Chapter-2-Part-III-Adv ersarial search

Learning objectives.

This chapter :

- Understand the concept of game search
- Apply various algorithms to check the optimality of the game tree
- Comprehend optimal strategies
- Interpret alpha-beta pruning

- "Unpredictable" opponent □ specifying a move for every possible opponent’s reply.

- Time limits □ unlikely to find goal, one must approximate

Introduction

- Games, sometimes to a frightening amount, amuse the human intellect since the starting of the civilisation.
- Probably, it arrives from the aspect that a game can be understood as an idealised world in which the opposing players compete with each other and face it.
- Competition is present at every field of the life.
- Ever since the advent of artificial intelligence (AI), game playing has been one of the most interesting applications of AI.
- In 1950, Claude Shannon and Alan Turing wrote the first game program for chess.
- It was the time when almost as soon as the computers became programmable.
- Games such as chess, tic-tac-toe, and Go are interesting as these games offer a pure abstraction of the competition between the two armies.

Optimal strategies

Mainly, the games of strategy with the following characteristics 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 maximise the payment.

Game Vs search problem:

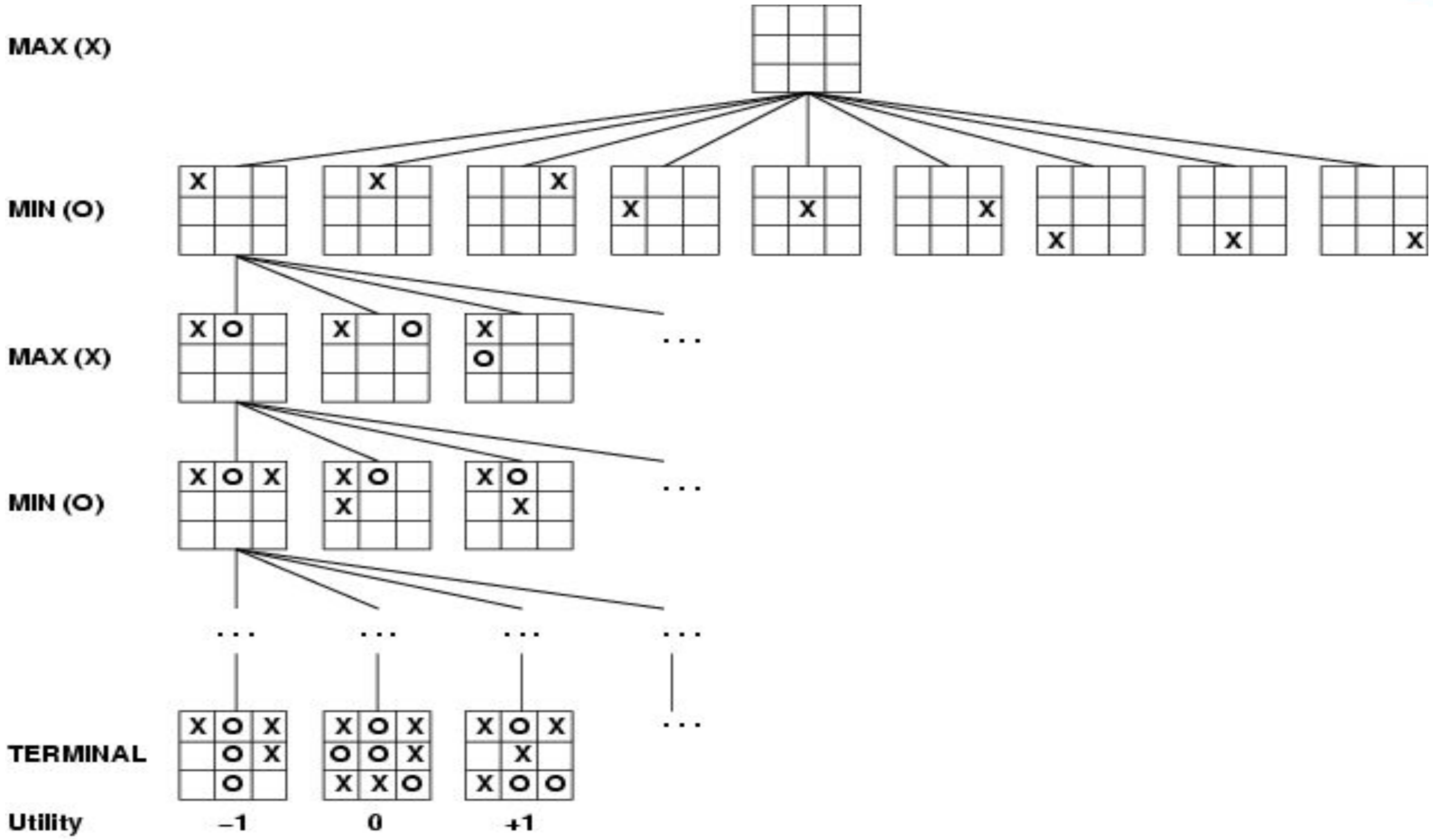
Unpredictable opponent: Specifies a move for every possible opponent reply.

Time limits: Unlikely to find goal, must be approximate.

Game tree

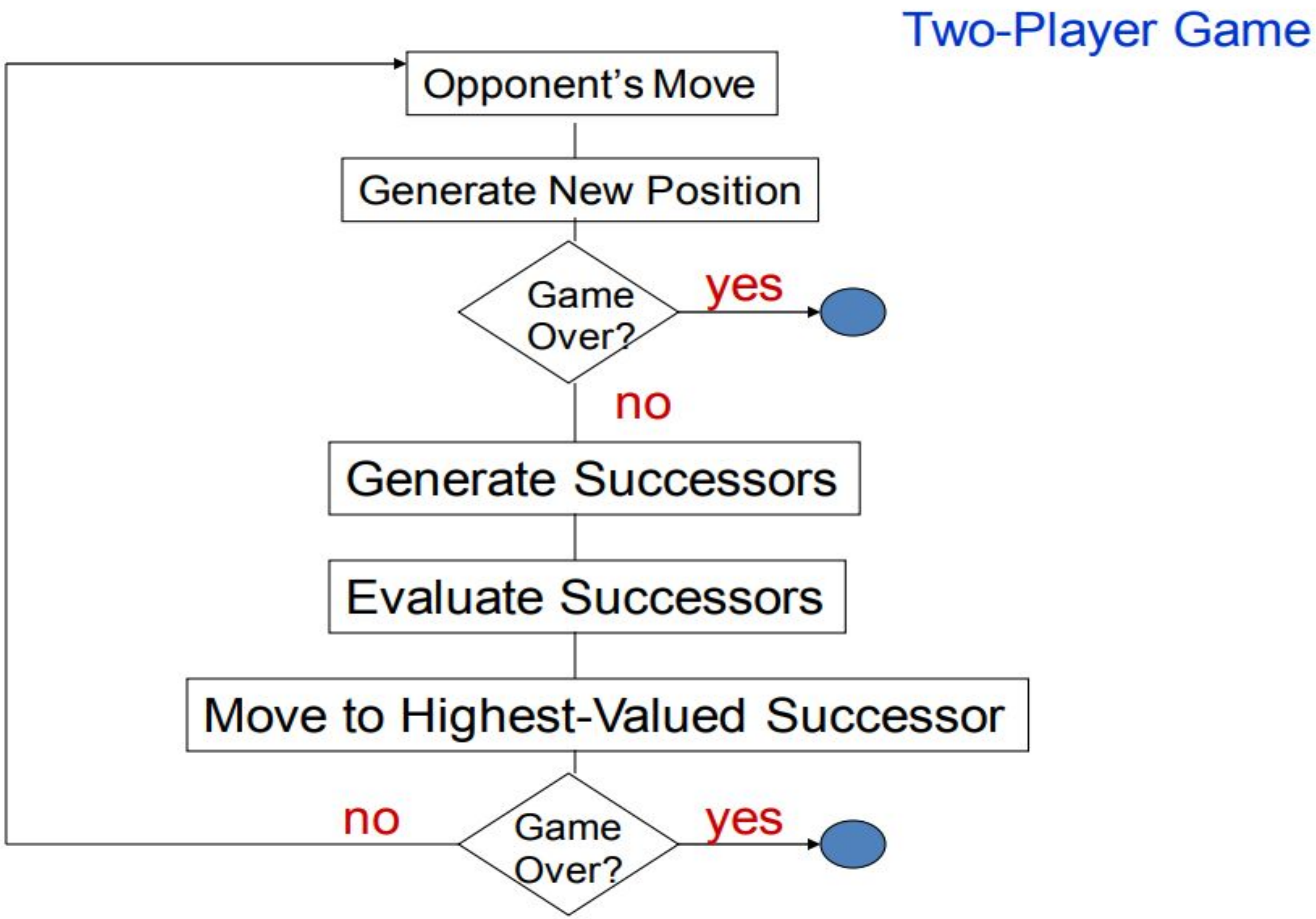
A game with two players (MAX and MIN, MAX moves first, taking turn) can be defined as a search problem with the following:

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.



How do we search this tree to find the optimal move?

Optimal strategies



The minmax algorithm

Minimax is a recursive algorithm that is used to choose an optimal move for a player assuming that the other player is also playing optimally. It is used in games such as tic-tac-toe, Go, chess, isola, checkers and various other two-player games. Such games are known as games of perfect information as it is possible to see all the possible moves of a particular game. There can be two-player games, which are not of perfect information such as Scrabble as the move of the opponent cannot be predicted.

The players in the game are referred to as MAX and MIN. MAX represents the person who is trying to win the game and hence maximise his or her score. MIN represents the opponent who is trying to minimise the score of MAX.

This algorithm is used to look ahead and decide which move to make first. If the game space is small enough the entire space can be generated and the leaf nodes can be allocated a win (1) or loss (0) value

The minmax algorithm

These values might be propagated back up the tree to decide which node to use. In propagating the values back up the tree, a MAX node is appointed the maximum value of all its children and MIN node is appointed the minimum values of all its children.

```
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)
```

The minmax algorithm

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.

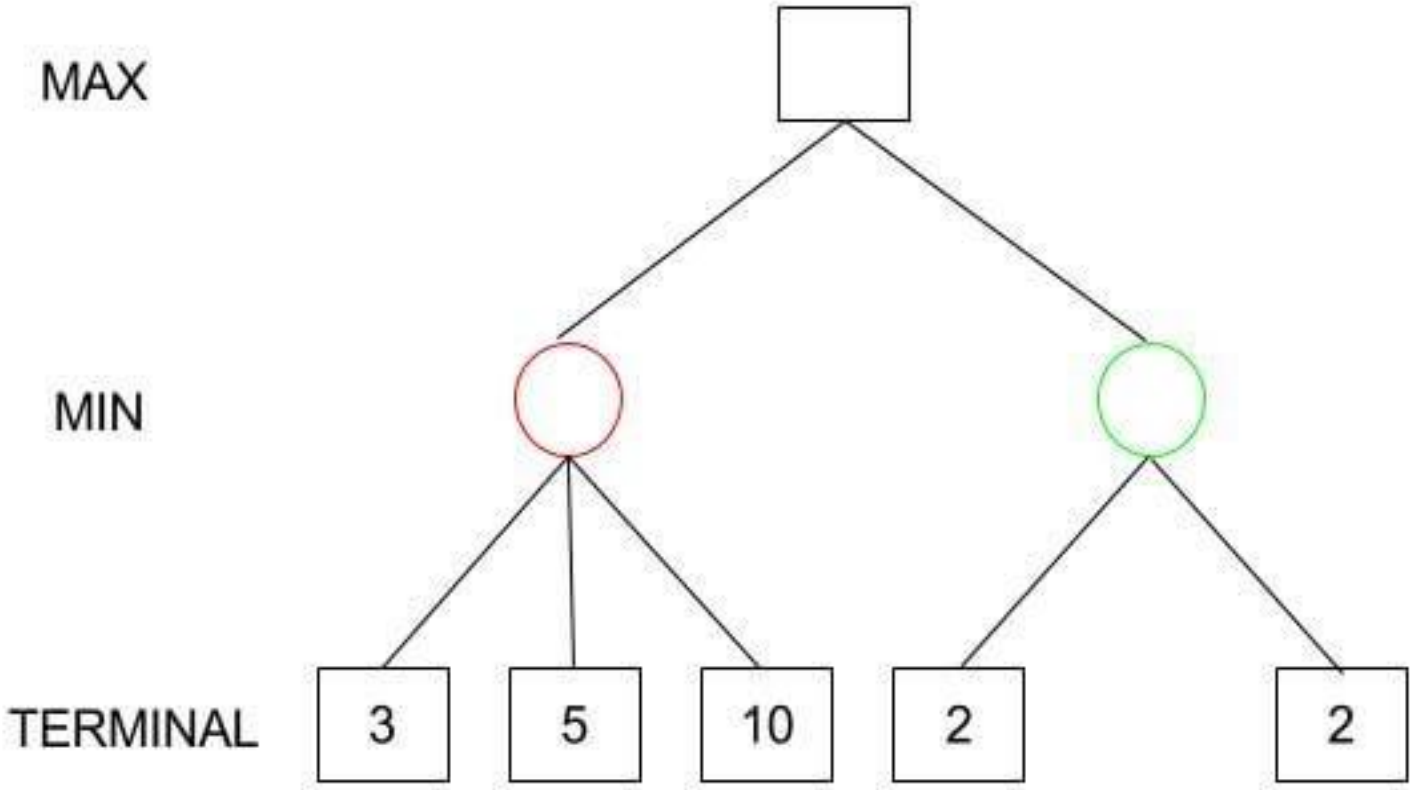
Step 2: Otherwise, if the level is a minimising level, use the minimax on the children of the current position. The minimum value of the results is reported here

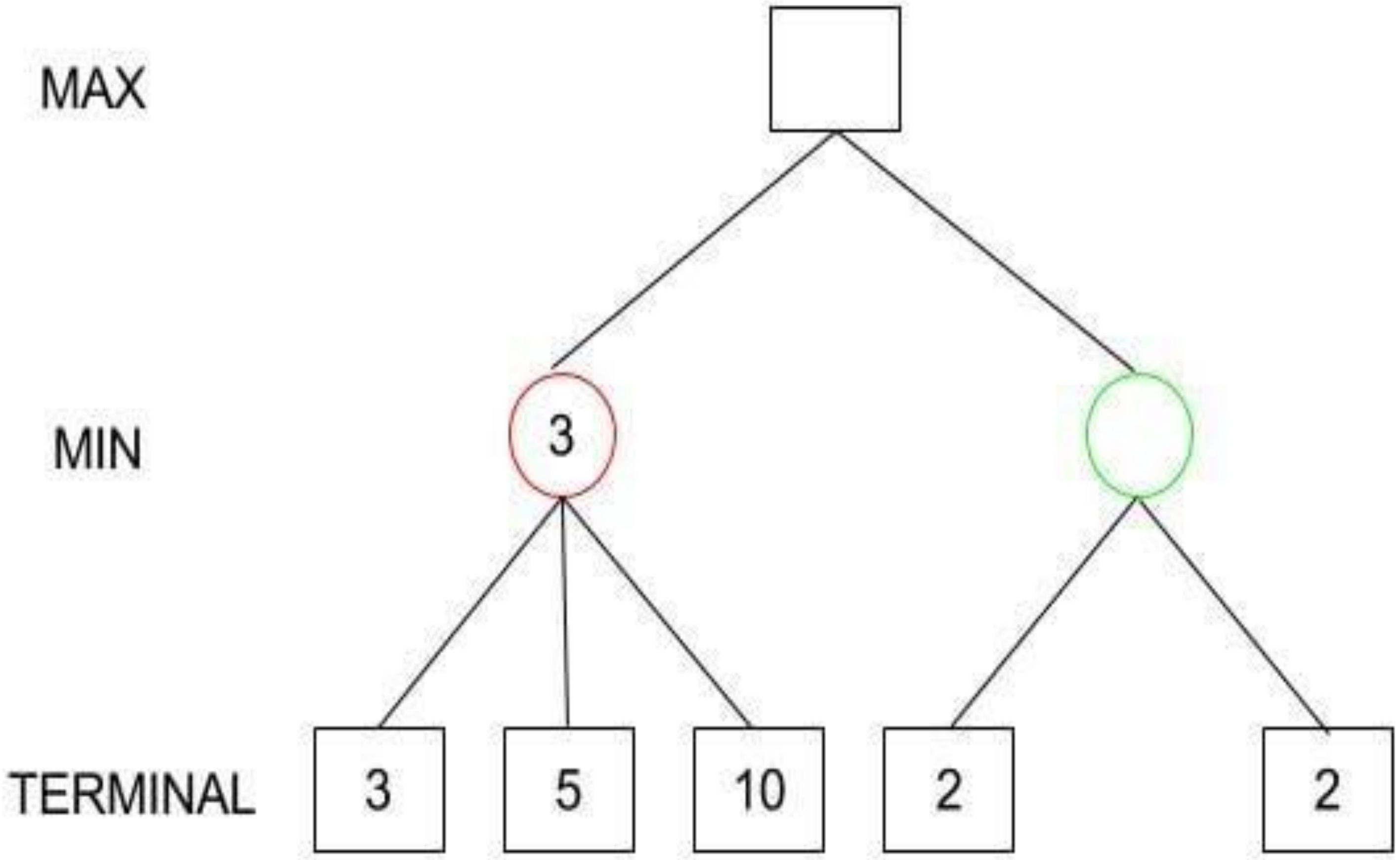
Step 3: Also, if the level is a maximising level, use the minimax on the children of the current position. The maximum of the results is reported here.

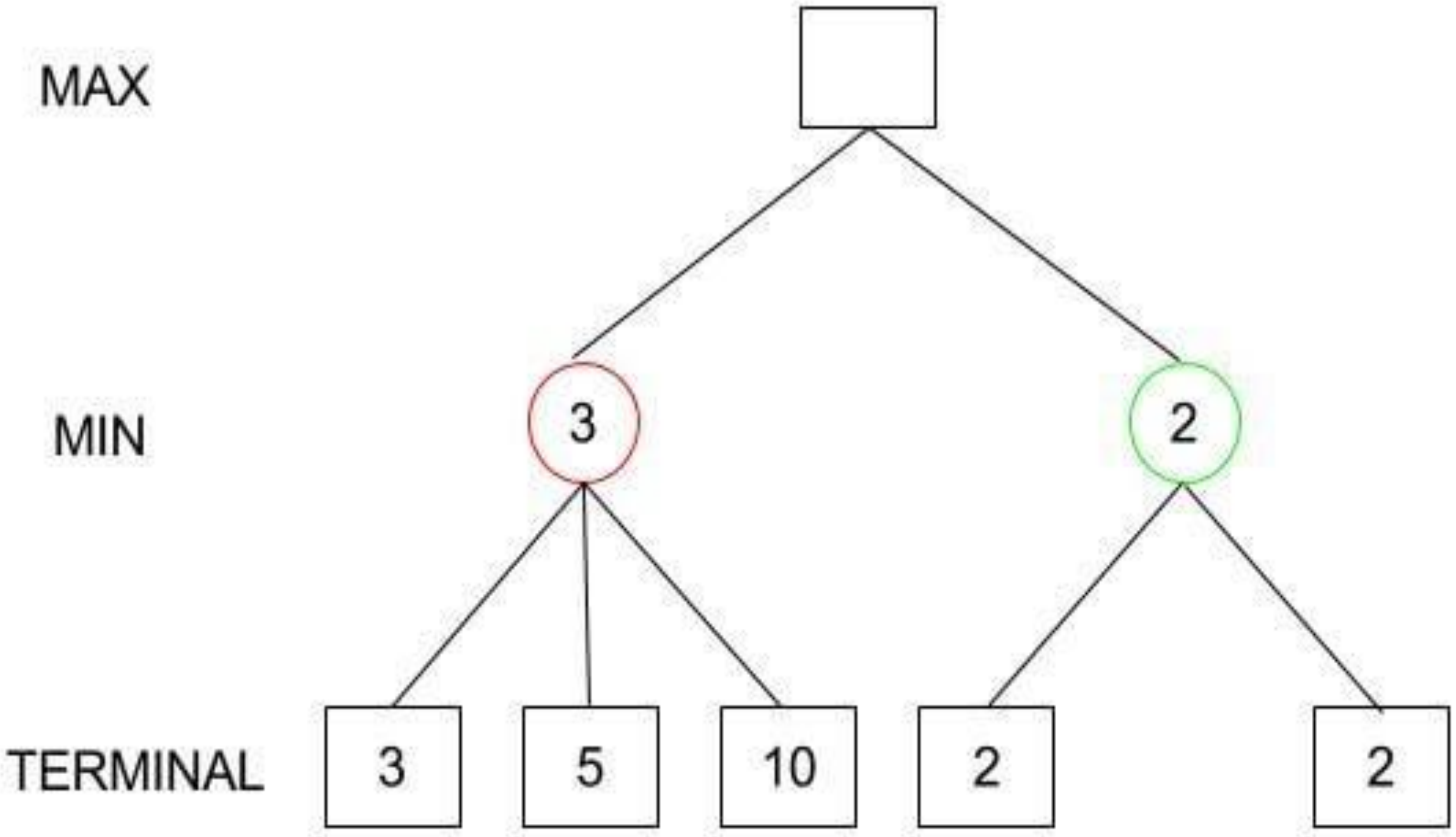
Step 4: The utility values is calculated with the help of leaves considering one layer at a time until the root of the tree.

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.

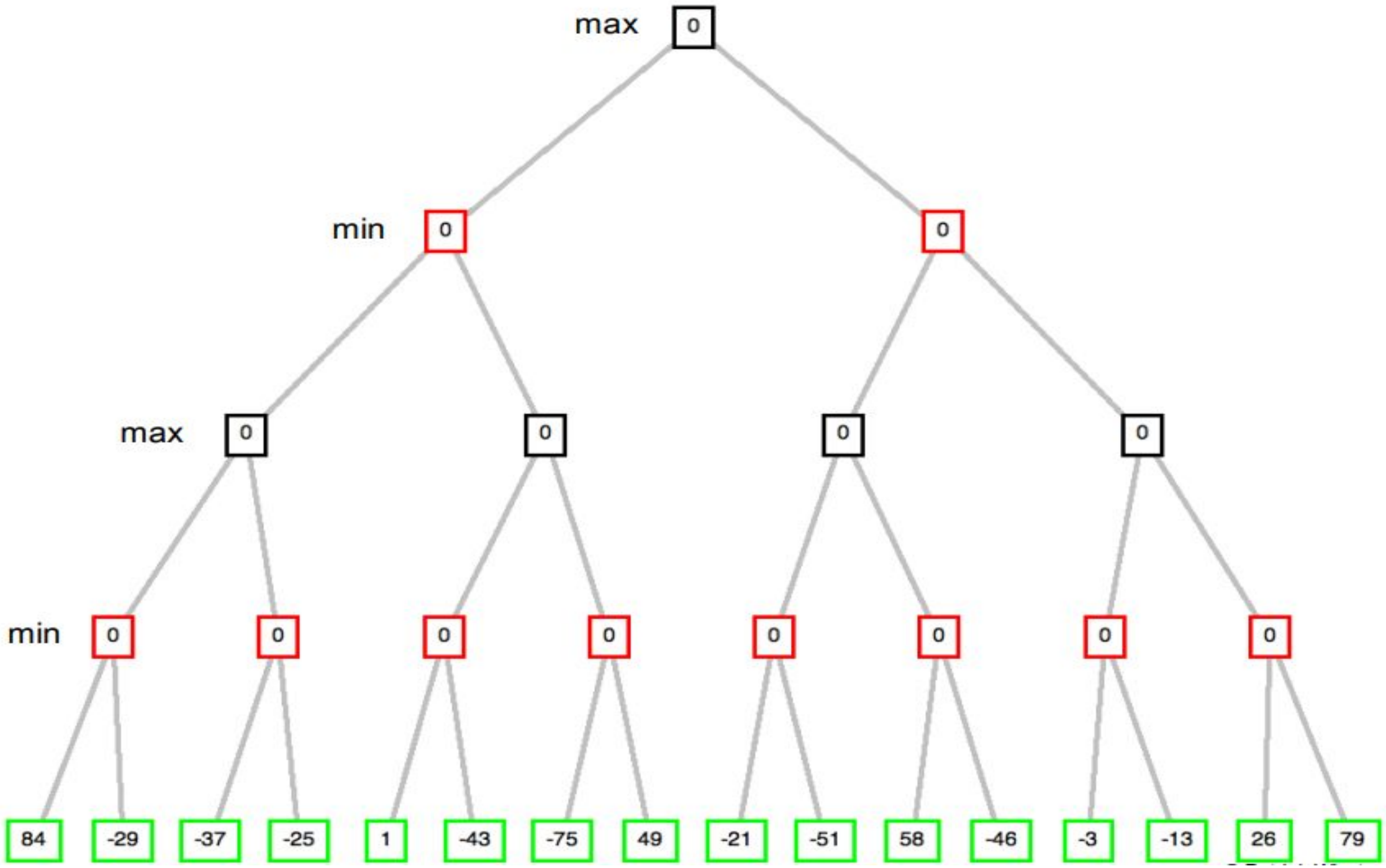
The minmax algorithm

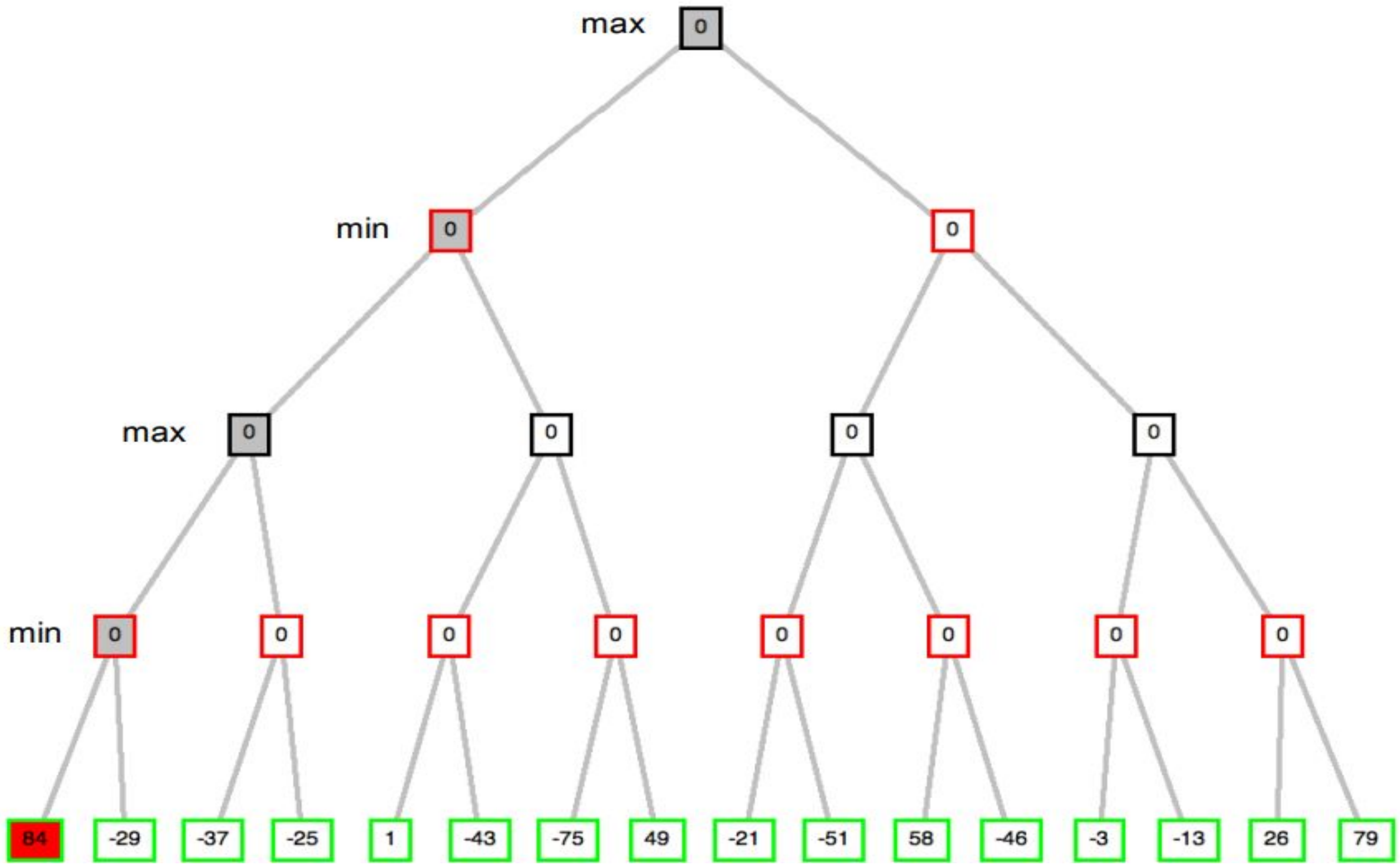


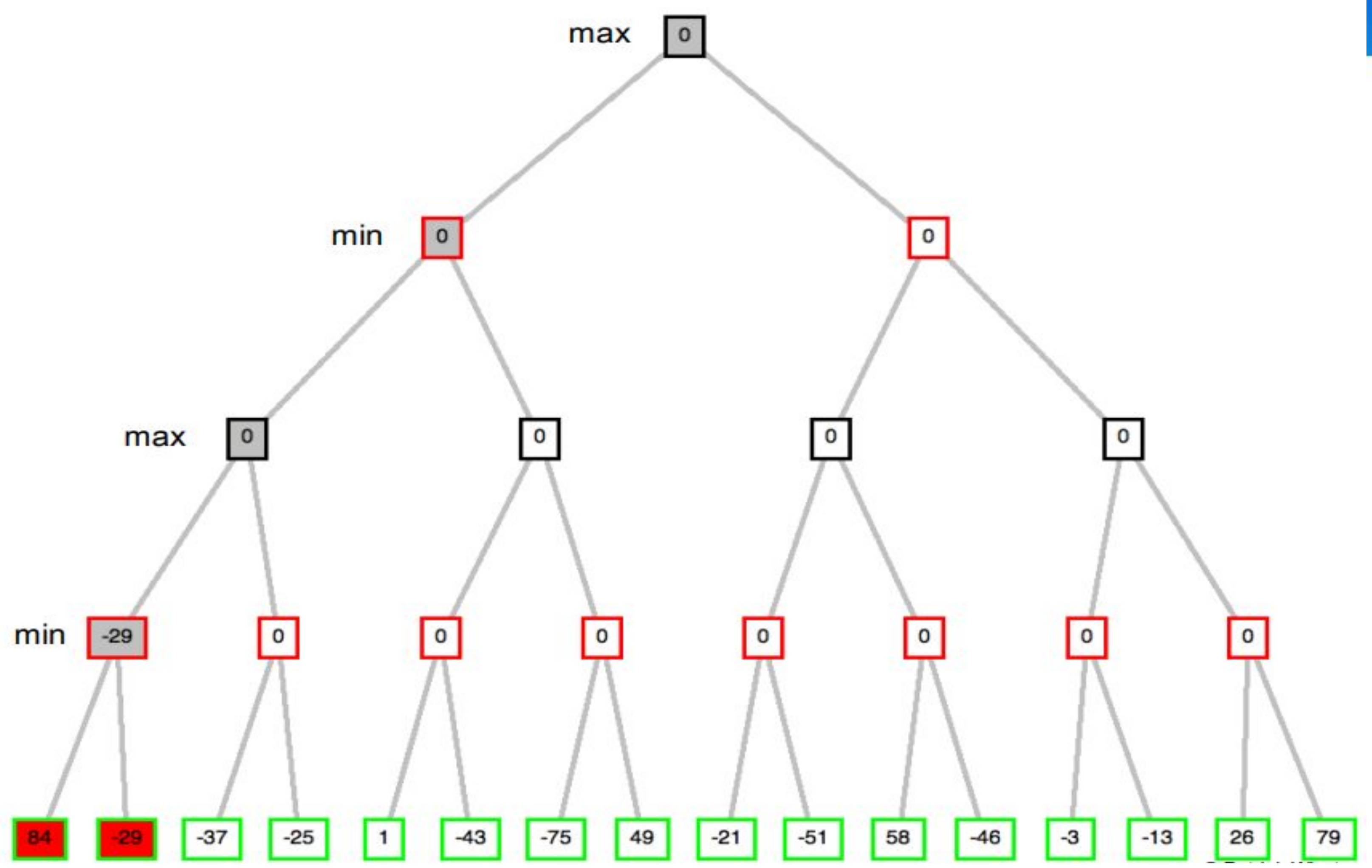

MAX
MIN
TERMINAL
3
3
5
10
2
2

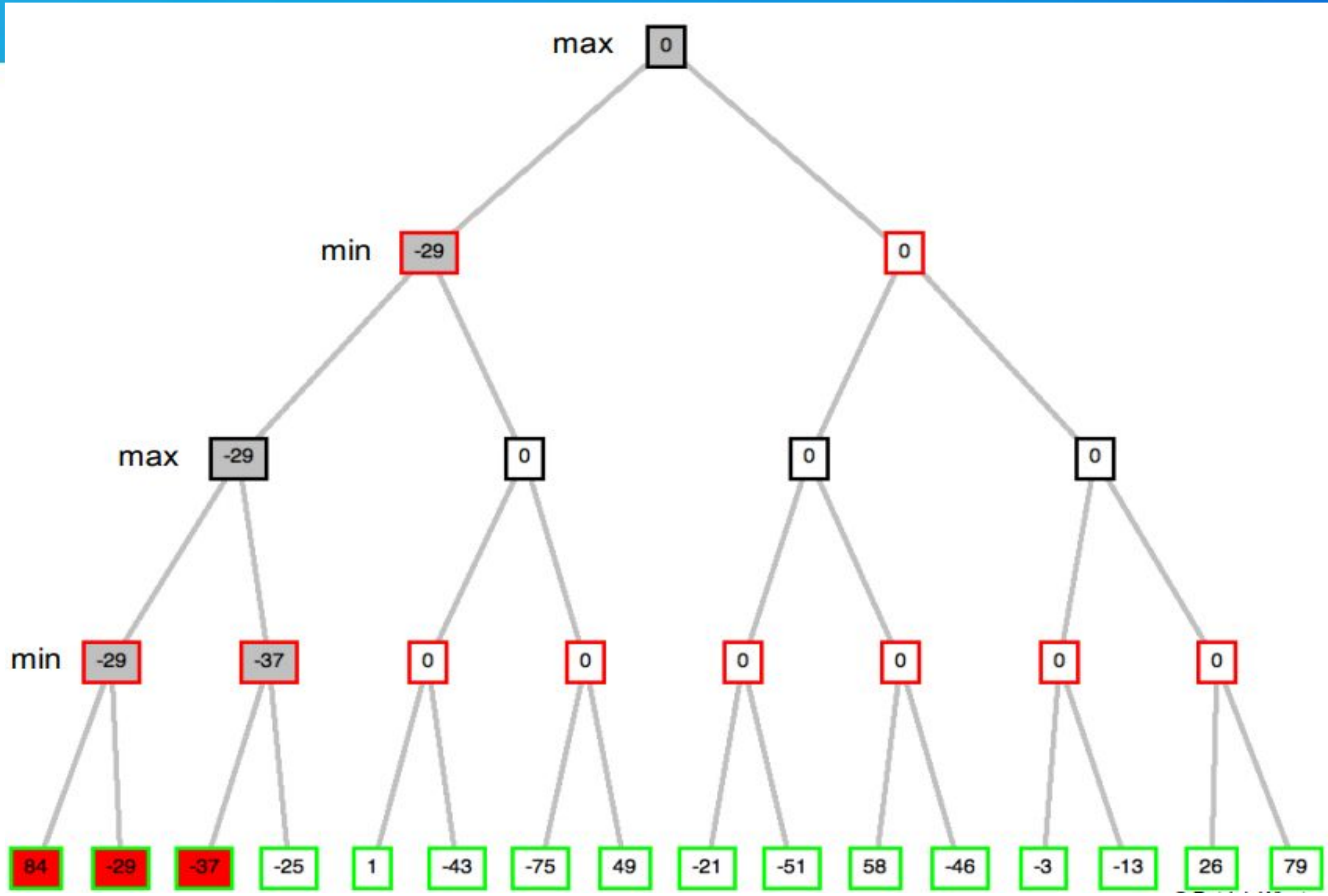


MAX
MIN
3
2
TERMINAL
3
5
10
2
2

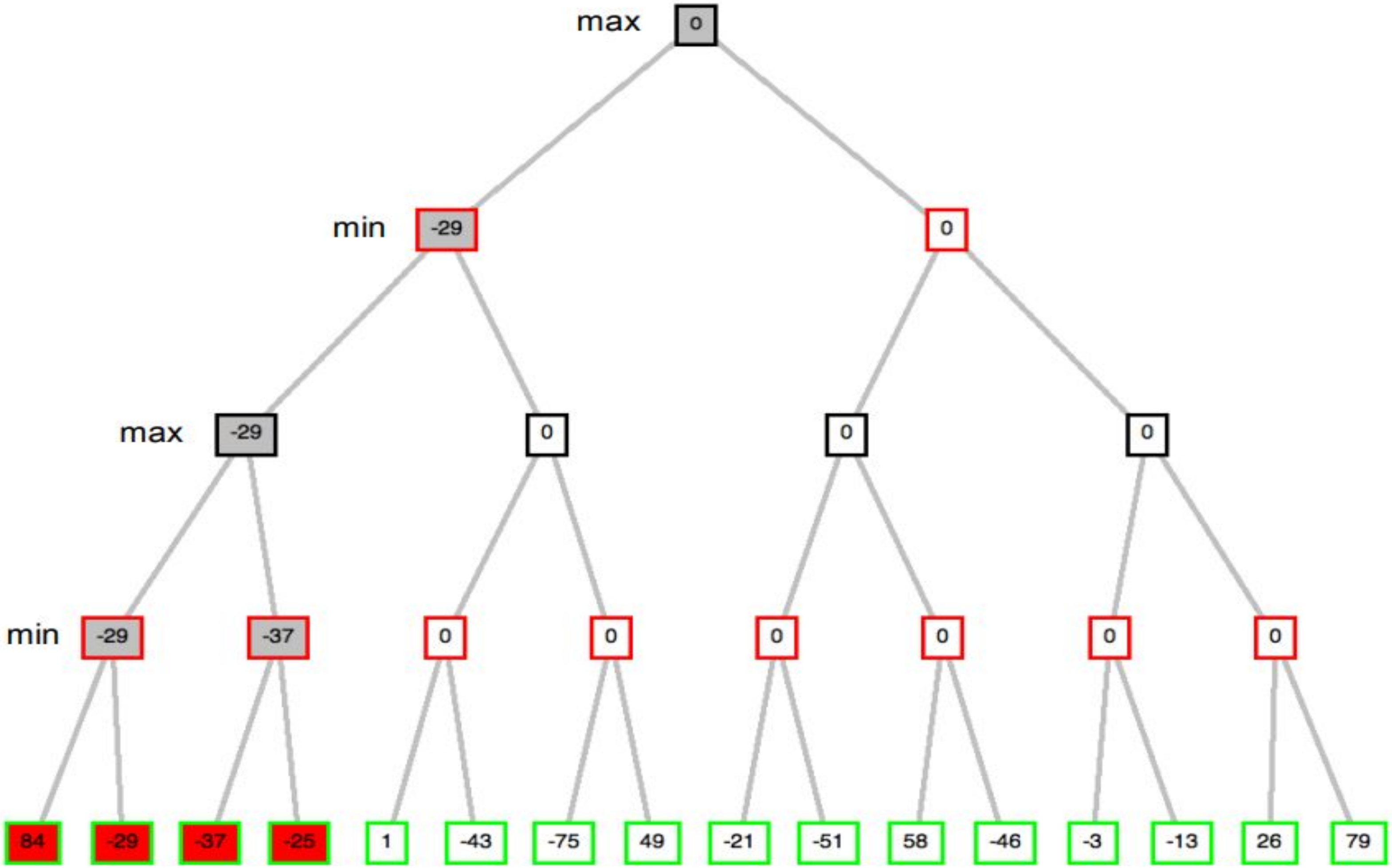

max
0
min
0
0
max
0
0
0
0
min
0
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

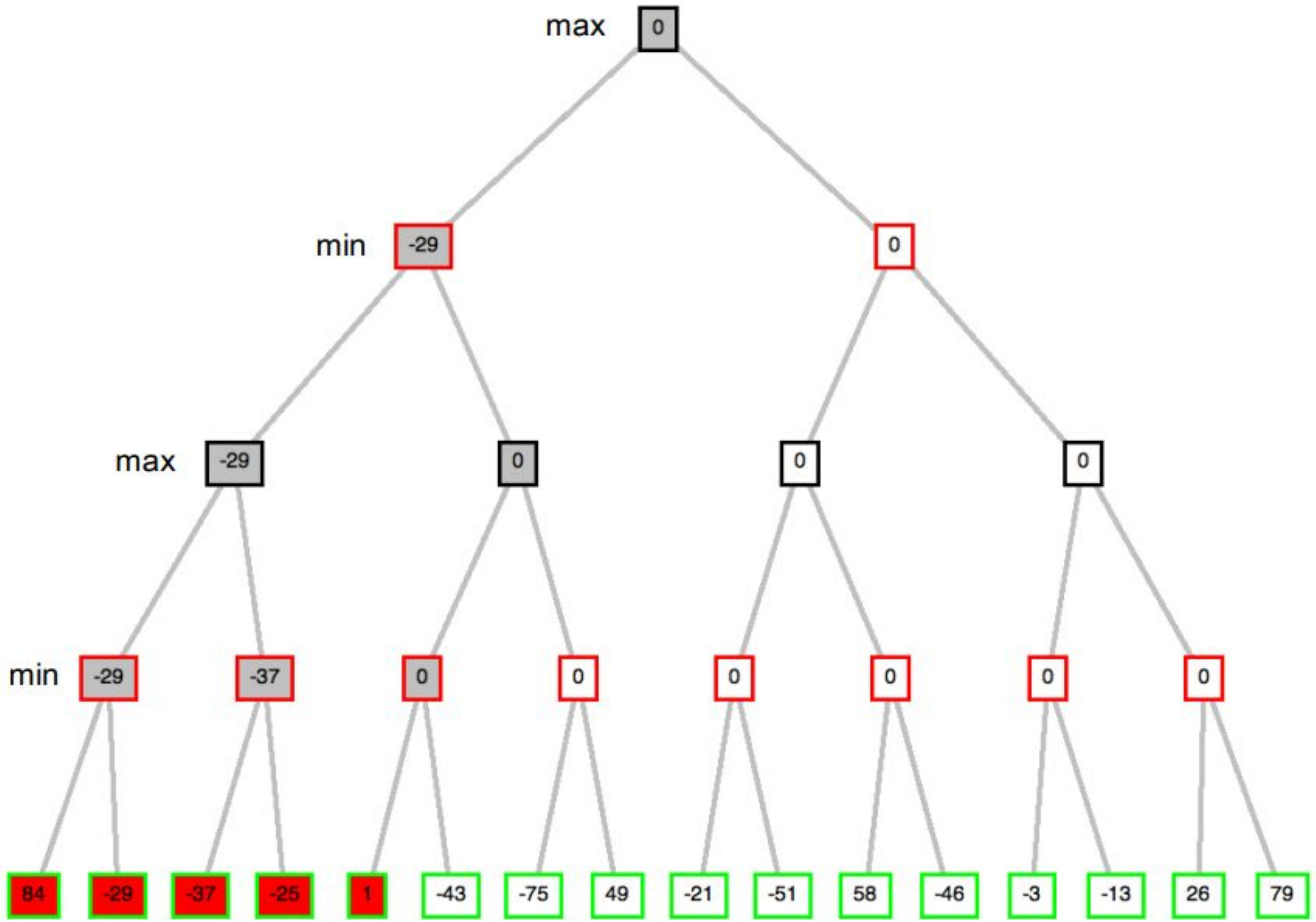

max
0
min
0
0
max
0
0
0
0
min
0
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

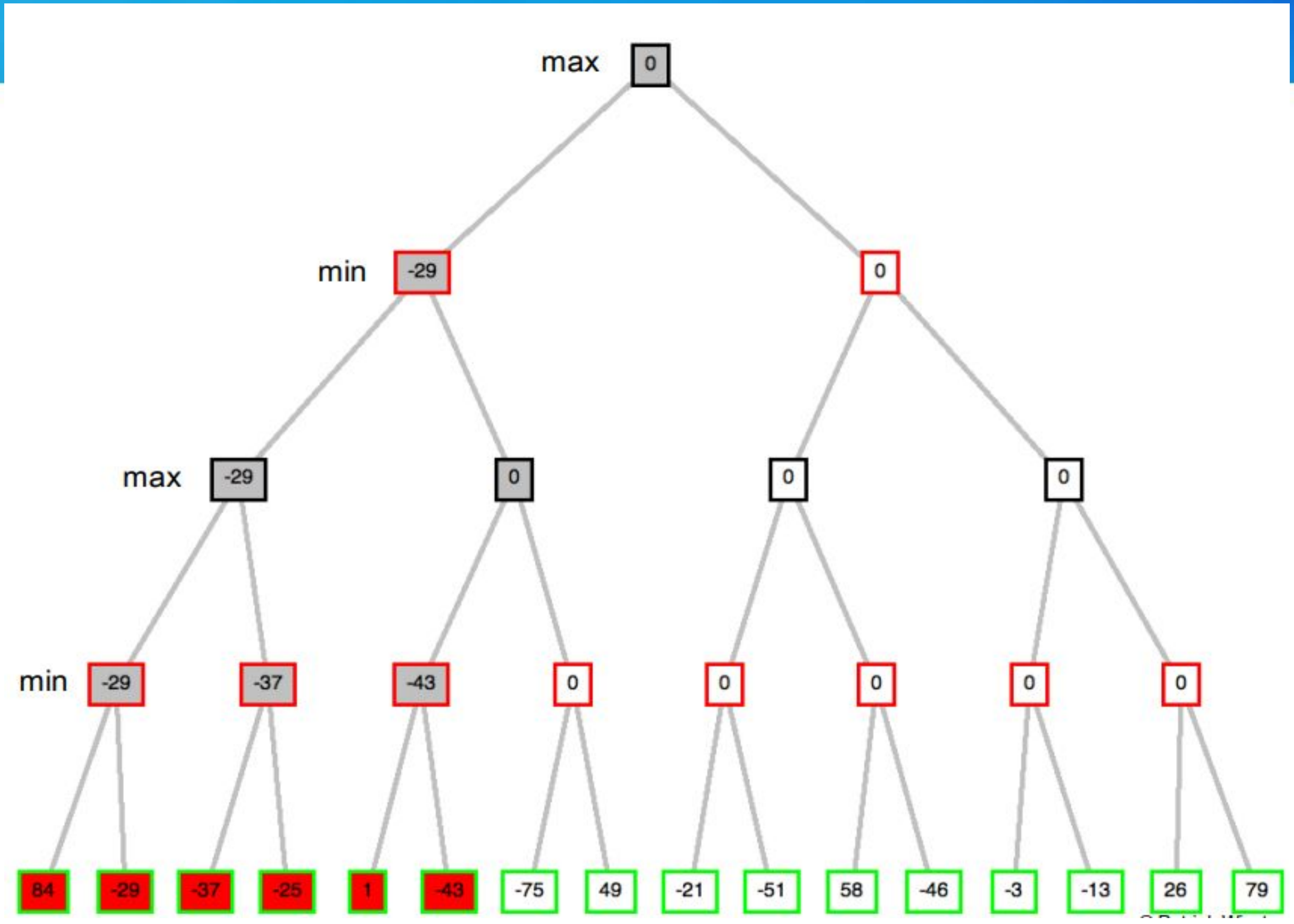

max
0
min
0
0
max
0
0
0
0
min
-29
0
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

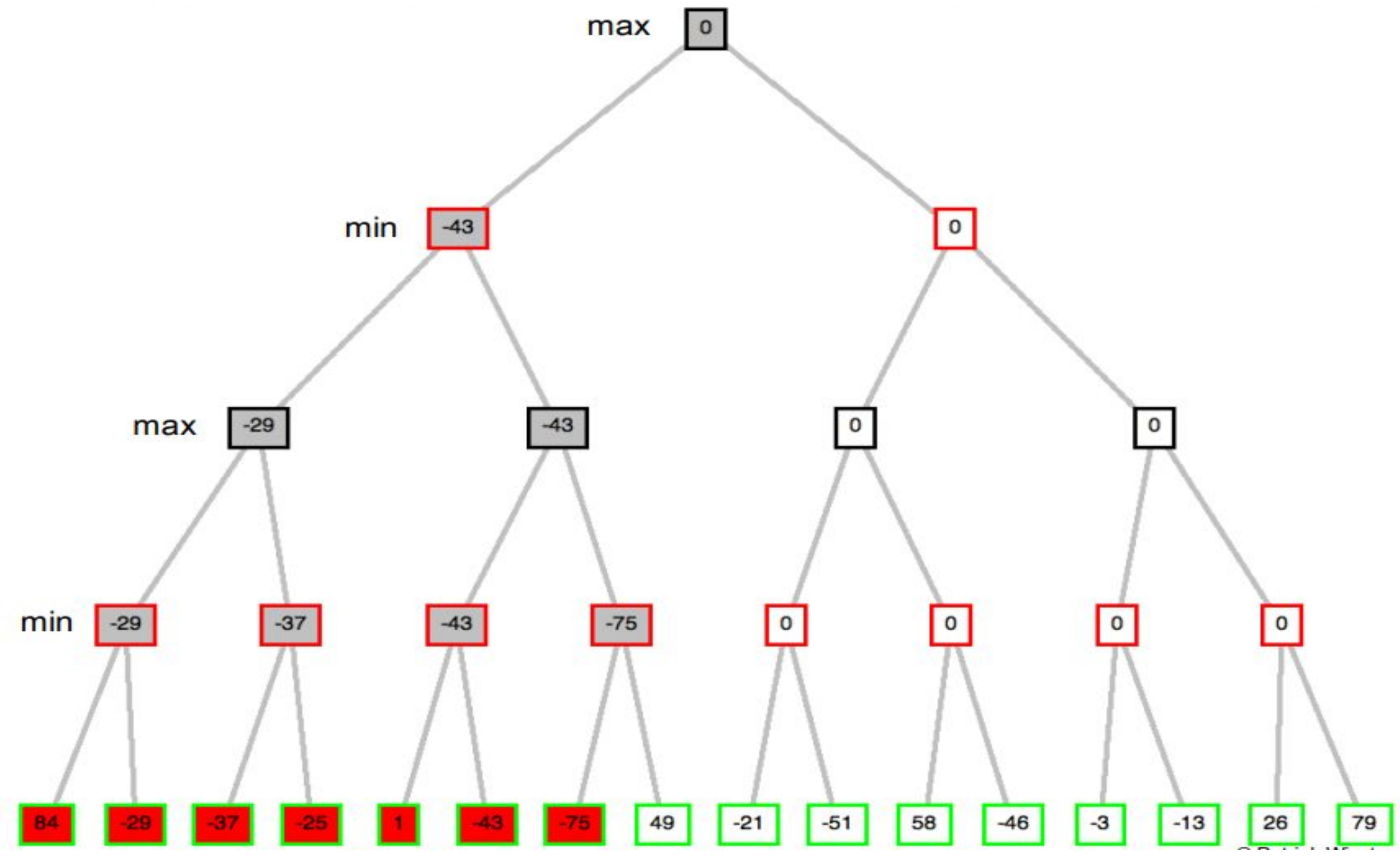


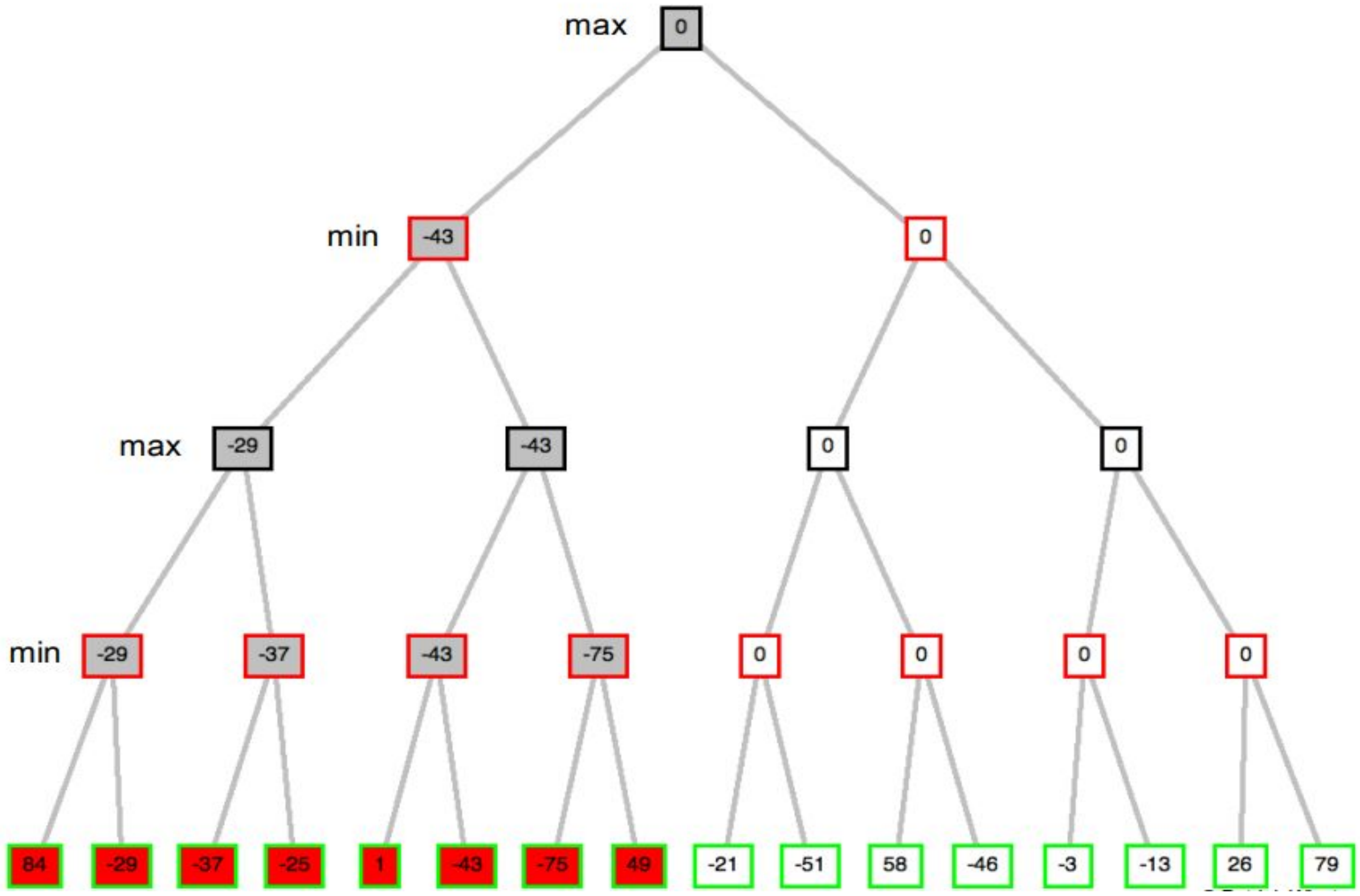
max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

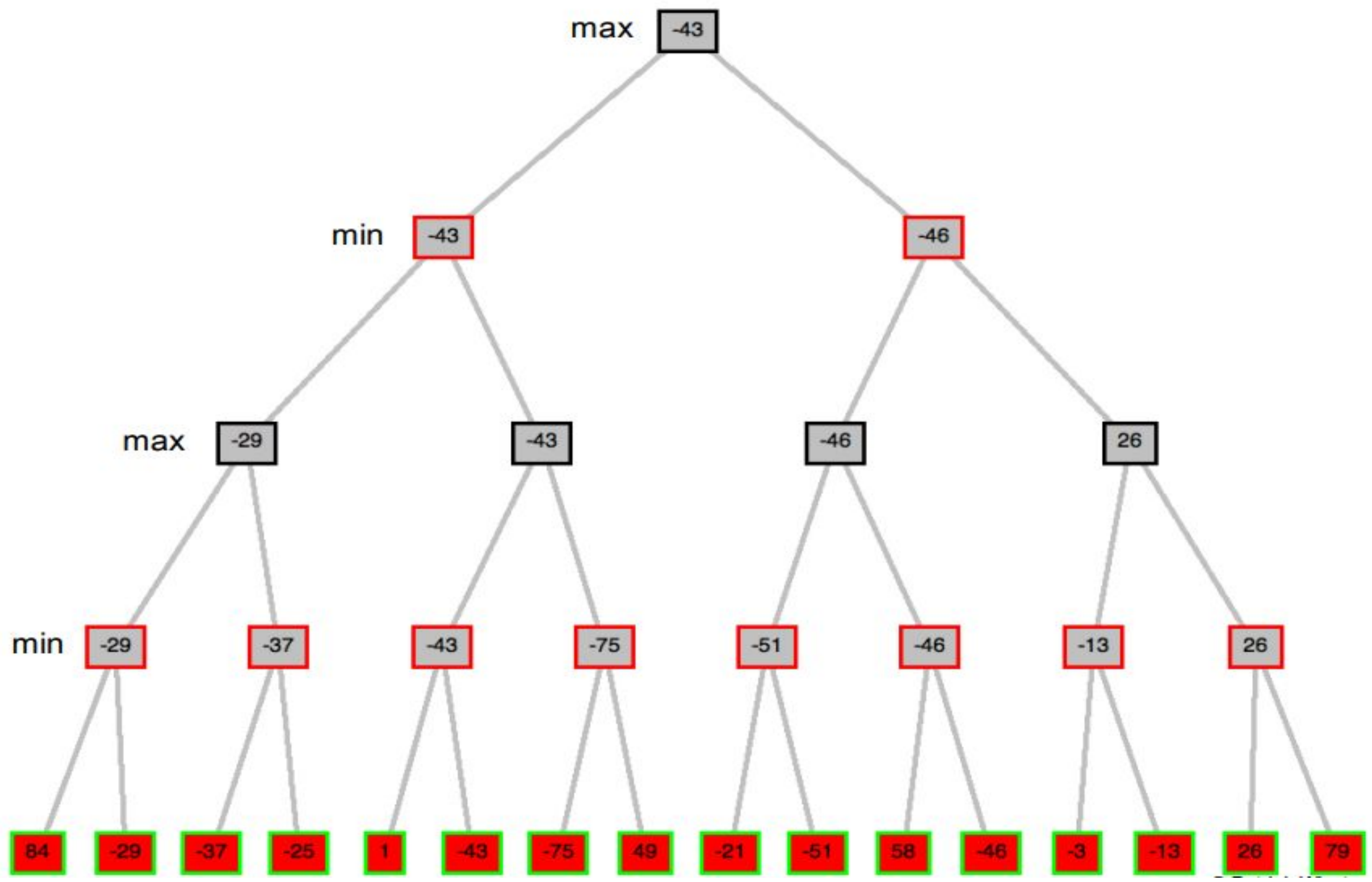

max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

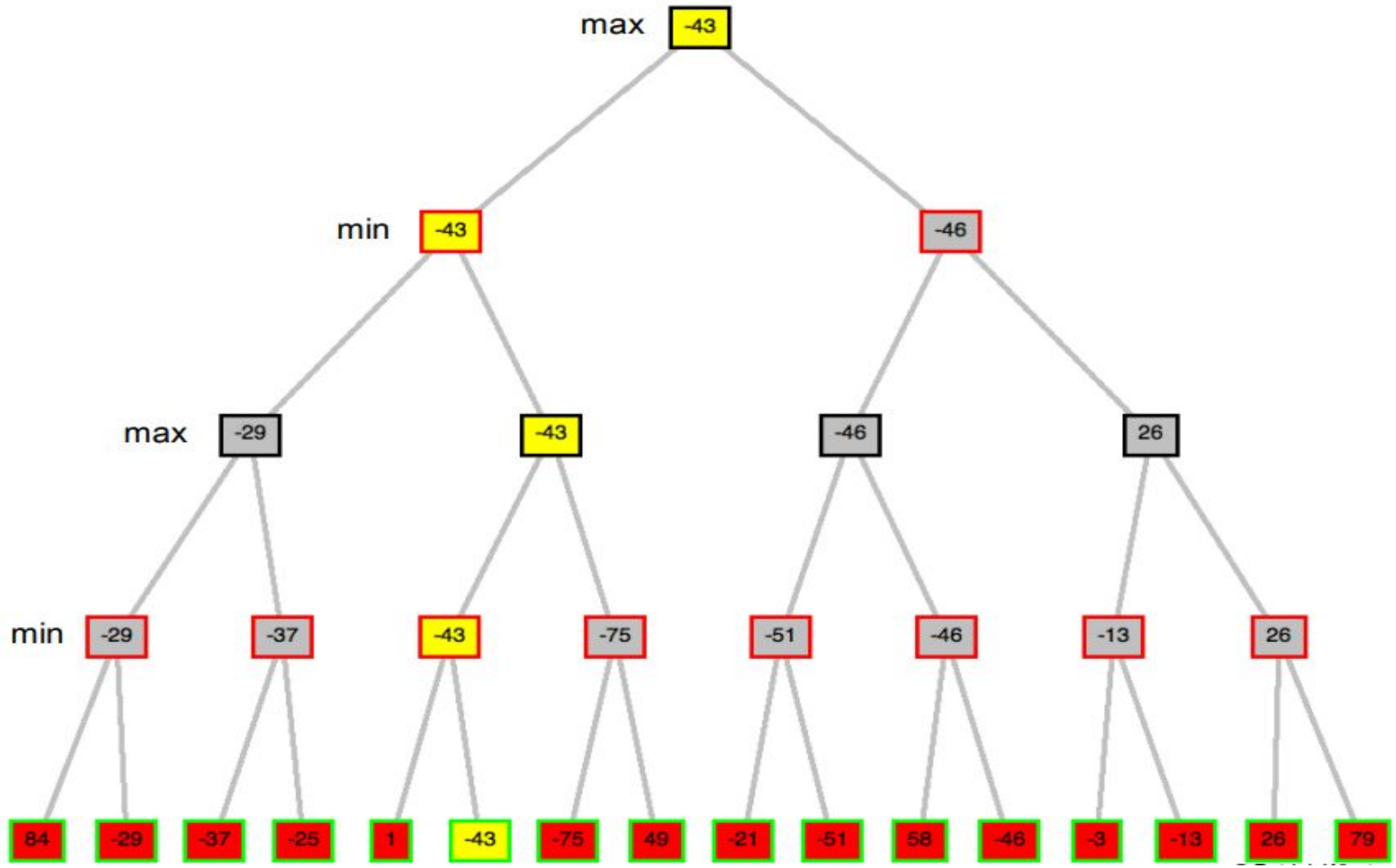

max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
0
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

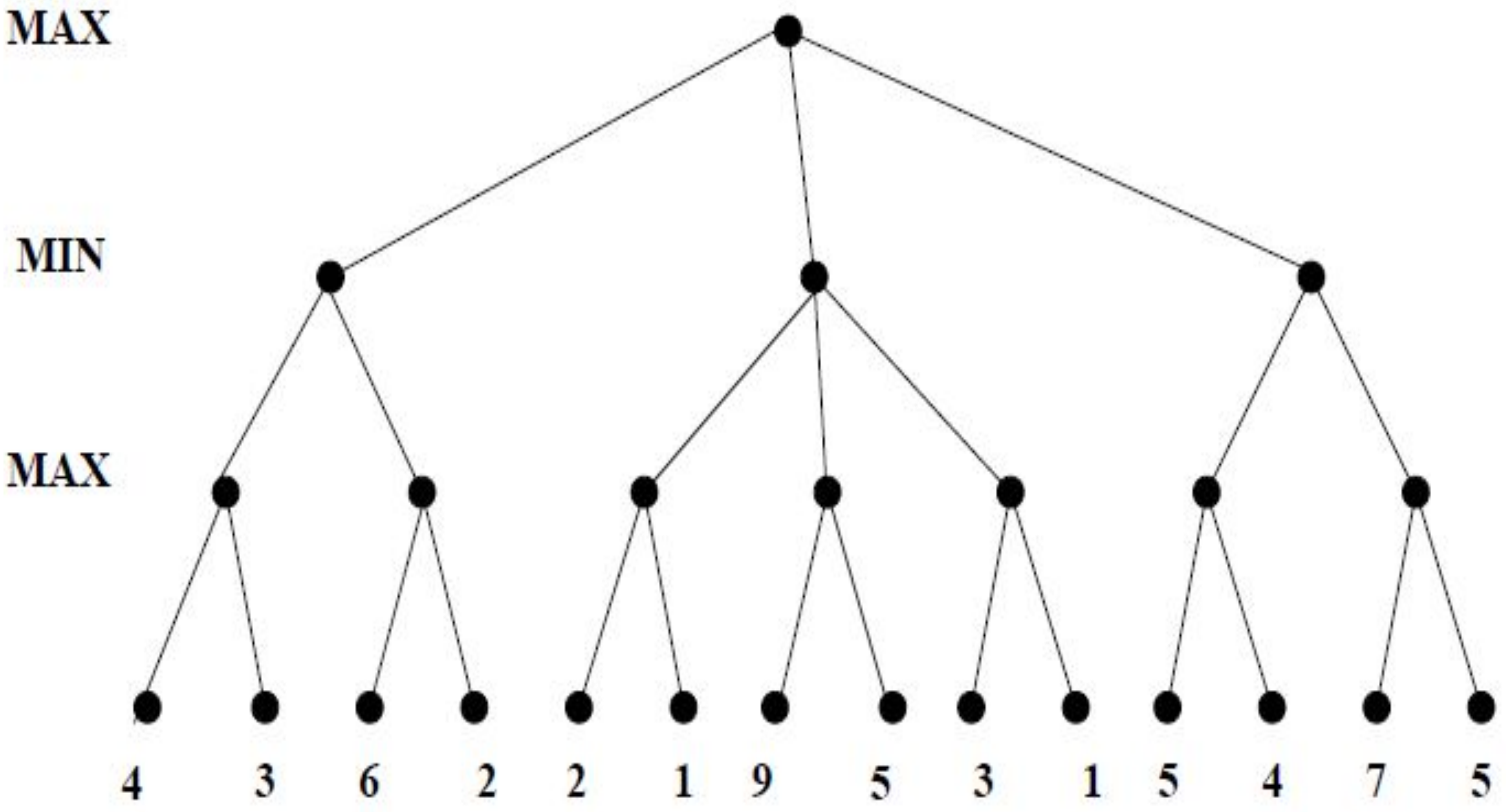

max
0
min
-29
0
max
-29
0
0
0
min
-29
-37
-43
0
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

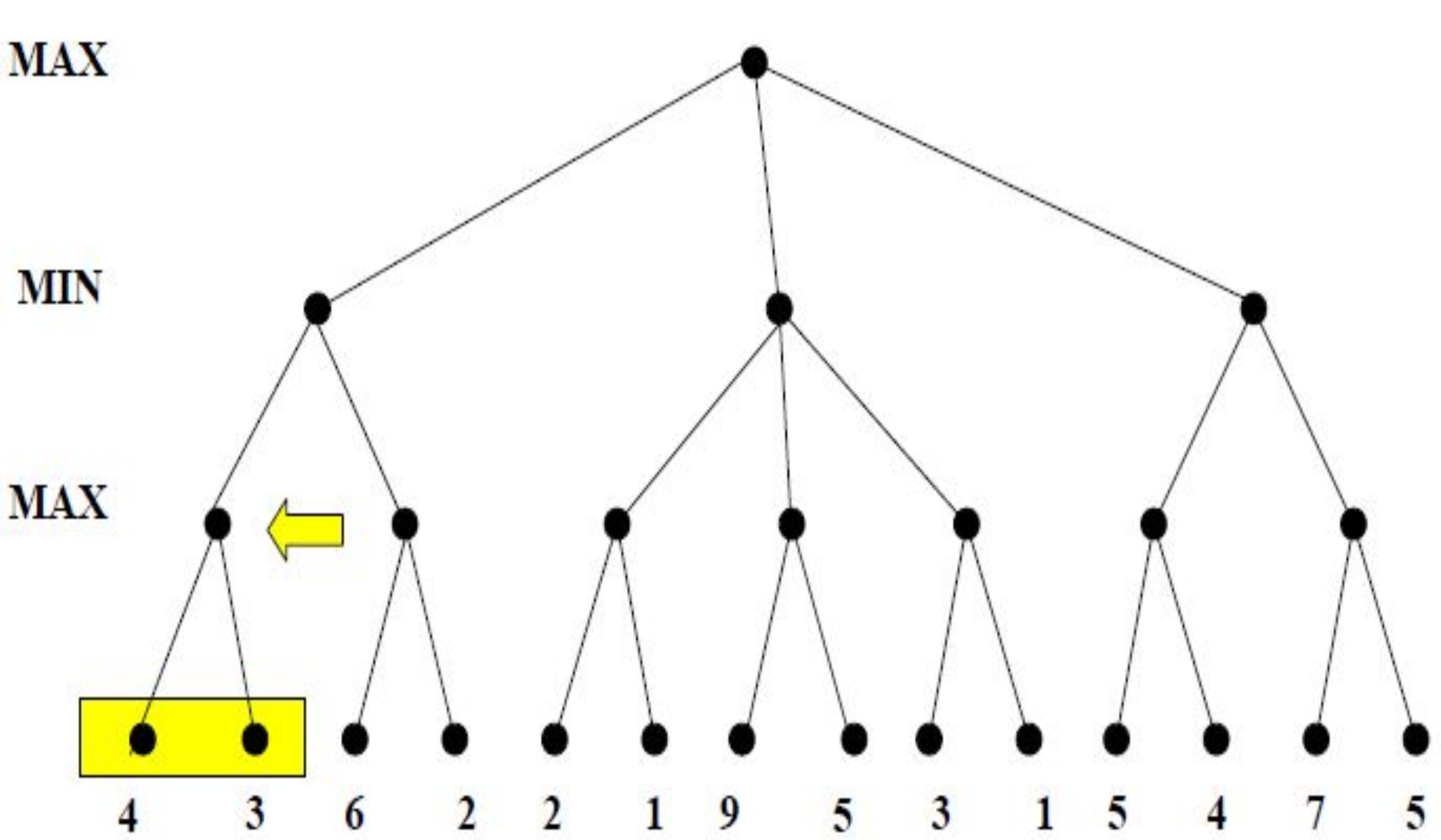

max
0
min
-43
0
max
-29
-43
0
0
min
-29
-37
-43
-75
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

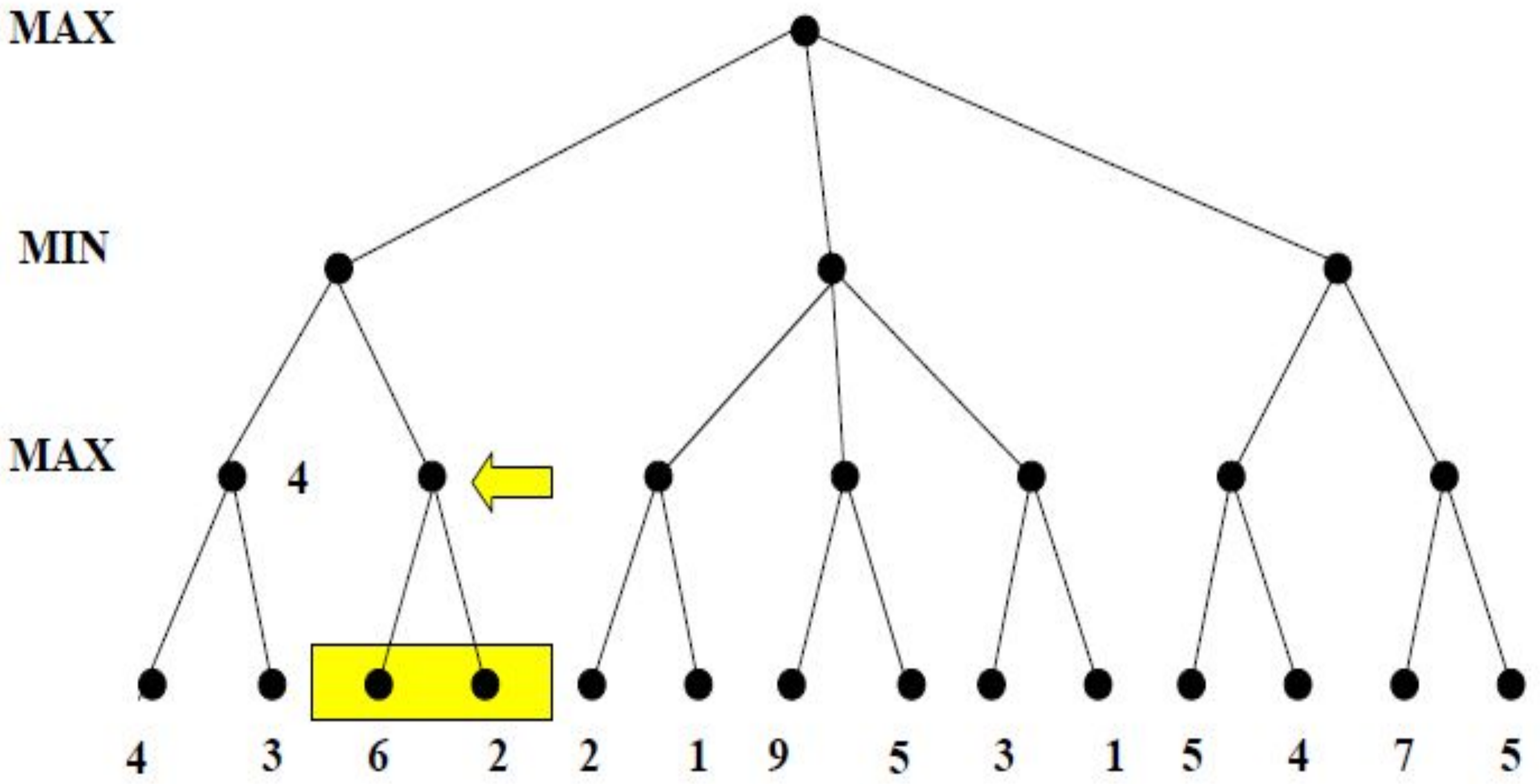

max
0
min
-43
0
max
-29
-43
0
0
min
-29
-37
-43
-75
0
0
0
0
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

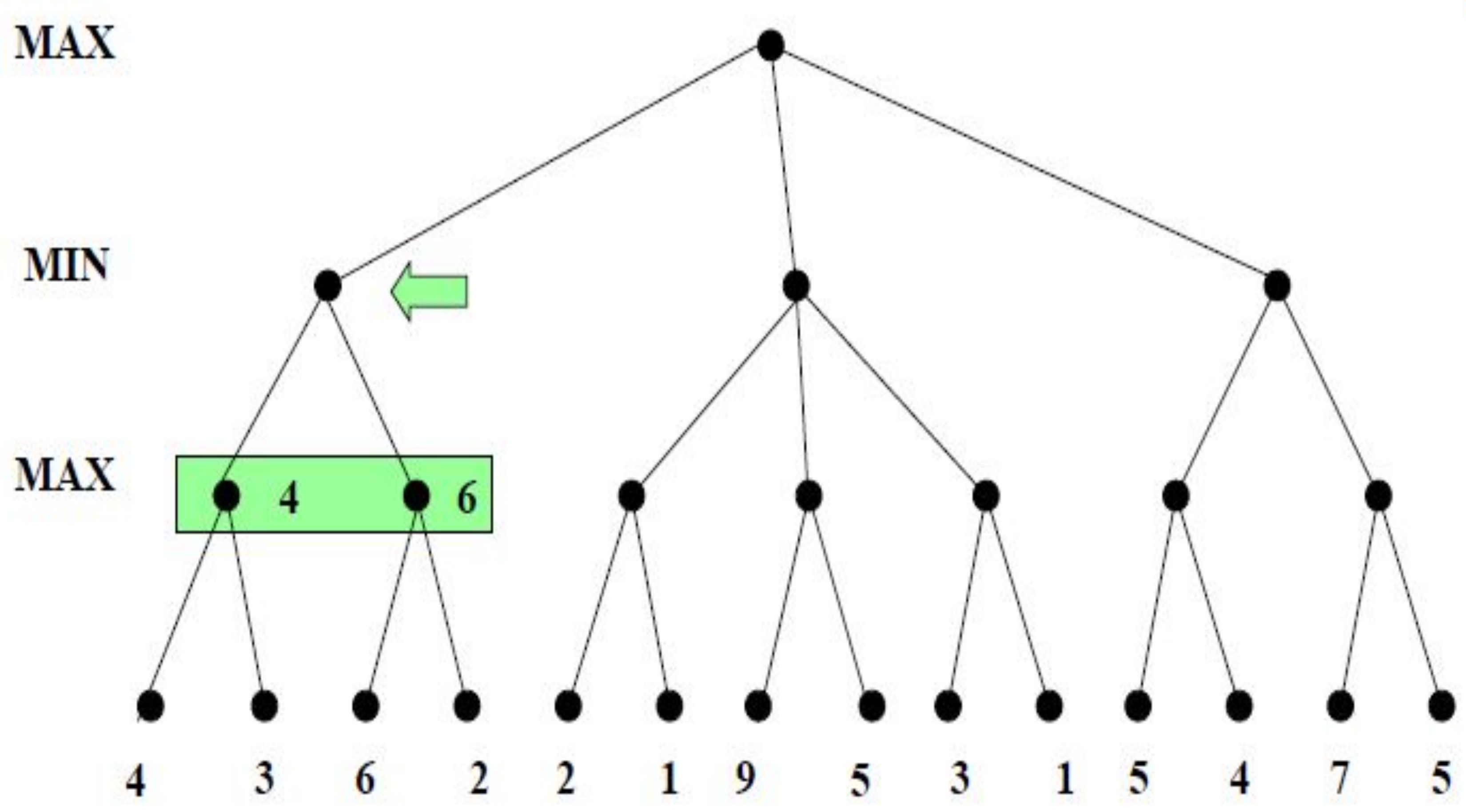

max
-43
min
-43
-46
max
-29
-43
-46
26
min
-29
-37
-43
-75
-51
-46
-13
26
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

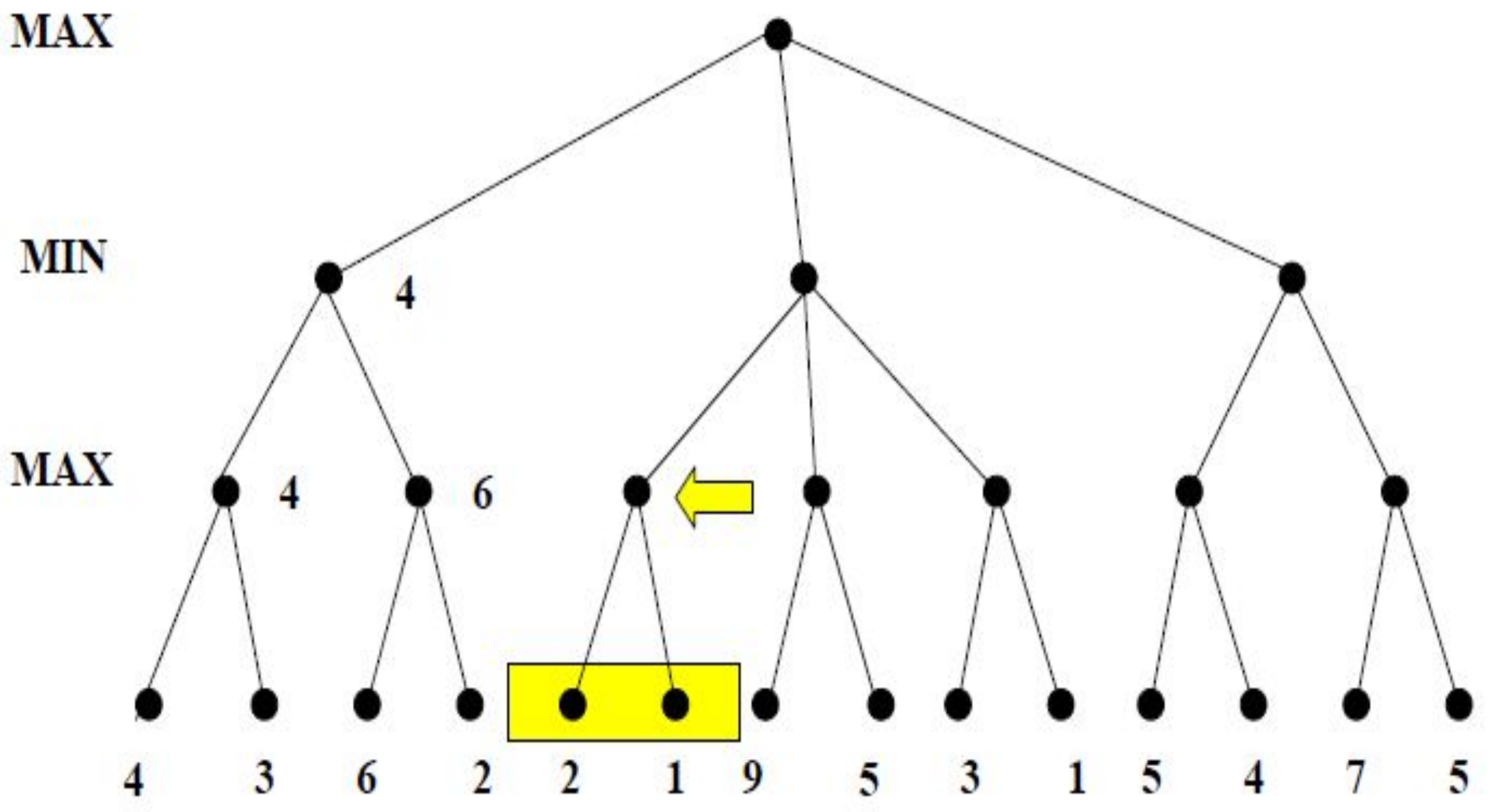

max
-43
min
-43
-46
max
-29
-43
-46
26
min
-29
-37
-43
-75
-51
-46
-13
26
84
-29
-37
-25
1
-43
-75
49
-21
-51
58
-46
-3
-13
26
79

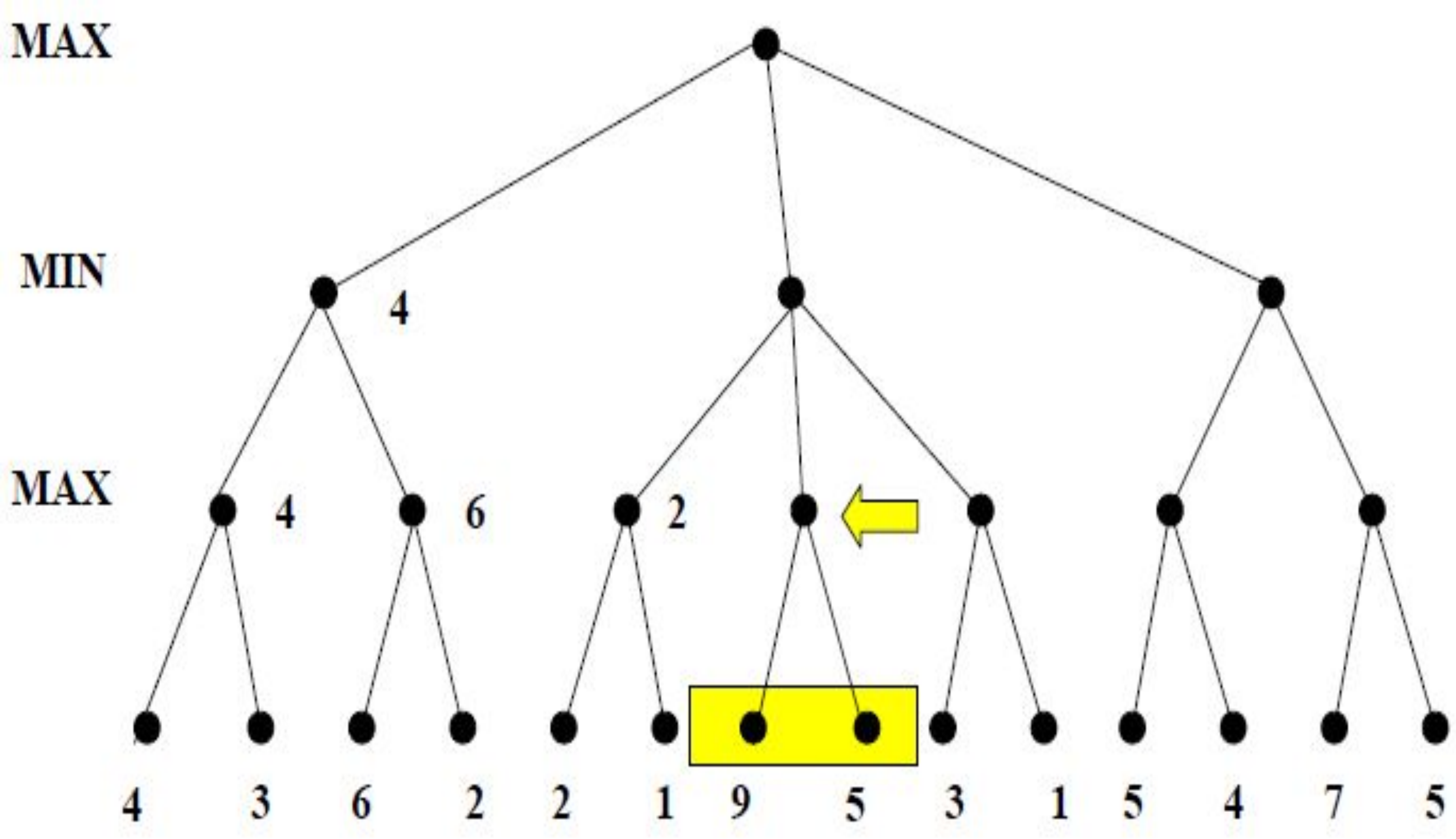

MAX
MIN
MAX
4 3 6 2 2 1 9 5 3 1 5 4 7 5

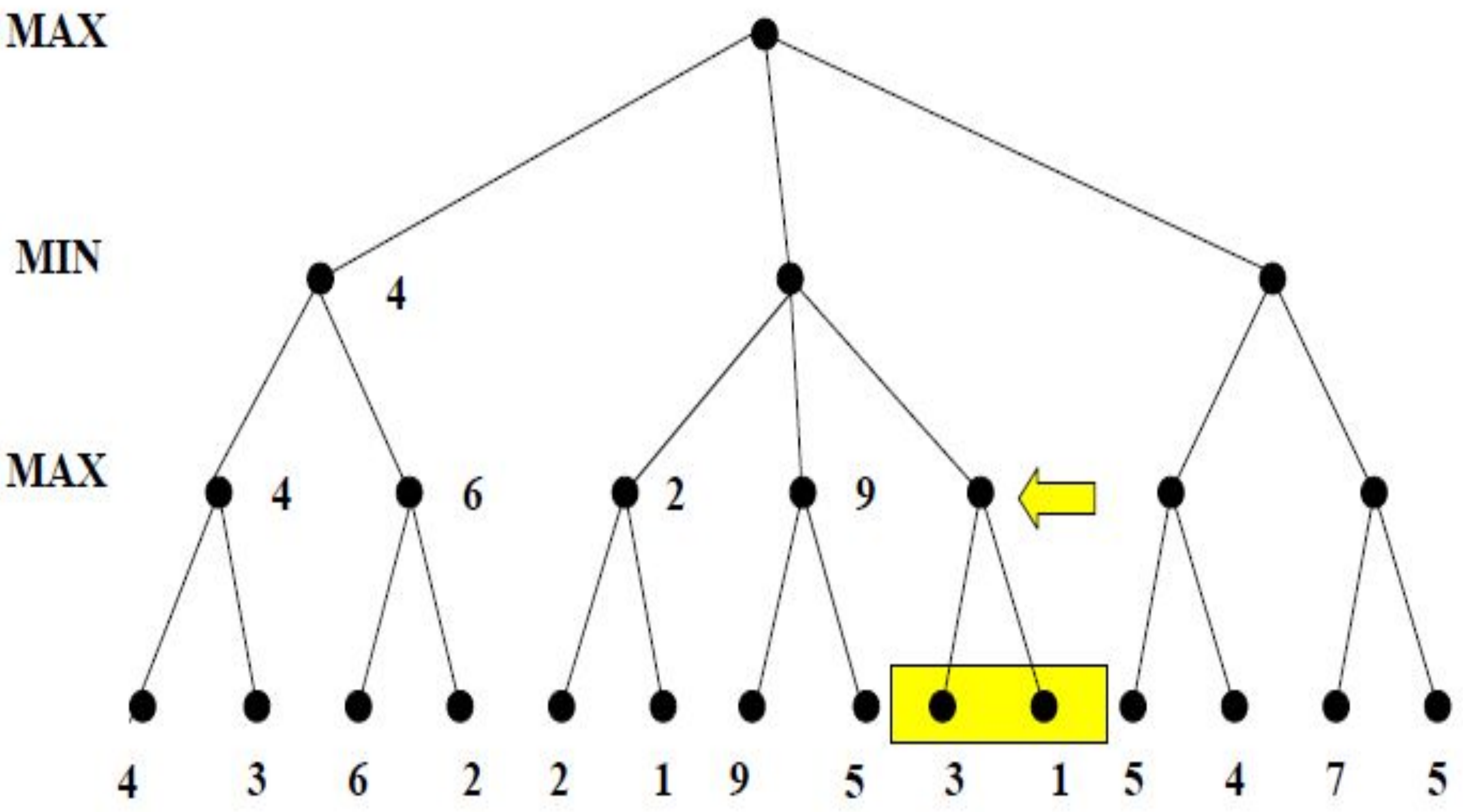

MAX
MIN
MAX
4
3
6
2
2
1
9
5
3
1
5
4
7
5

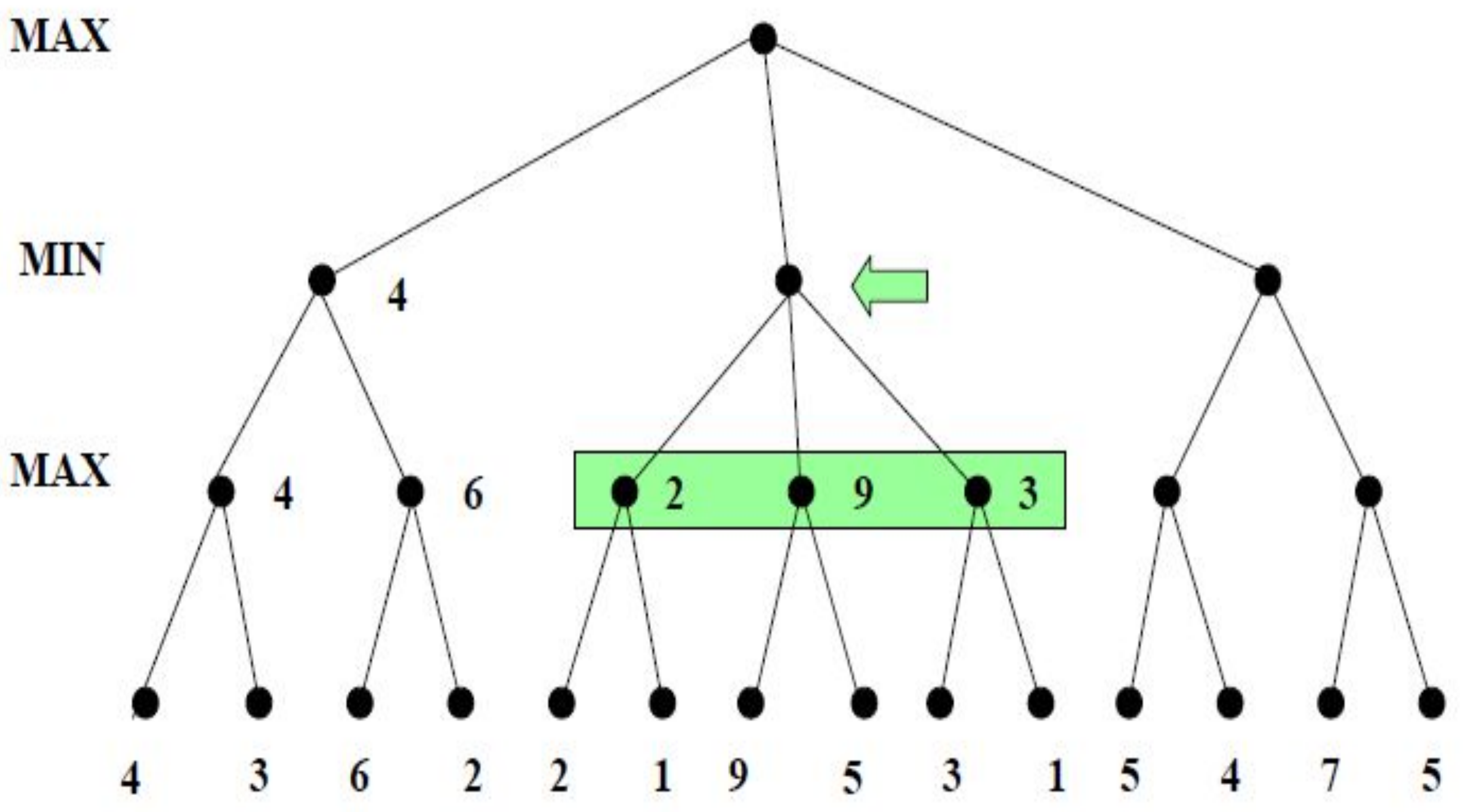

MAX
MIN
MAX
4
4 3 6 2 2 1 9 5 3 1 5 4 7 5

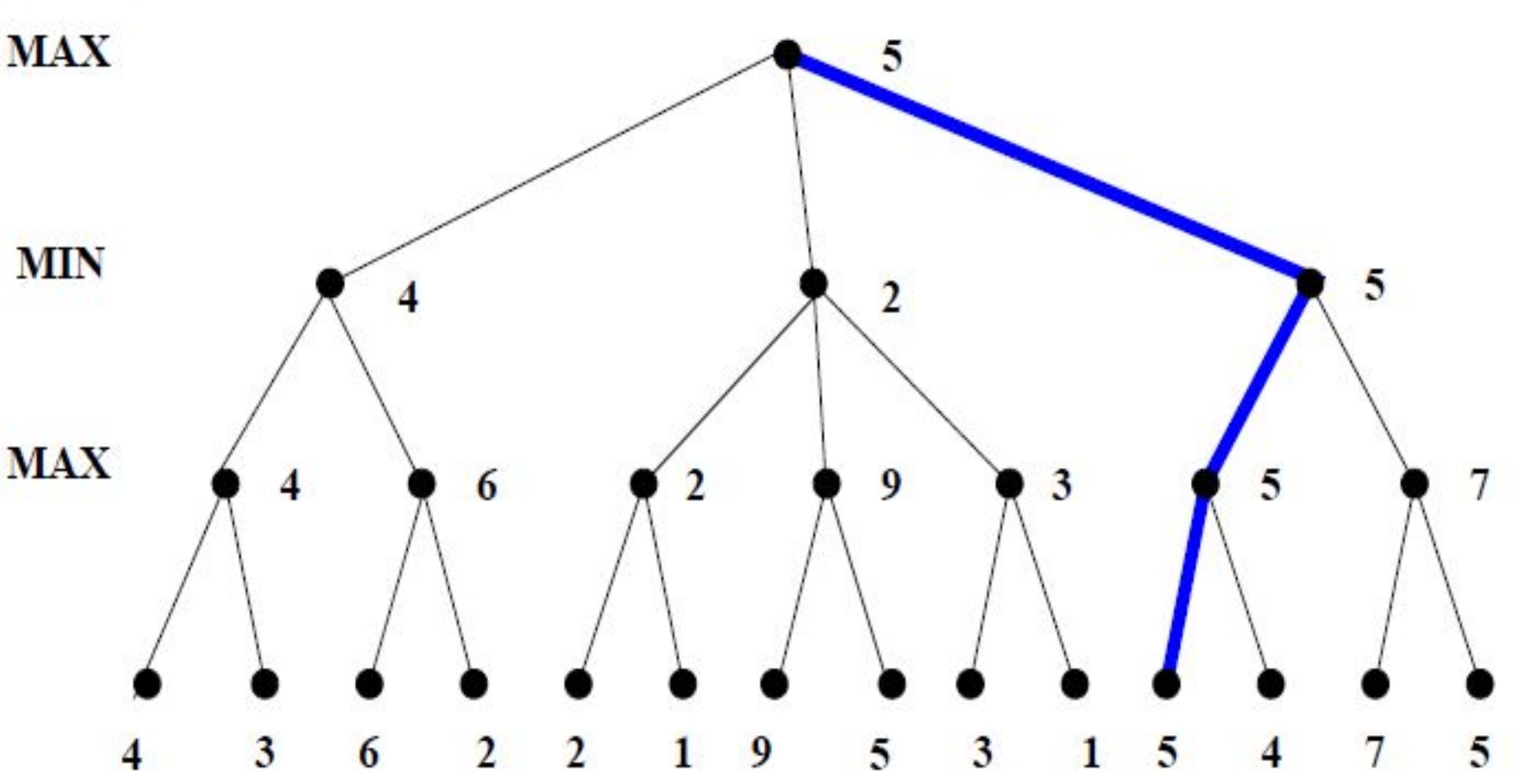

MAX
MIN
MAX
4
6
4 3 6 2 2 1 9 5 3 1 5 4 7 5


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
4
MAX
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

Properties of minmax

1. **Complete:** It is complete (if the tree is finite).
2. **Optimal:** The minimax is optimal against an optimal opponent and is not optimal where it does not exploit opponent weakness against suboptimal opponent.
3. **Time complexity:** The time complexity is defined as $O(bm)$.
4. **Space complexity:** The space complexity is written as $O(bm)$ (depth-first exploration).

Advantages

- It is the simplest possible (reasonable) game search algorithm.
- It returns an optimal action, assuming perfect opponent play.
- There is no need to rerun Minimax while starting every game: It should be run once, offline before the game starts.

Advantages

- During the actual game, the branches taken in the tree is followed. Whenever it’s your turn, choose an action maximising the value of the successor states.

Disadvantages

- In practice, it is completely infeasible.
- The main disadvantage of the minimax algorithm is that all the nodes in the game tree cutoff to a certain depth are examined.

Alpha-beta pruning

Alpha is the best choice so far for player MAX. The highest possible value for this is demanded here.

Beta proves to be the best choice so far for MIN, and it has to be the lowest possible value.

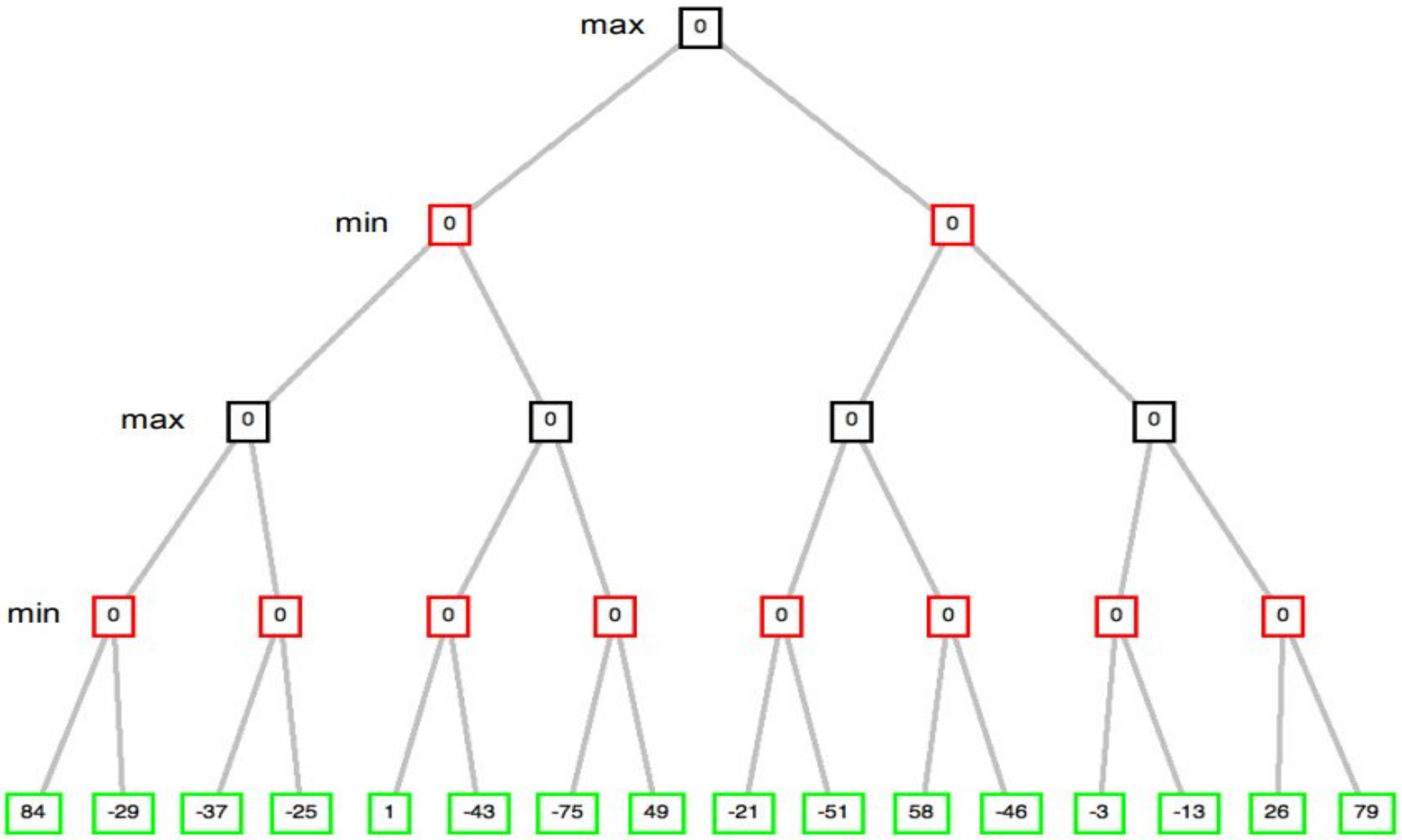
ALGORITHM

- 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.

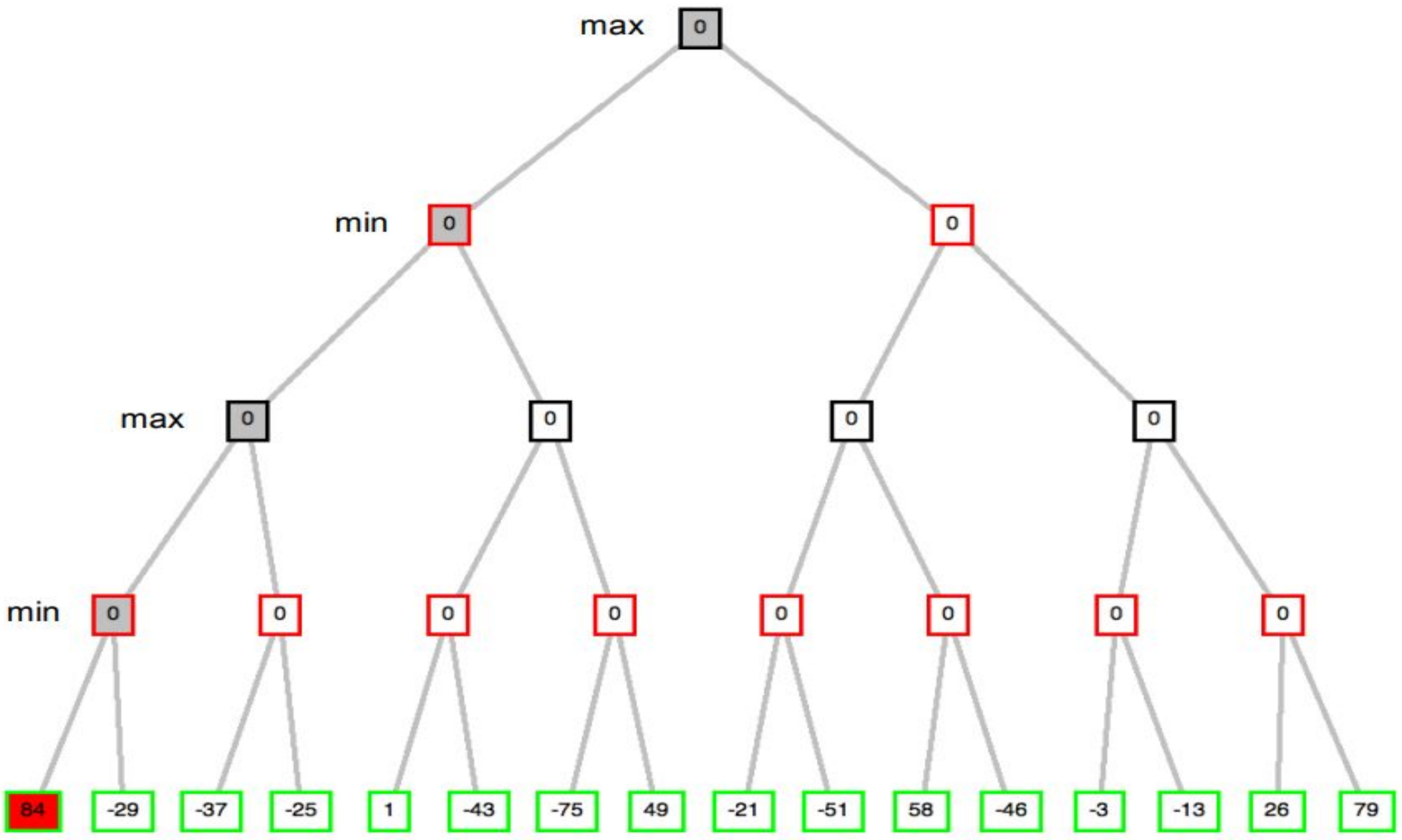
Alpha-beta pruning

- 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.

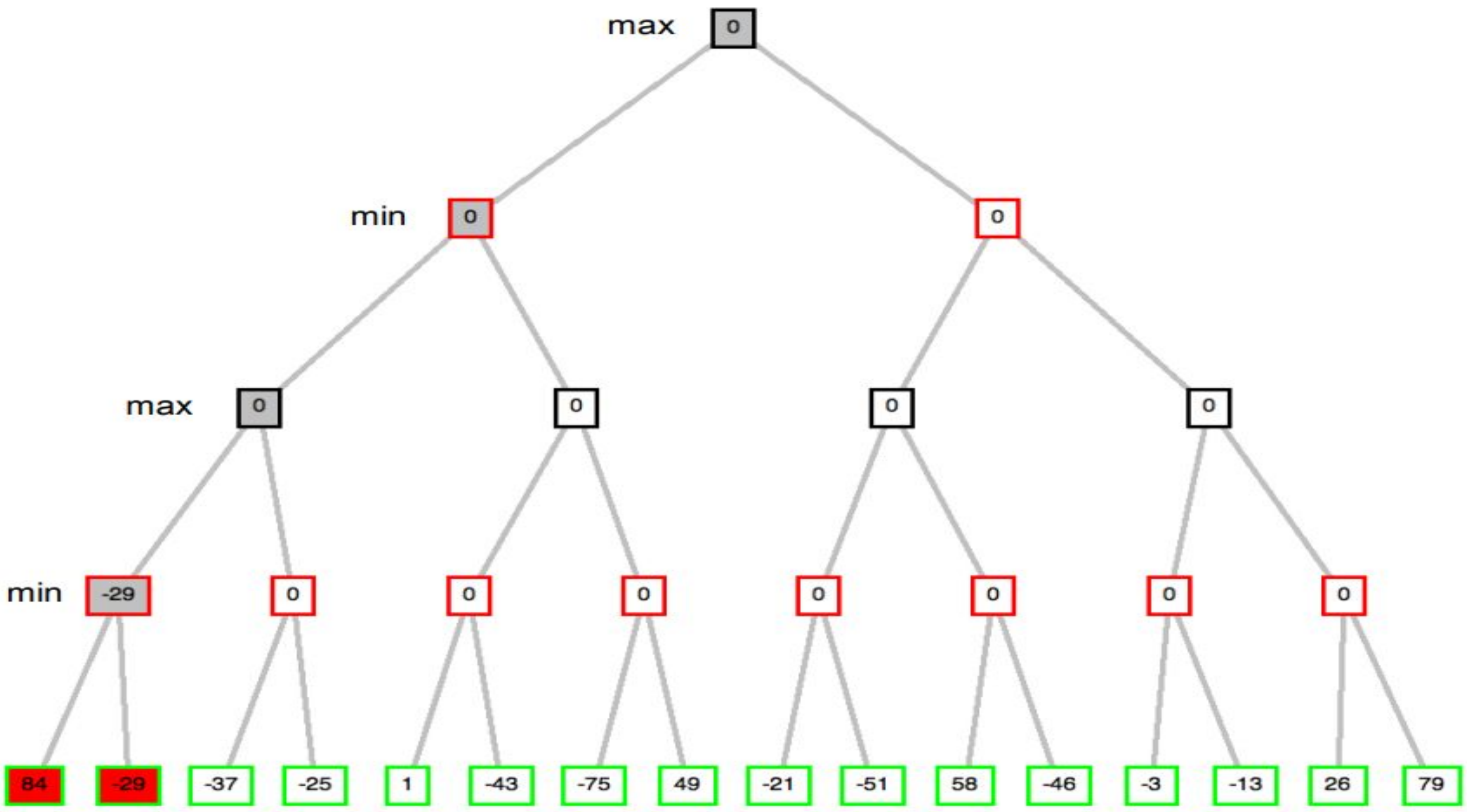
Alpha-beta pruning



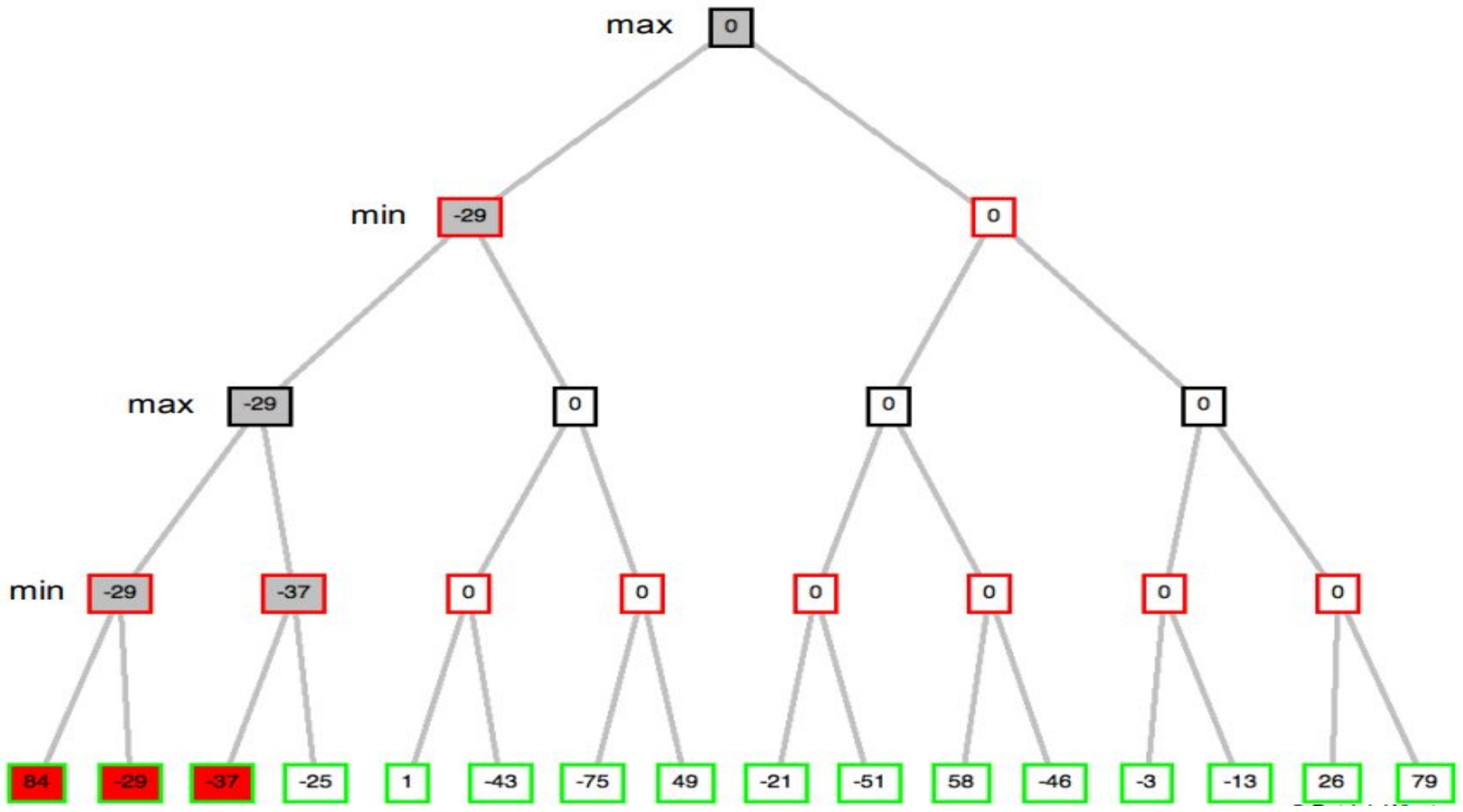
Alpha-beta pruning



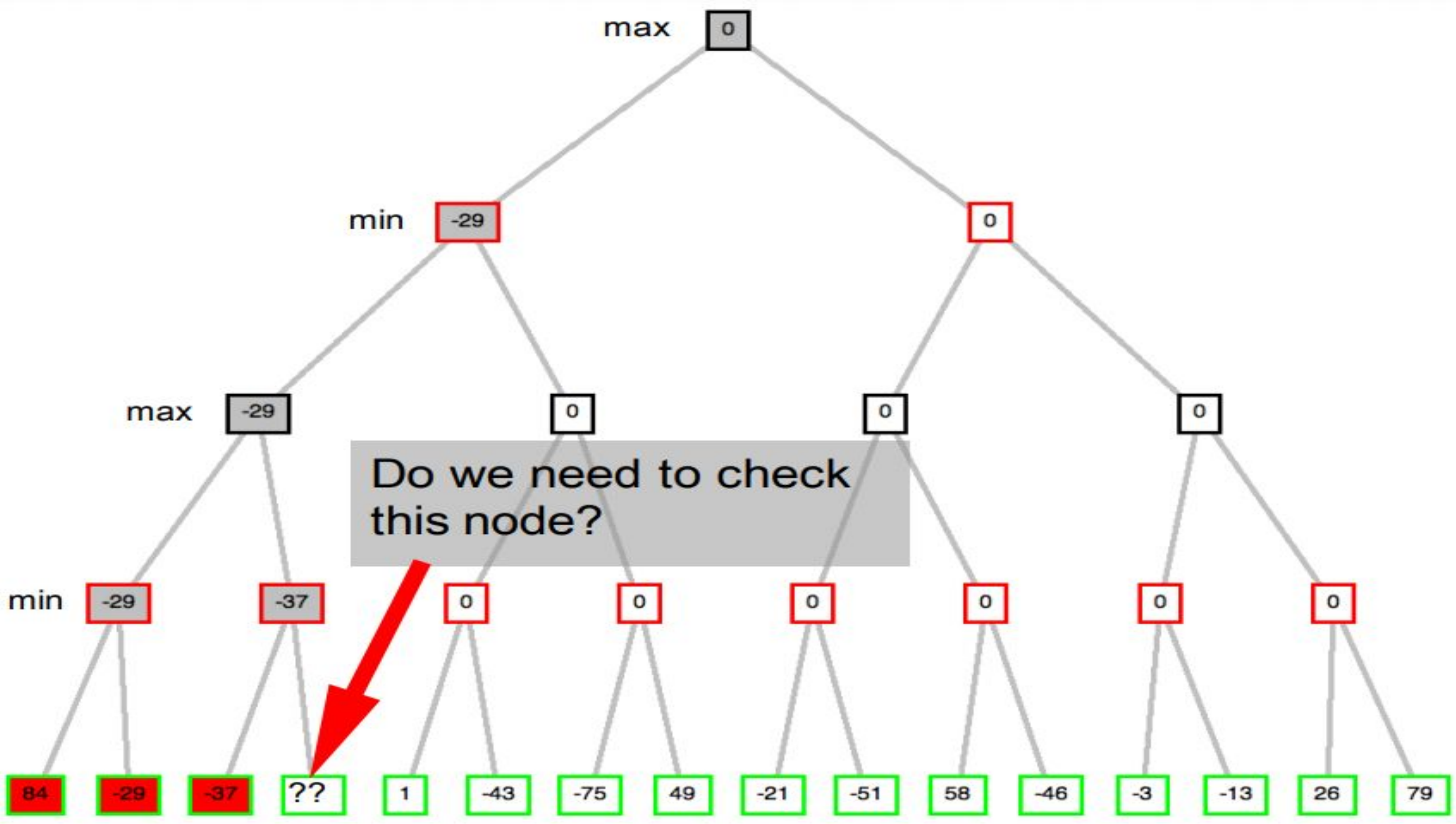
Alpha-beta pruning



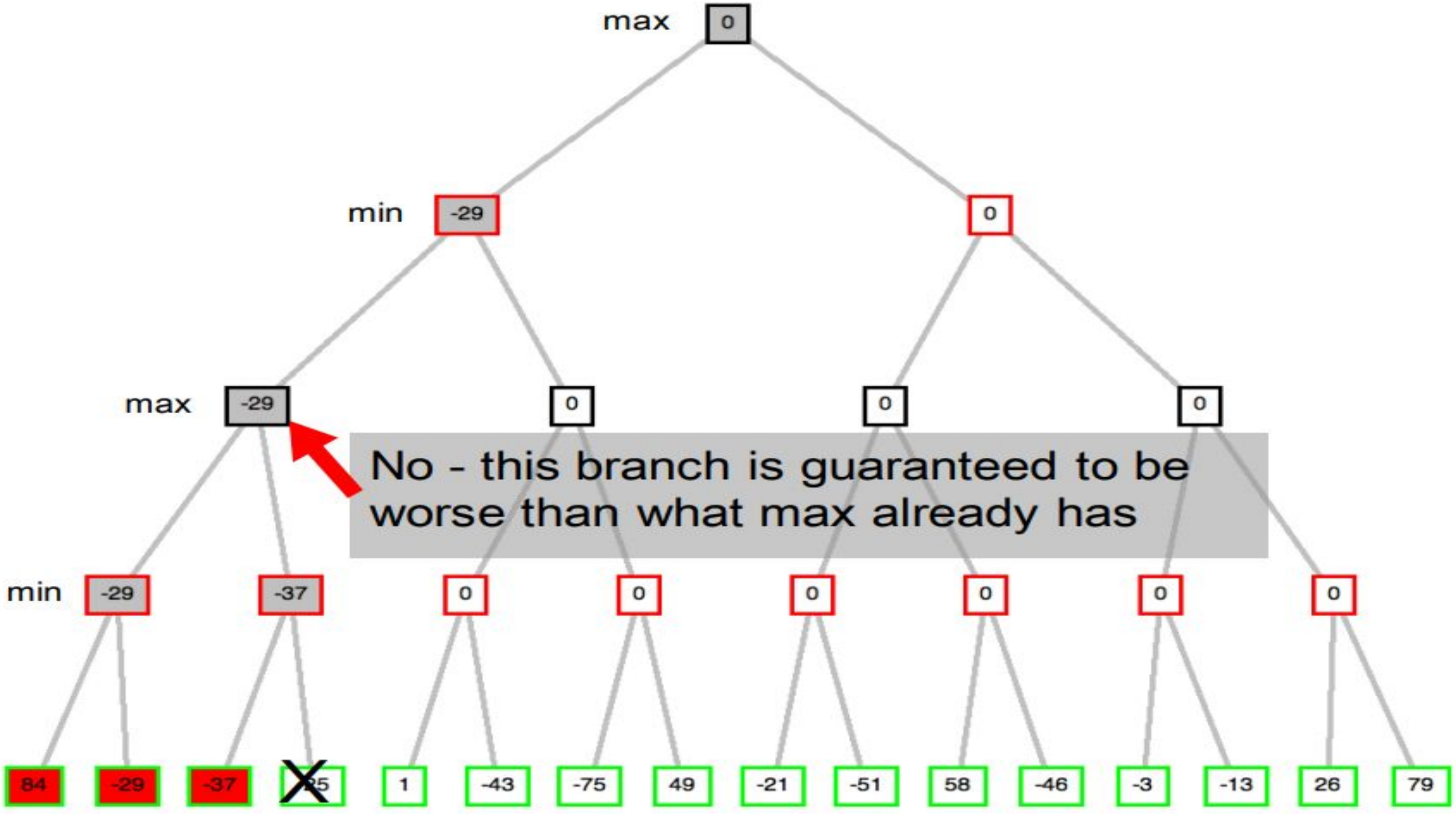
Alpha-beta pruning



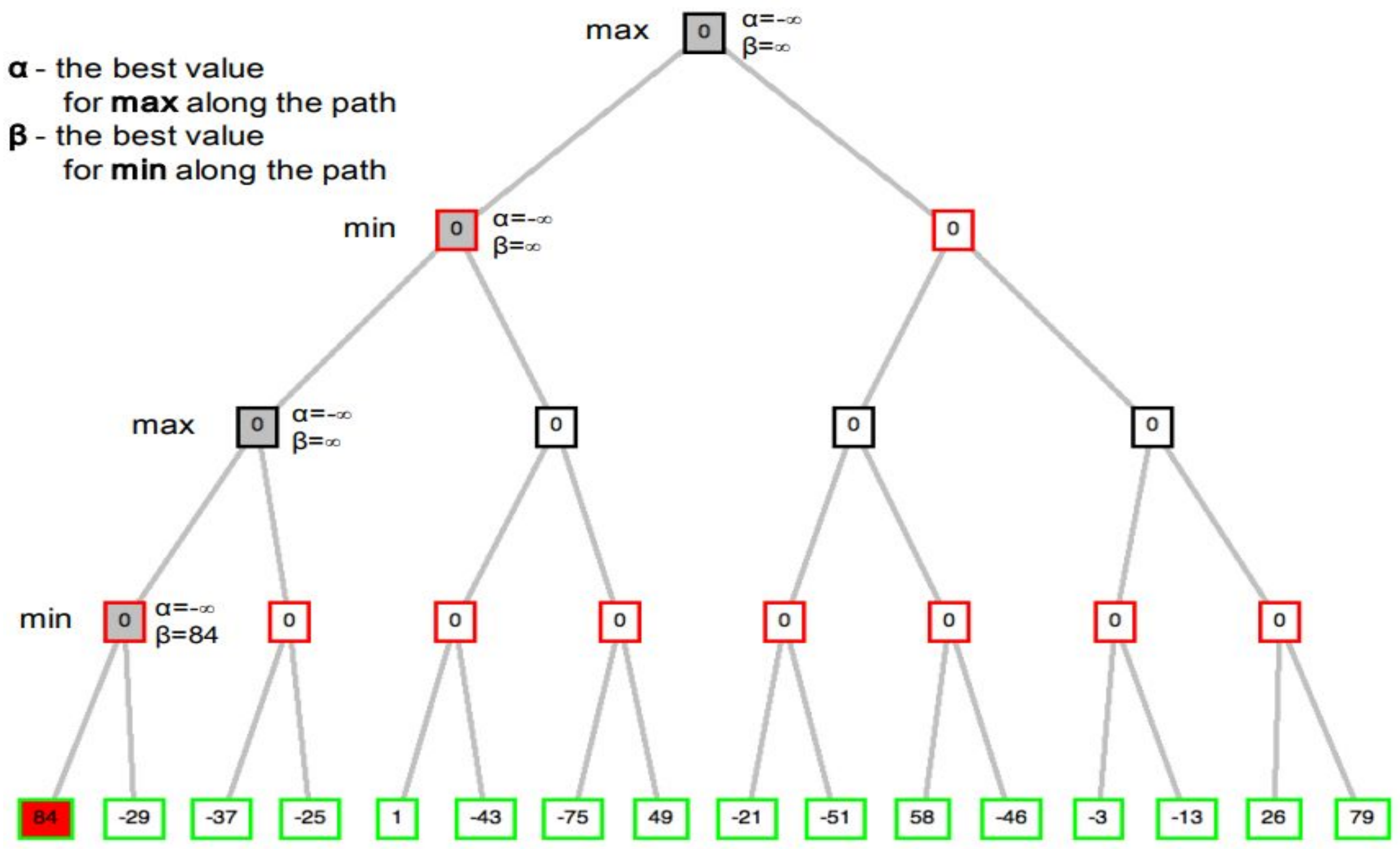
Alpha-beta pruning



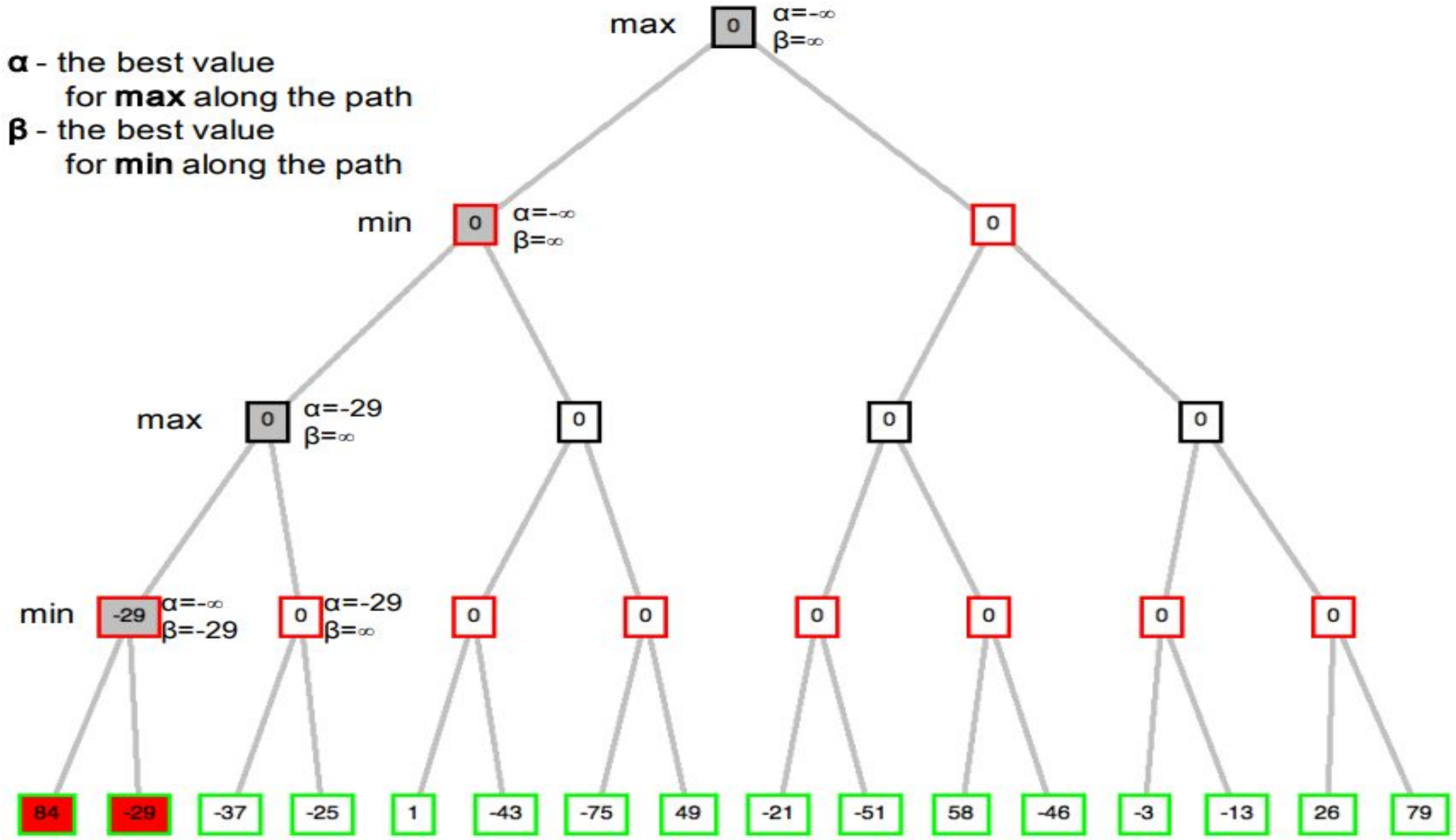
Alpha-beta pruning



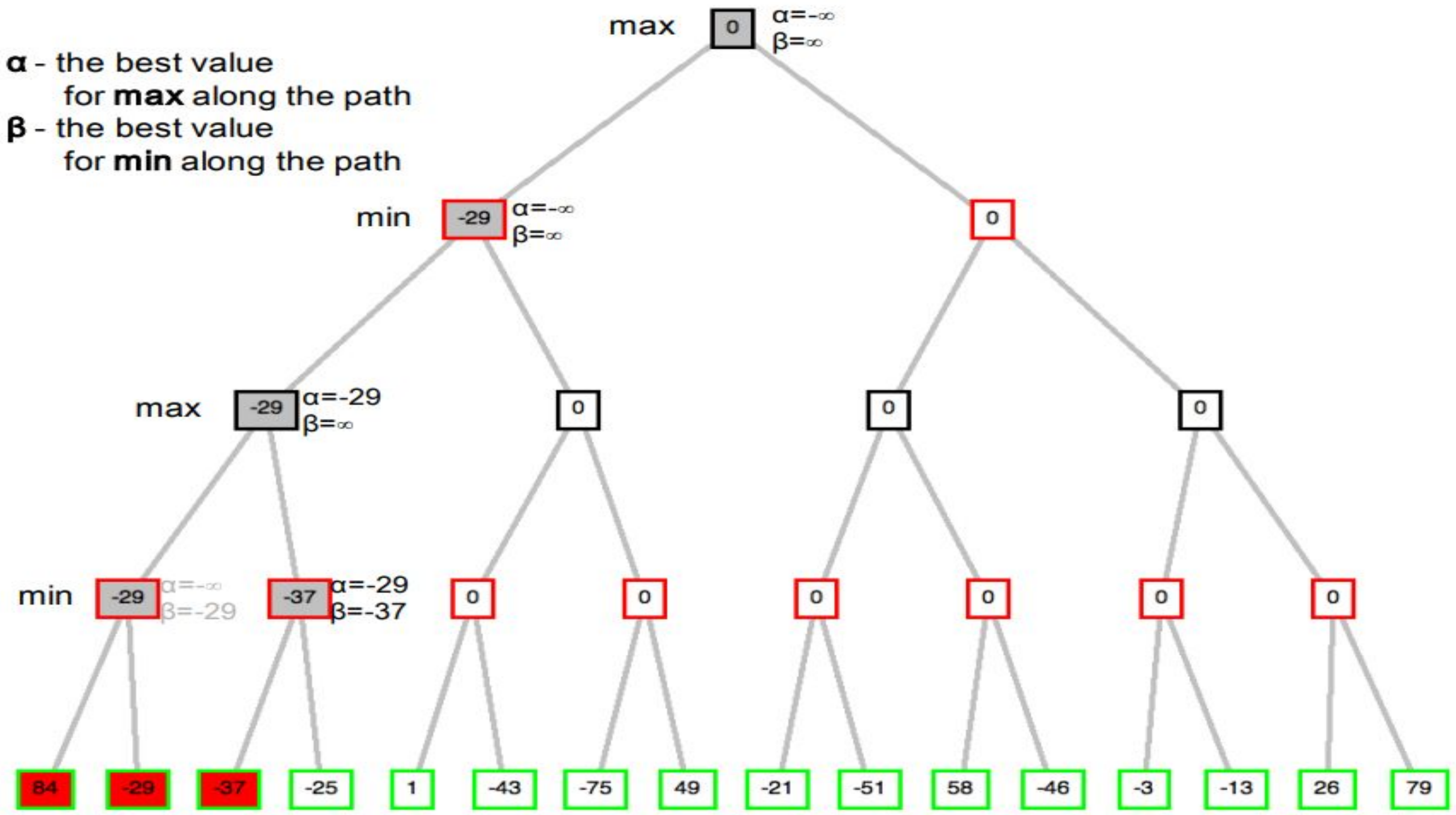
Alpha-beta pruning



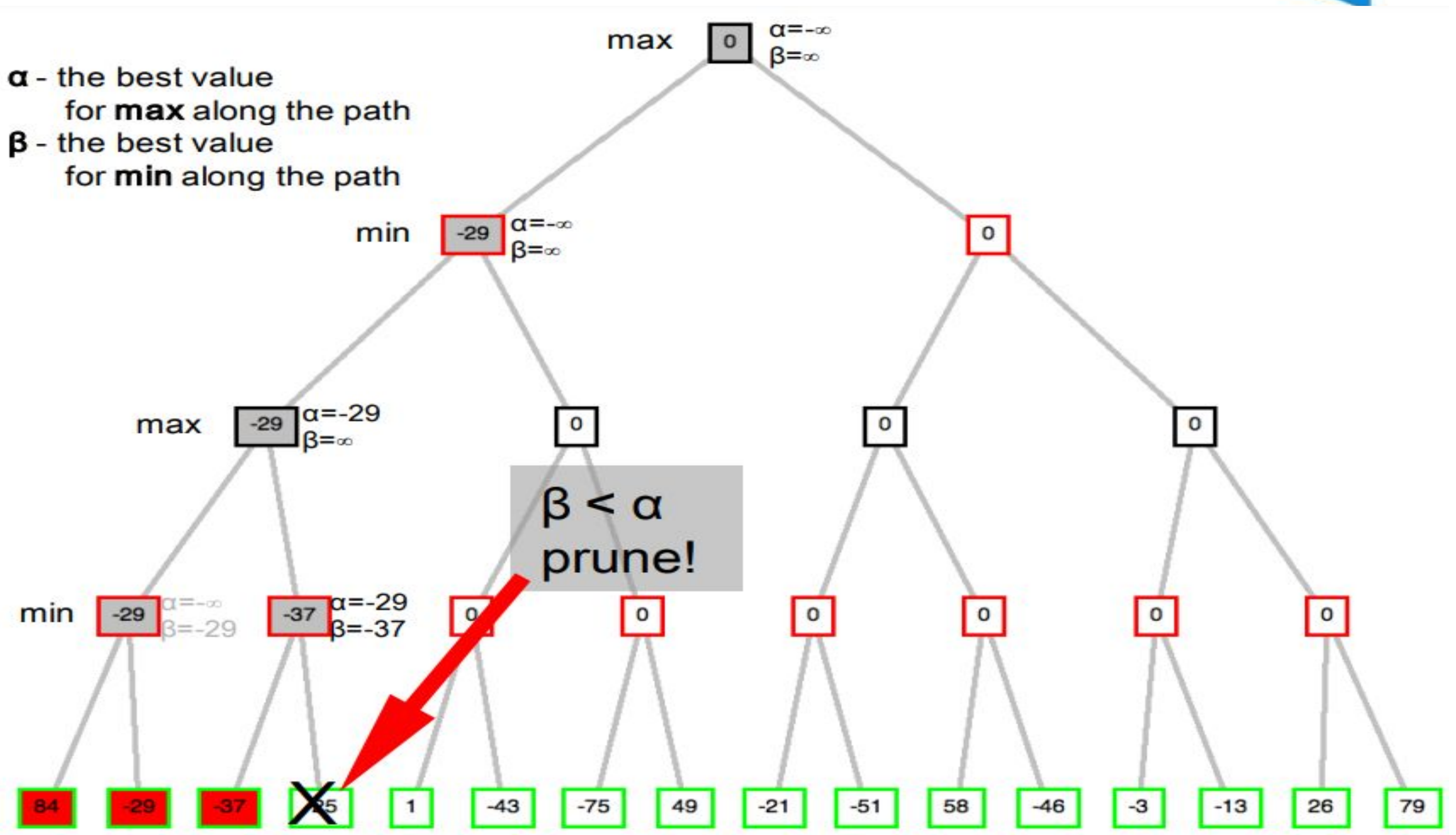
Alpha-beta pruning



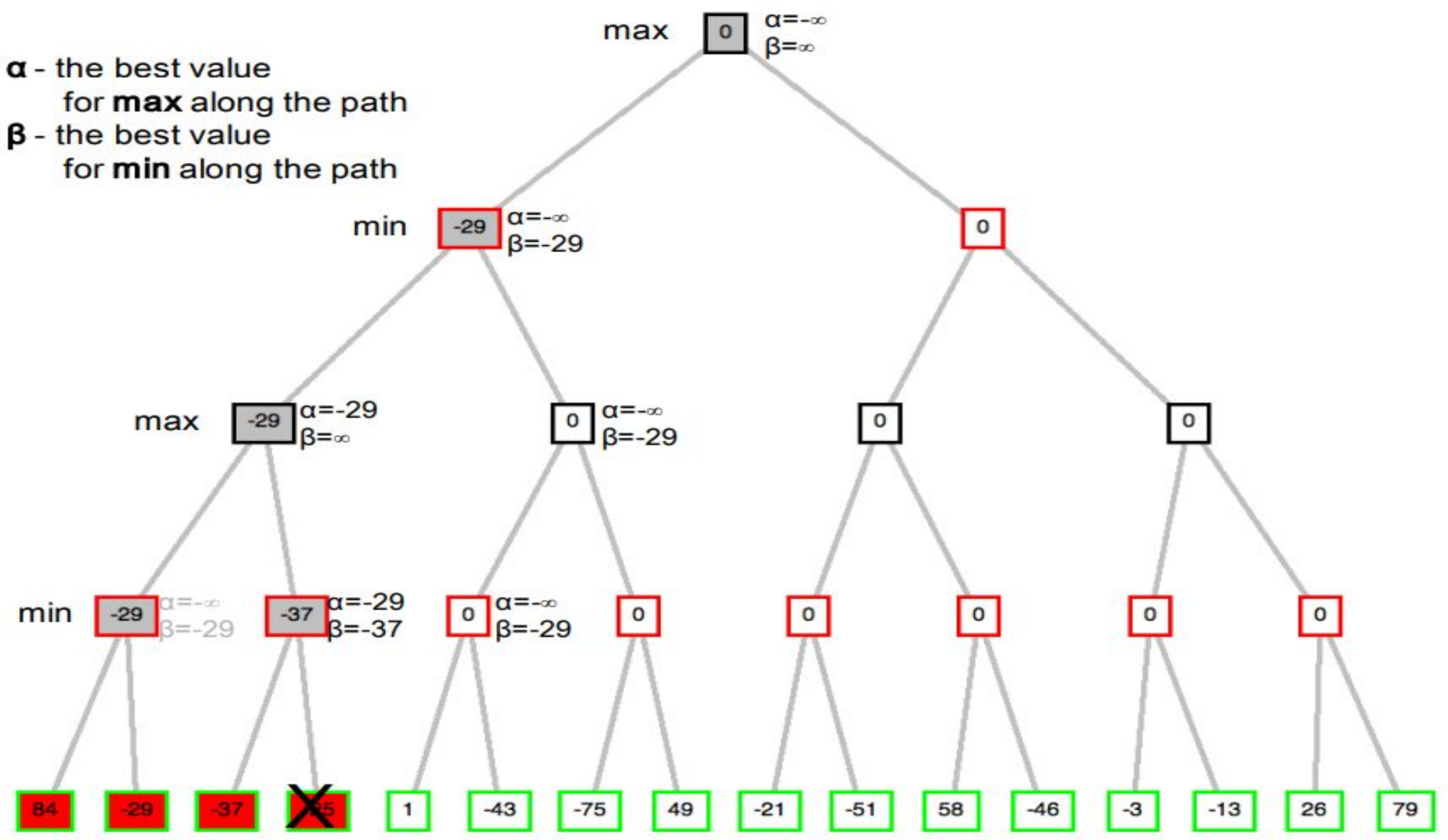
Alpha-beta pruning



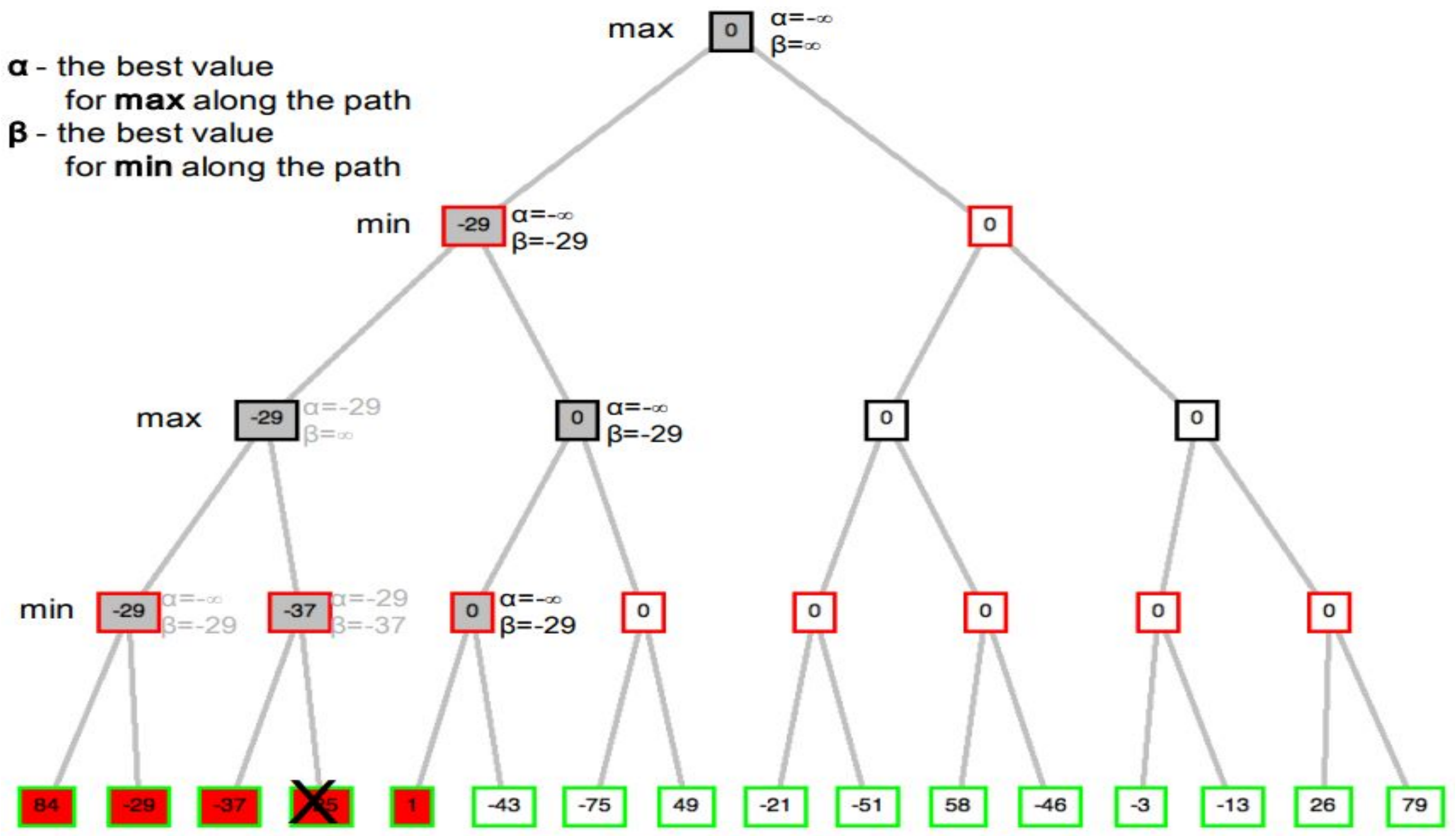
Alpha-beta pruning



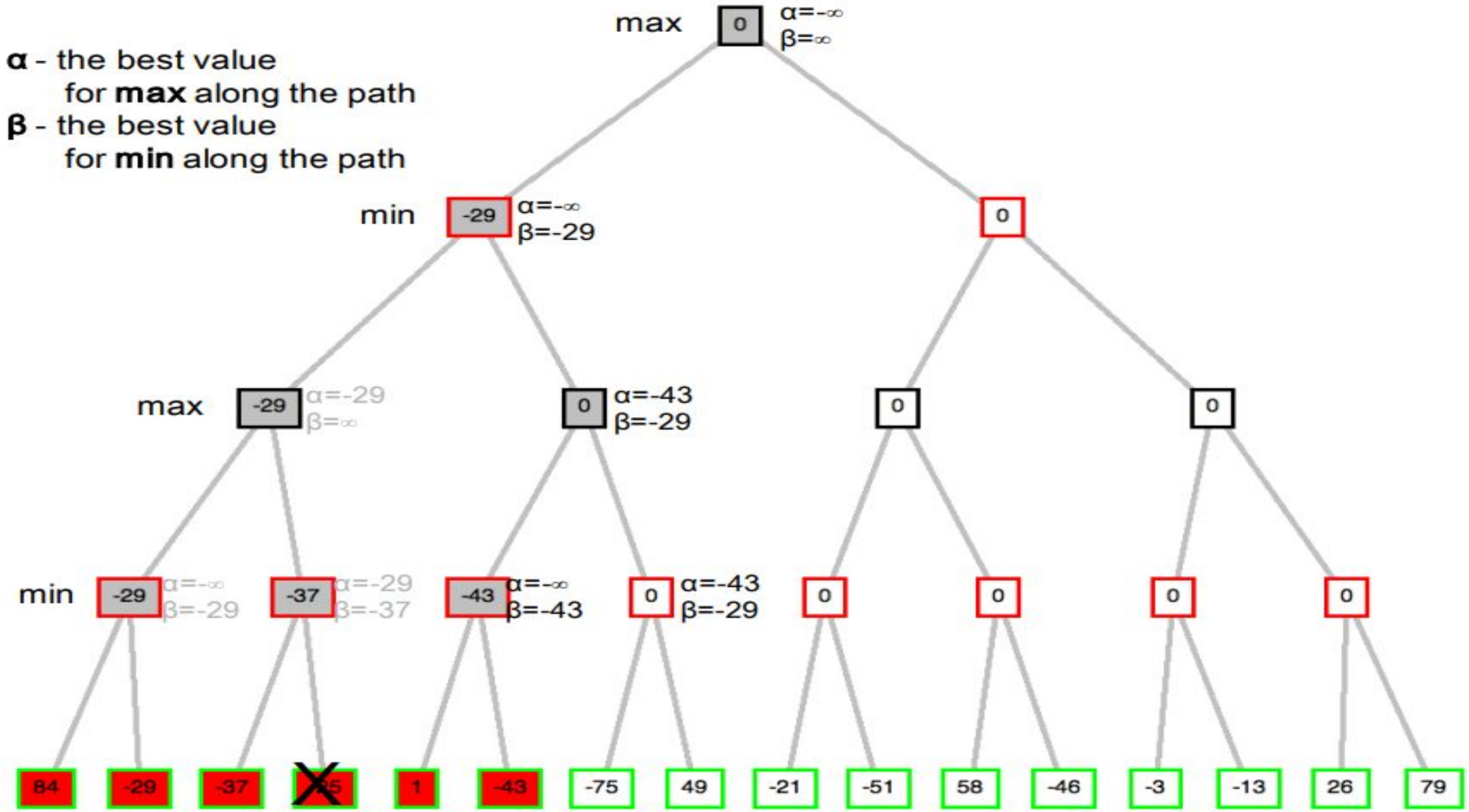
Alpha-beta pruning



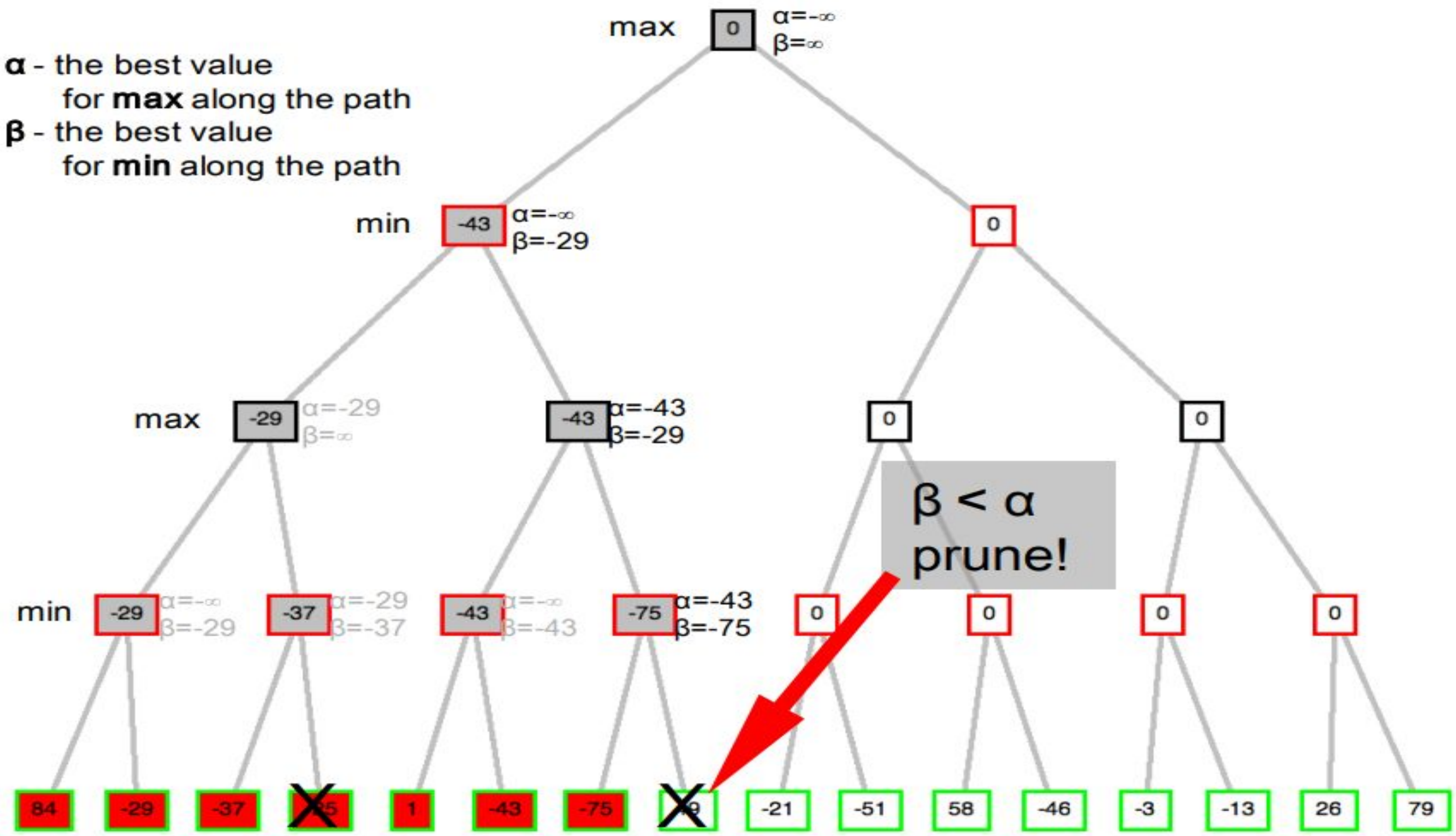
Alpha-beta pruning



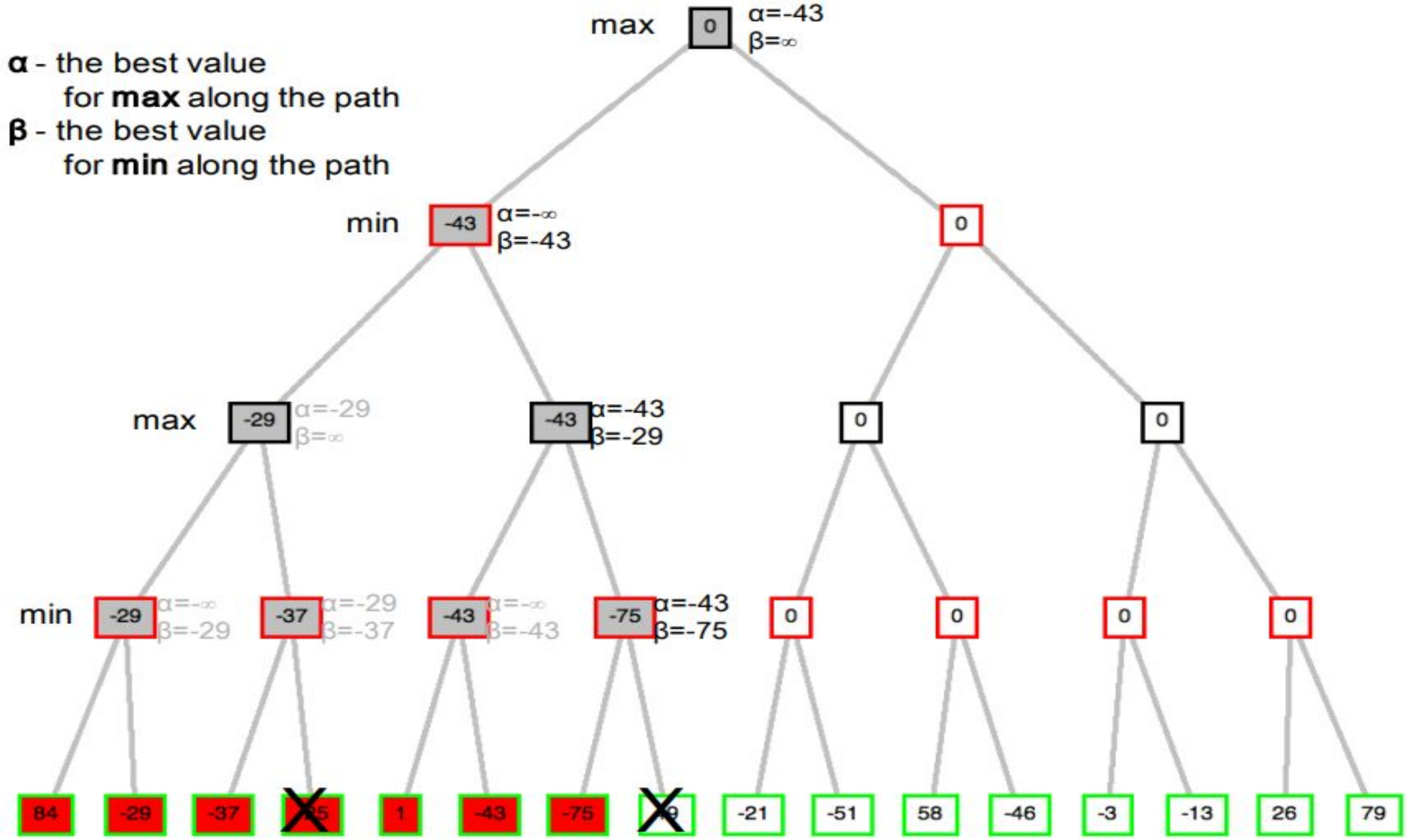
Alpha-beta pruning



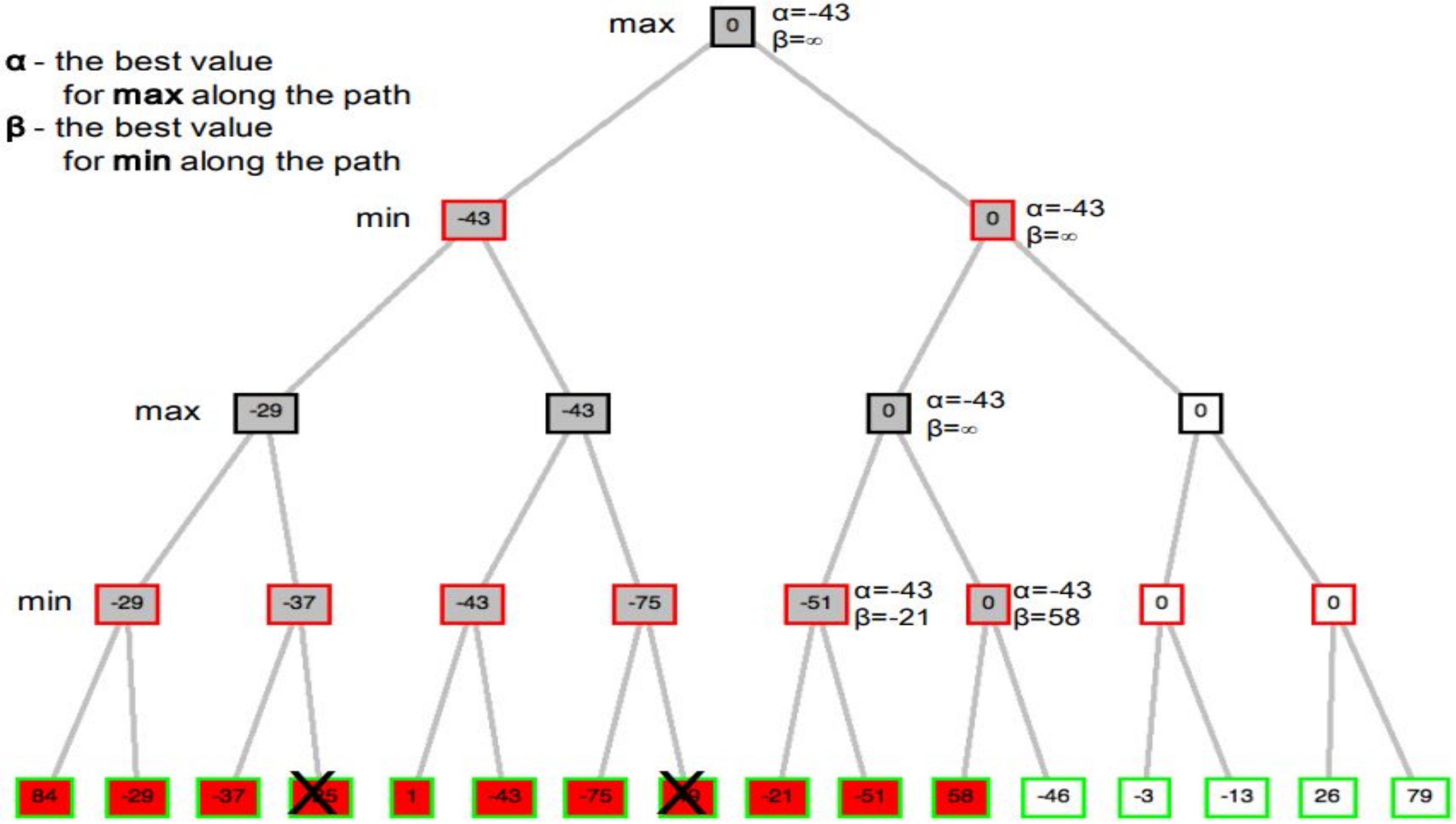
Alpha-beta pruning



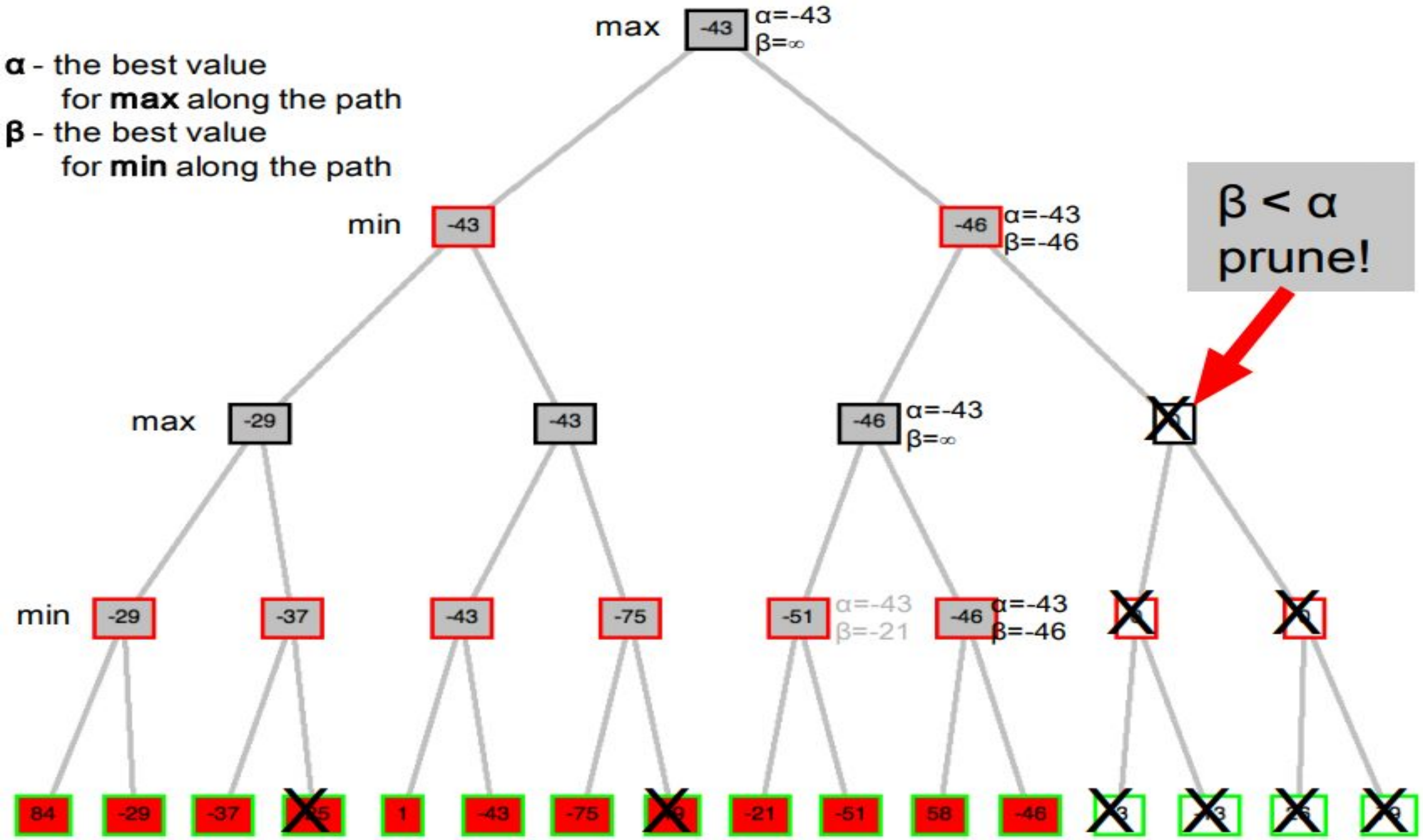
Alpha-beta pruning



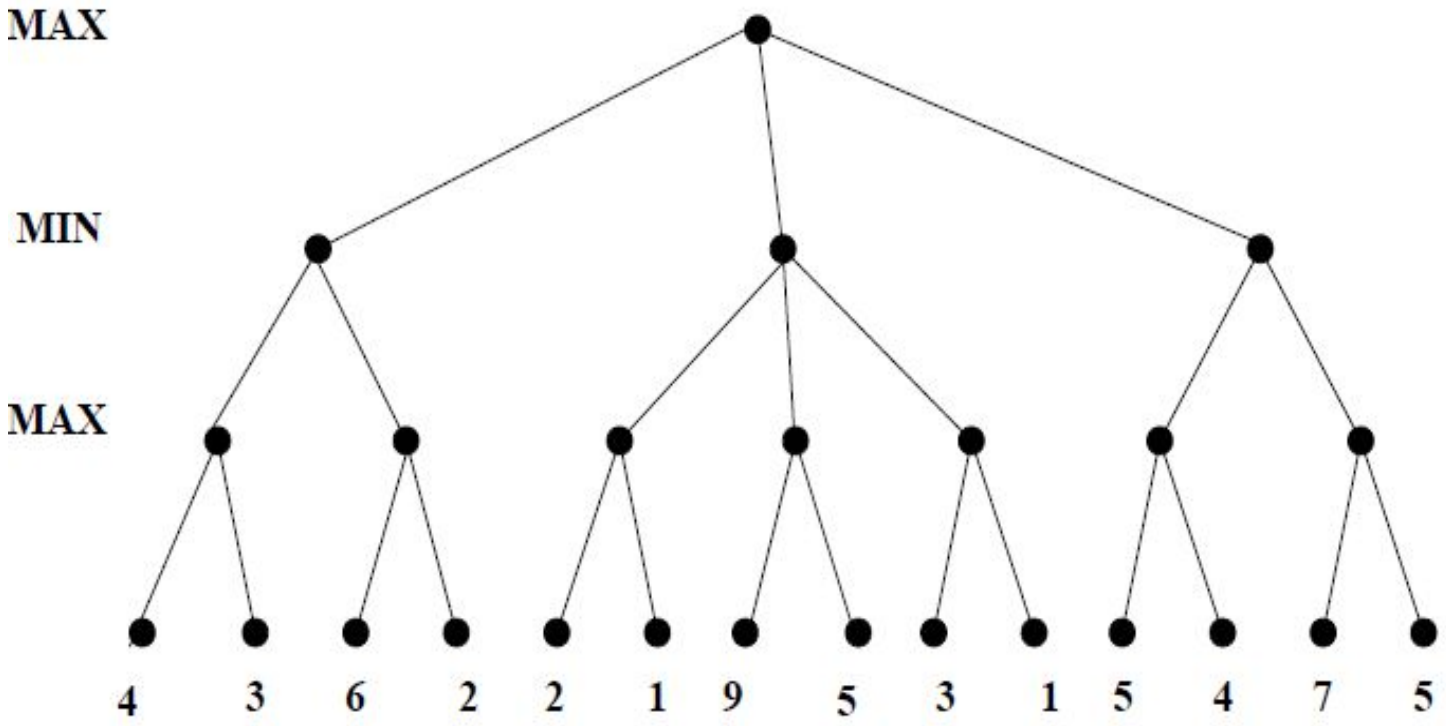
Alpha-beta pruning



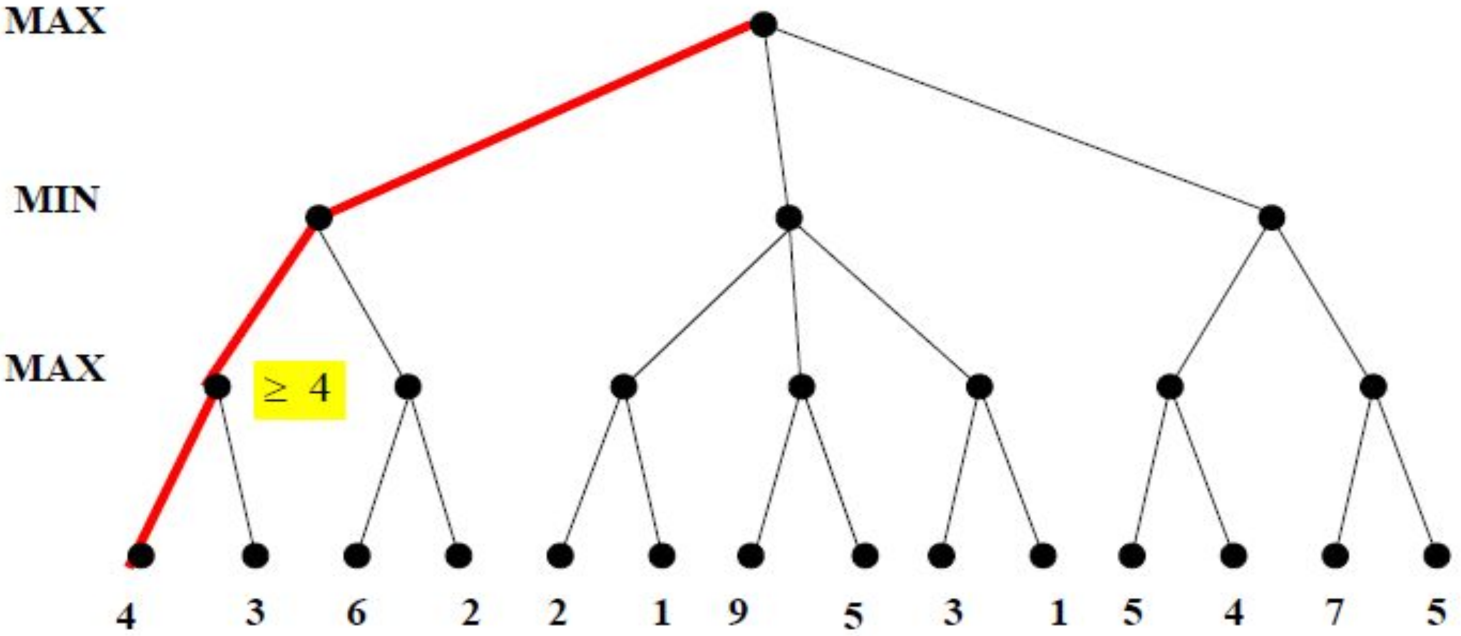
Alpha-beta pruning



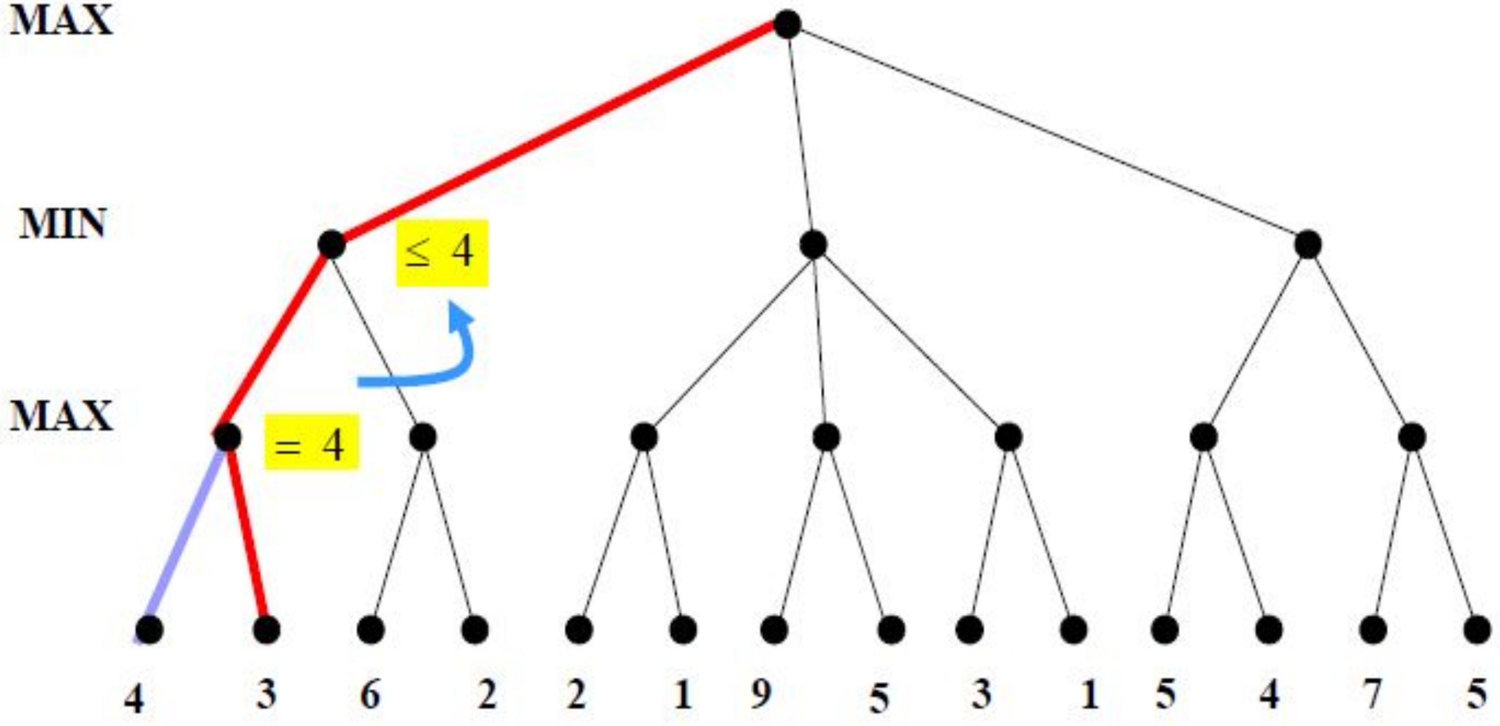
Alpha-beta pruning Example 2



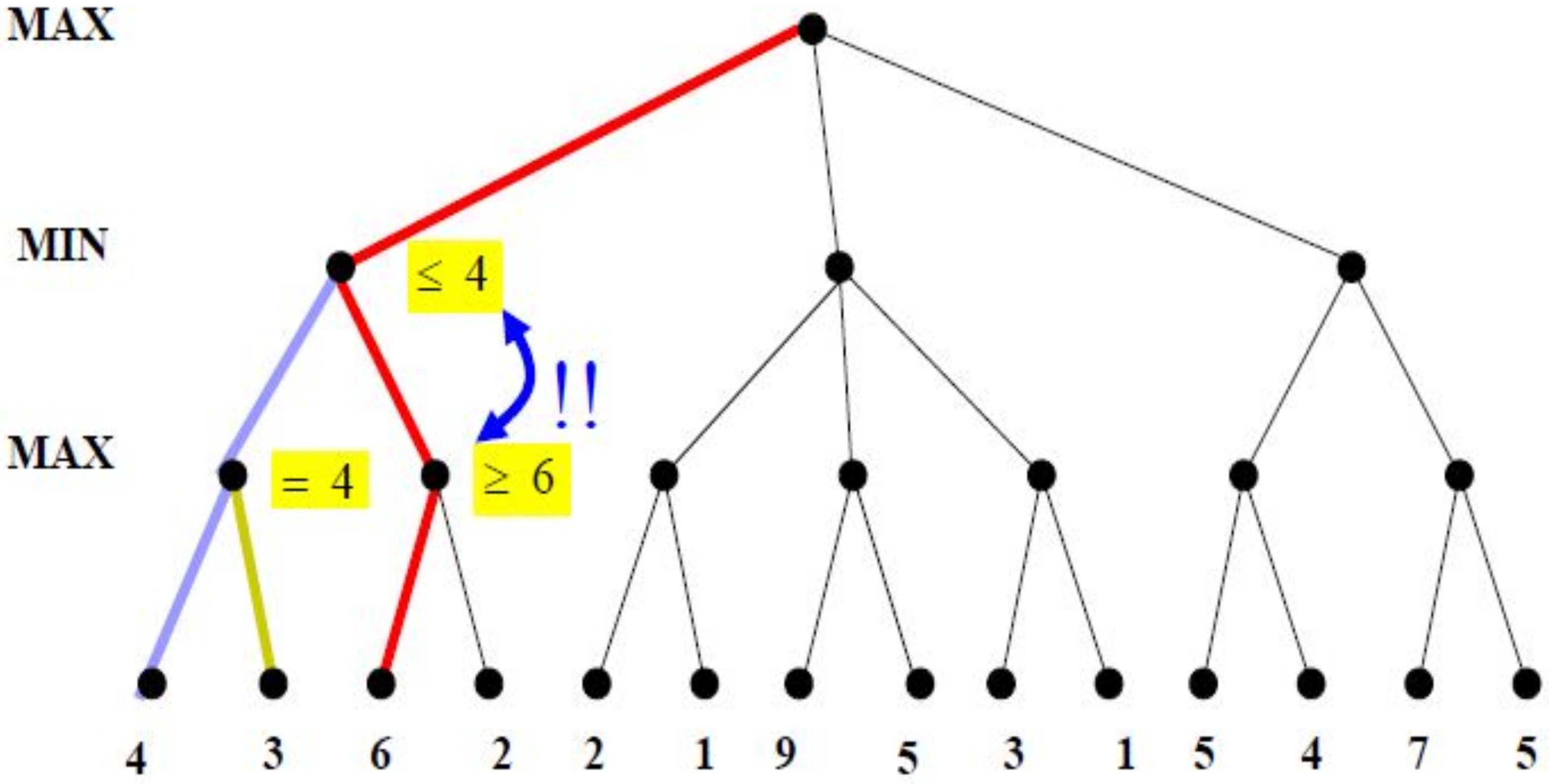
Alpha-beta pruning



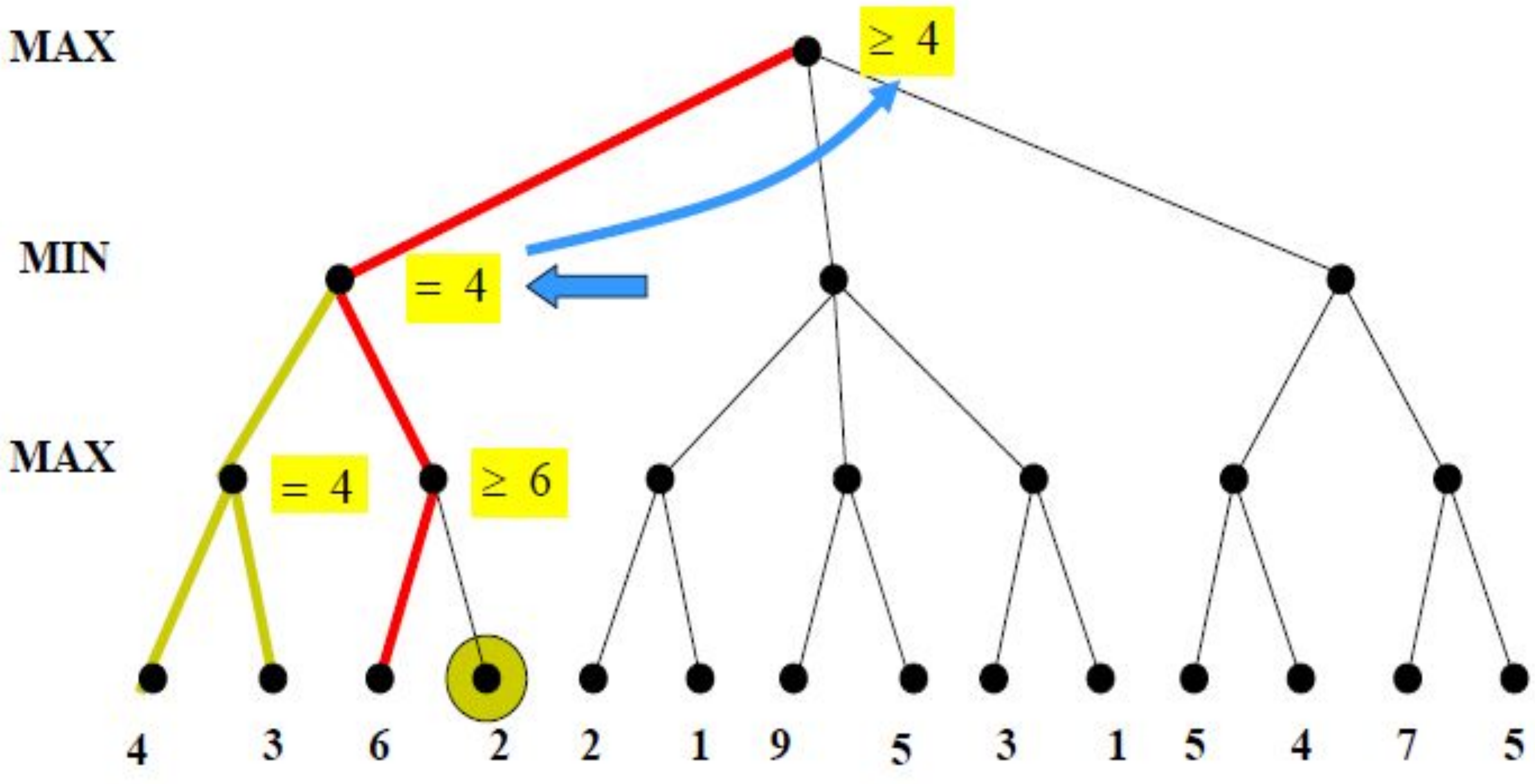
Alpha-beta pruning



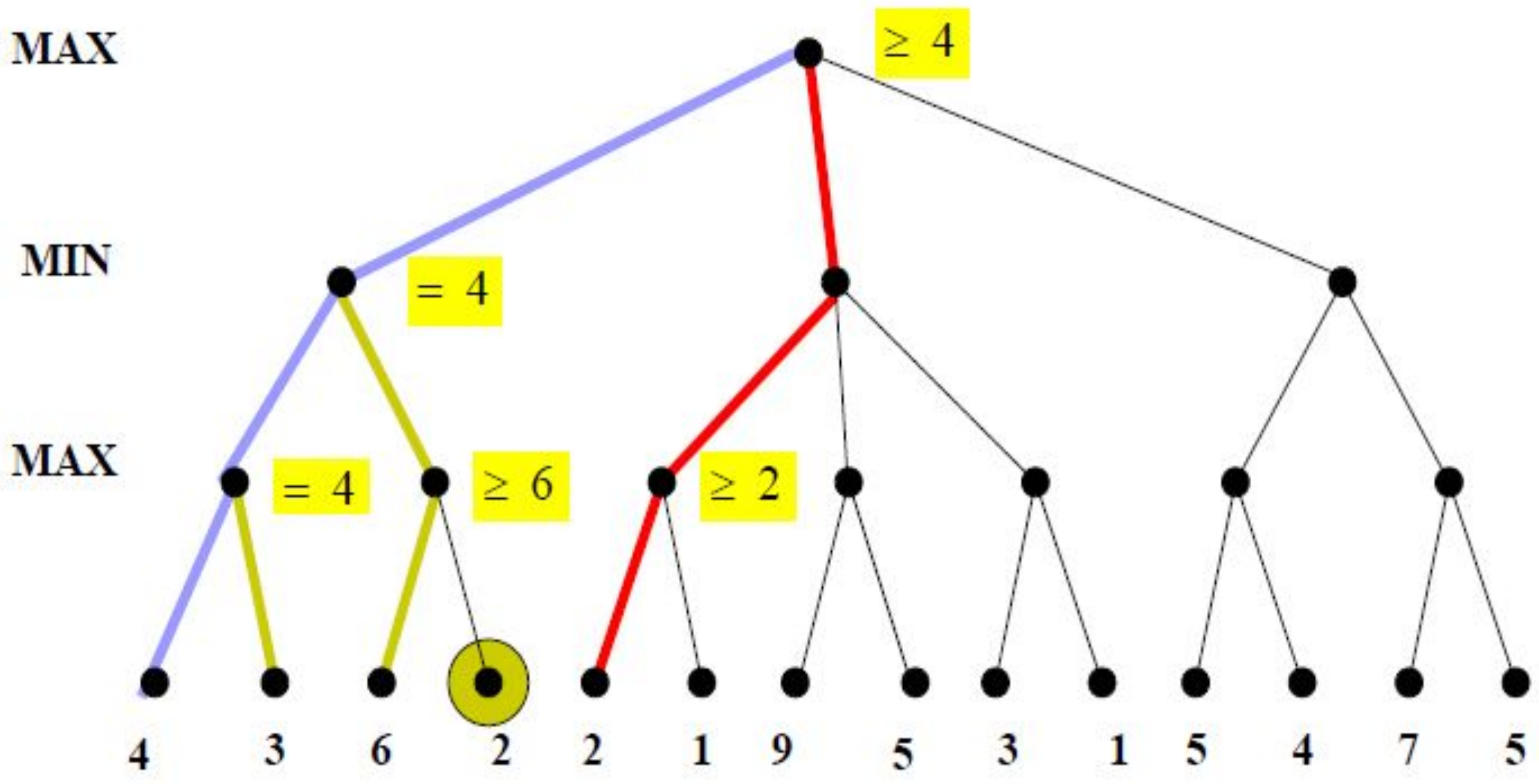
Alpha-beta pruning



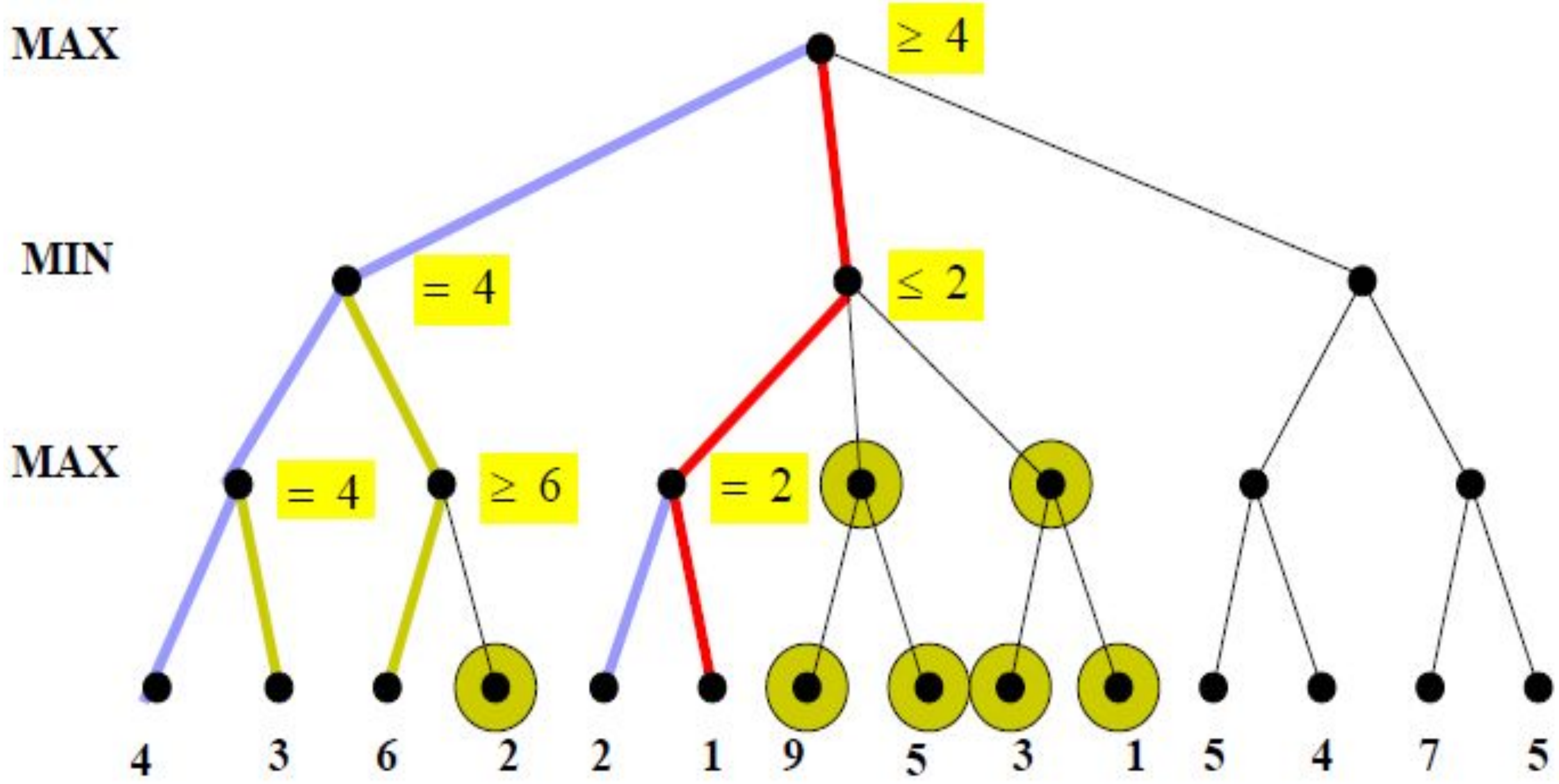
Alpha-beta pruning



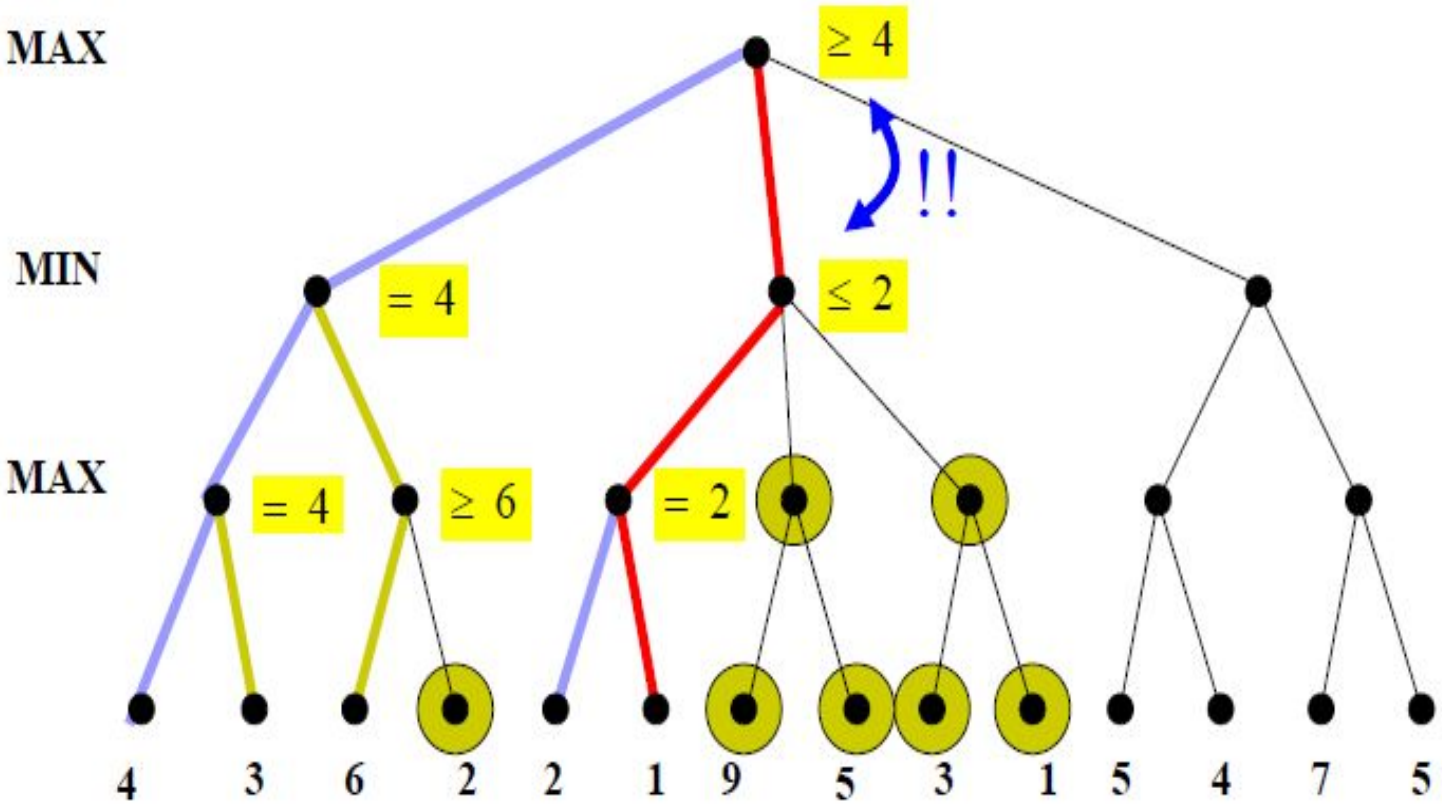
Alpha-beta pruning



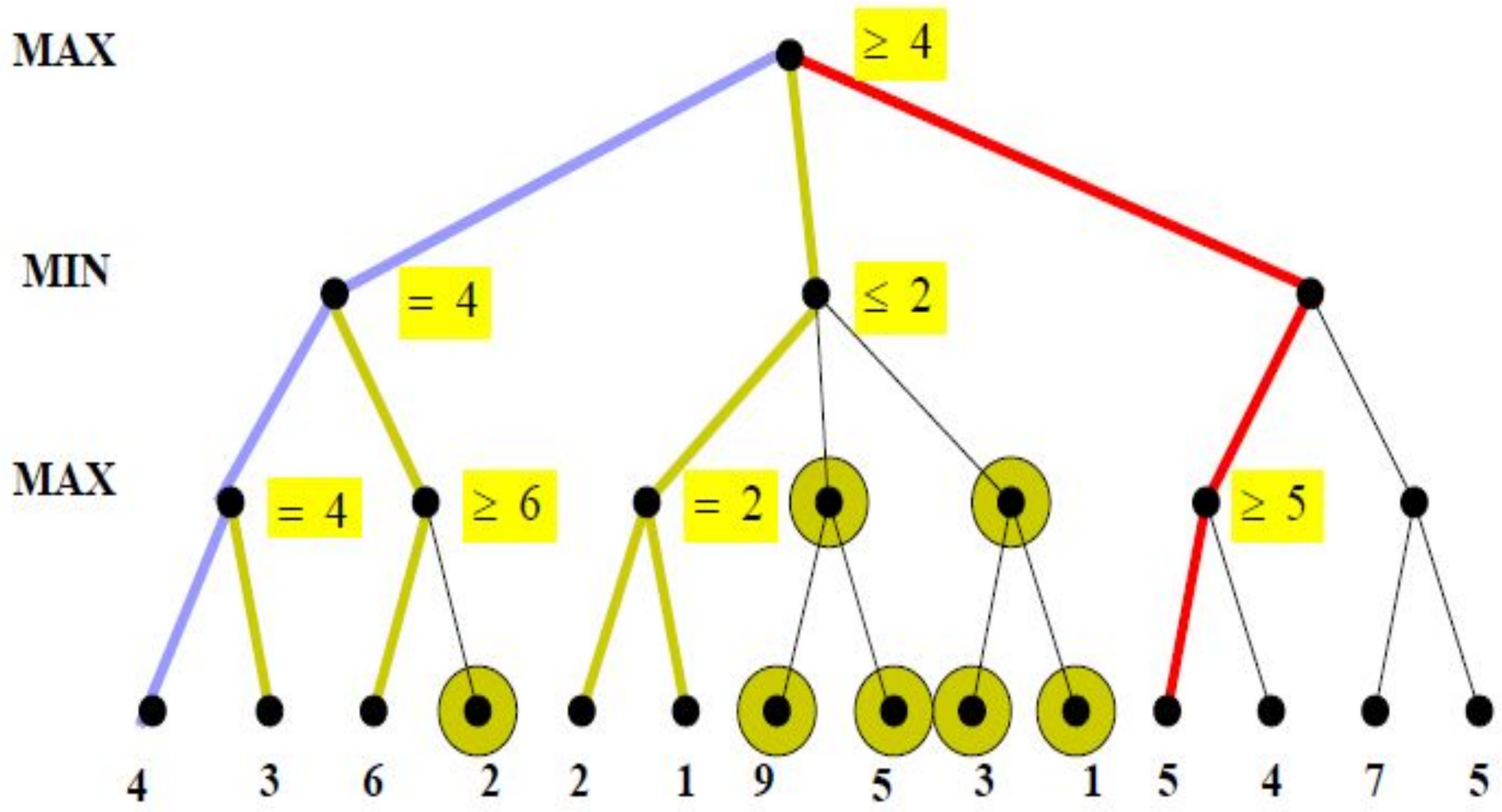
Alpha-beta pruning



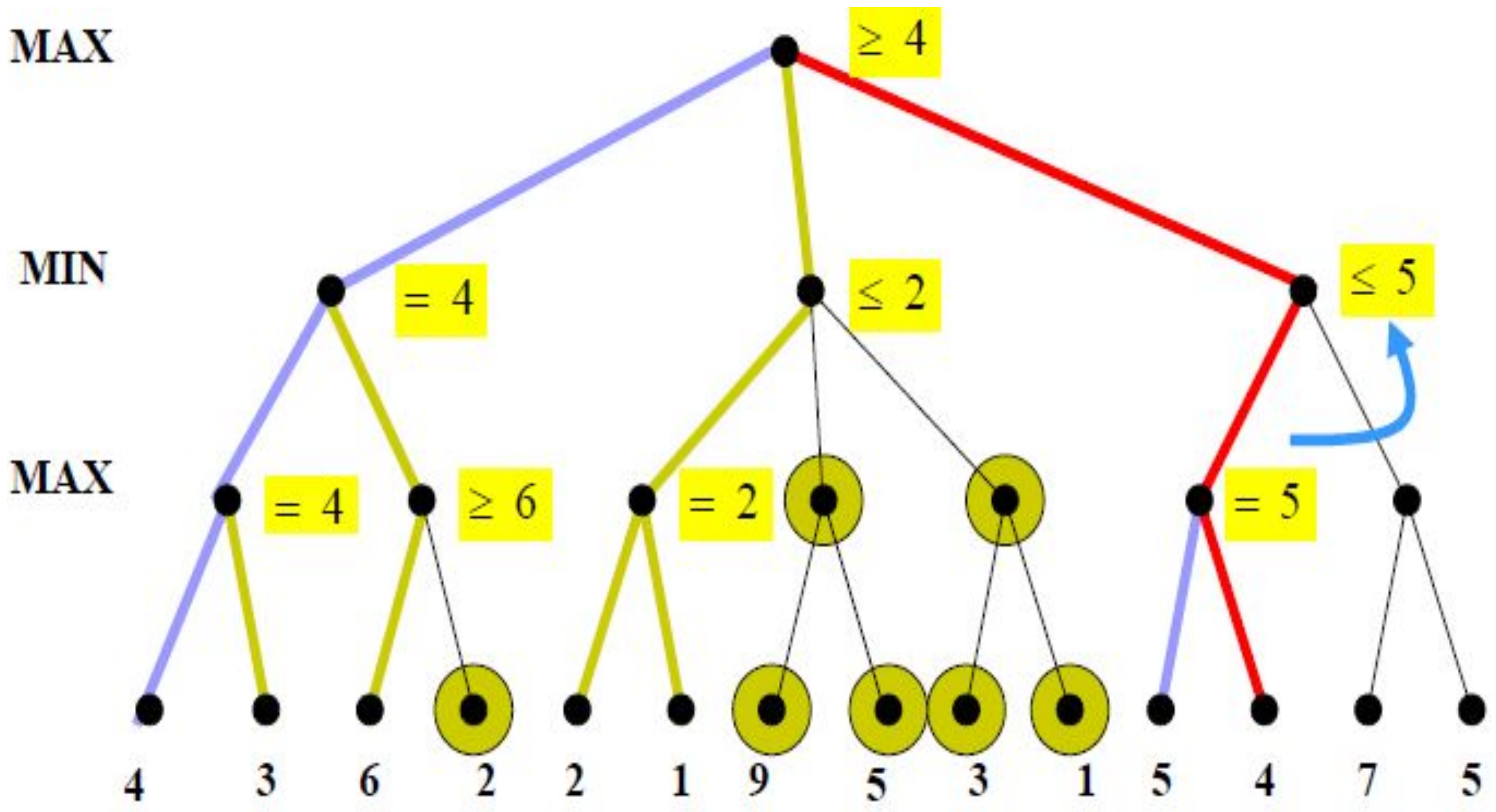
Alpha-beta pruning



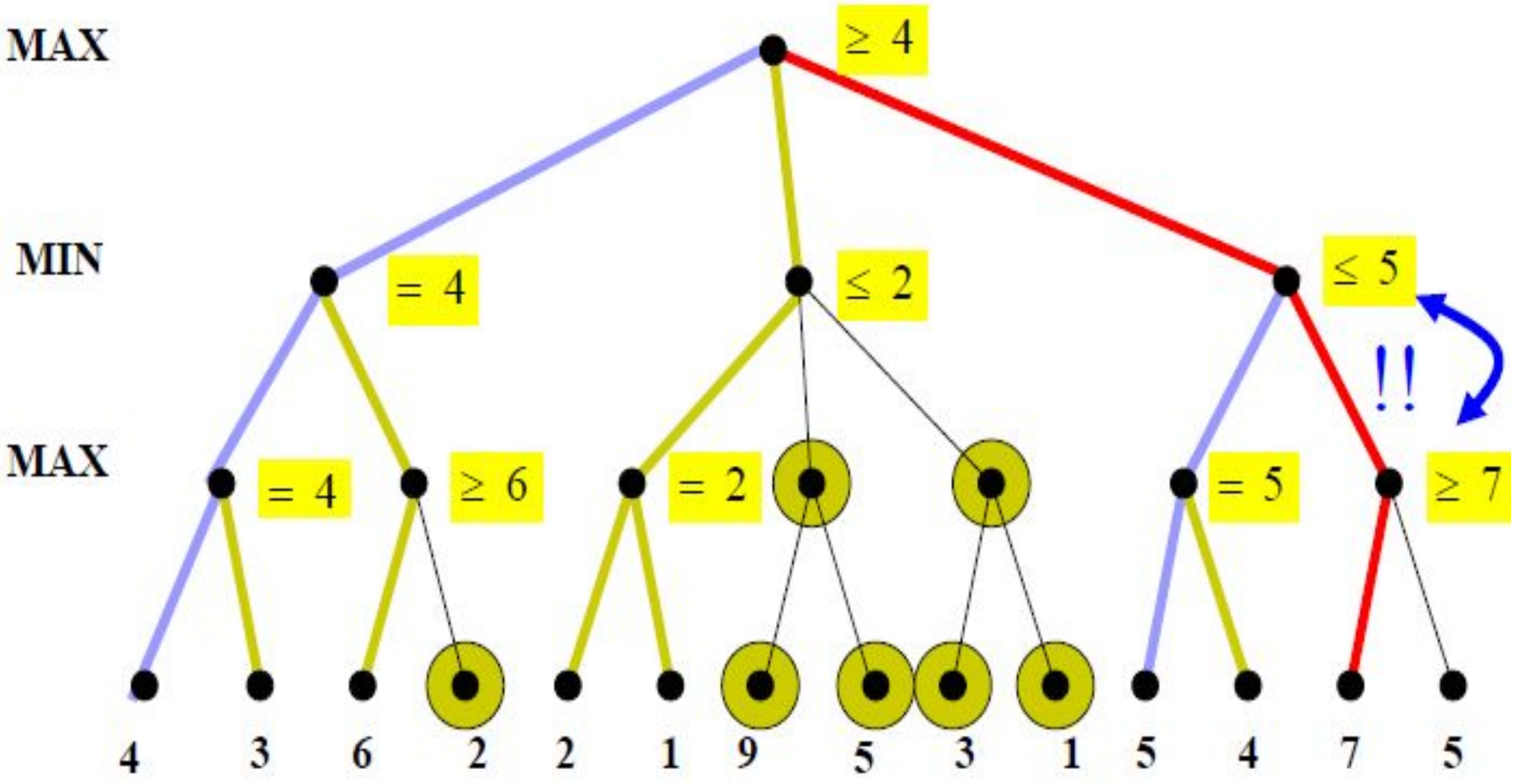
Alpha-beta pruning



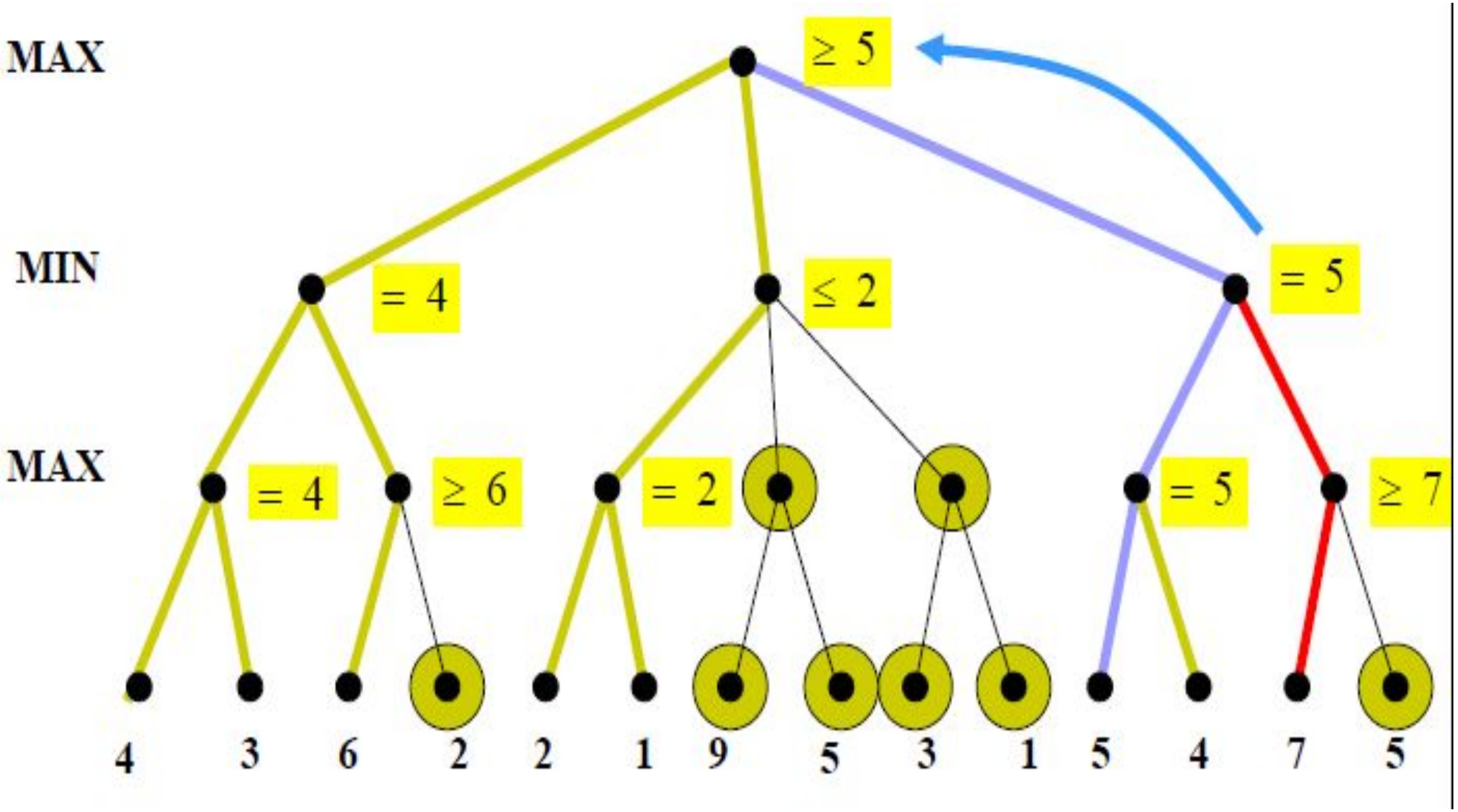
Alpha-beta pruning



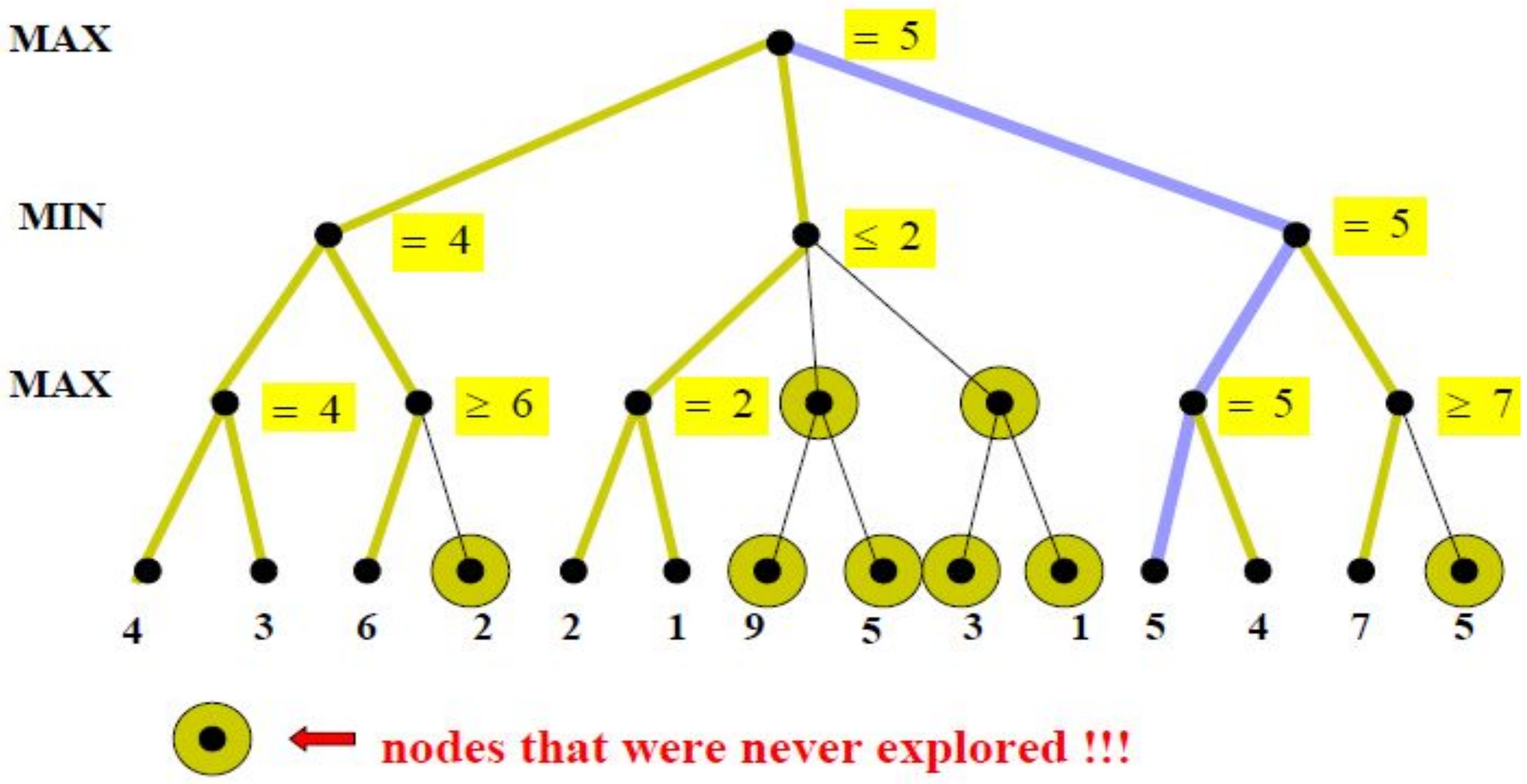
Alpha-beta pruning



Alpha-beta pruning



Alpha-beta pruning



Properties of Alpha-beta pruning

- **Completeness:** There is no completeness.
- **Optimality:** This is against an optimal player. However, the fixed-depth version is not optimal.
- **Time complexity:** With the perfect child ordering, $O(b^{m/2})$, where b is the branching factor and m is the maximum depth or depth limit. Child ordering has a huge impact on the effectiveness of the pruning.
- **Space complexity:** $O(bm)$.
- Pruning does not affect the final result (it is exact).
- Good-move ordering improves effectiveness of the pruning (see the last branch in examples).
- Different orderings of the sequences of the moves may lead to the same state. Save the value of these “transpositions” to avoid the double work. With “perfect ordering,” time complexity = $O(b^{m/2})$ doubles depth of the search.

Thank you