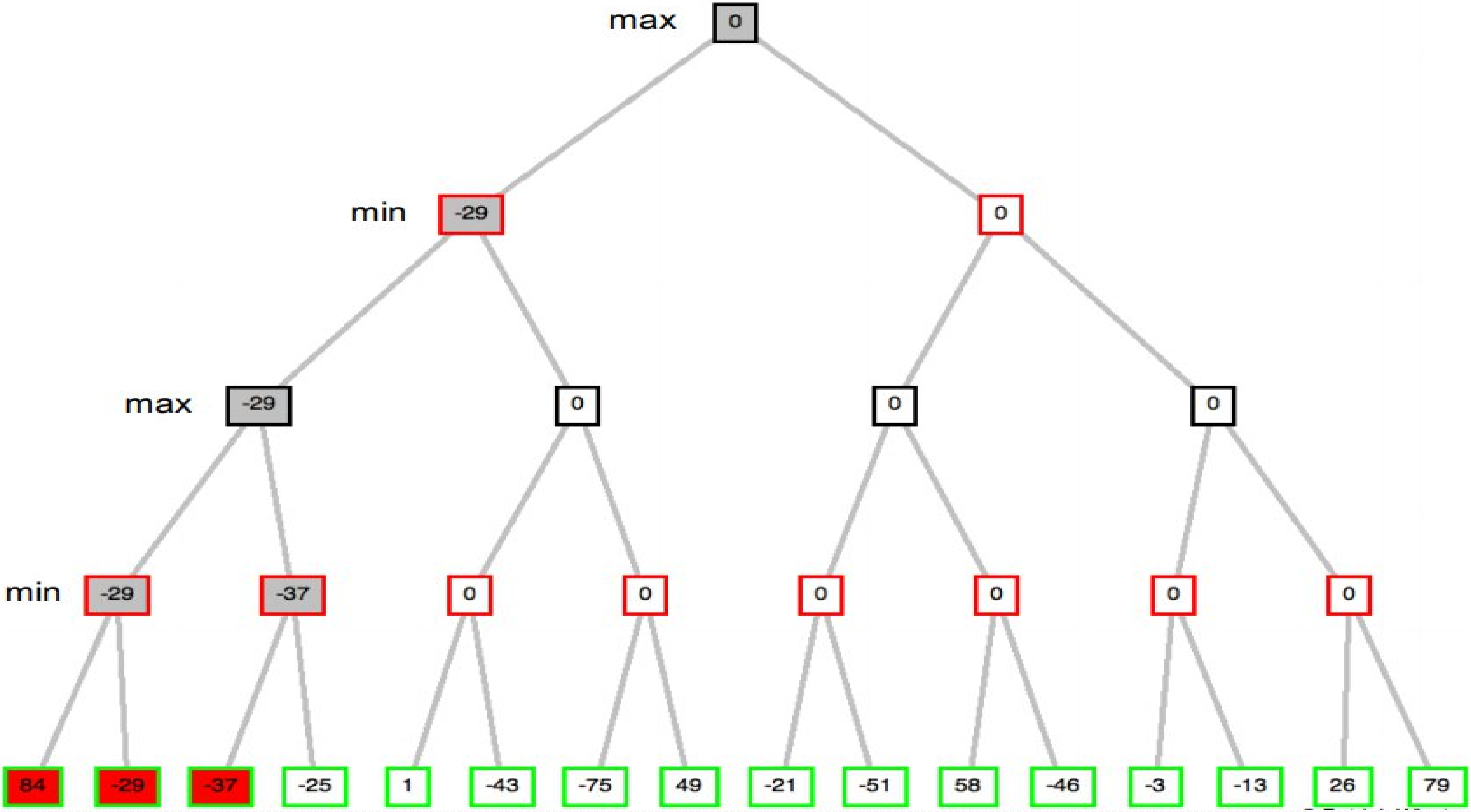
- Alpha is set to -infinity and beta to infinity.
- If the node is a leaf node, the value is returned.
- If the node is a min node, then for each children the minimax algorithm is applied with the alpha–beta pruning.
- If the value returned by a child is less, the beta sets beta to this value.
- If at any stage, beta is less than or equal to alpha do not examine any more children
- If the node is a max node, the value of beta is returned. For each of the children apply the minimax algorithm with the alpha– beta pruning.
- If the value returned by a child is greater, the alpha set alpha to this value.
- If at any stage alpha is greater than or equal to beta, more children are not examined, and the value of alpha is returned.

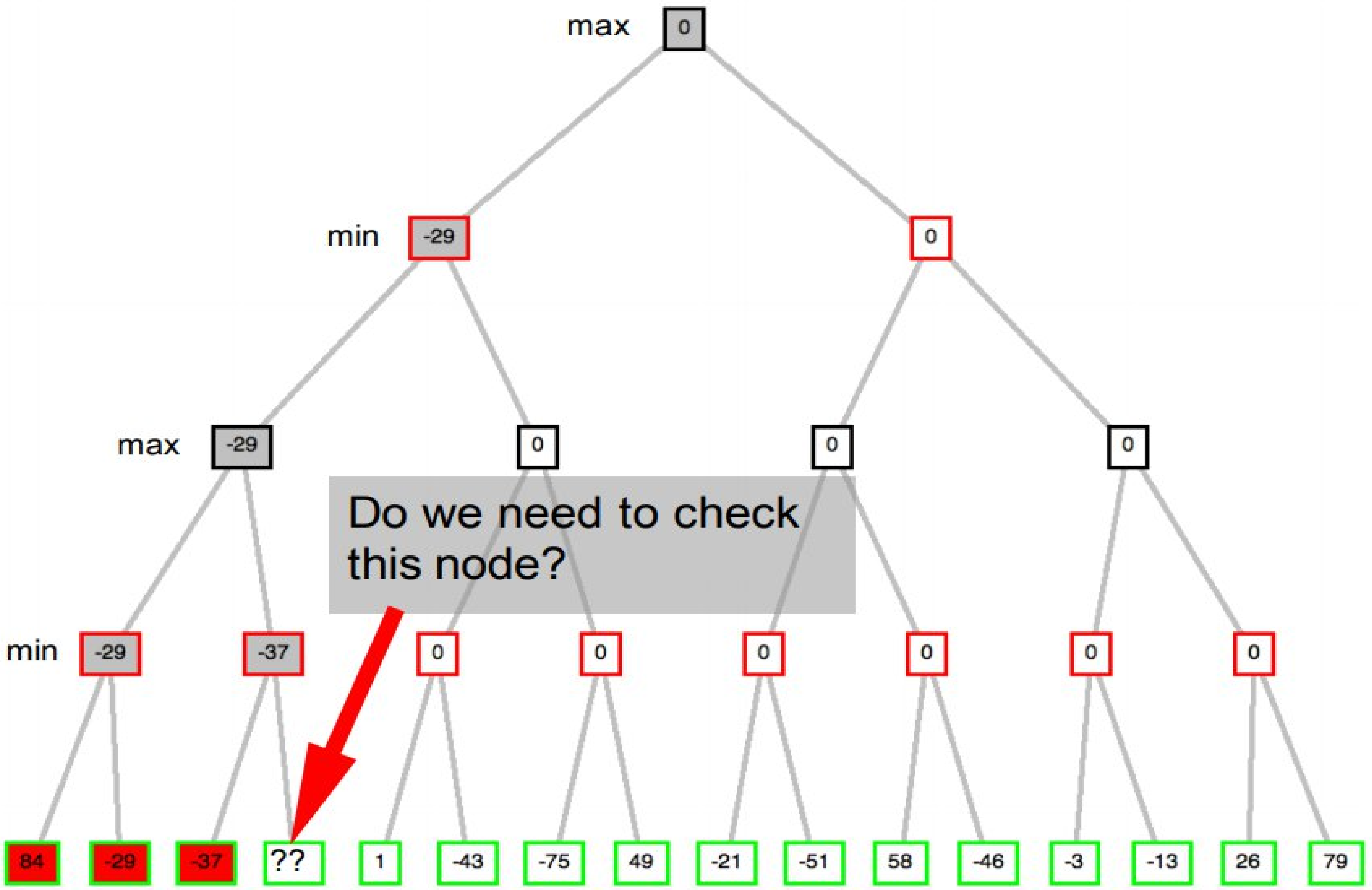
Alpha-beta pruning example



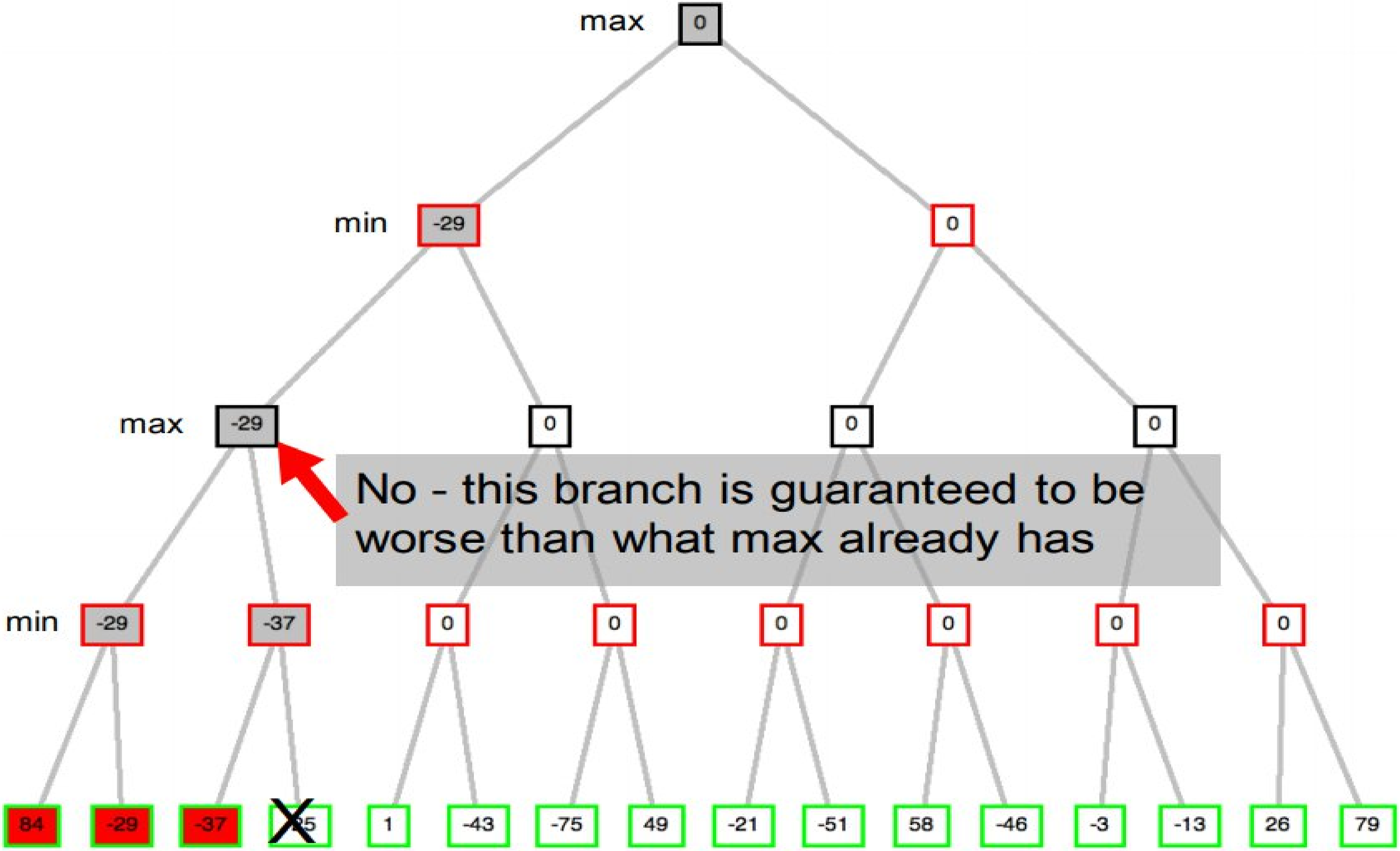

max
0
min
0
0
max
0
0
0
0
min
0
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79


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



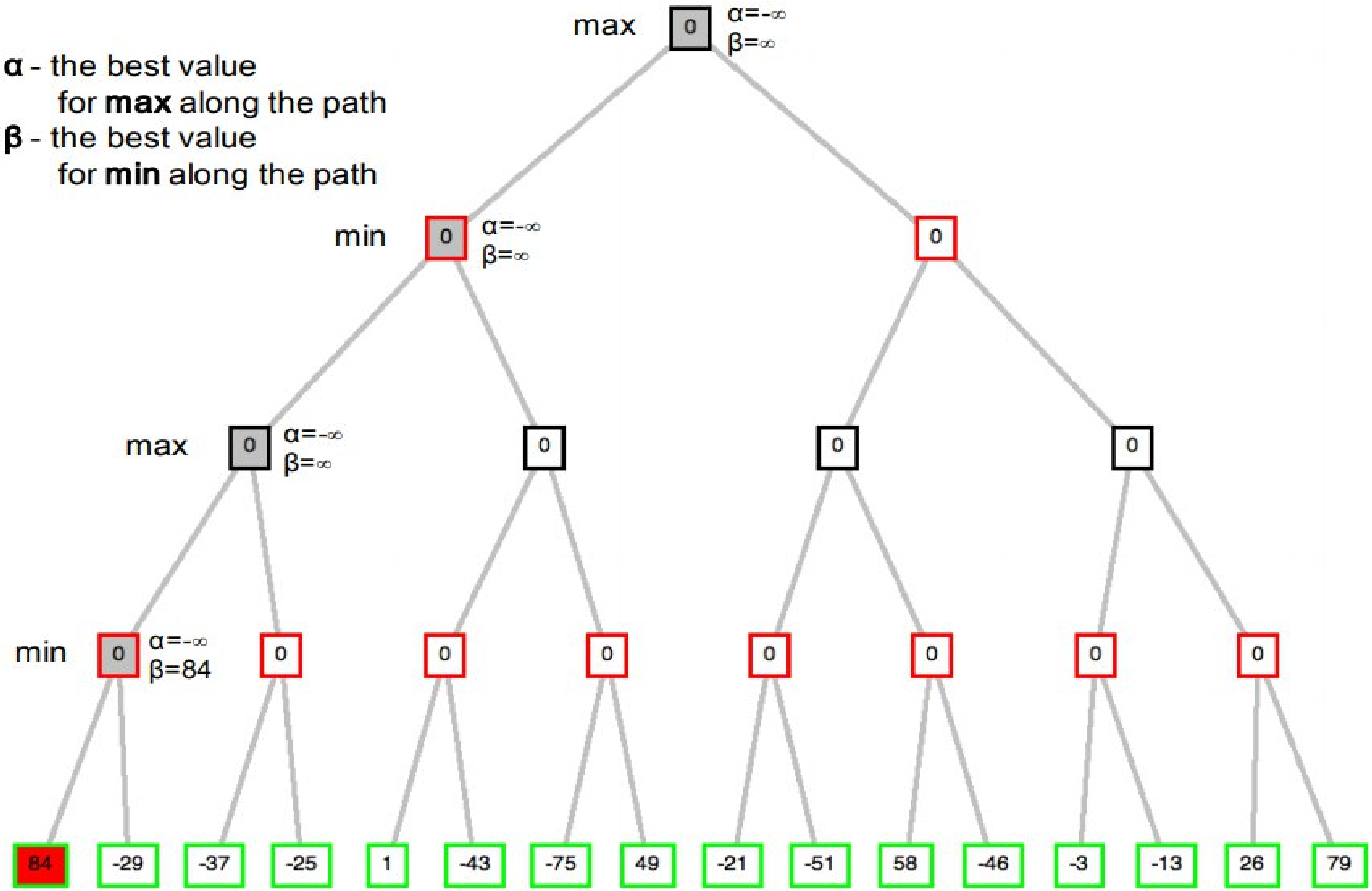
Do we need to check this node?



No - this branch is guaranteed to be worse than what max already has

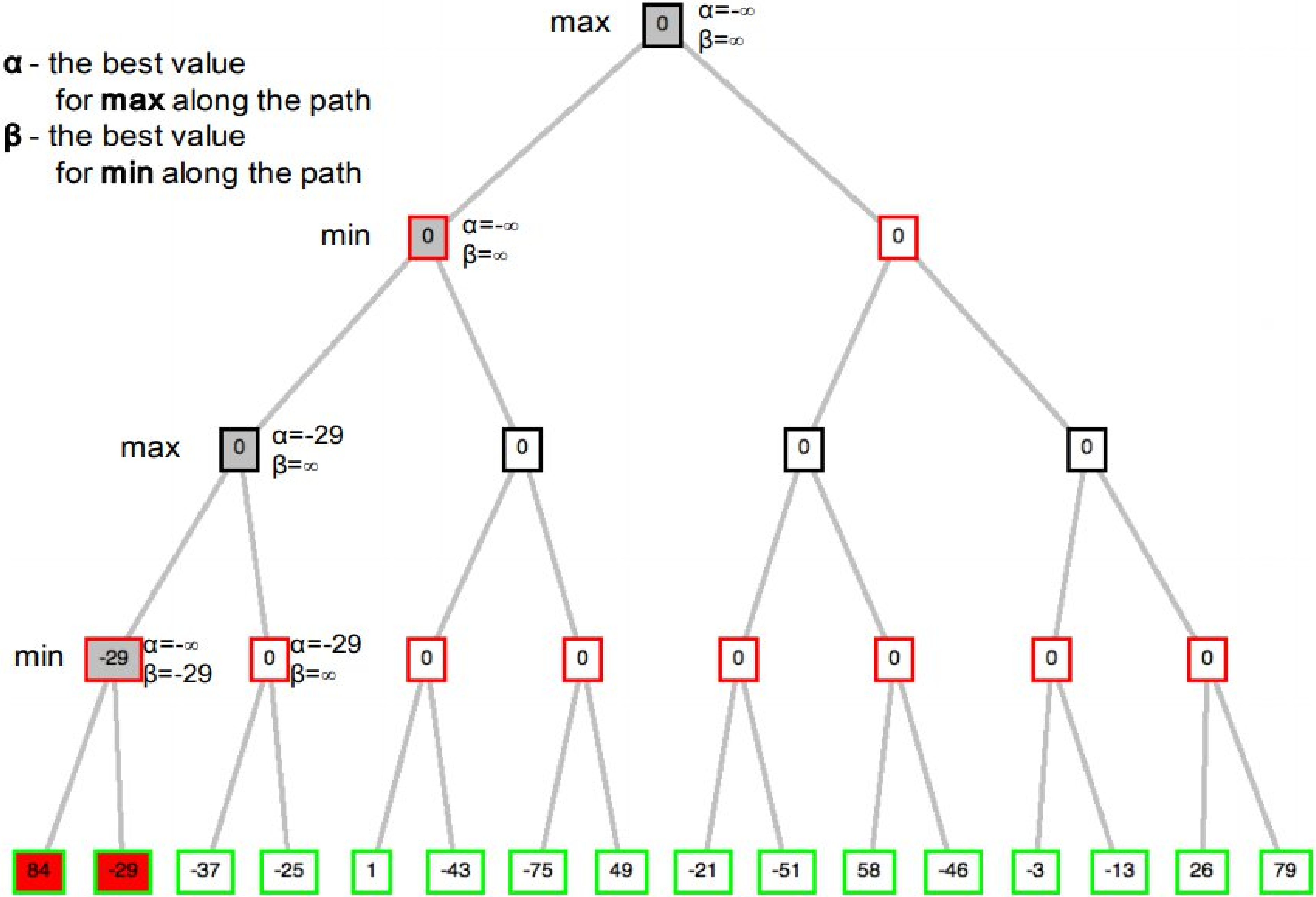
α - the best value
for **max** along the path

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



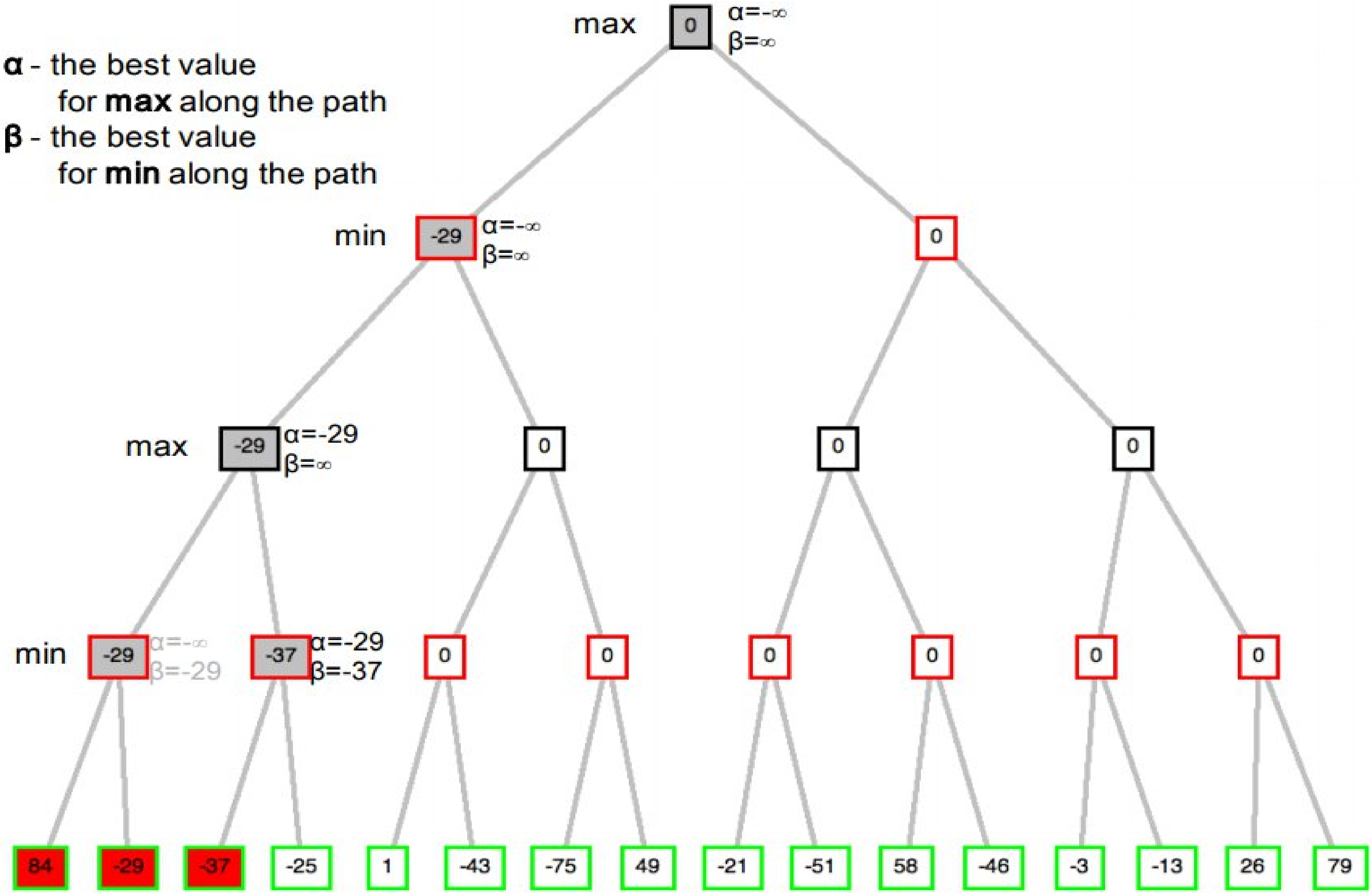
α - the best value
for **max** along the path

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



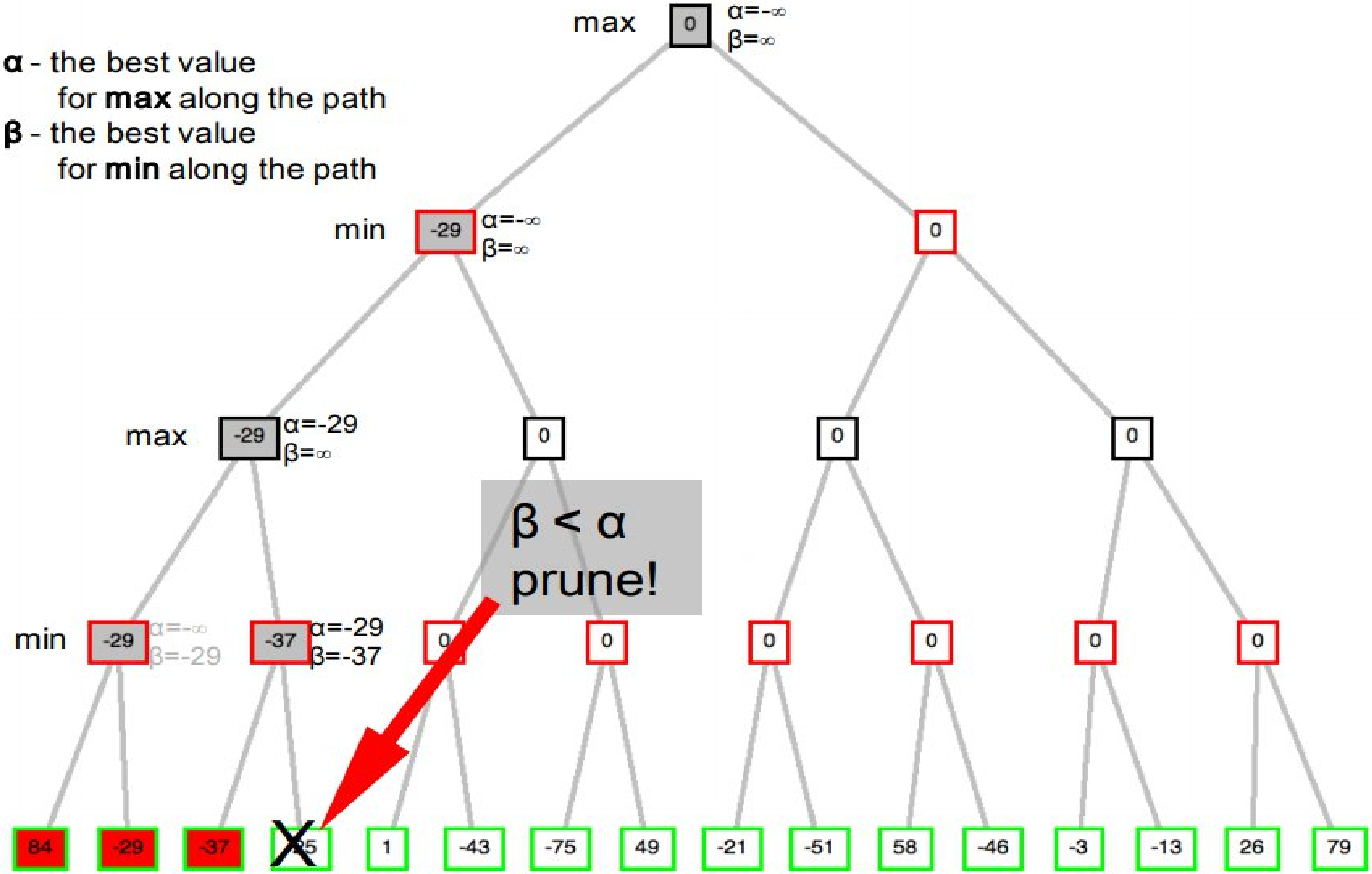
α - the best value
for **max** along the path

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



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

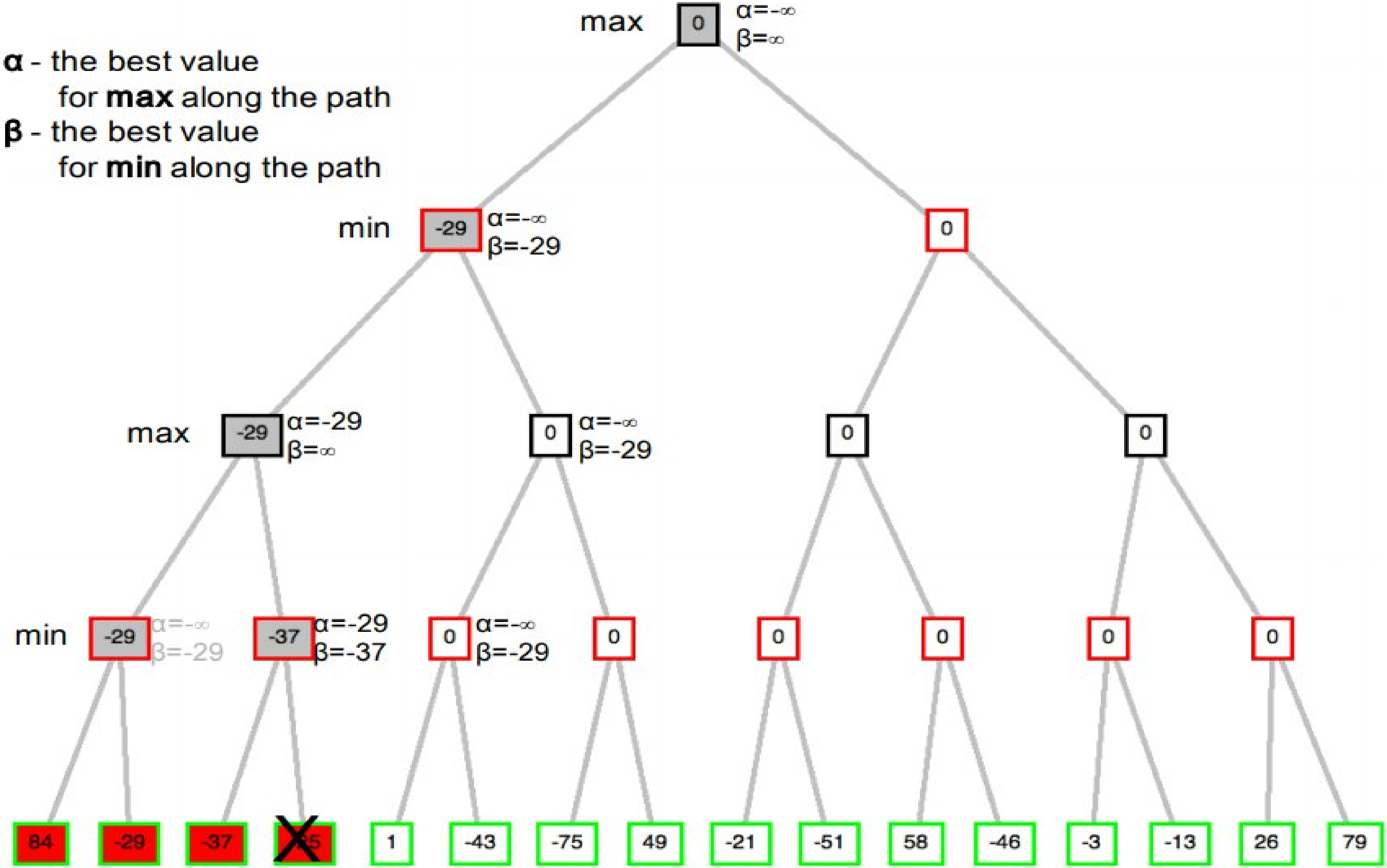
β - the best value
for **min** along the path



β < α
prune!

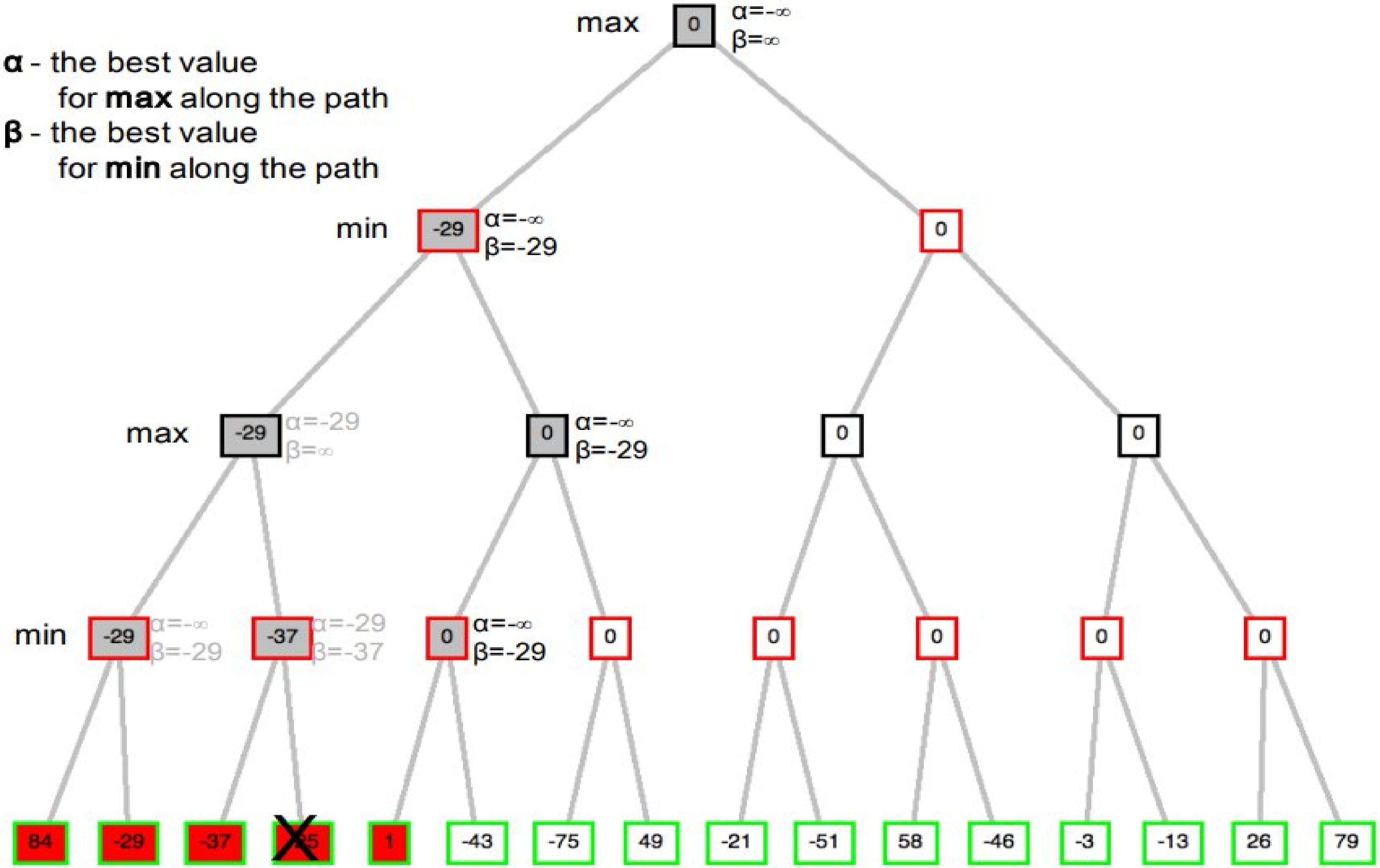
α - the best value
for **max** along the path

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



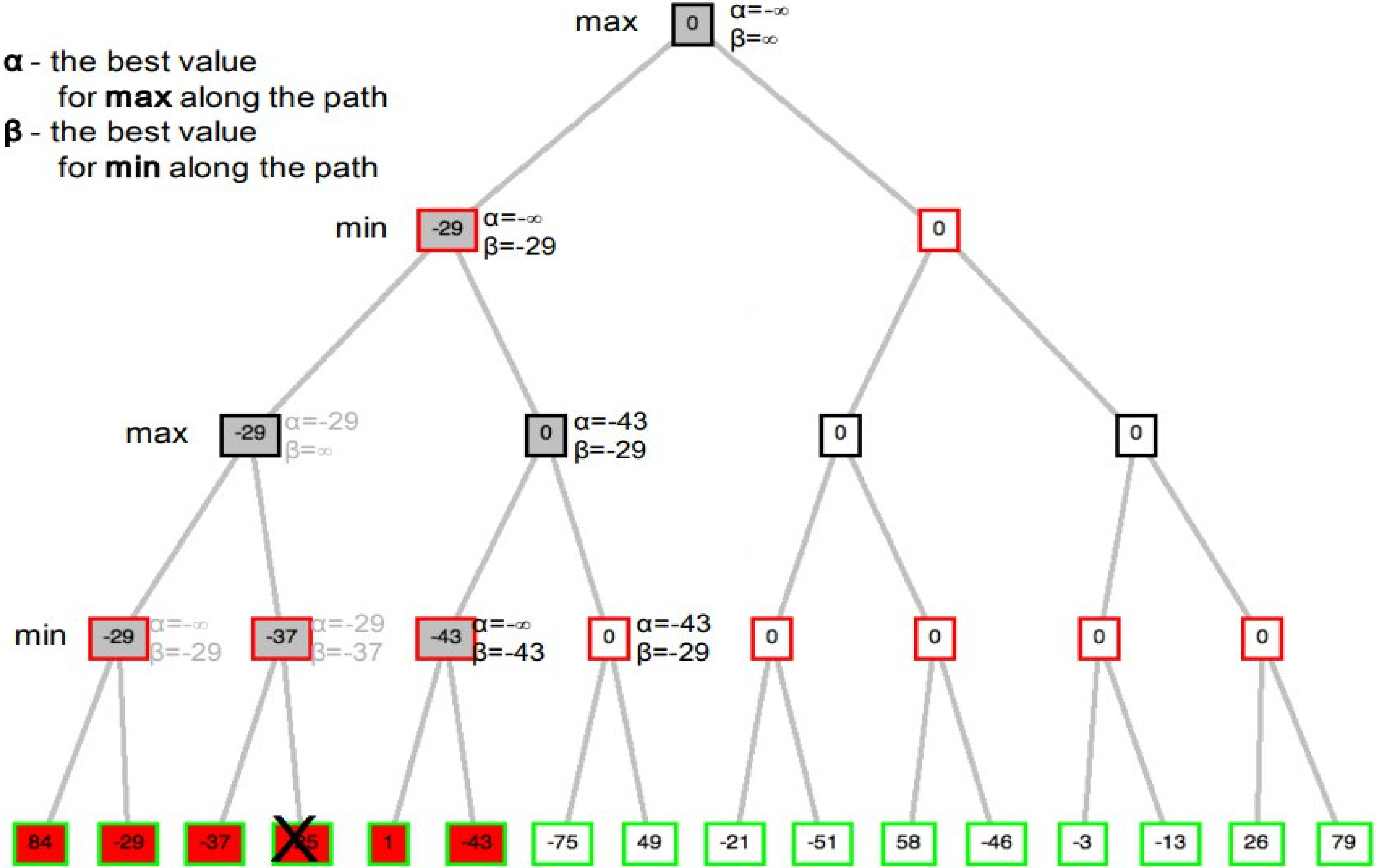
α - the best value for **max** along the path

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



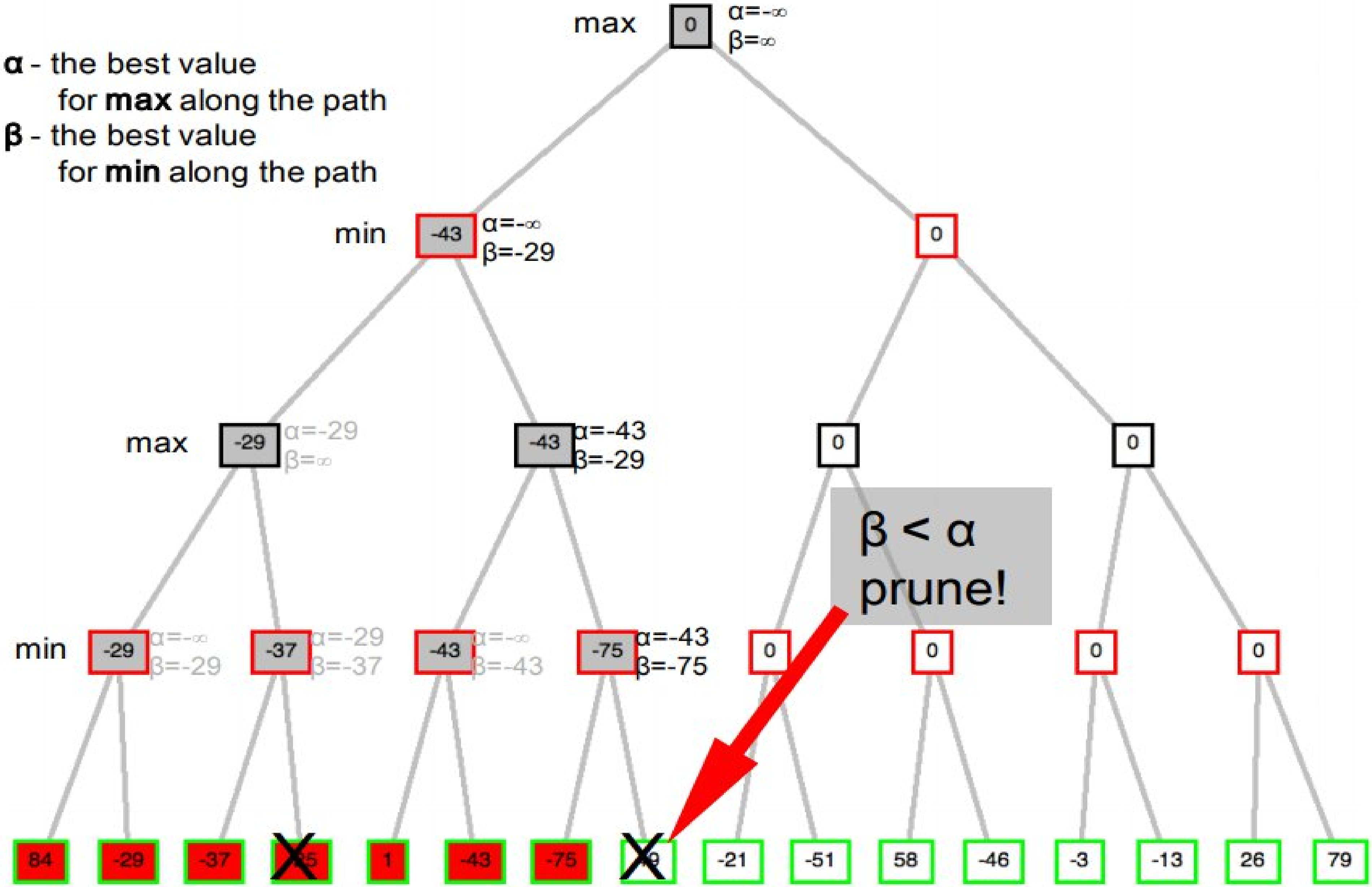
α - the best value
for **max** along the path

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



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

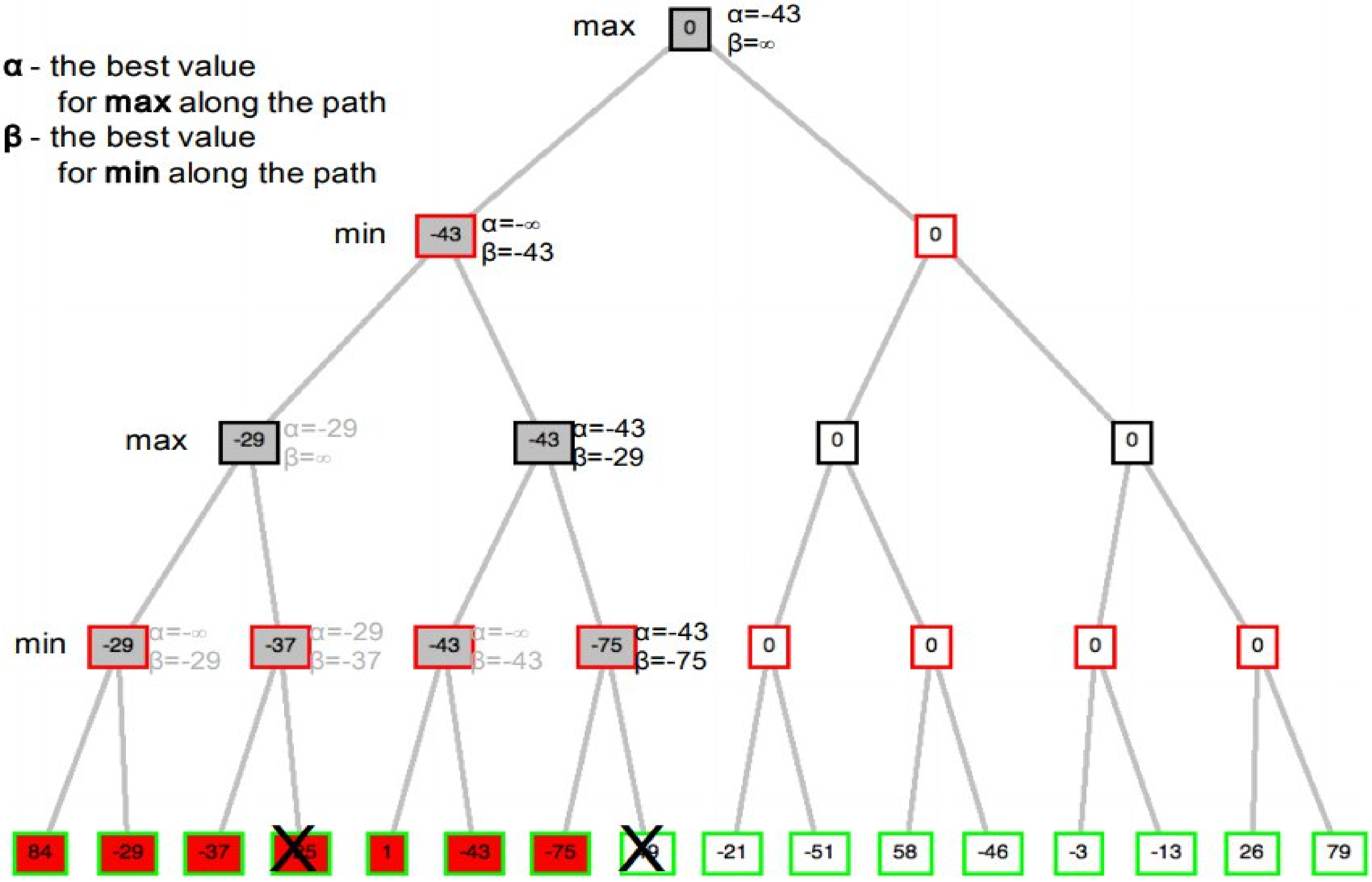
β - the best value
for **min** along the path



β < α
prune!

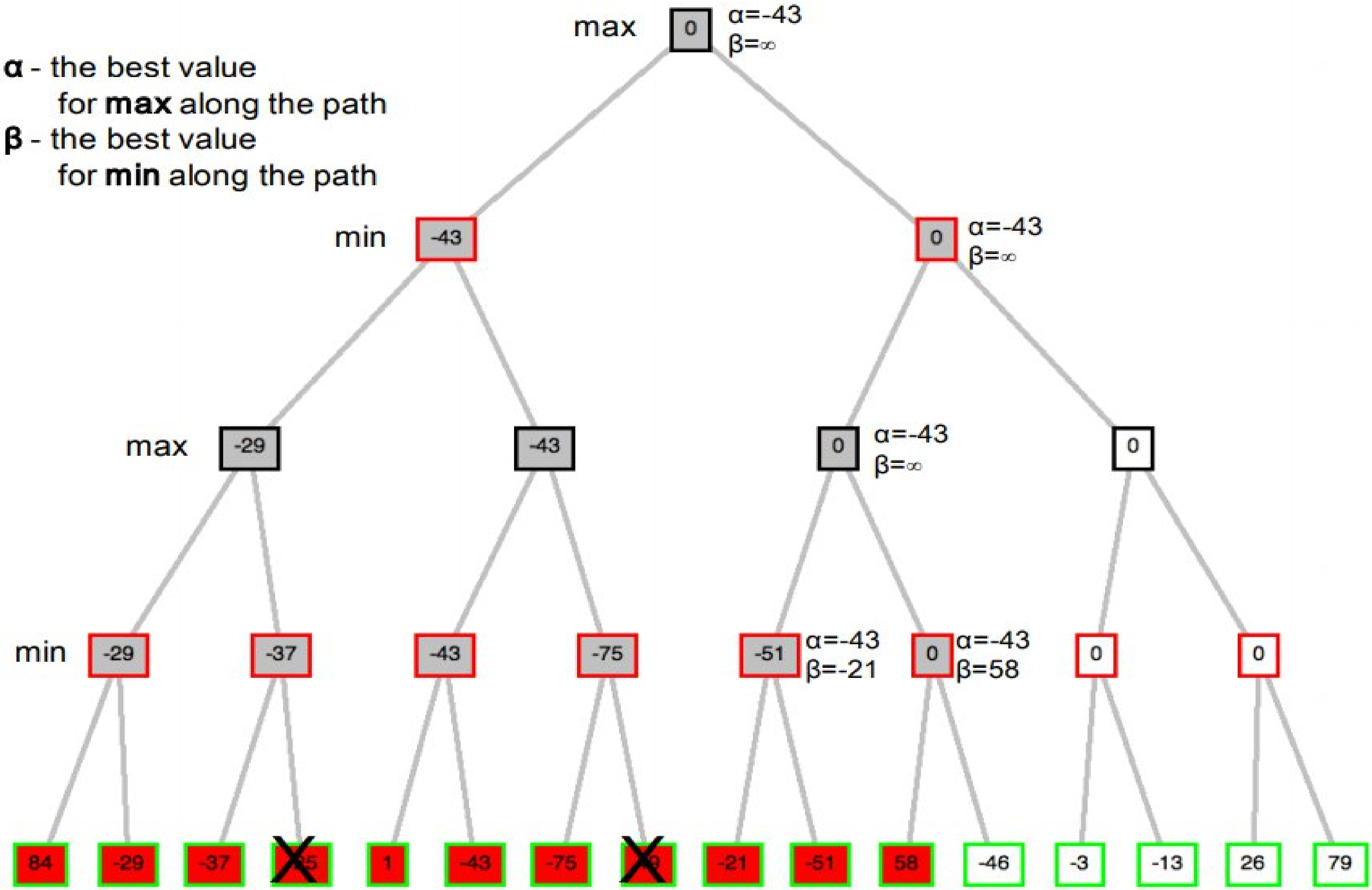
α - the best value
for **max** along the path

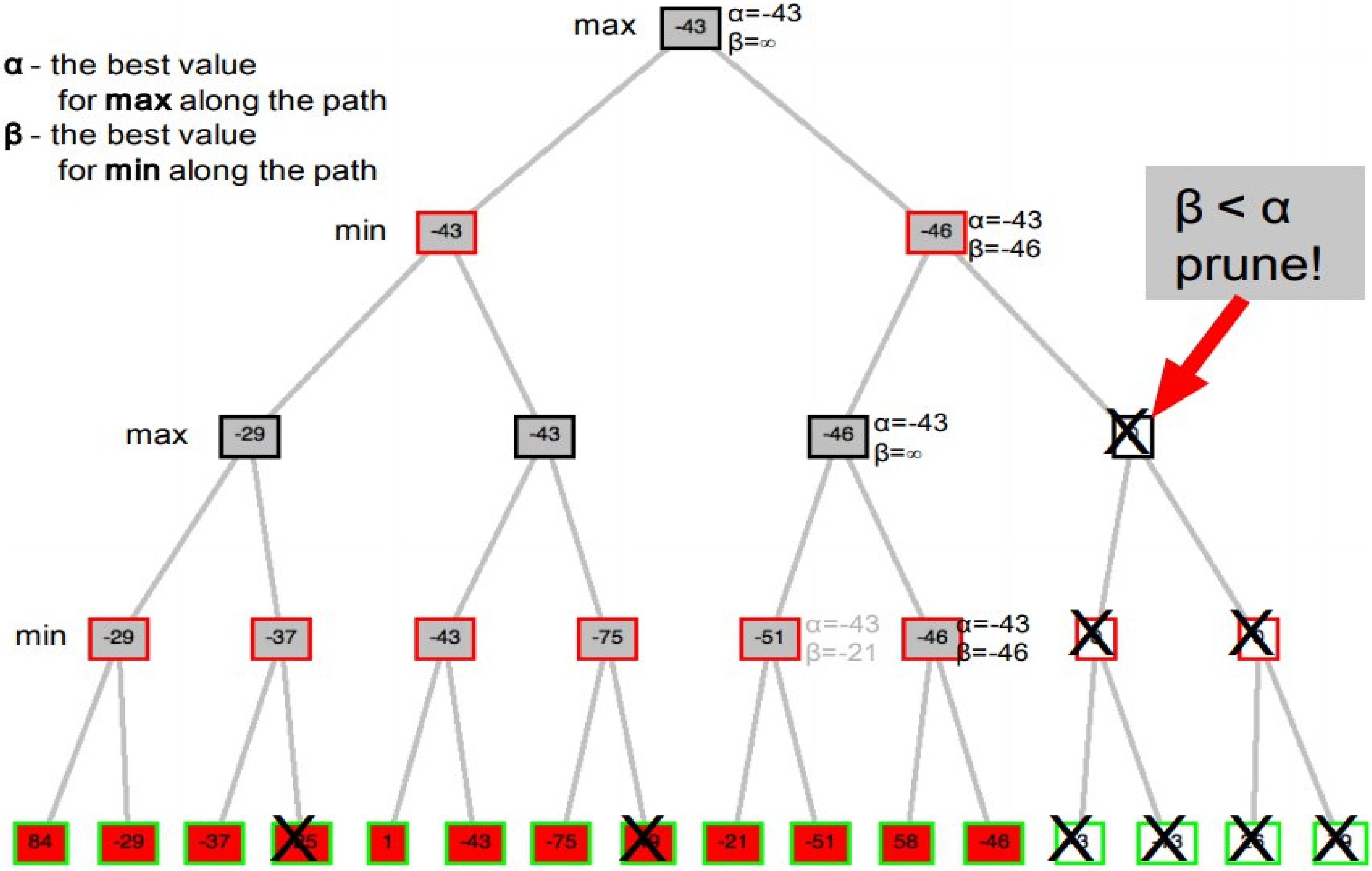
β - the best value
for **min** along the path



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

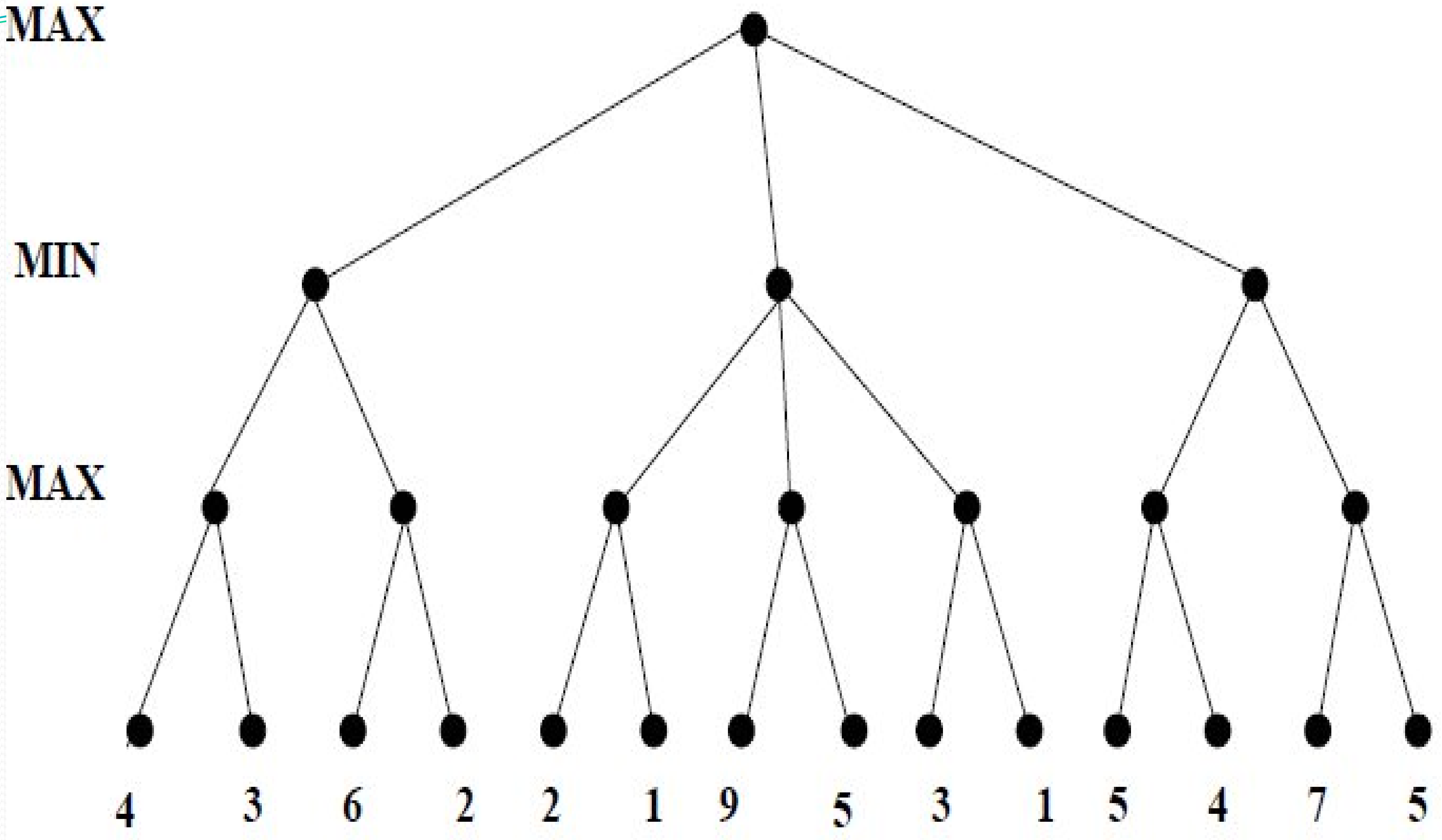
β - the best value
for **min** along the path

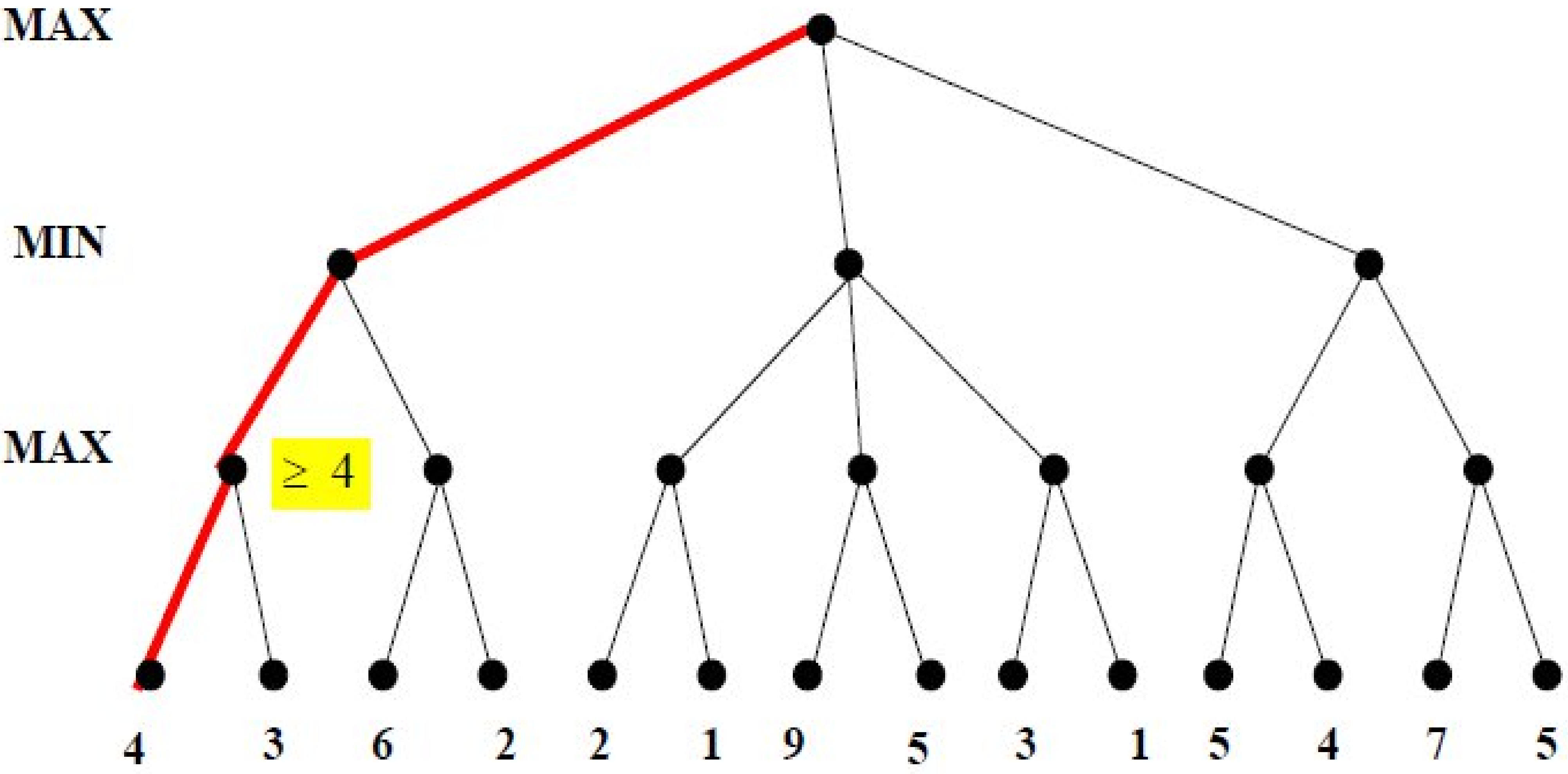


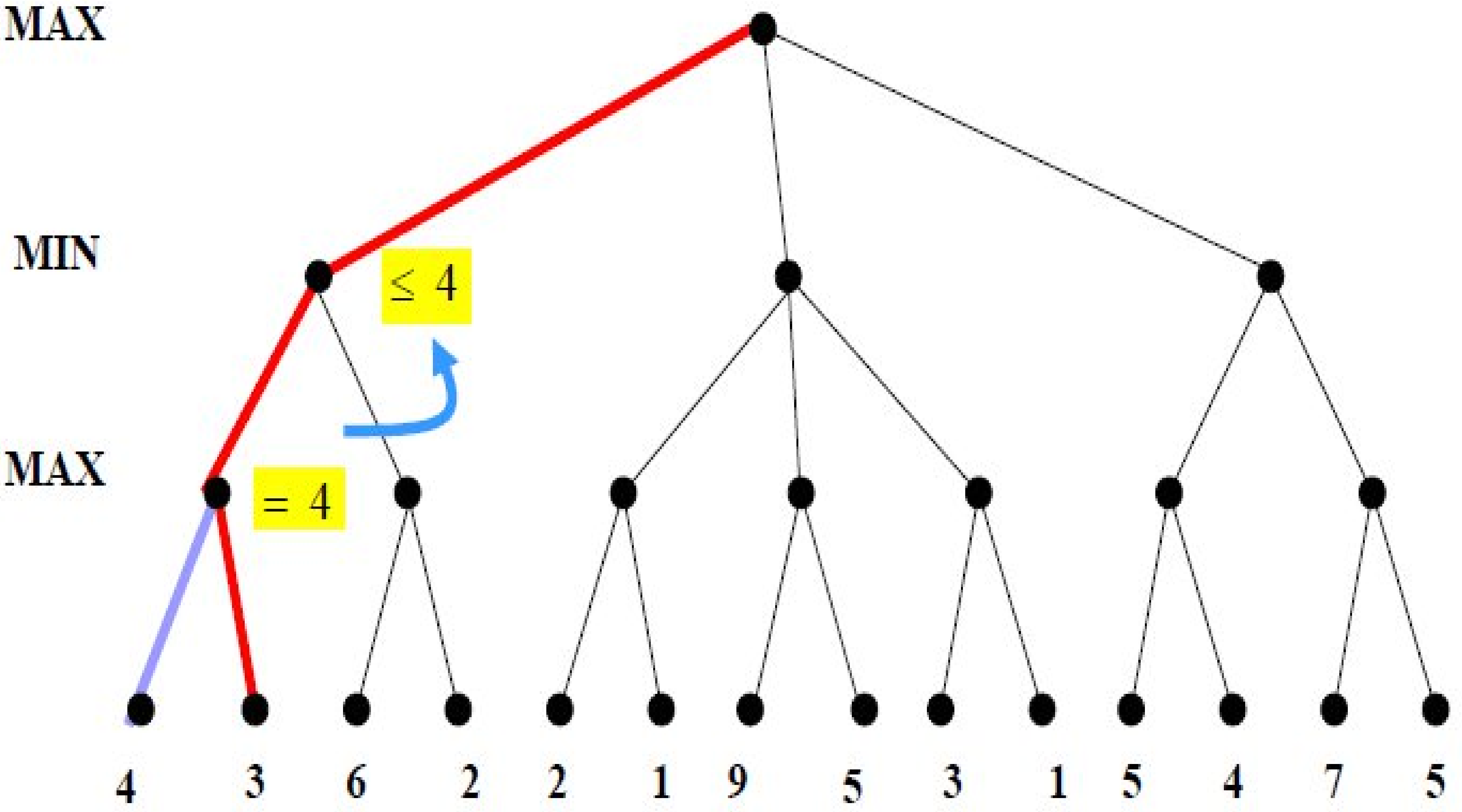


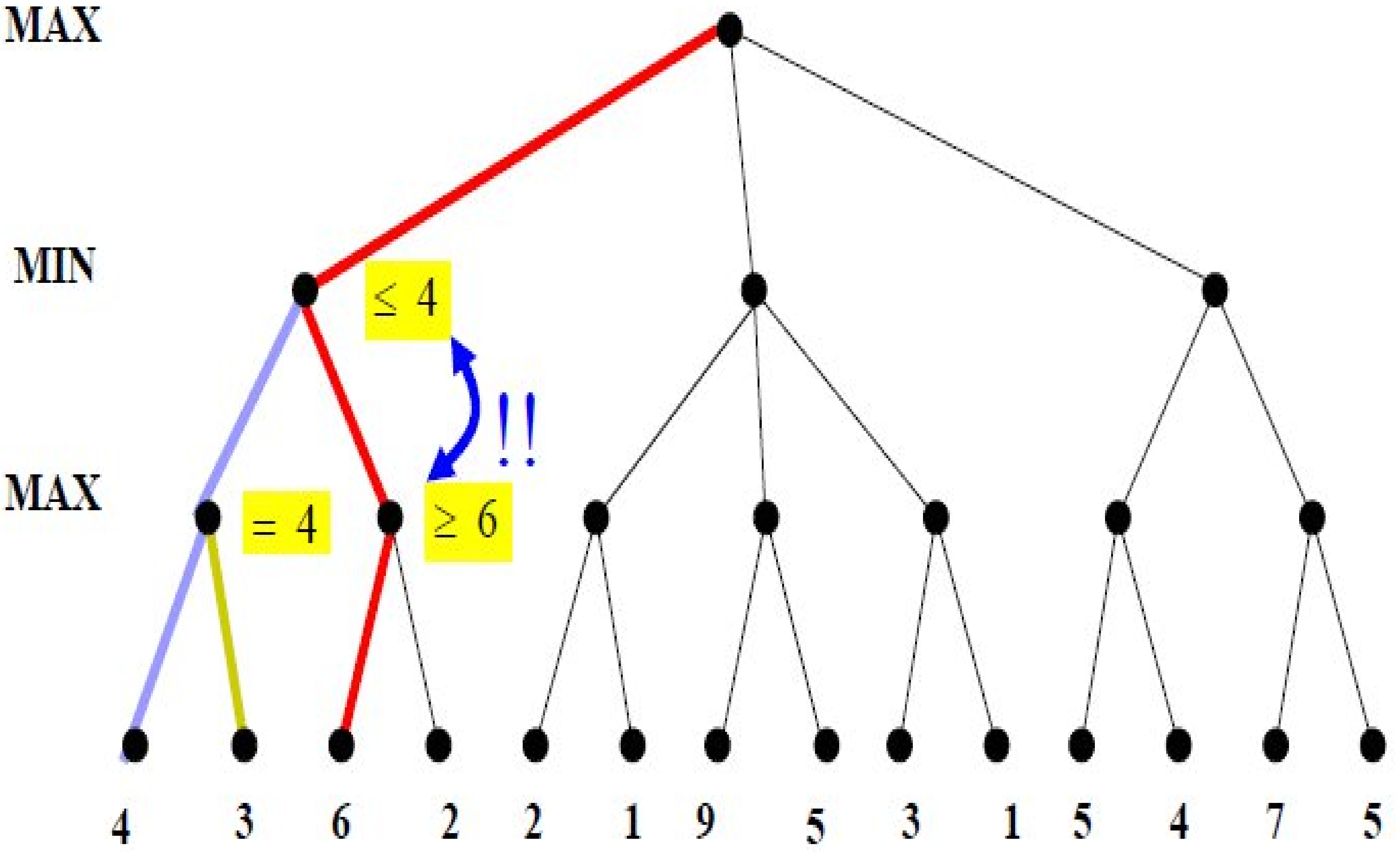
α - the best value
for **max** along the path
β - the best value
for **min** along the path

β < α
prune!

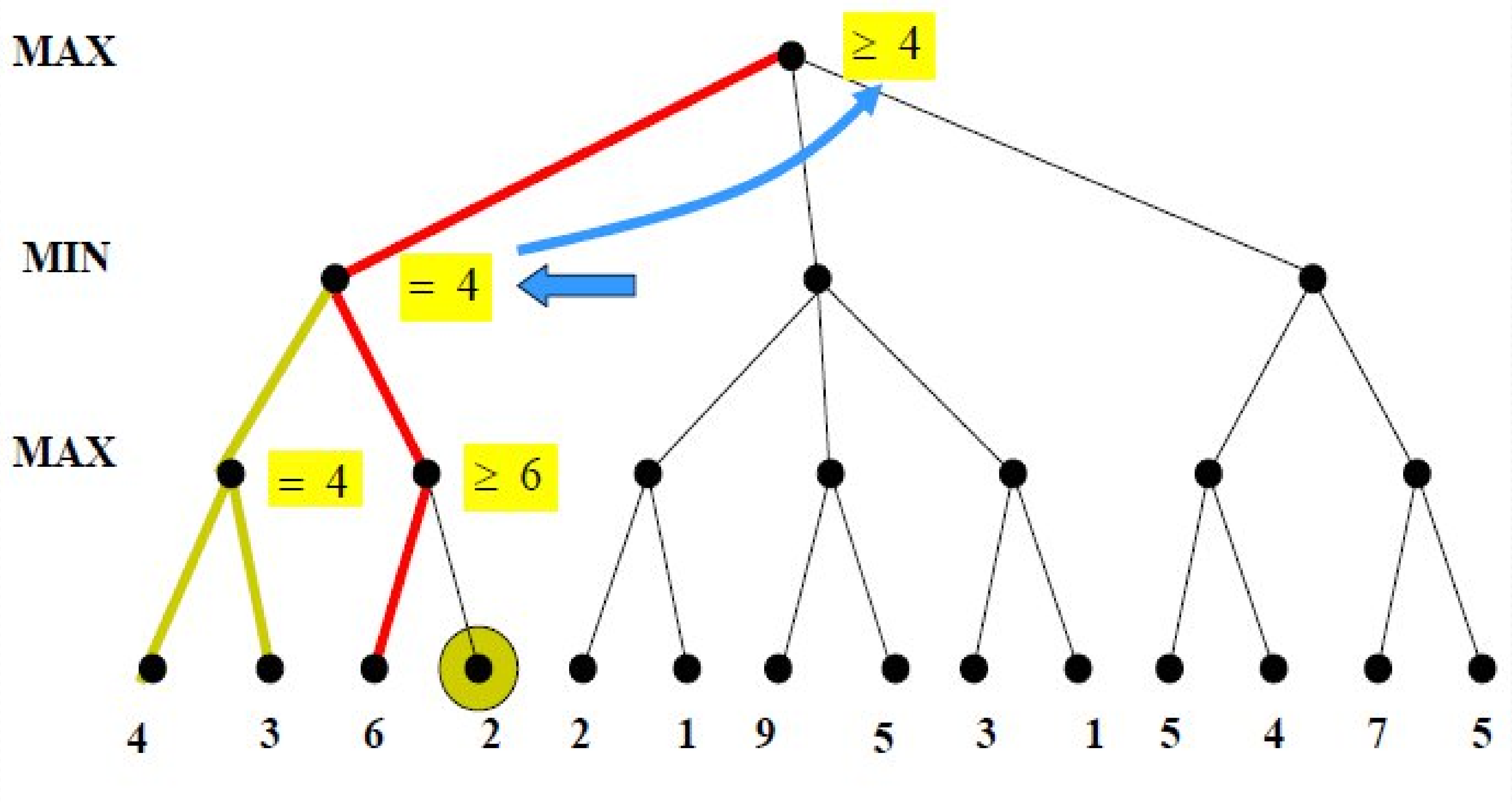

MAX
MIN
MAX
4
3
6
2
2
1
9
5
3
1
5
4
7
5


MAX
MIN
MAX
≥ 4
4
3
6
2
2
1
9
5
3
1
5
4
7
5

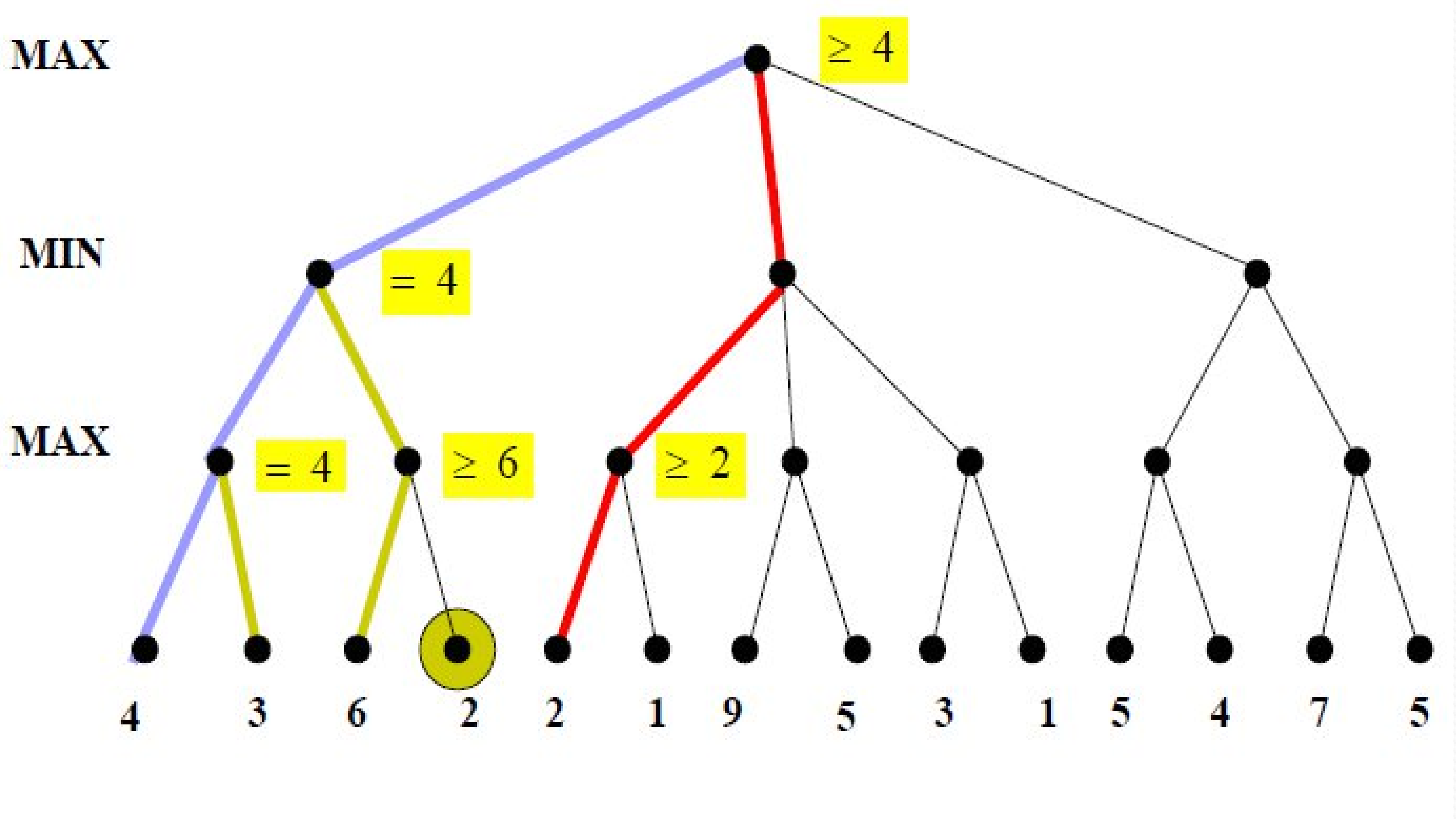

MAX
MIN
MAX
≤ 4
= 4
4
3
6
2
2
1
9
5
3
1
5
4
7
5


MAX
MIN
MAX
≤ 4
!!
= 4
≥ 6
4
3
6
2
2
1
9
5
3
1
5
4
7
5

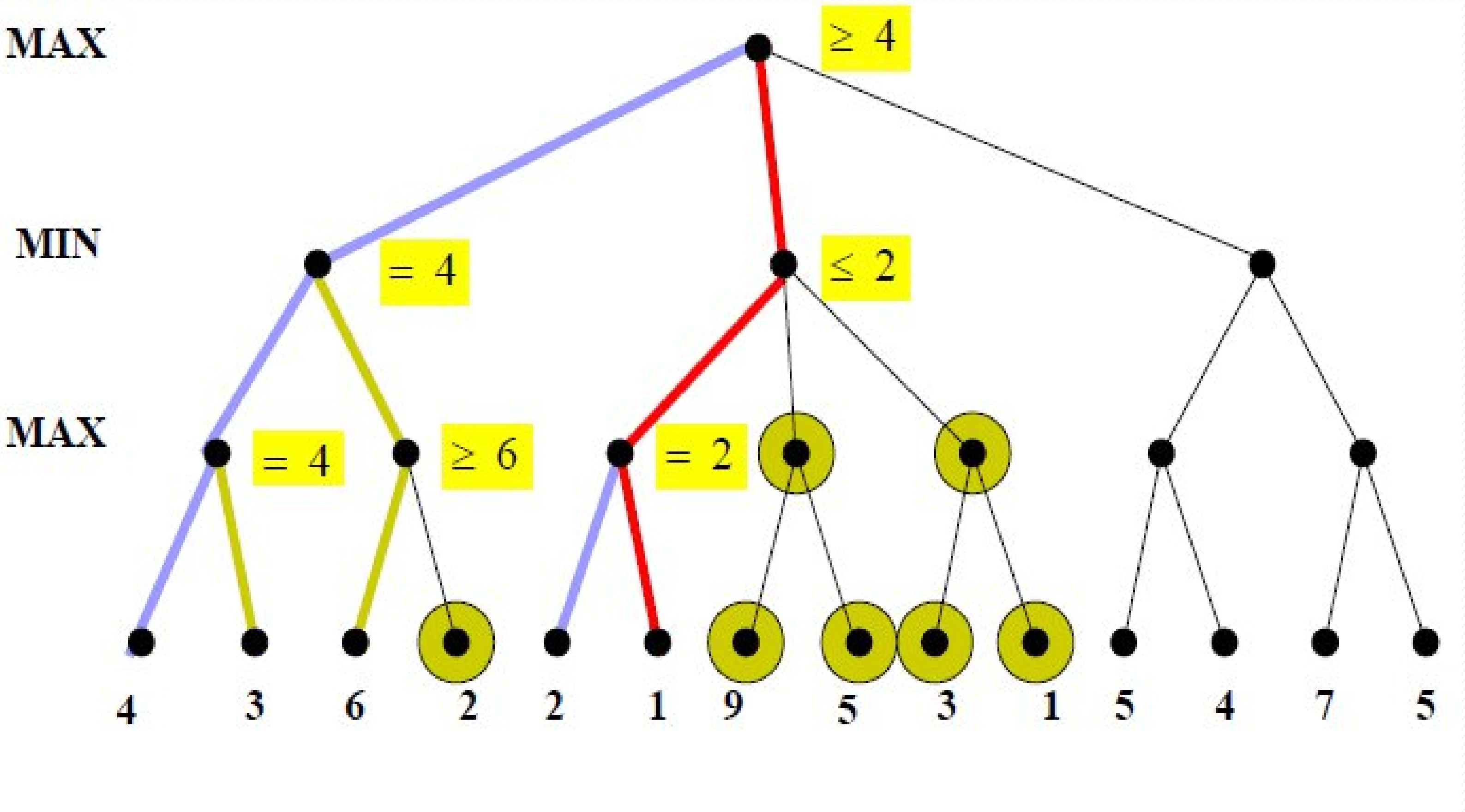
Alpha-beta pruning



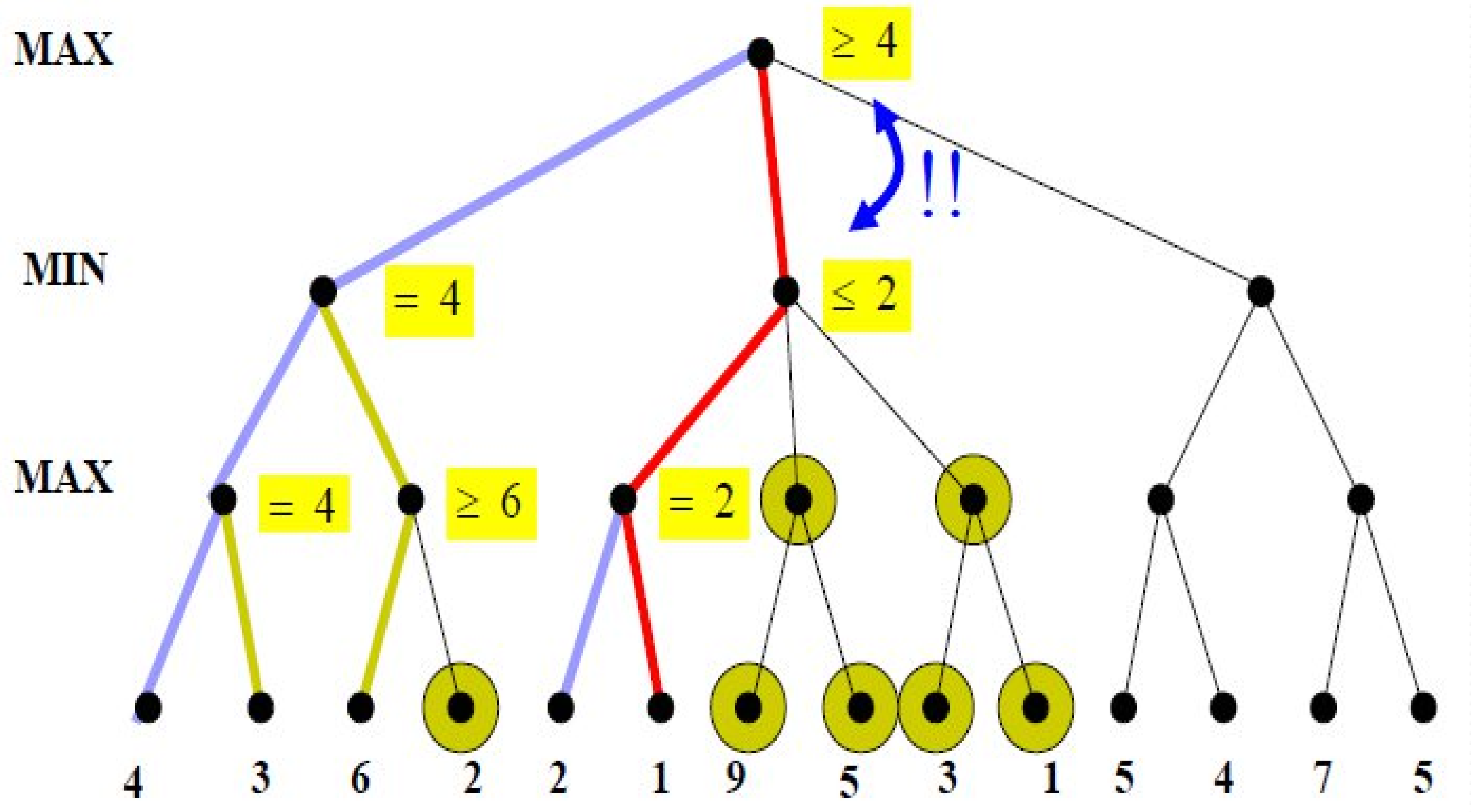
Alpha-beta pruning



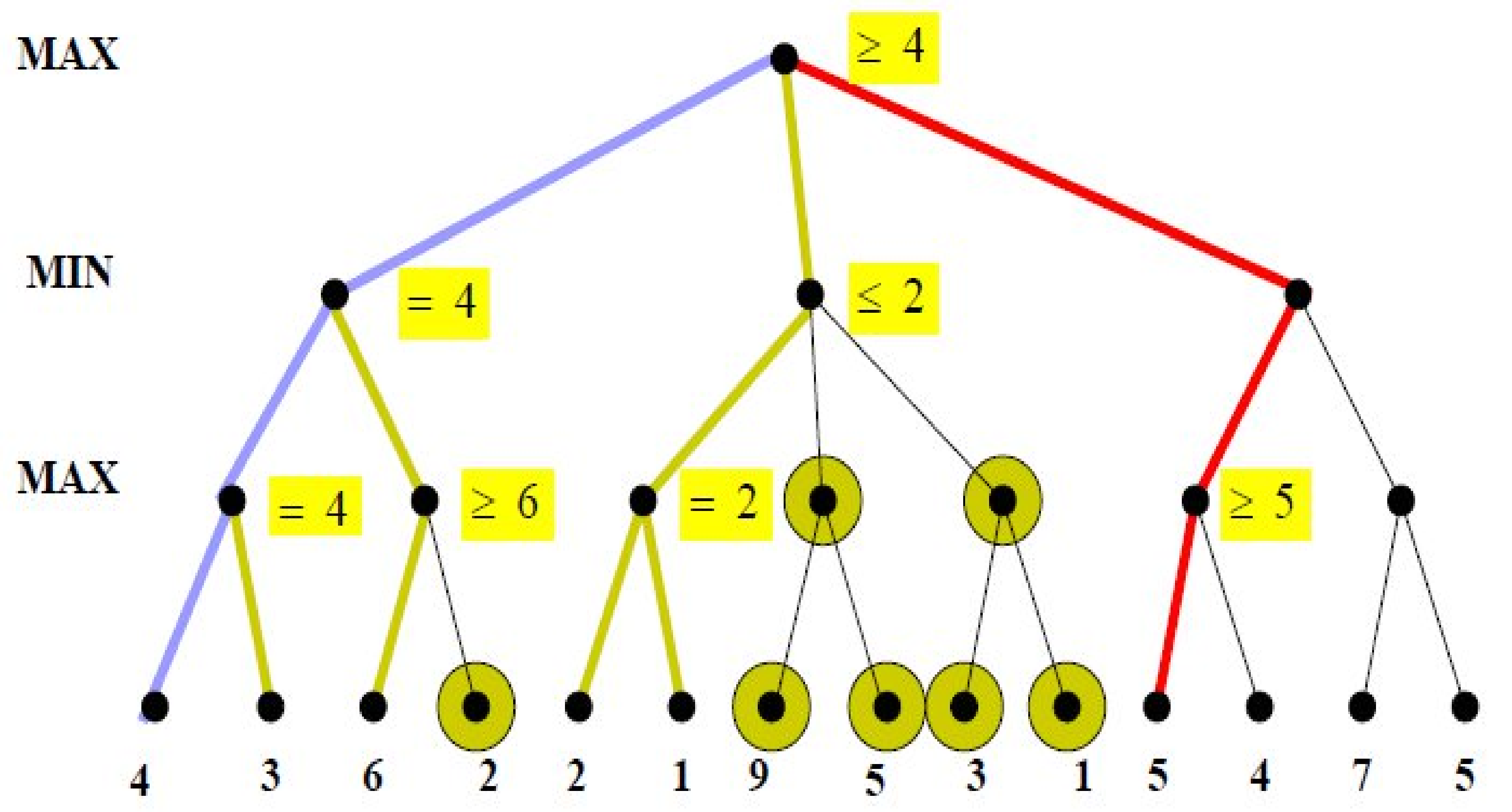
Alpha-beta pruning



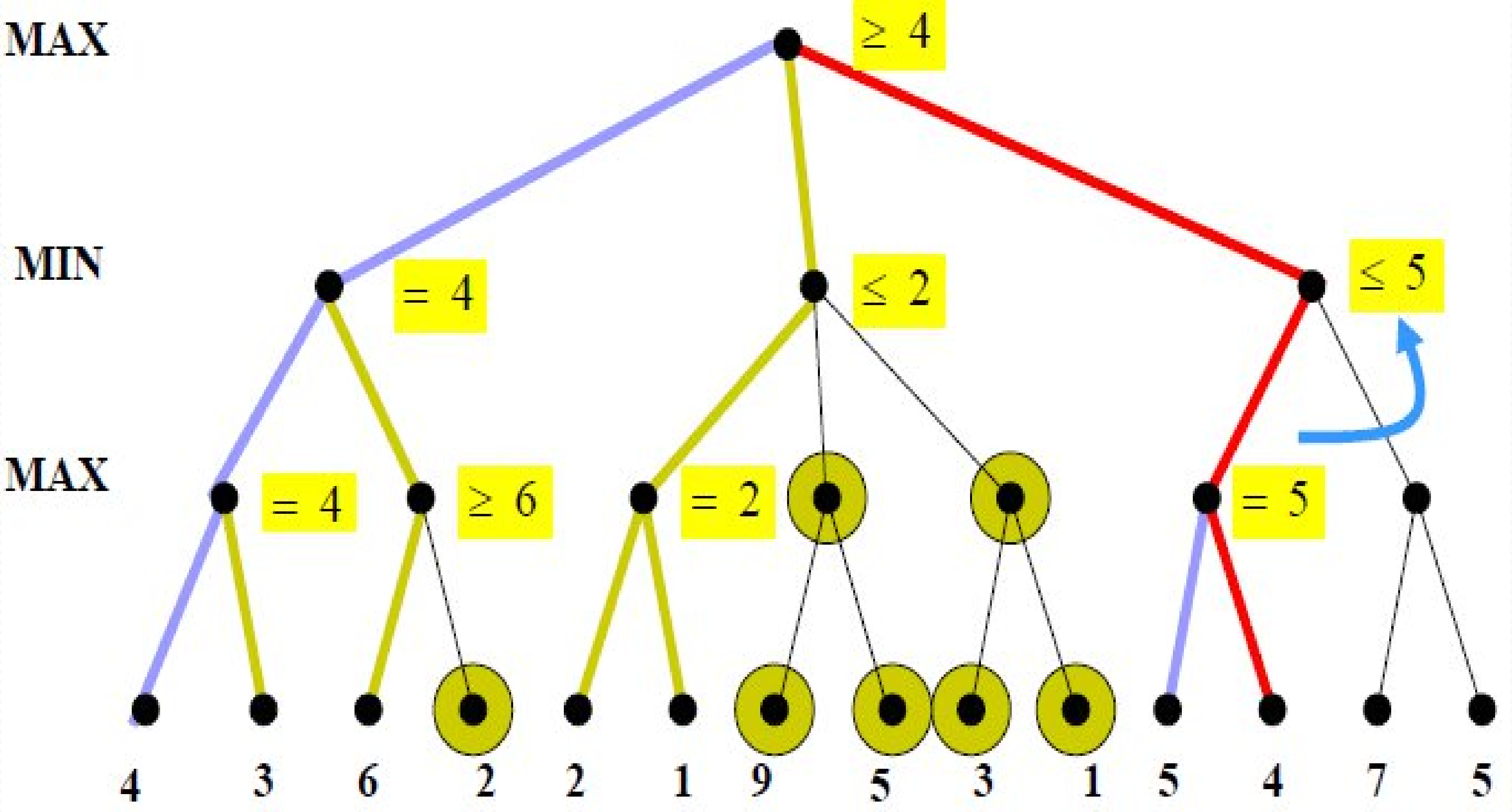
Alpha-beta pruning



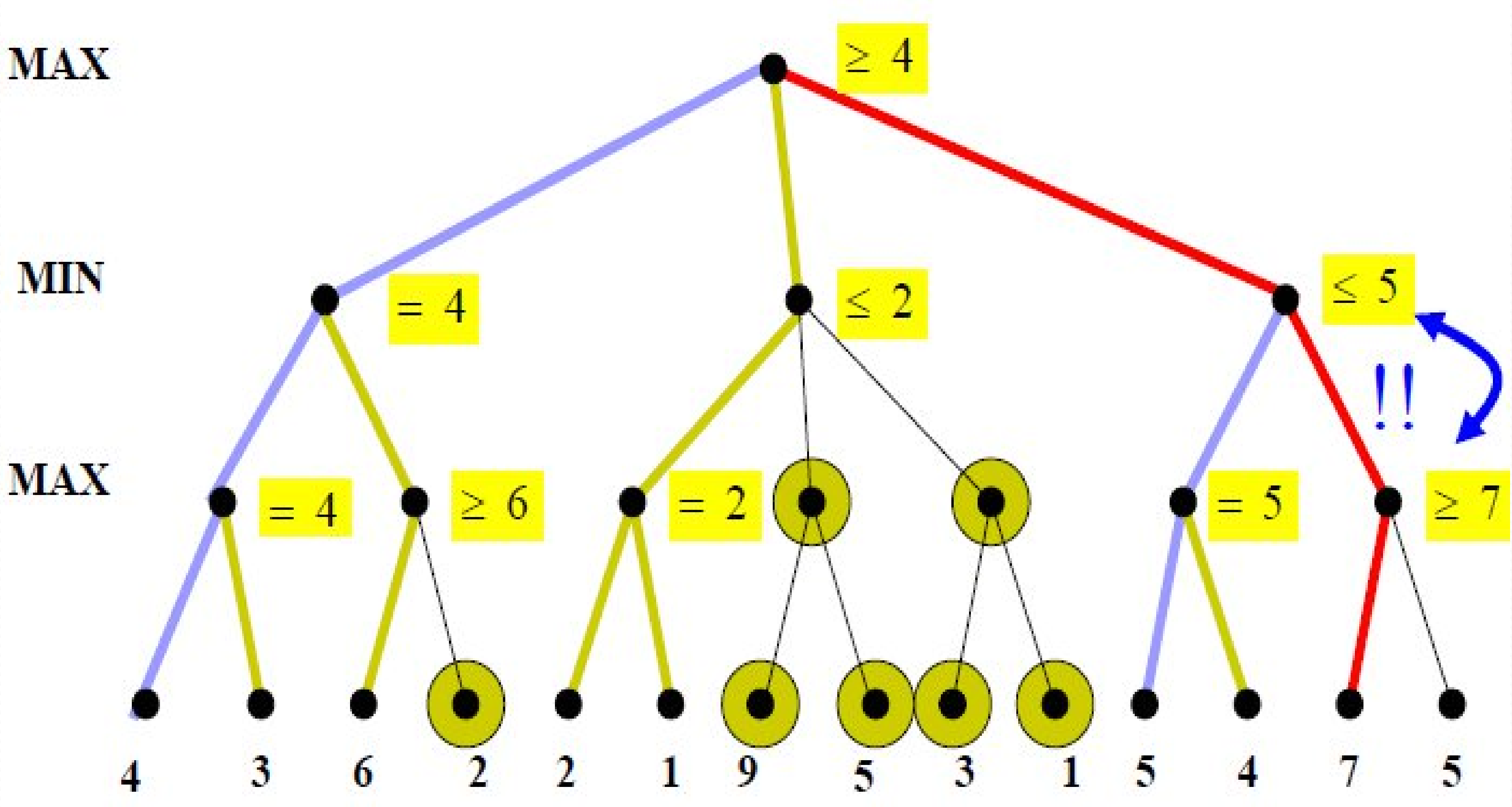
Alpha-beta pruning



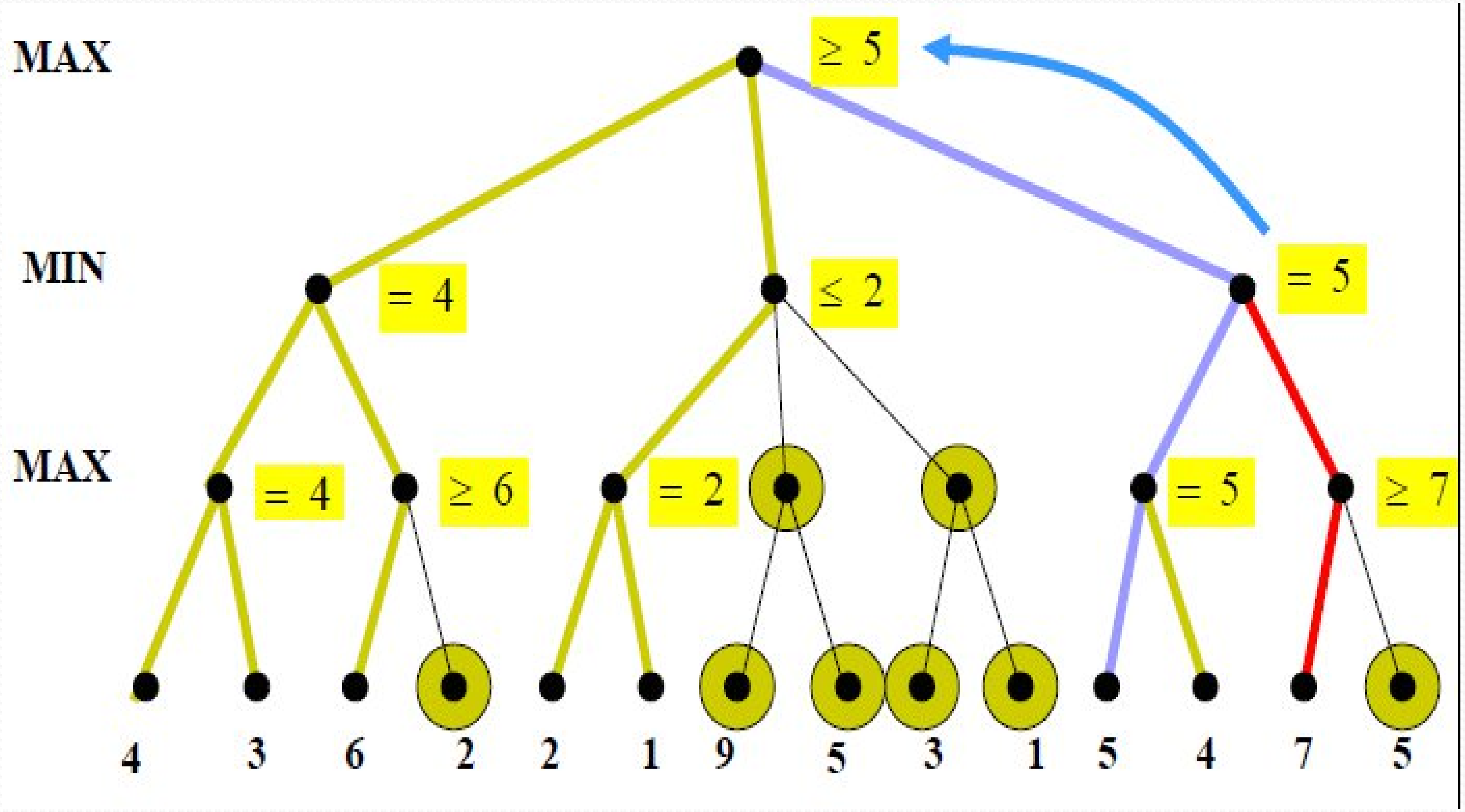
Alpha-beta pruning



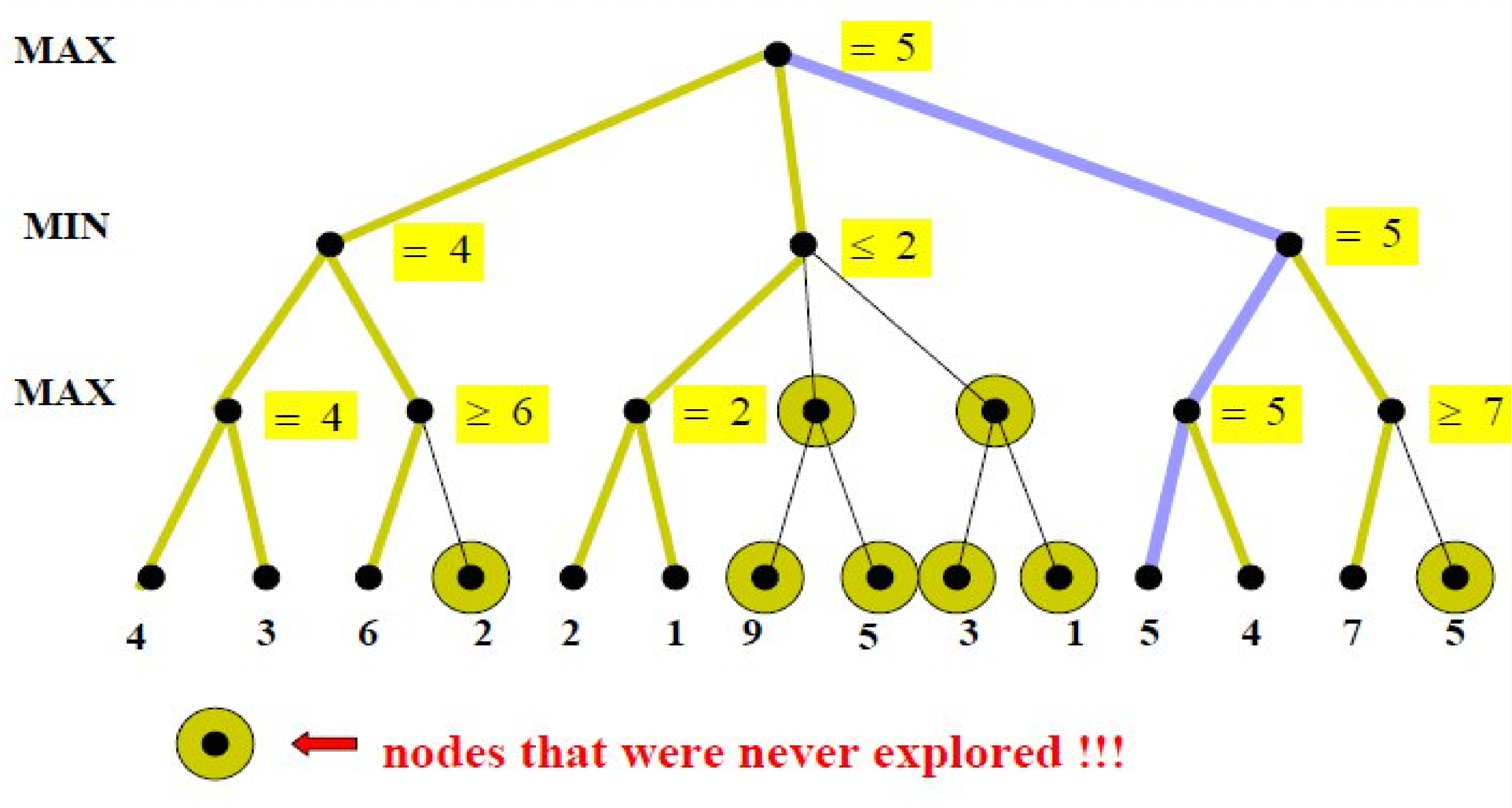
Alpha-beta pruning



Alpha-beta pruning



Alpha-beta pruning



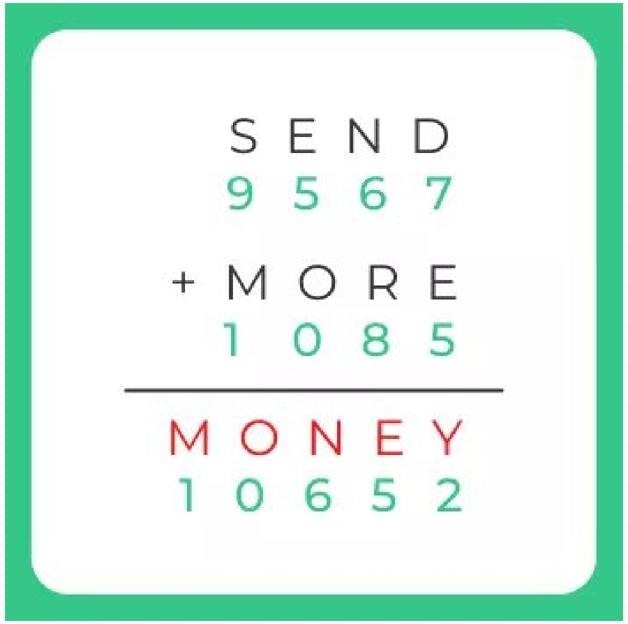
Cryptarithmetic problem

- The **Crypt-Arithmetic problem** is a type of encryption problem in which the written message in an alphabetical form which is easily readable and understandable is converted into a numeric form which is neither easily readable nor understandable.
- The constraints which this problem follows during the conversion is as follows:

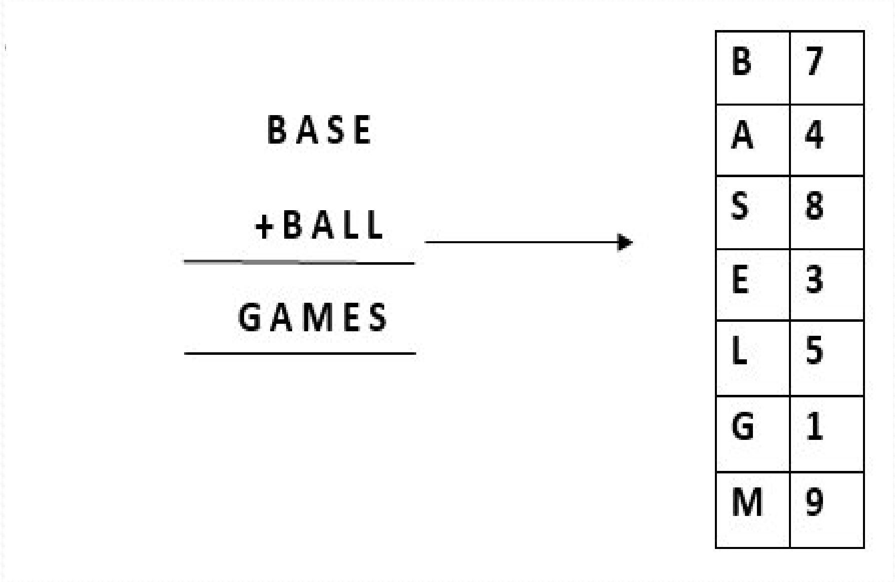
1. A number 0-9 is assigned to a particular alphabet.
2. Each different alphabet has a unique number.
3. All the same, alphabets have the same numbers.
4. The numbers should satisfy all the operations that any normal number does.

Crypt-Arithmetic problem

- Let us take an example of the message: SEND MORE MONEY.
- Here, to convert it into numeric form, we first split each word separately and represent it as follows:

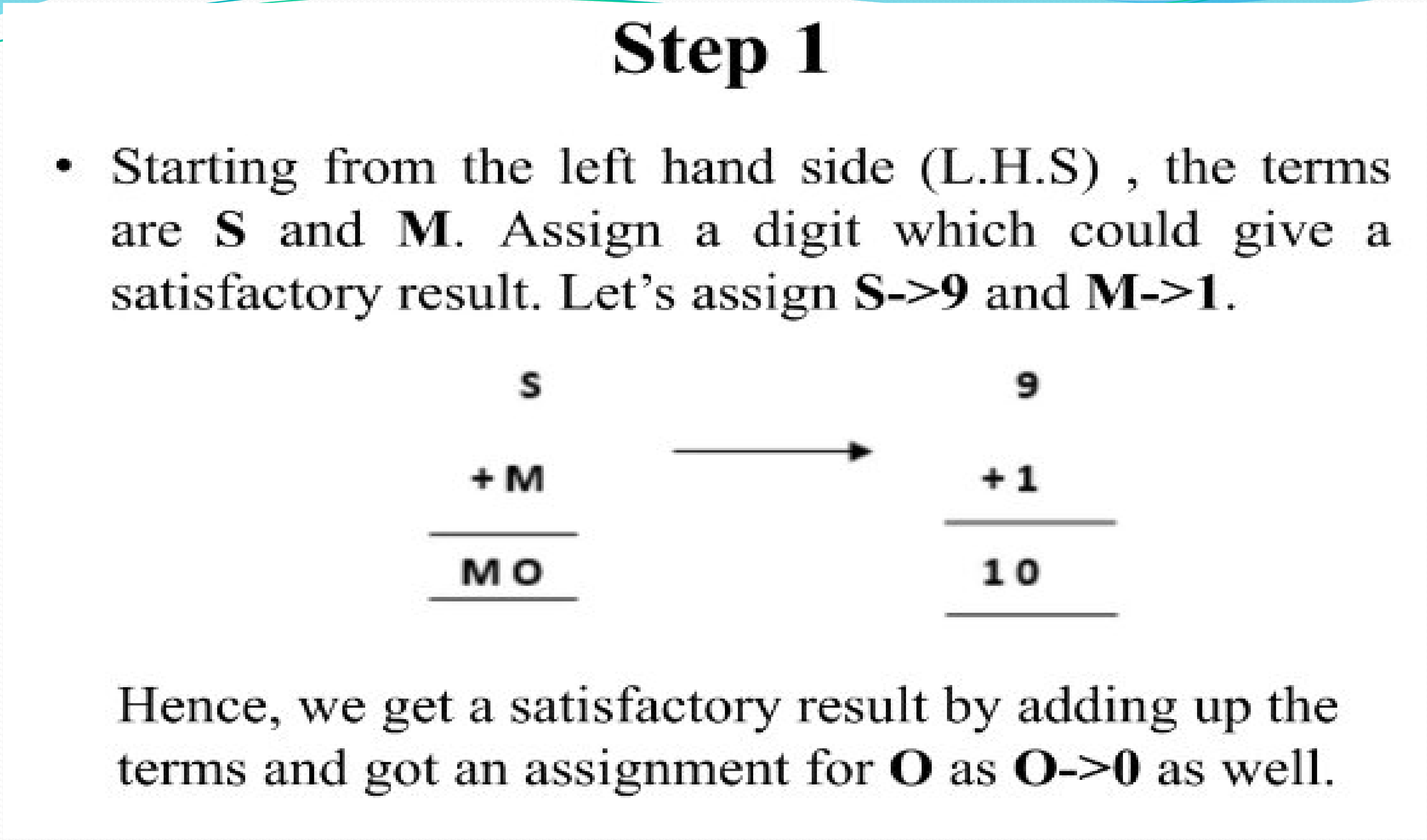


```
   S E N D
   9 5 6 7
 + M O R E
   1 0 8 5
 ---------
 M O N E Y
 1 0 6 5 2
```



```
  BASE
 +BALL
 -----
 GAMES
```

B	7
A	4
S	8
E	3
L	5
G	1
M	9



Step 1

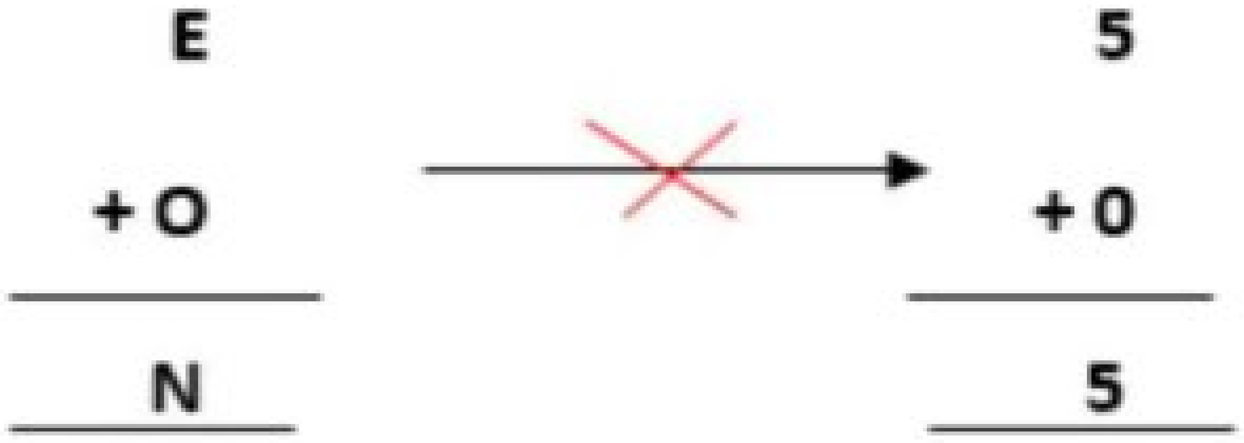
- Starting from the left hand side (L.H.S) , the terms are **S** and **M**. Assign a digit which could give a satisfactory result. Let’s assign **S->9** and **M->1**.

$$\begin{array}{r} S \\ +M \\ \hline MO \\ \hline \end{array} \longrightarrow \begin{array}{r} 9 \\ +1 \\ \hline 10 \\ \hline \end{array}$$

Hence, we get a satisfactory result by adding up the terms and got an assignment for **O** as **O->0** as well.

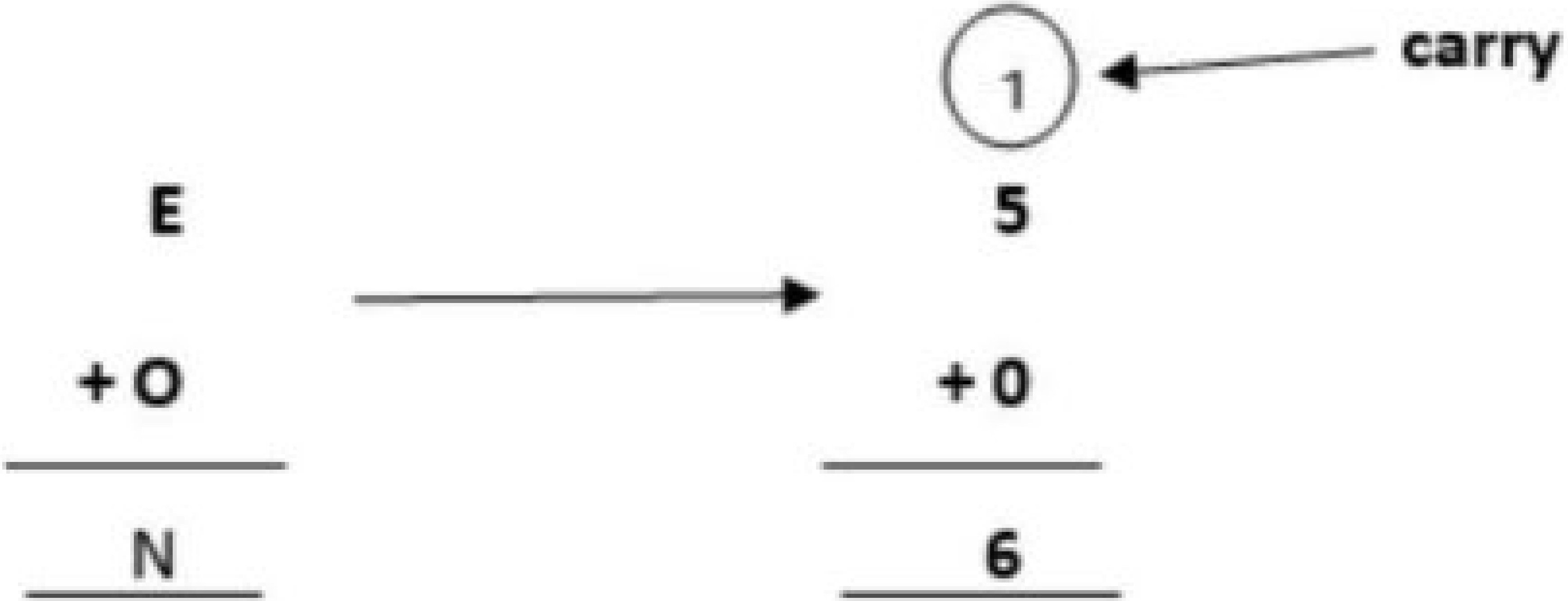
Step 2

- Now, move ahead to the next terms **E** and **O** to get **N** as its output.



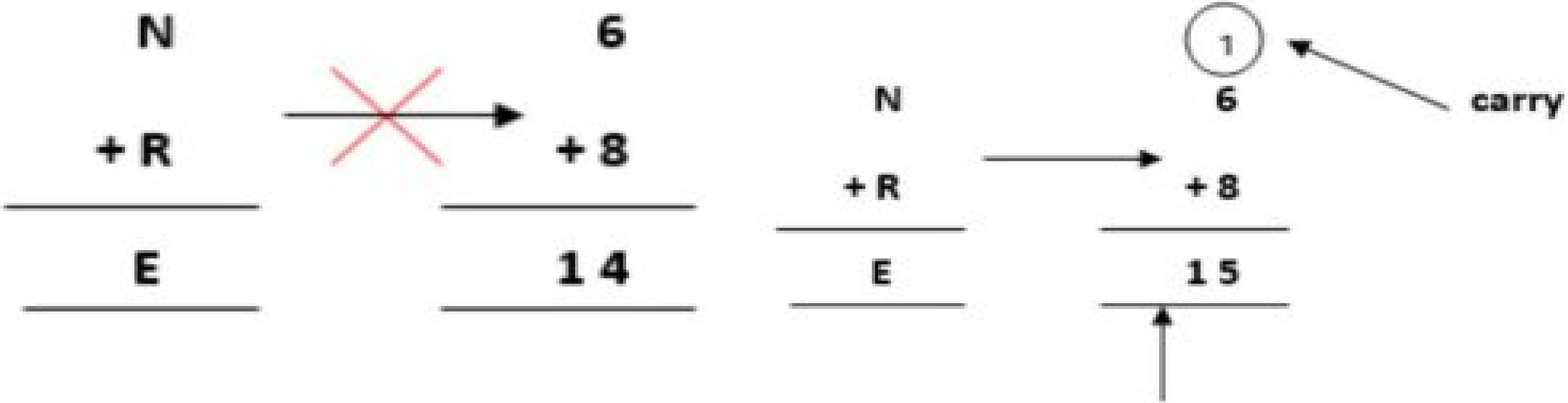
Adding E and O, which means 5+0=0, which is not possible because we cannot assign the same digit to two letters.

- **Add carry 1 to the value E to change the value of alphabet.**



Step 3

- Further, adding the next two terms **N** and **R** we get,



N

+ R

E

6

+ 8

14

N

+ R

E

1

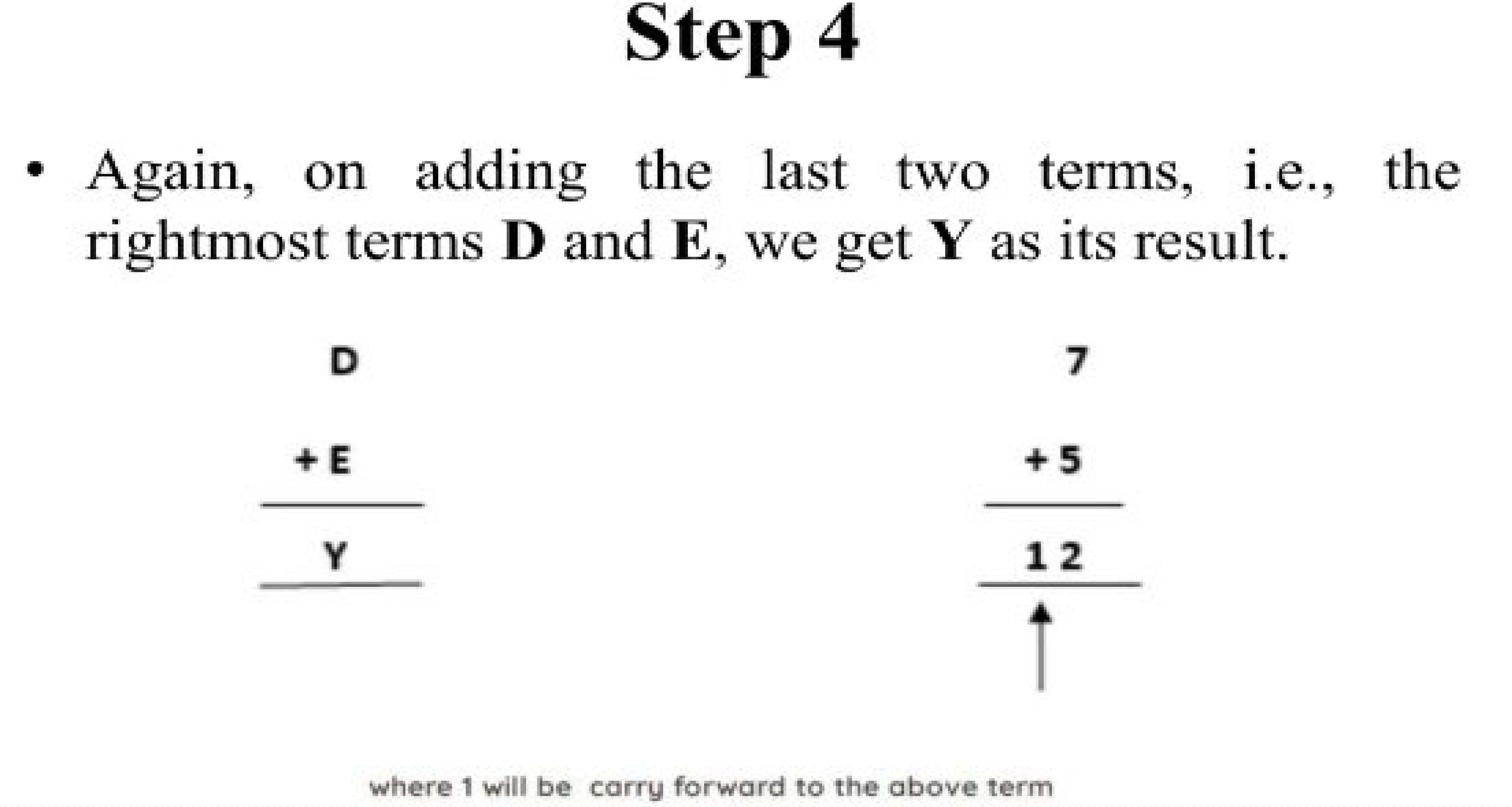
carry

6

+ 8

15

But, we have already assigned **E->5**. Not possible with **5 to E**
Again, after solving the whole problem, we will get a carryover on this term, so our answer will be satisfied.



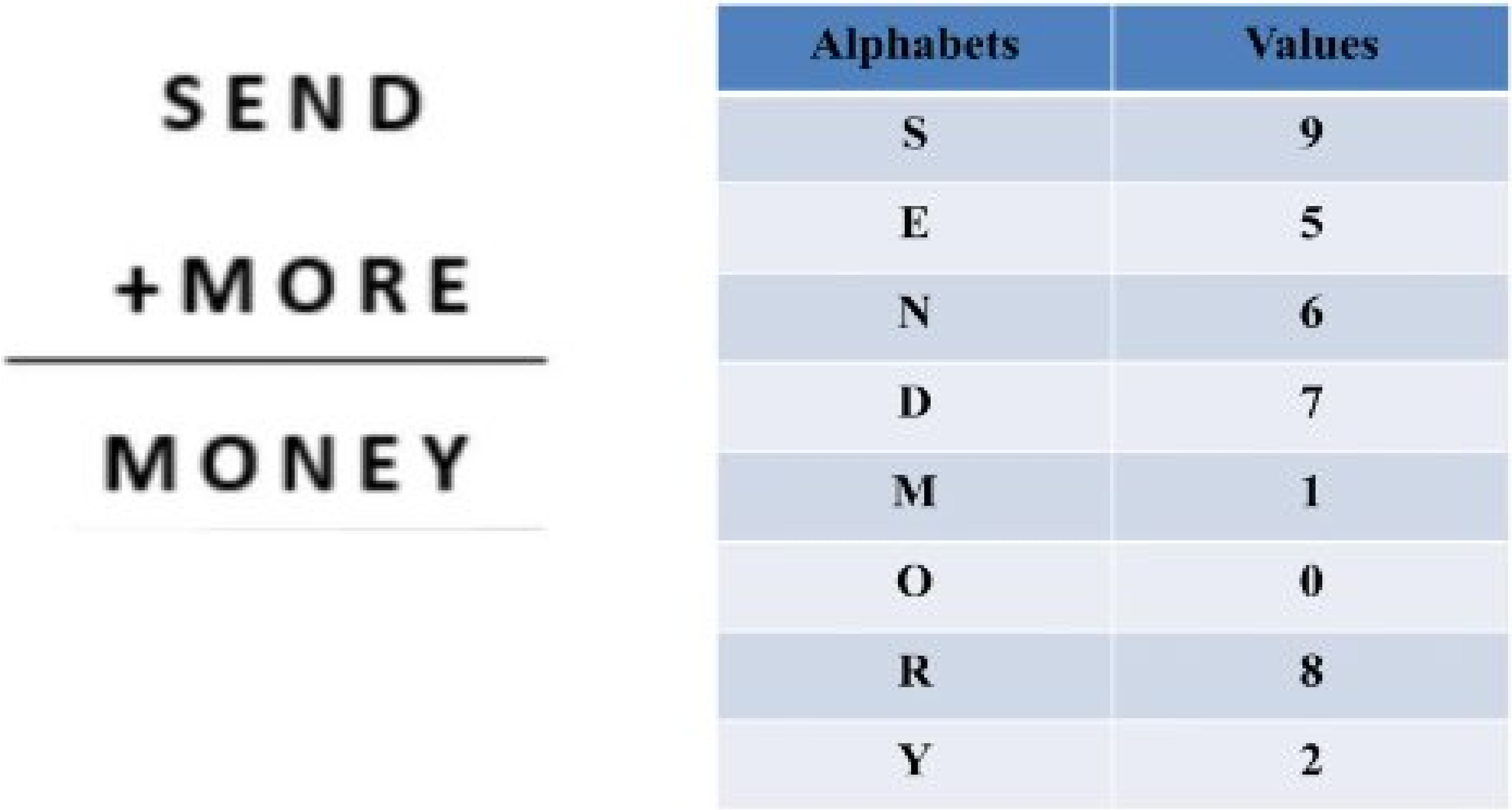
Step 4

- Again, on adding the last two terms, i.e., the rightmost terms **D** and **E**, we get **Y** as its result.

$$\begin{array}{c} D \\ +E \\ \hline Y \\ \hline \end{array} \qquad\qquad \begin{array}{c} 7 \\ +5 \\ \hline 12 \\ \hline \uparrow \end{array}$$

where 1 will be carry forward to the above term

- Keeping all the constraints in mind, the final resultant is as follows:



SEND

+MORE

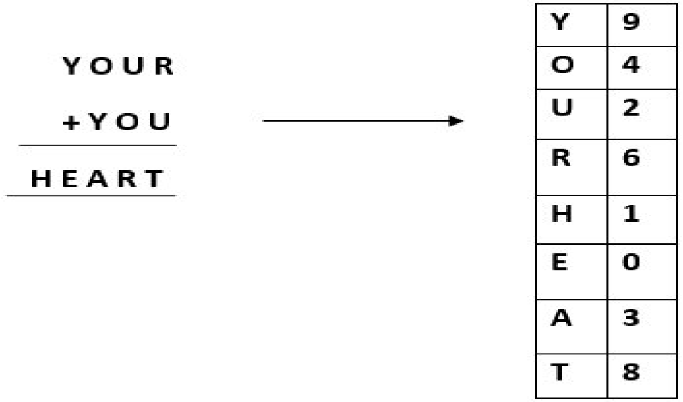
MONEY

Alphabets	Values
S	9
E	5
N	6
D	7
M	1
O	0
R	8
Y	2

Examples of Cryptarithmetic:

```
BLACK          7 9 2 0 8
GREEN          5 3 4 4 6
------------  ----------------
ORANGE       1 3 2 6 5 4
```

```
CRASH              3 6 8 4 5
HACKER         5 8  3 9 2 6
------------  -------------------
REBOOT        6  2  0 7 7 1
```



```
  YOUR
 +YOU
 HEART
```

Y	9
O	4
U	2
R	6
H	1
E	0
A	3
T	8

```
CROSS            9 6 2 3 3
ROADS            6 2 5 1 3
------------  ----------------
DANGER      1 5 8 7 4 6
```