Adversarial Search (Game)

https://www.javatpoint.com/ai-adversarial-search
https://www.javatpoint.com/mini-max-algorithm-in-ai
https://www.javatpoint.com/ai-alpha-beta-pruning

Review: blind search and heuristic search

- In **previous topics**
  - search strategies which are **only associated with a single agent** that aims to find the solution
  - **expressed in the form of a sequence of actions**.

- But, there might be some situations where **more than one agent is searching for the solution in the same search space**.
- This situation **usually occurs in game playing** → **Adversarial Search**

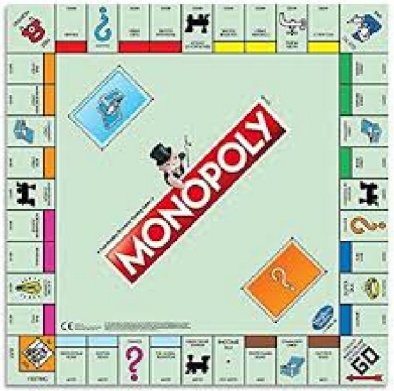
Adversarial Search

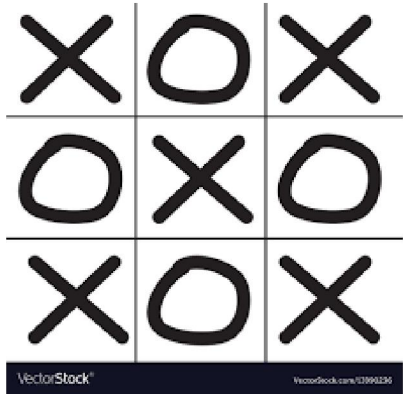
- **Adversarial search** is **search** when there is an "enemy" or "opponent" changing the state of the problem every step in a direction you do not want.
- Examples: Chess, business, trading, war.
- You change state, but then you don't control the next state.
- Opponent will change the next state in a way: **unpredictable**.
- **We try to plan ahead of the world and other agents are planning against us**.

- 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





1. **Perfect information:** A game with the perfect information is that in which agents can look into the complete board. Agents have all the information about the game, and they can see each other moves also. Examples are Chess, Checkers, Go, etc.
2. **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, Battleship, blind, Bridge, etc.
3. **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.
4. **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. Such games are also called as stochastic games.
   Example: Backgammon, Monopoly, Poker, etc.

Why new techniques for games?

⊙ "Contingency" problem:

- We don't know the opponents move !

- The size of the search space:
  - Chess : ~15 moves possible per state, 80 ply
    - 15^{80} nodes in tree
  - Go : ~200 moves per state, 300 ply
    - 200^{300} nodes in tree

- Game playing algorithms:
  - Search tree only up to some depth bound
  - Use an evaluation function at the depth bound
  - Propagate the evaluation upwards in the tree

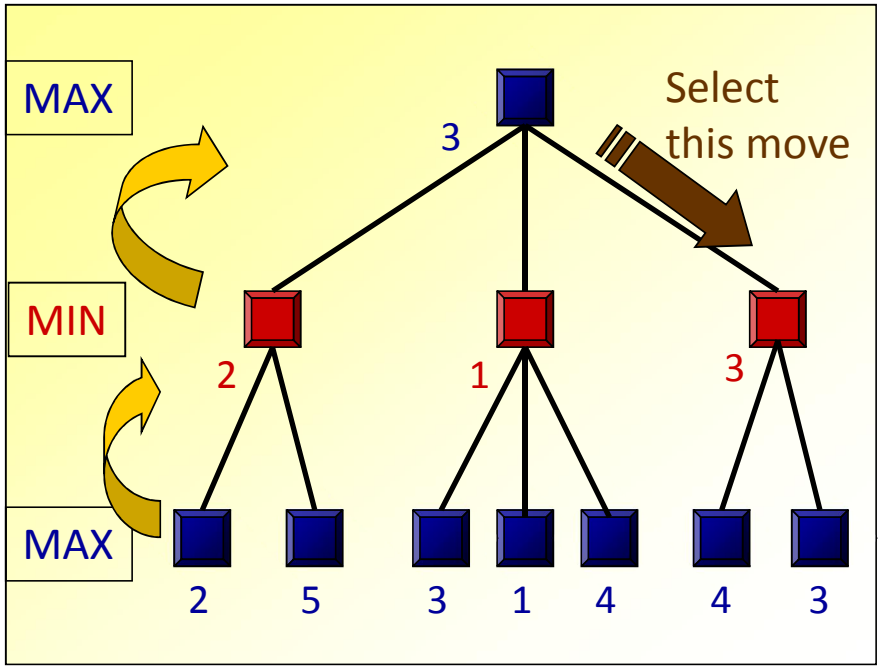
Mini-Max Algorithm

MINI MAX

- Restrictions:
  - 2 players: MAX (computer) and MIN (opponent)

  ➔ deterministic, perfect information

⊙ Select a depth-bound (say: 2) and evaluation function



- Construct the tree up till the depth-bound

- Compute the evaluation function for the leaves

- Propagate the evaluation function upwards:
  - taking minima in MIN
  - taking maxima in MAX

The MINI-MAX algorithm:

```
Initialise depthbound;

Minimax (board, depth) =

IF depth = depthbound
    THEN    return static_evaluation(board);
    ELSE IF   maximizing_level(depth)
          THEN FOR EACH   child child of board
                    compute Minimax(child, depth+1);
           return maximum over all children;
    ELSE IF   minimizing_level(depth)
          THEN FOR EACH   child child of board
                    compute Minimax(child, depth+1);
           return minimum over all children;
```

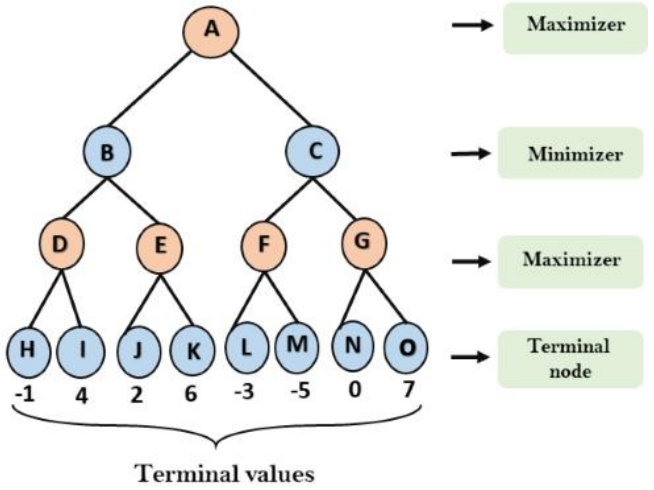
Call: Minimax(current_board, 0)

Resume of Mini-Max Algorithm

- In this algorithm two players play the game, one is called MAX and other is called MIN.
- Both the players fight it as the opponent player gets the minimum benefit while the player get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.
- The minimax algorithm performs a depth-first search (DFS) algorithm for the exploration of the complete game tree.
- Example: Chess, Checkers, tic-tac-toe, go, and various two-players game.

Working of Min-Max Algorithm:

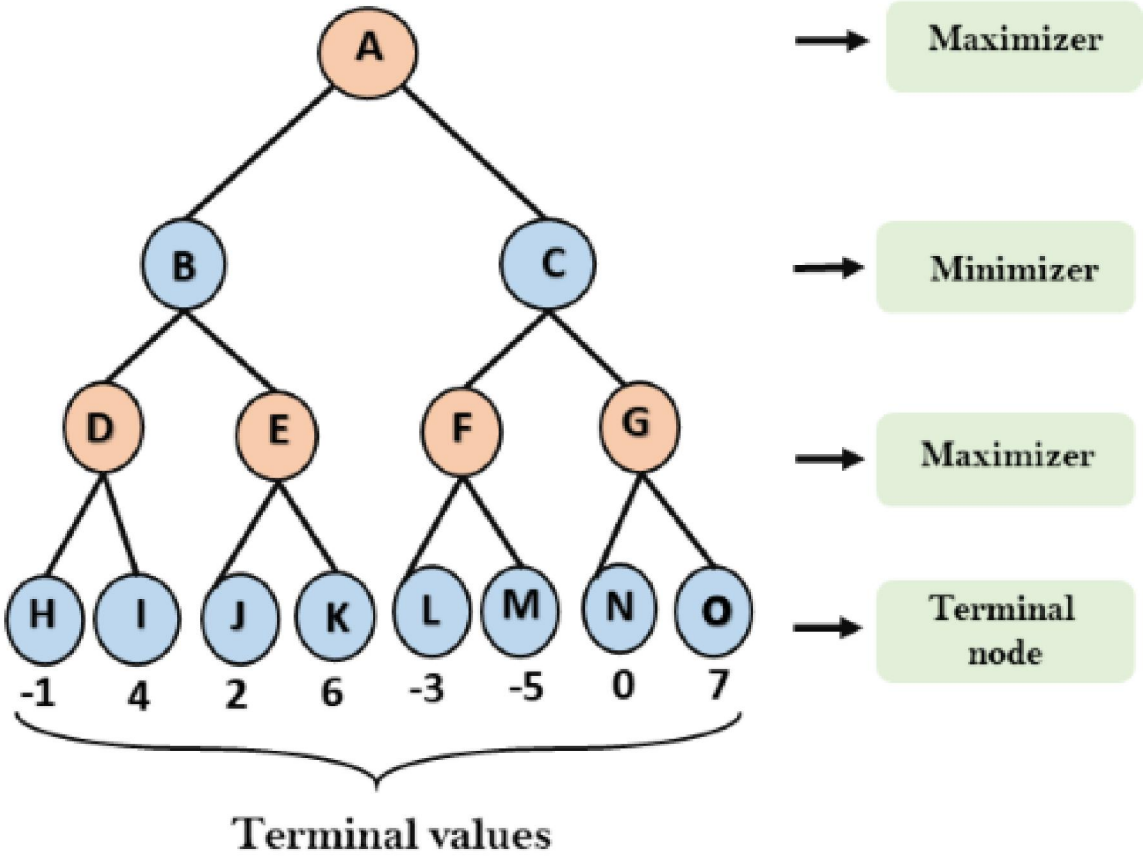
- Below figure we have taken an example of game-tree which is representing the two-player game.
- In this example, there are two players one is called Maximizer and other is called Minimizer.
- Maximizer will try to get the Maximum possible score, and Minimizer will try to get the minimum possible score.
- This algorithm applies DFS, so in this game-tree, we have to go all the way through the leaves to reach the terminal nodes.
- At the terminal node, the terminal values are given so we will compare those value and backtrack the tree until the initial state occurs.



Example 2: Minimax algorithm.

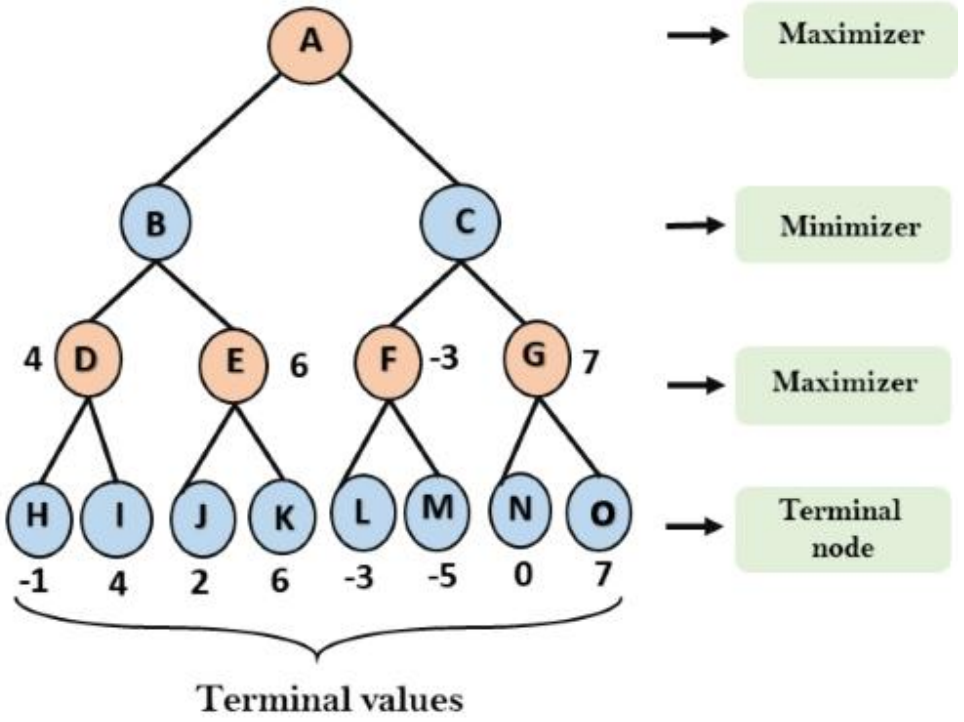
Step-1:

- In the first step, the algorithm **generates the entire game-tree** and apply the utility function to get the utility values for the terminal states.
- In the below tree diagram, let's take A is the initial state of the tree.
- Suppose :
  - maximizer takes first turn which has worst-case initial value =- infinity, and
  - minimizer will take next turn which has worst-case initial value = +infinity.



Step 2:

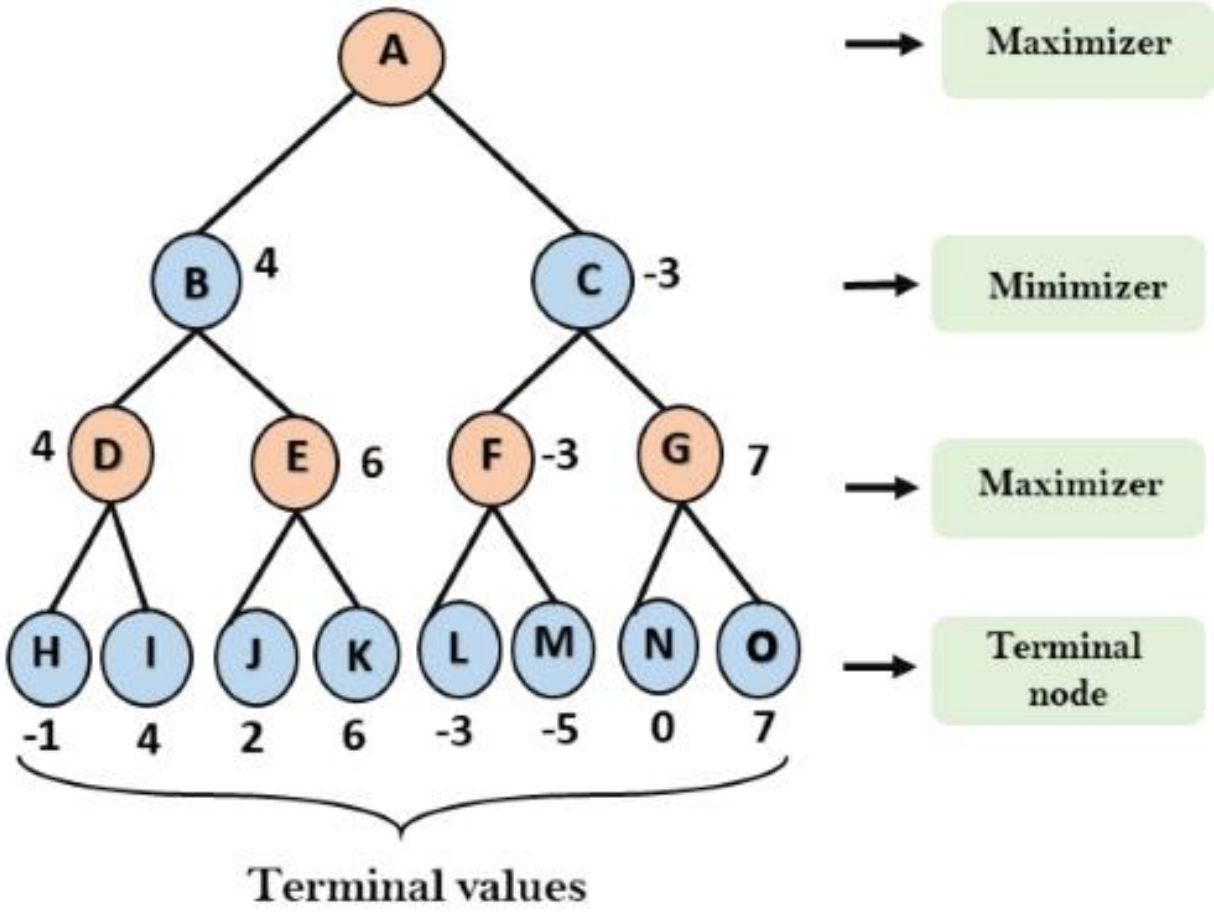
- Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.
- For node D max(-1, -∞) => max(-1,4) = 4
- For Node E max(2, -∞) => max(2, 6) = 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) => max(0, 7) = 7



Step 3:

In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

- For node B= min(4,6) = 4
- For node C= min (-3, 7) = -3

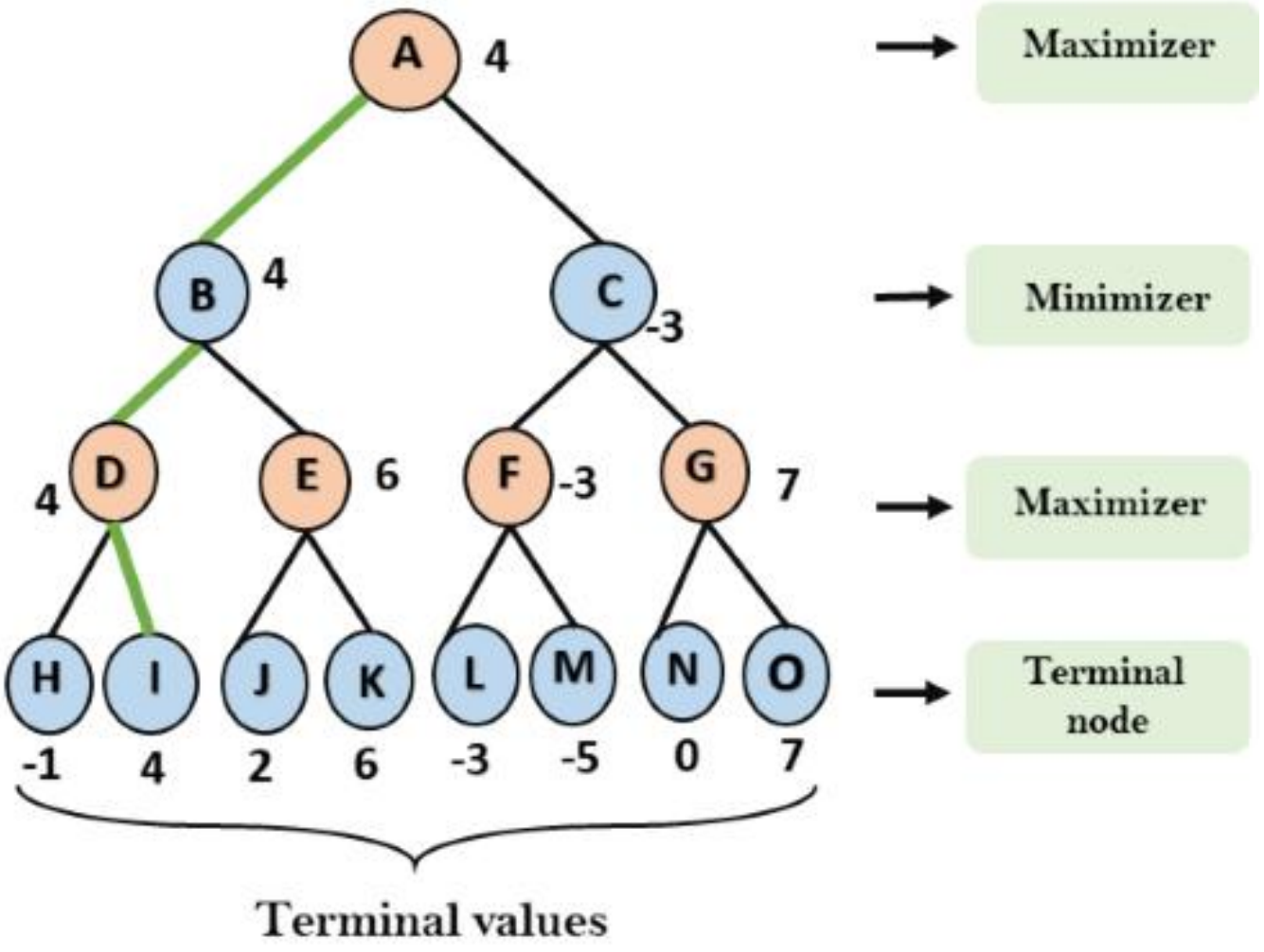


Step 4:

Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node.

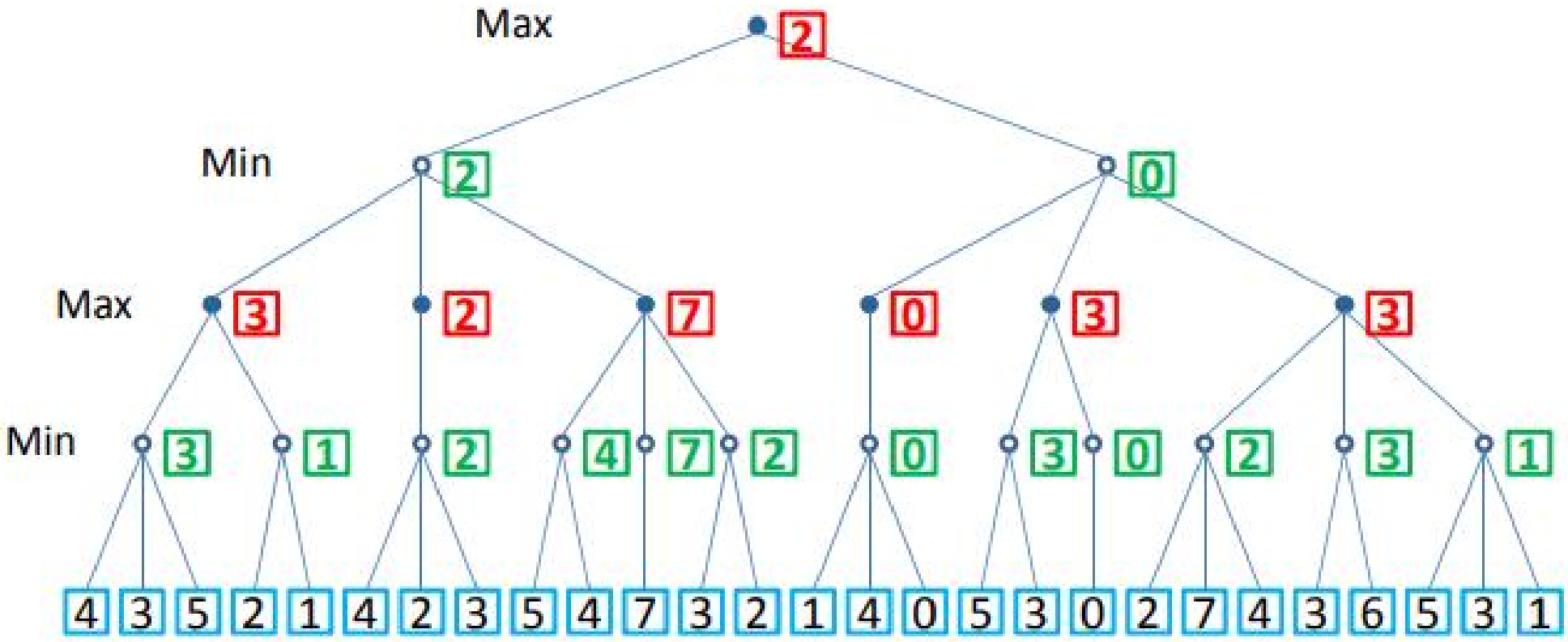
In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

- For node A max(4, -3)= 4



That was the complete workflow of the minimax two player game.
Rute hasil = A-B-D-I
Dengan nilai = 4

Example 3: Minimax algorithm



Properties of Mini-Max algorithm:

- **Complete:** Min-Max algorithm is Complete. It will definitely find a solution (if exist), in the finite search tree.
- **Optimal:** Min-Max algorithm is optimal if both opponents are playing optimally.
- **Time complexity:** As it performs DFS for the game-tree, so the time complexity of Min-Max algorithm is **$O(b^m)$**, where b is branching factor of the game-tree, and m is the maximum depth of the tree.
- **Space Complexity:** Space complexity of Mini-max algorithm is also similar to DFS which is **O(bm)**.

Limitation of the minimax Algorithm:

- The main drawback of the minimax algorithm is that it gets really slow for complex games such as Chess, go, etc.
- This type of games has a huge branching factor, and the player has lots of choices to decide.
- This limitation of the minimax algorithm can be improved from alpha-beta pruning which we have discussed in the next topic.

Alpha-Beta Pruning

Alpha-Beta Pruning

- Generally applied optimization on Mini-max.

- Instead of:
  - ➔ first creating the entire tree (up to depth-level)
  - ➔ then doing all propagation
- Interleave the generation of the tree and the propagation of values.
- Point:
  - ➔ some of the obtained values in the tree will provide information that other (non-generated) parts are redundant and do not need to be generated.

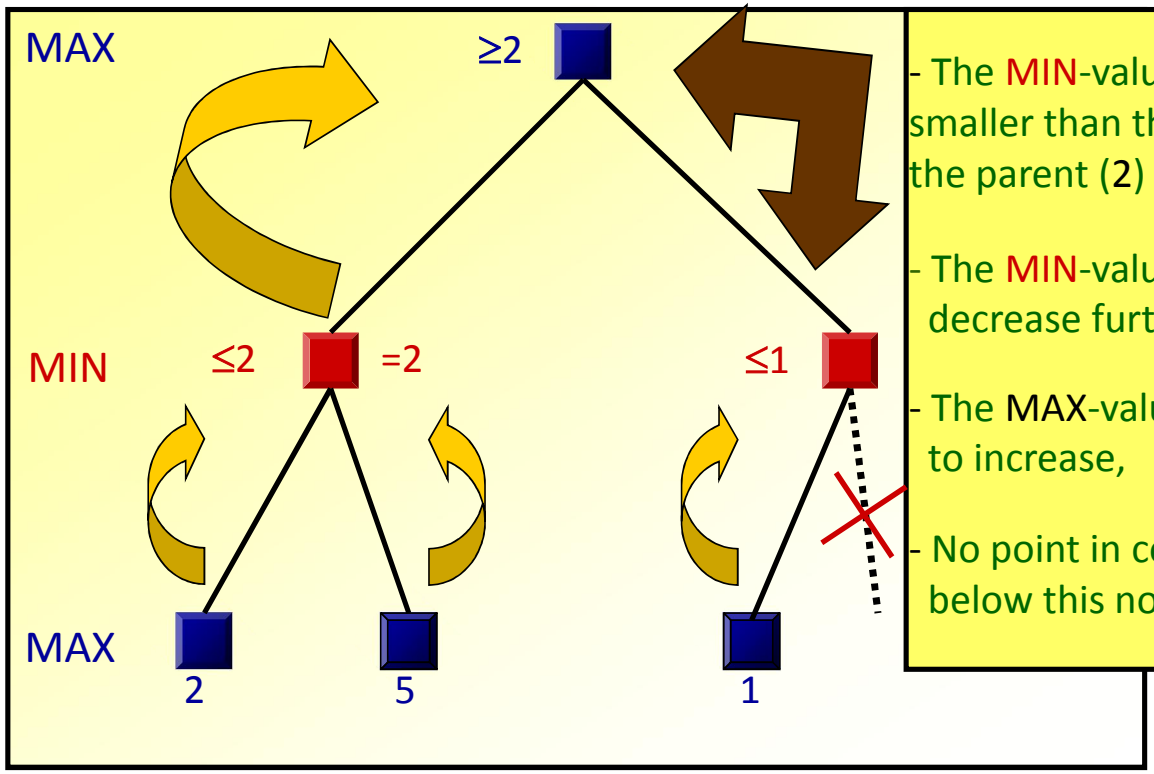
Alpha-Beta Pruning (Alpha-Beta Cut-off)

- Generally applied optimization on Mini-max.
- Instead of:
  - ➔ first creating the entire tree (up to depth-level)
  - ➔ then doing all propagation
- Interleave the generation of the tree and the propagation of values.
- Point:
  - ➔ some of the obtained values in the tree will provide information that other (non-generated) parts are redundant and do not need to be generated.

Alpha-Beta idea:

- Principles:
  - generate the tree depth-first, left-to-right

  ➔ propagate final values of nodes as initial estimates for their parent node.



- The MIN-value (1) is already smaller than the MAX-value of the parent (2)

- The MIN-value can only decrease further,

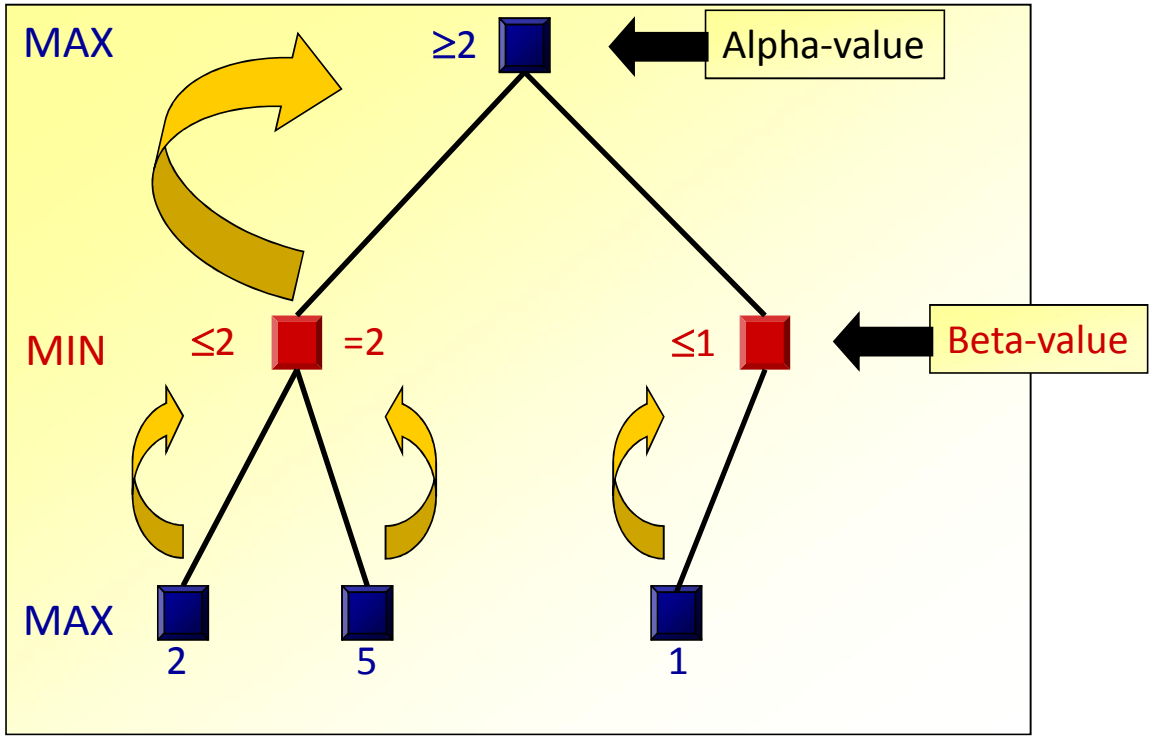
- The MAX-value is only allowed to increase,

- No point in computing further below this node

Terminology:

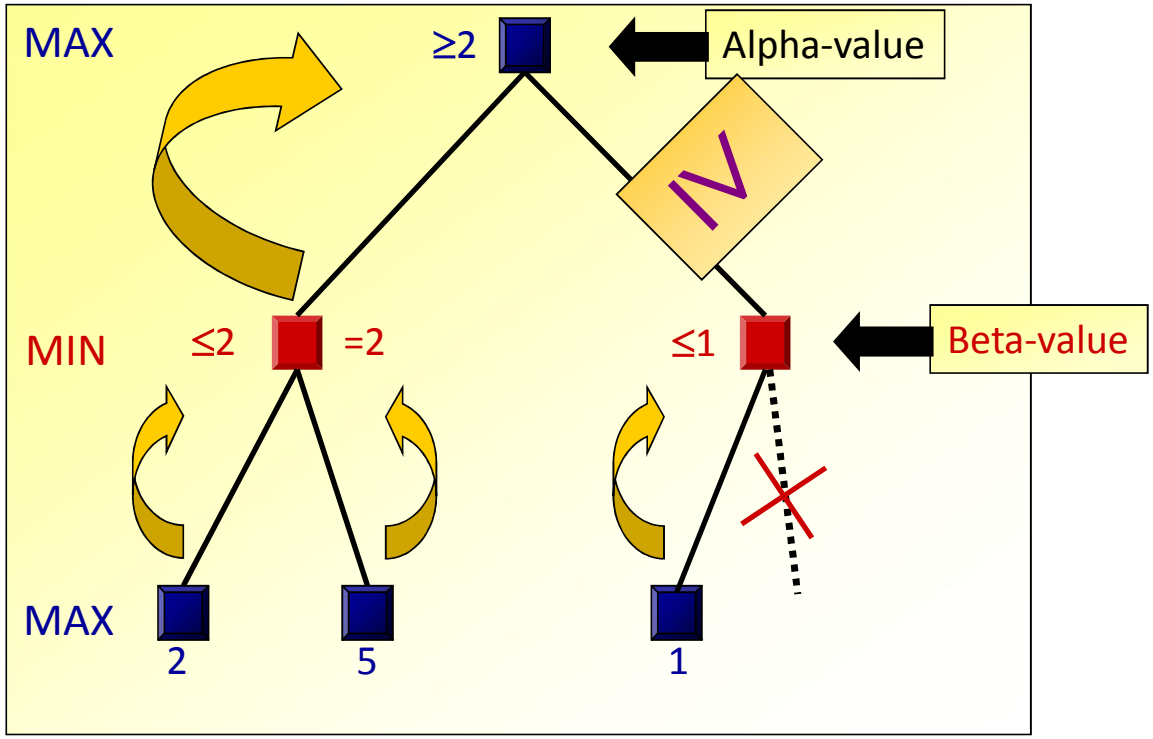
- The (temporary) values at MAX-nodes are ALPHA-values

- The (temporary) values at MIN-nodes are BETA-values



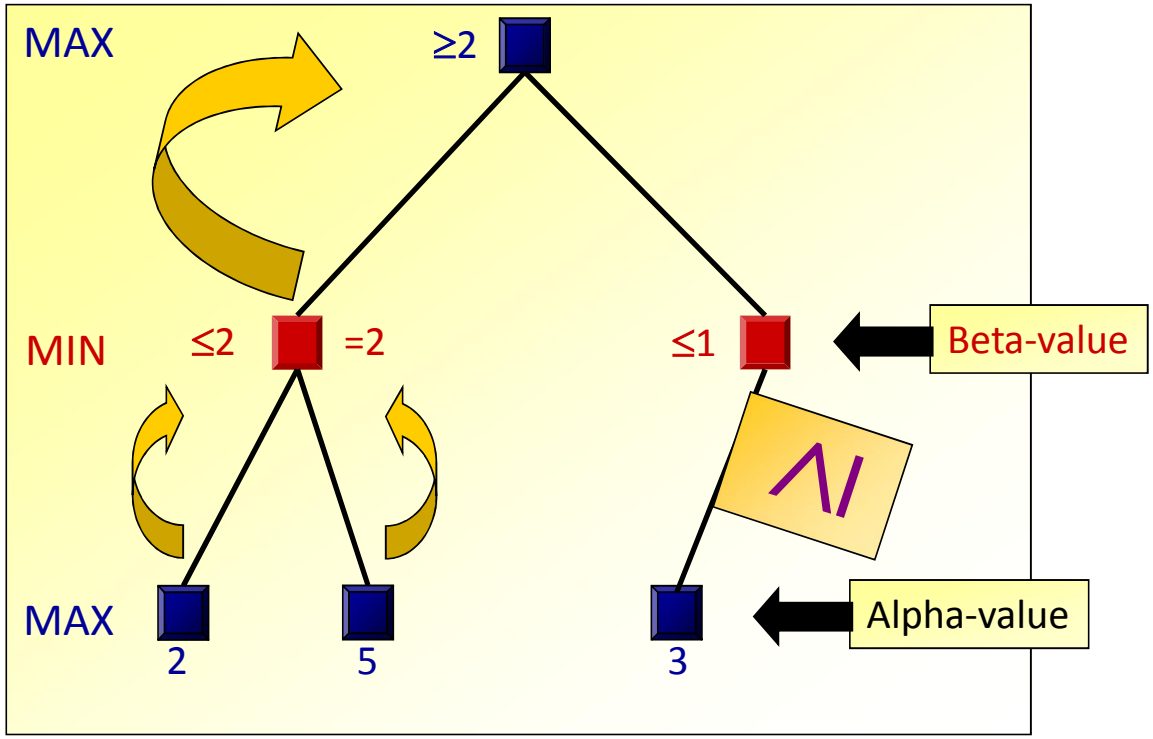
The Alpha-Beta principles (1):

- If an ALPHA-value is larger or equal than the Beta-value of a descendant node:
  stop generation of the children of the descendant

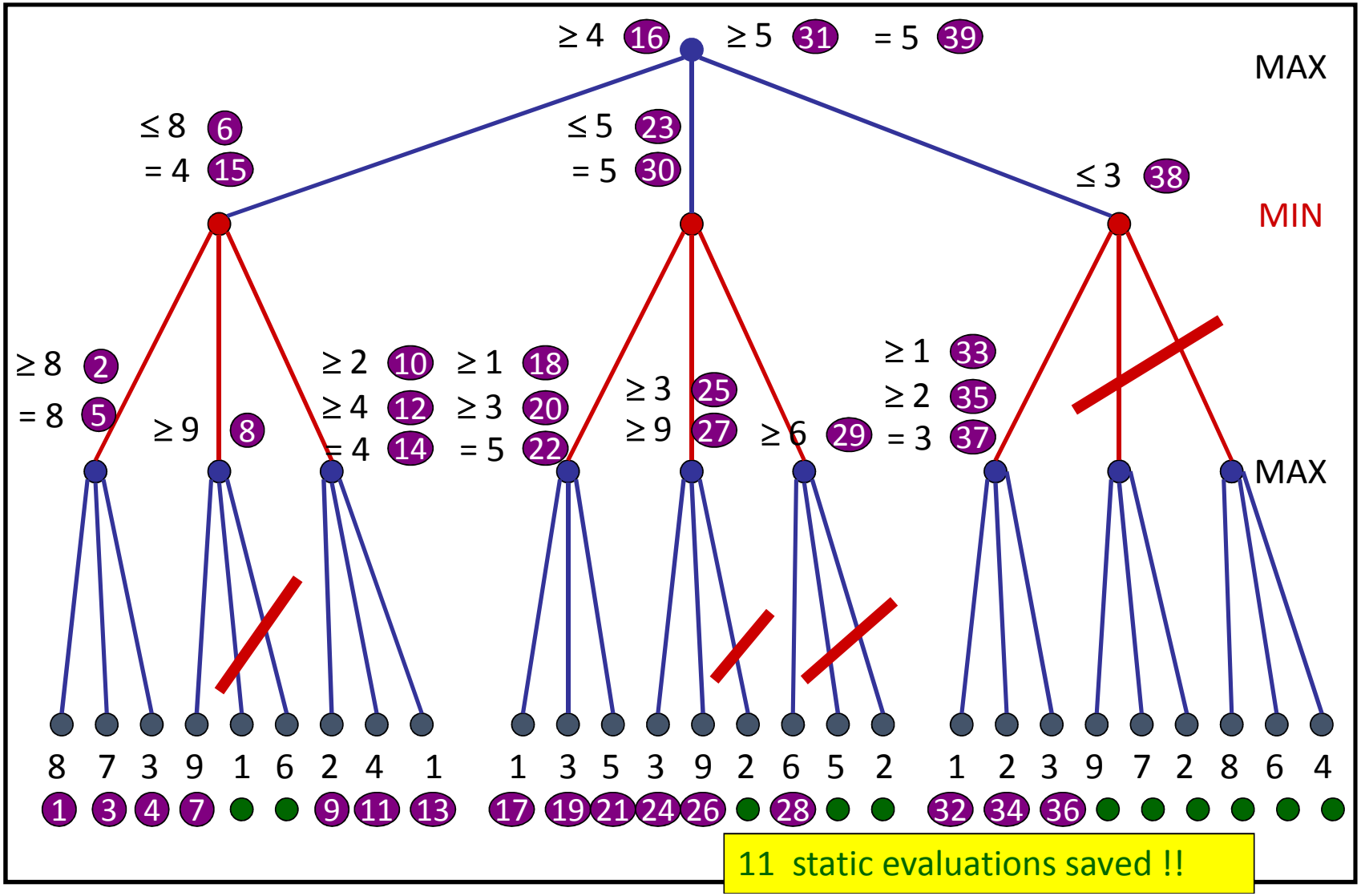


The Alpha-Beta principles (2):

- If an Beta-value is smaller or equal than the Alpha-value of a descendant node:
  stop generation of the children of the descendant

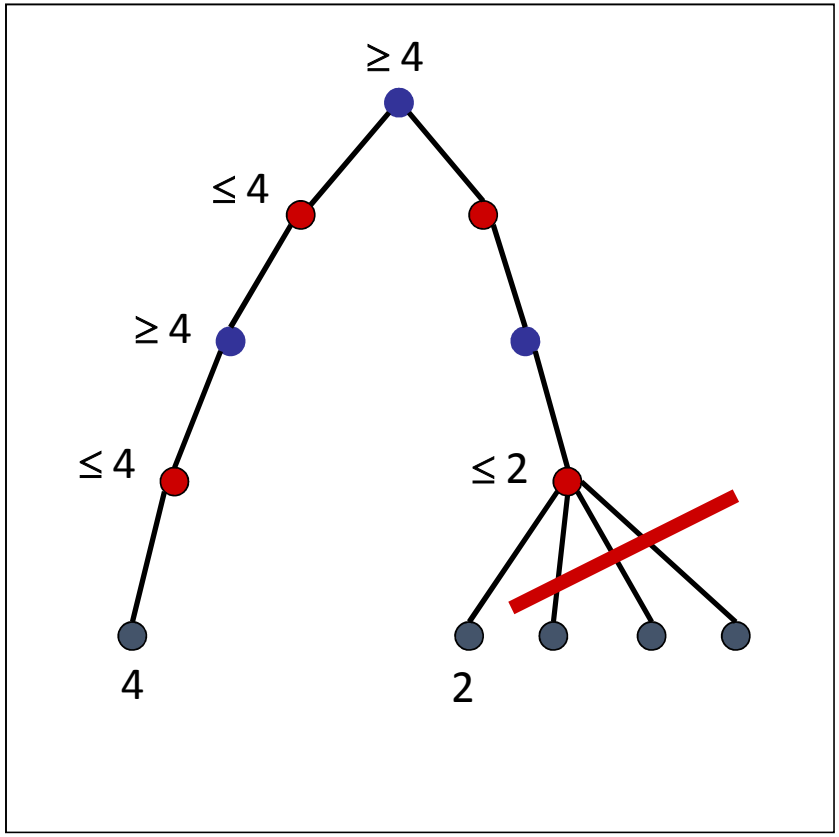


Mini-Max with α–β at work:



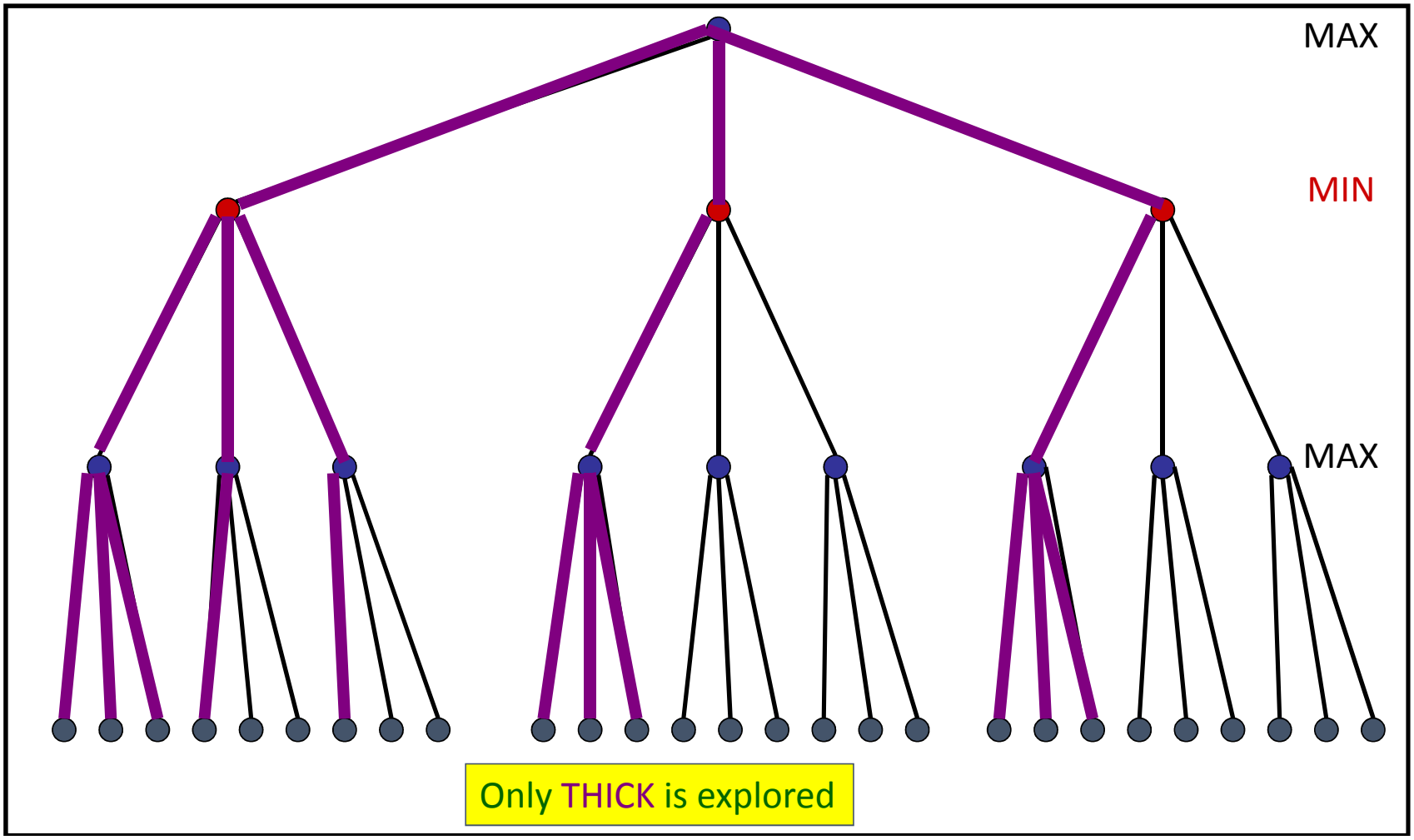
“DEEP” cut-offs

- For game trees with at least 4 Min/Max layers:
  the Alpha - Beta rules apply also to deeper levels.

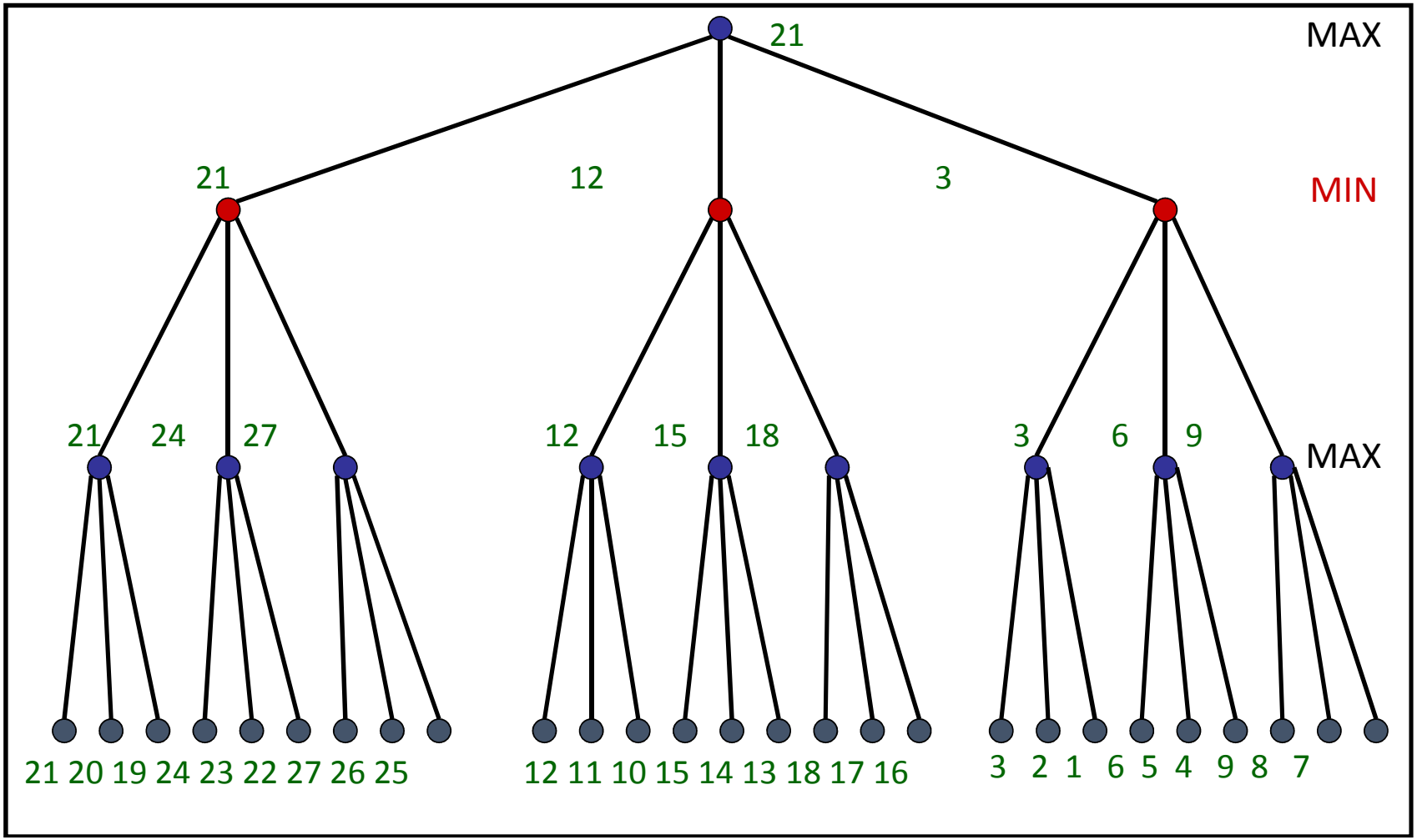


The Gain: Best case:

- If at every layer: the best node is the left-most one



Example of a perfectly ordered tree



Condition for Alpha-beta pruning

- The main condition which required for alpha-beta pruning is: $\alpha >= \beta$

Key points about alpha-beta pruning

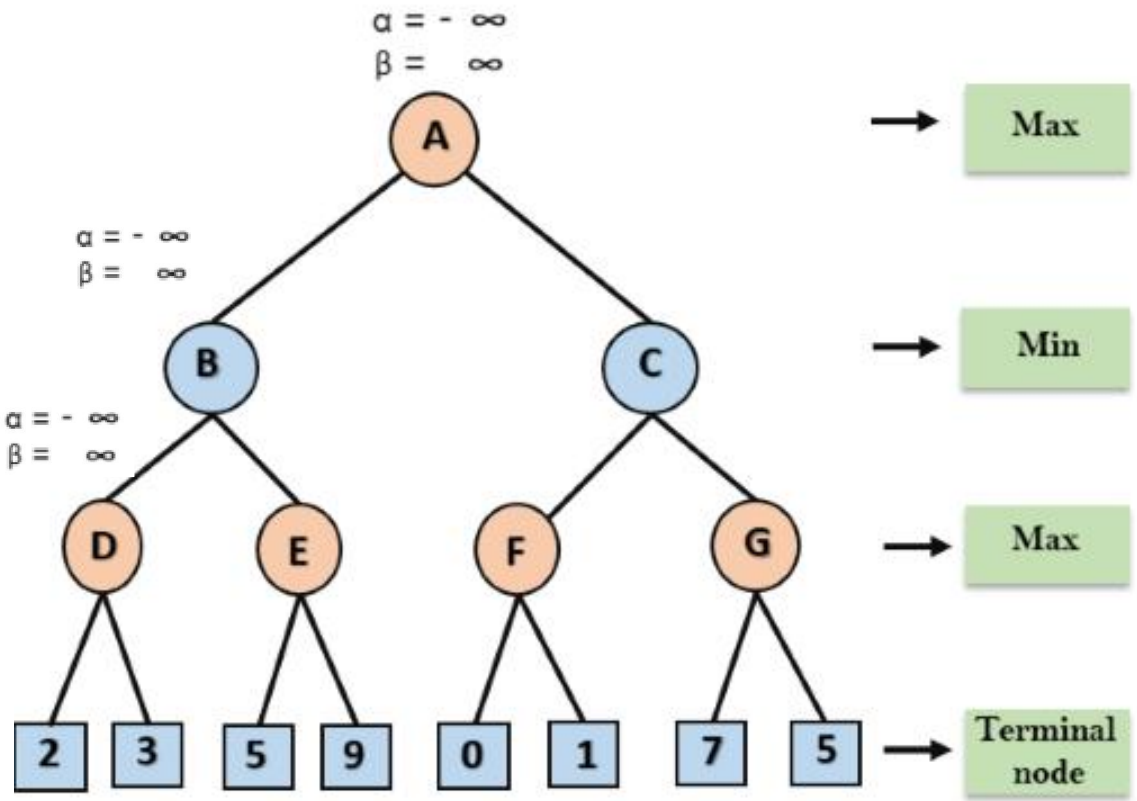
- The Max player will only update the value of alpha.
- The Min player will only update the value of beta.
- While backtracking the tree, the node values will be passed to upper nodes instead of values of alpha and beta.
- We will only pass the alpha, beta values to the child nodes.

Exersice 1: Working of Alpha-Beta Pruning

Let's take an example of two-player search tree to understand the working of Alpha-beta pruning

Step 1:

- At the first step the, Max player will start first move from node A where α= -∞ and β= +∞.
- These value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



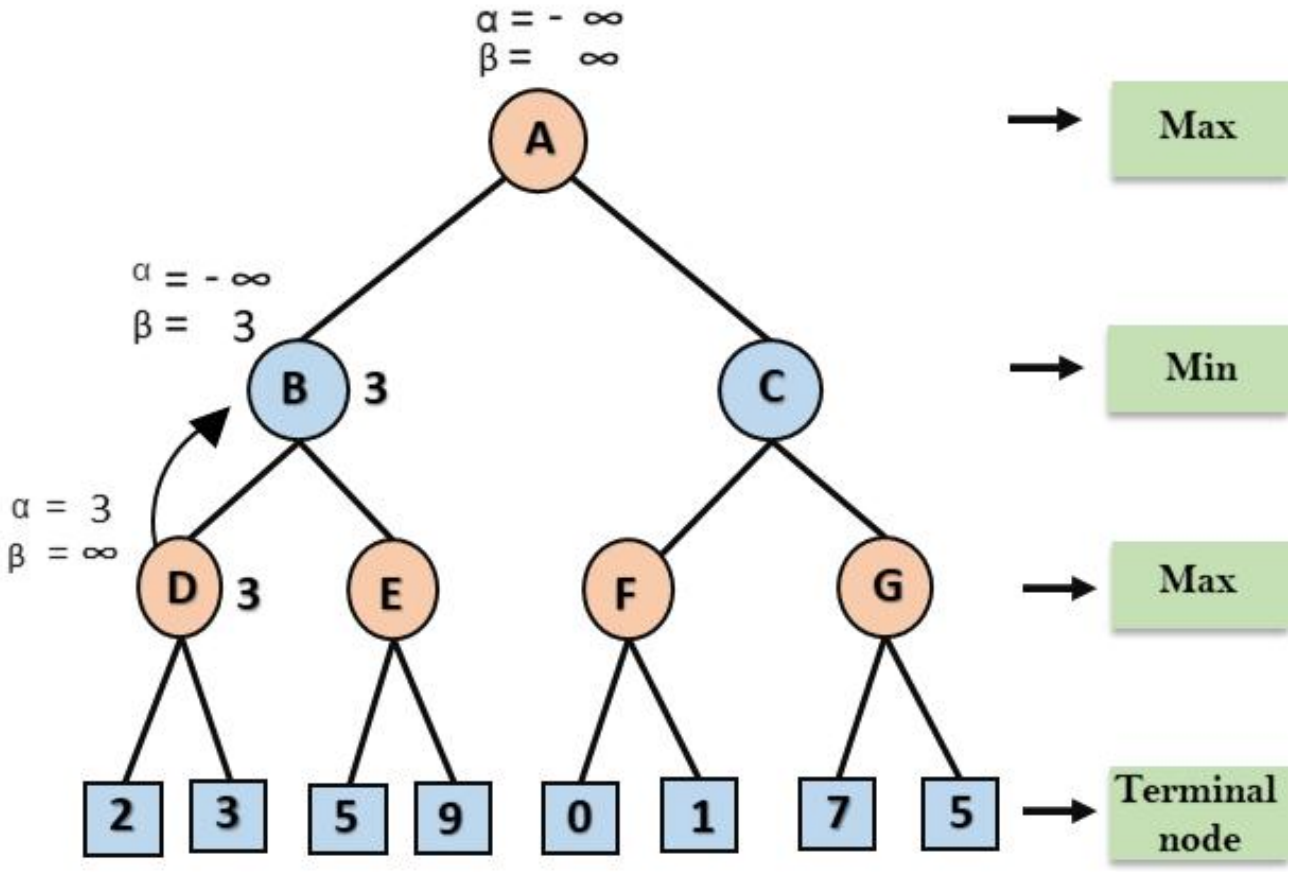
Step 2:

- At Node D, the value of α will be calculated as its turn for Max.
- The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

Step 3:

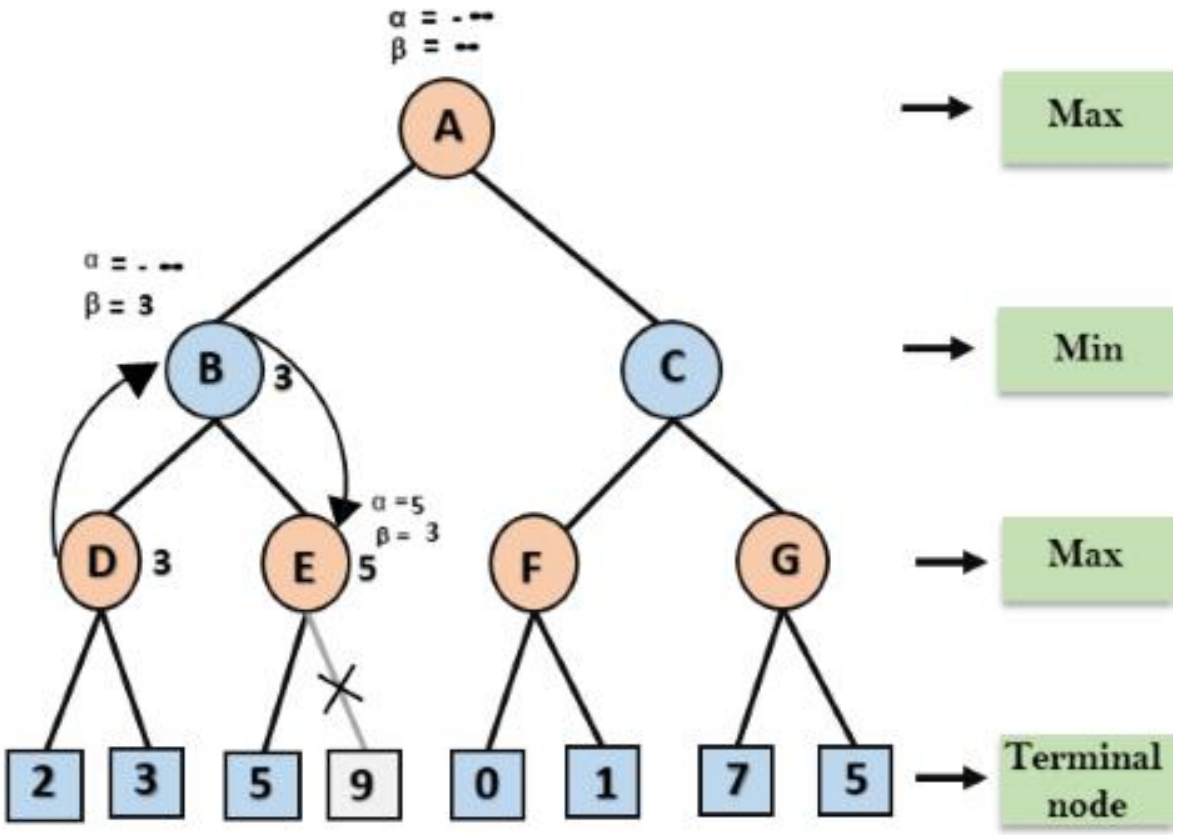
- Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min.
- Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.

In the **next step**, algorithm traverse the next successor of Node B which is **node E**, and the values of α= -∞, and β= 3 will also be passed.



Step 4:

- At node E, Max will take its turn, and the value of alpha will change.
- The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.

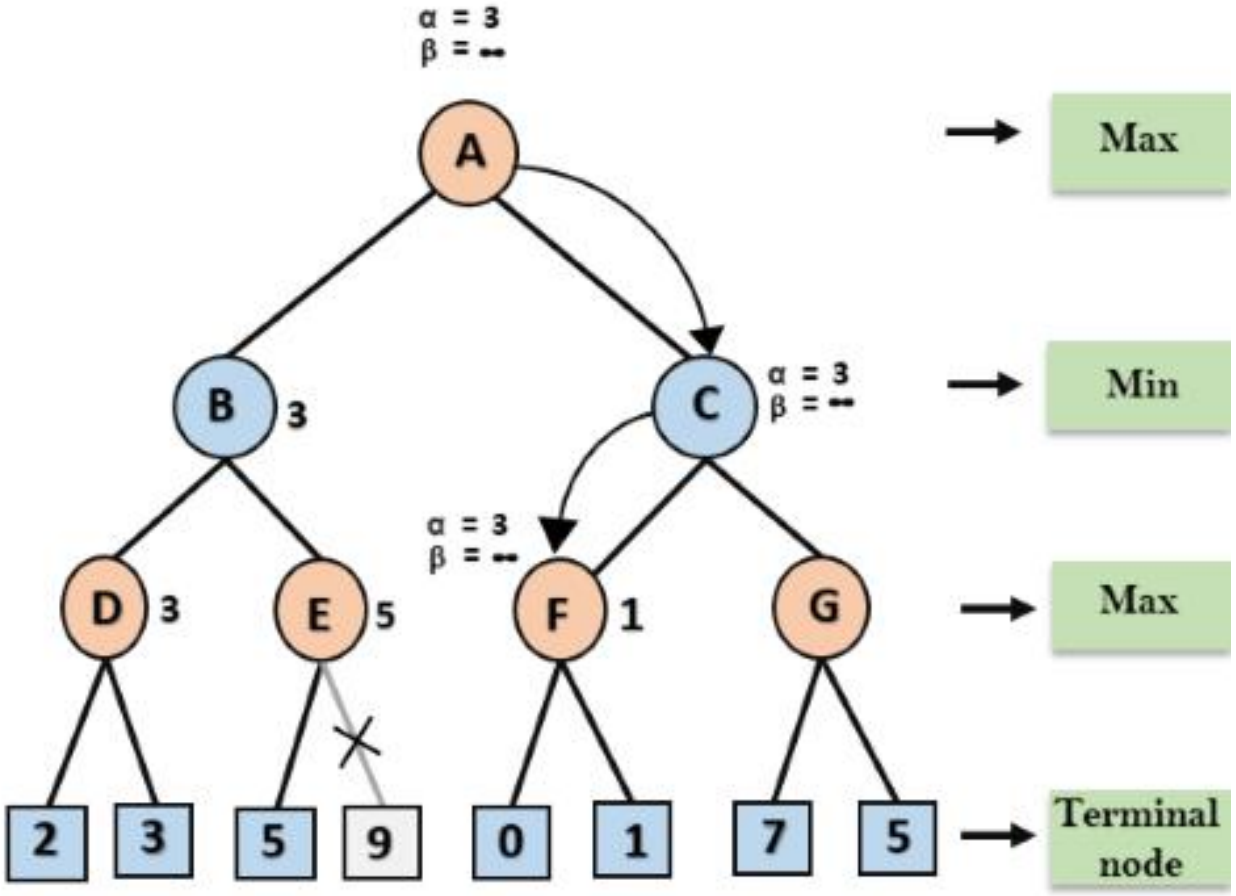


Step 5:

- At next step, algorithm again backtrack the tree, from node B to node A.
- At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.
- At node C, α=3 and β= +∞, and the same values will be passed on to node F.

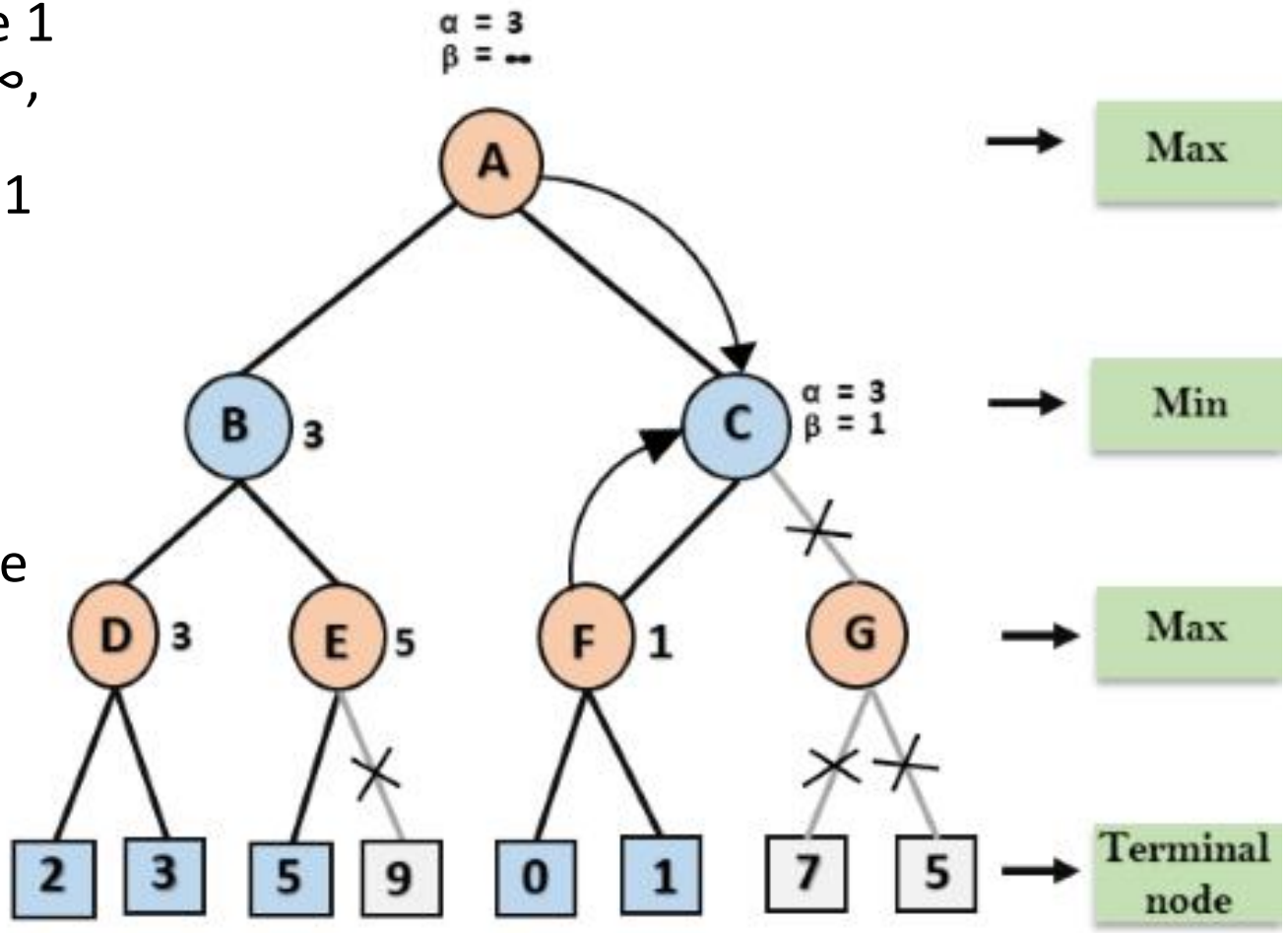
Step 6:

- At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



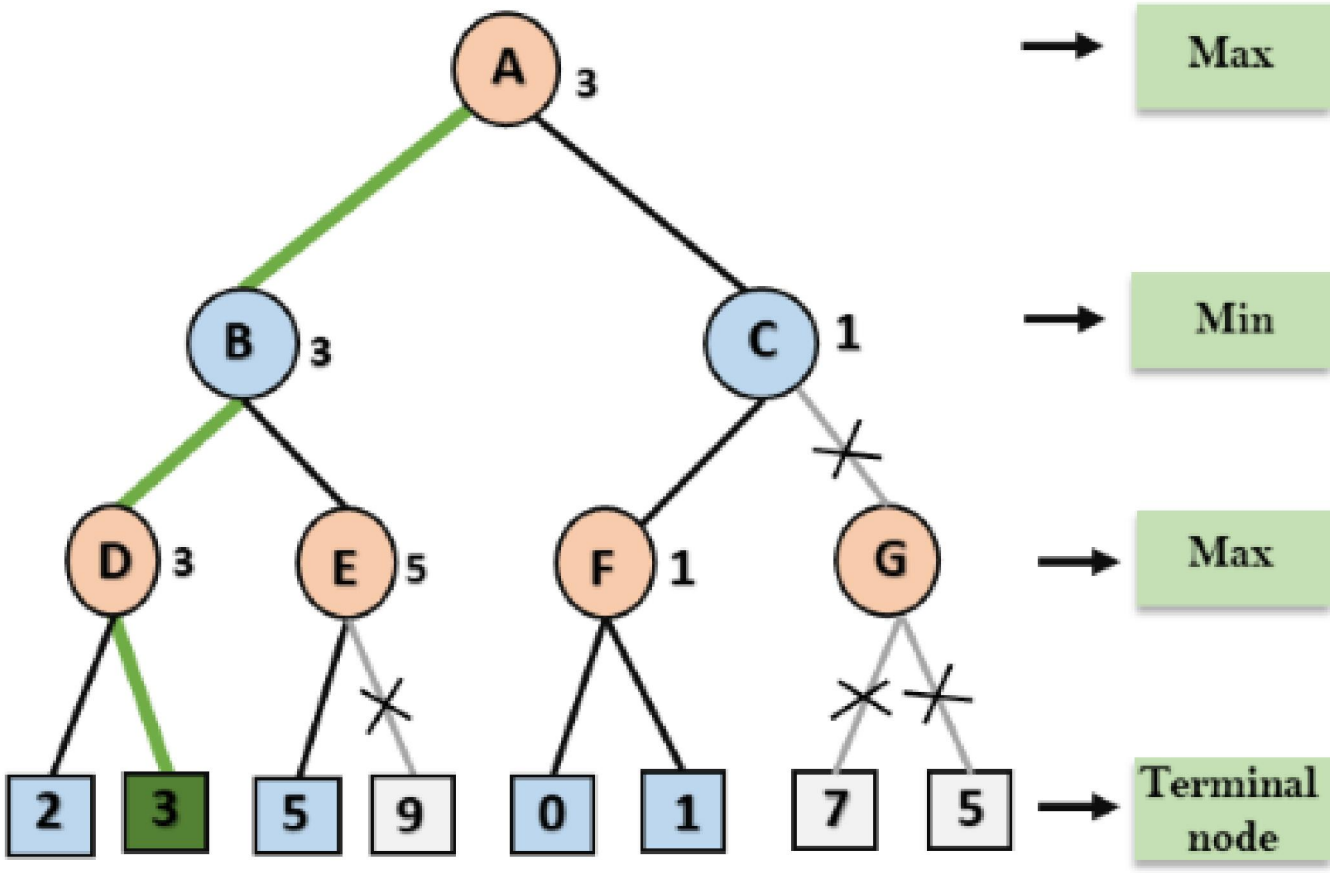
Step 7:

- Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1.
- Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



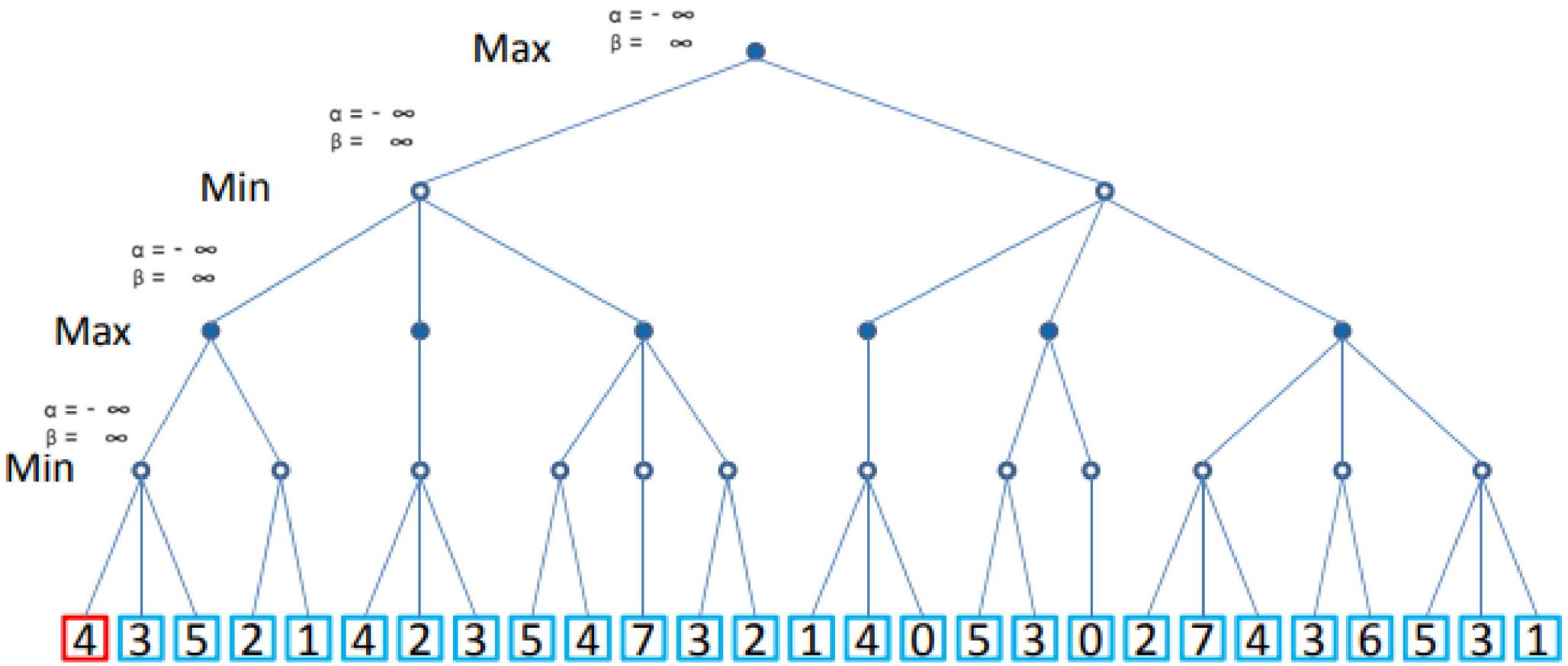
Step 8:

- C now returns the value of 1 to A here the best value for A is max (3, 1) = 3.
- Following is the final game tree which is showing the nodes which are computed and nodes which has never computed.
- Hence the optimal value for the maximizer is 3 for this example.

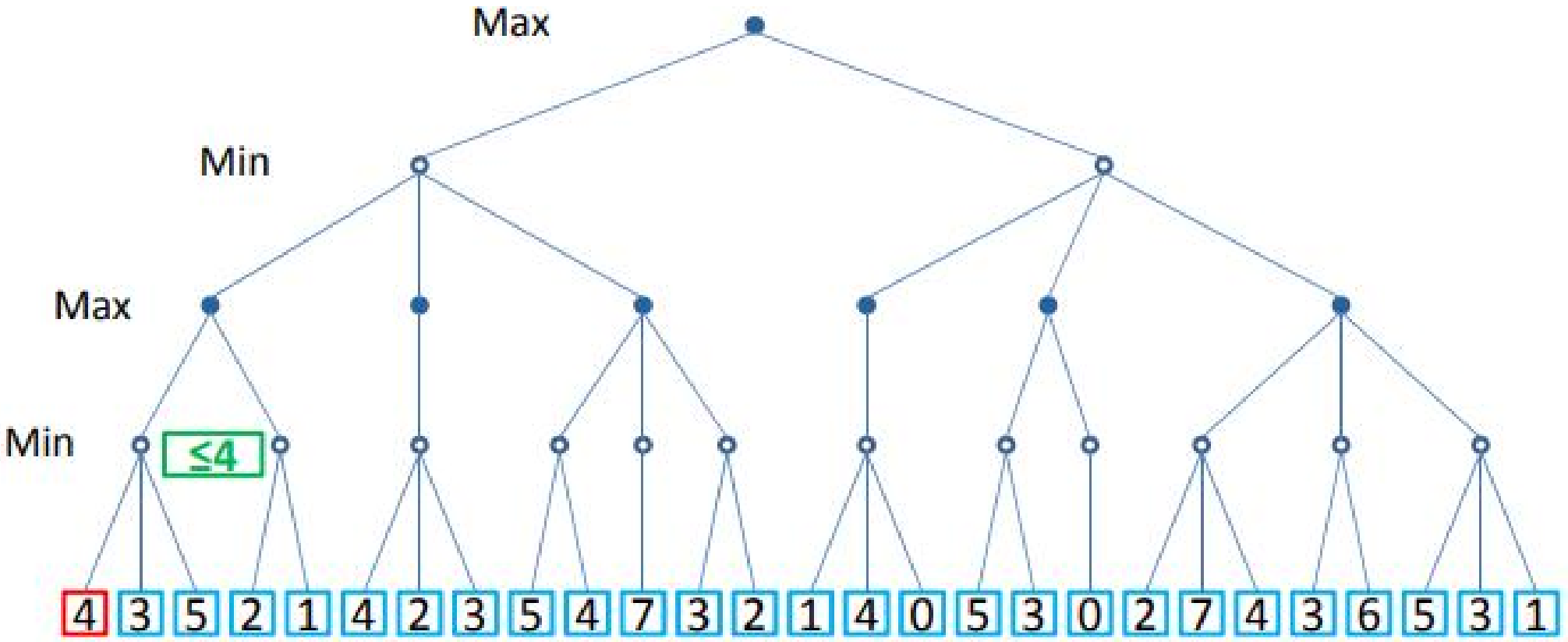


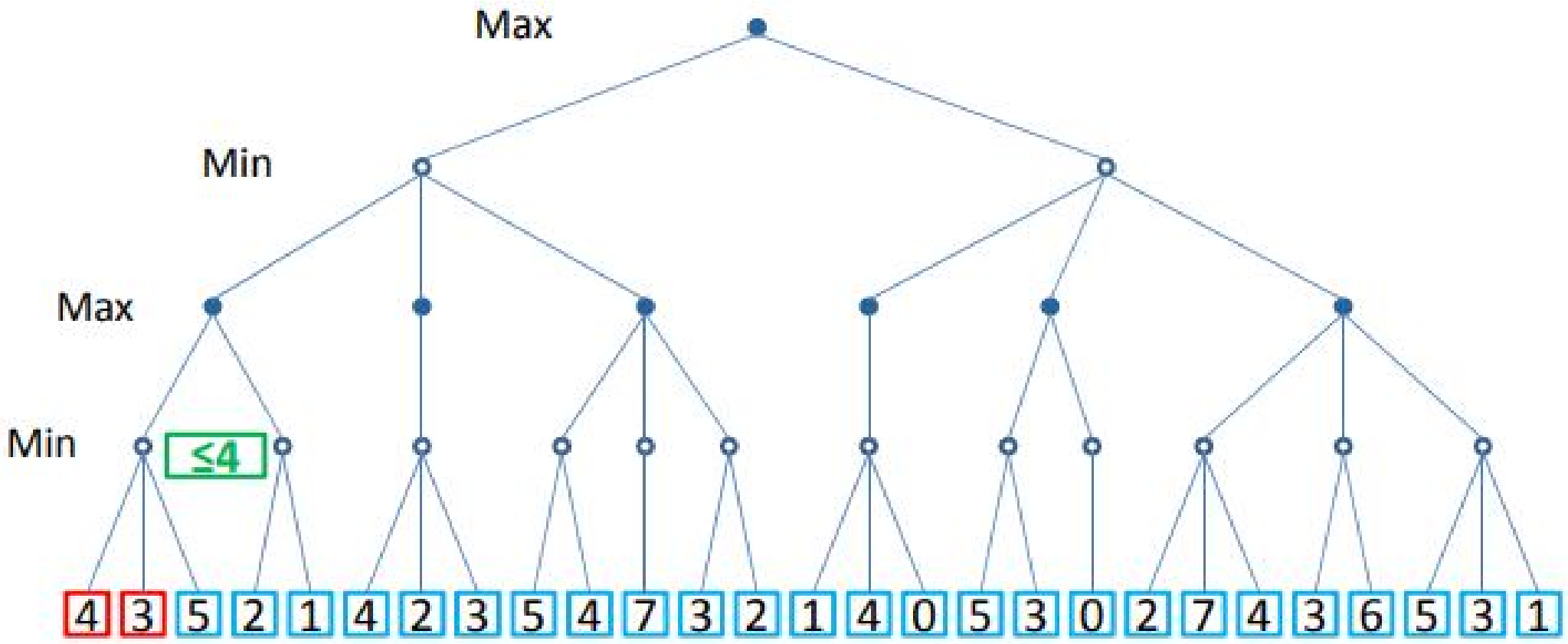
Exercise 2: Alpha-Beta pruning

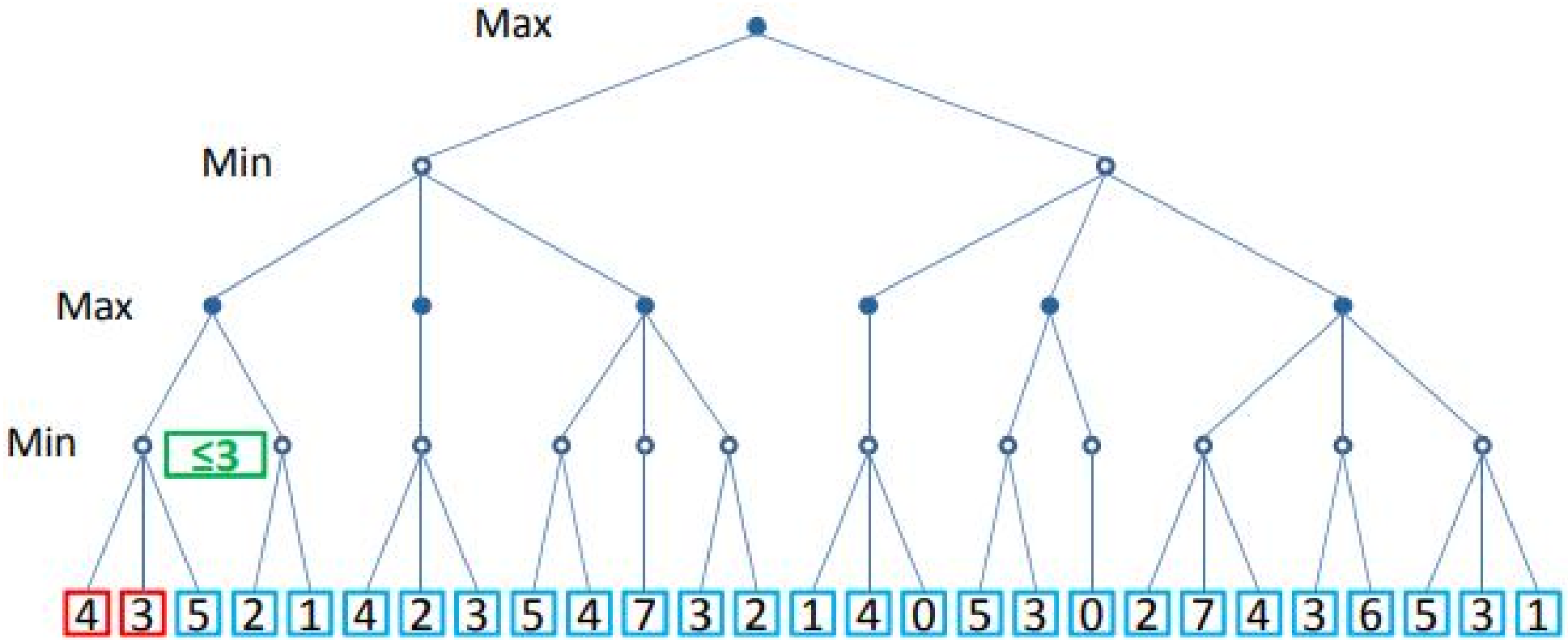
https://dtai.cs.kuleuven.be

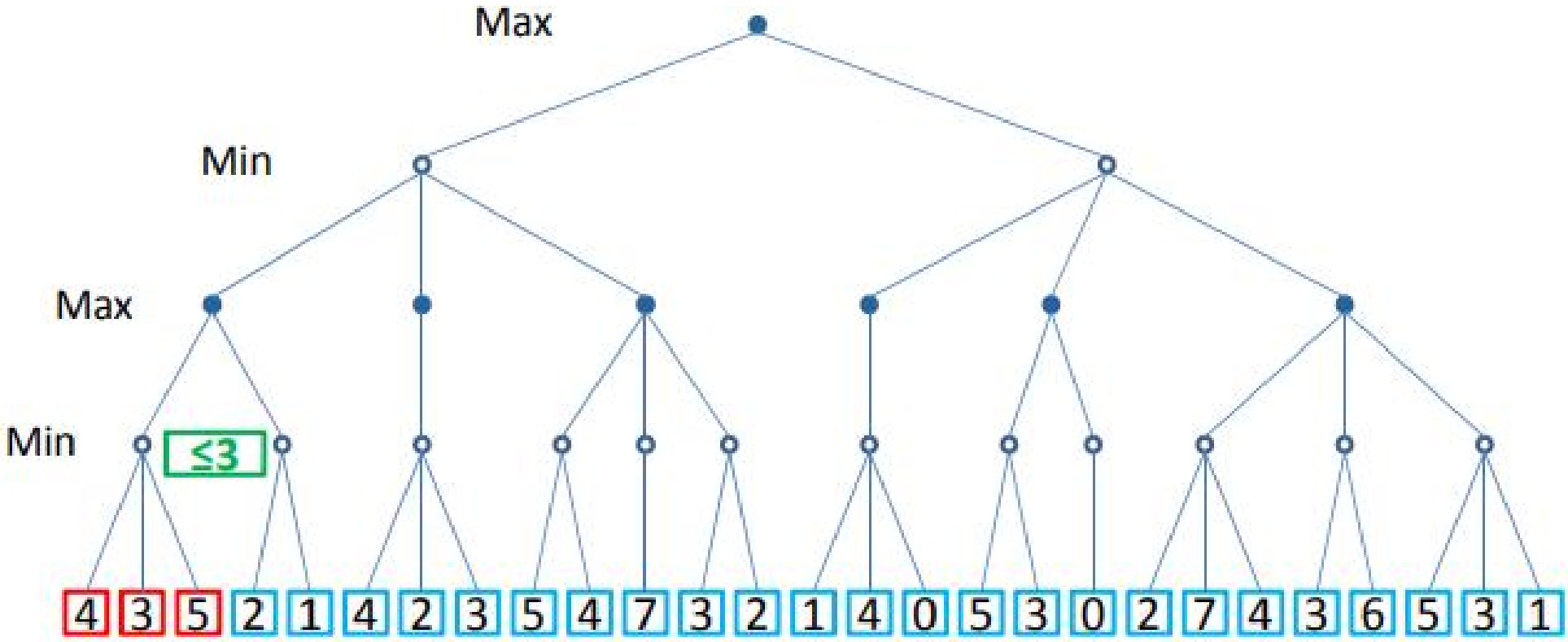


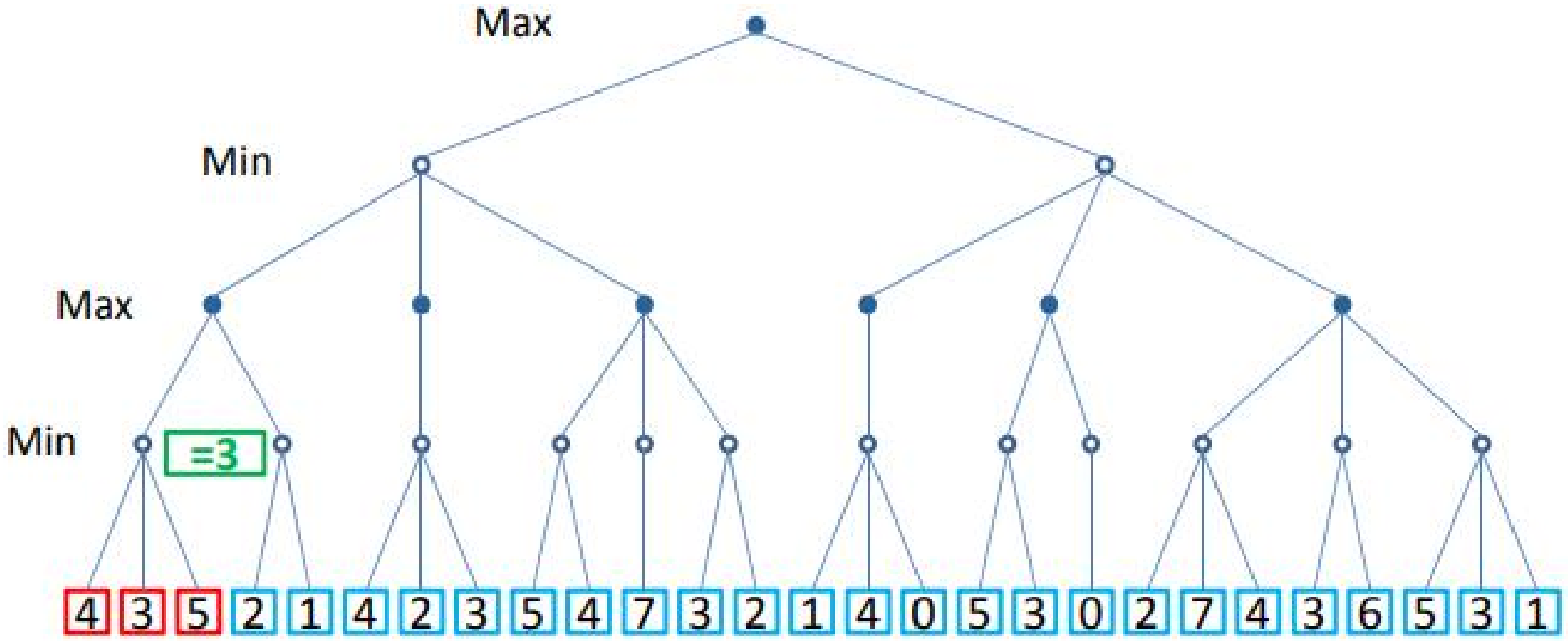
α-nodes: At MIN-nodes has minimum infinity number



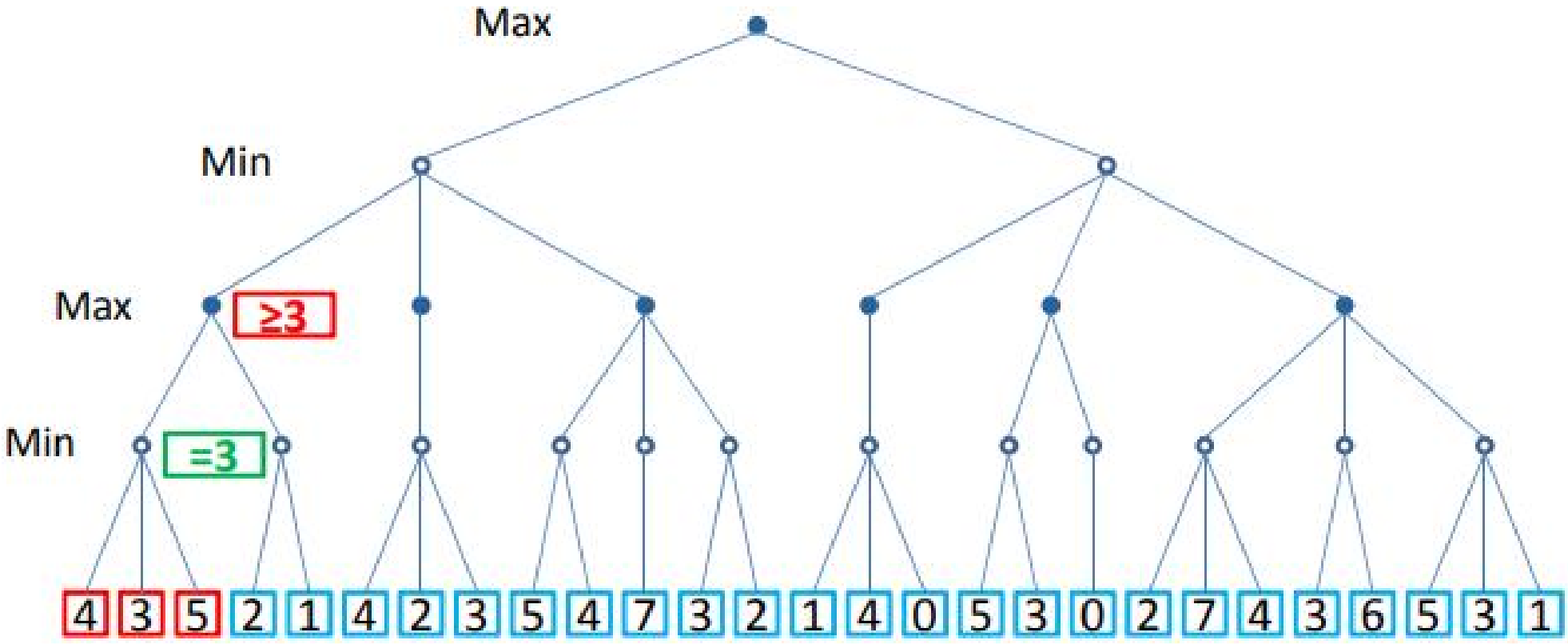

Max
Min
Max
Min
≤4
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

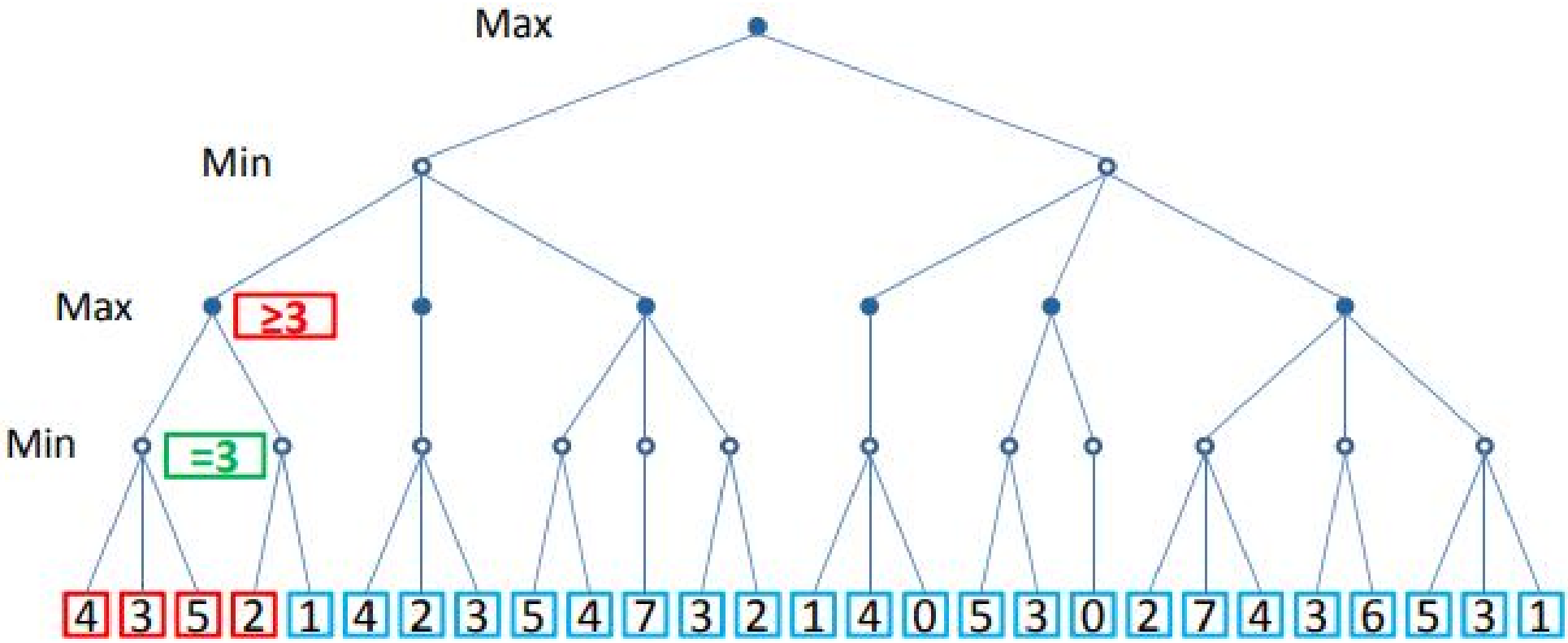

Max
Min
Max
Min
≤3
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

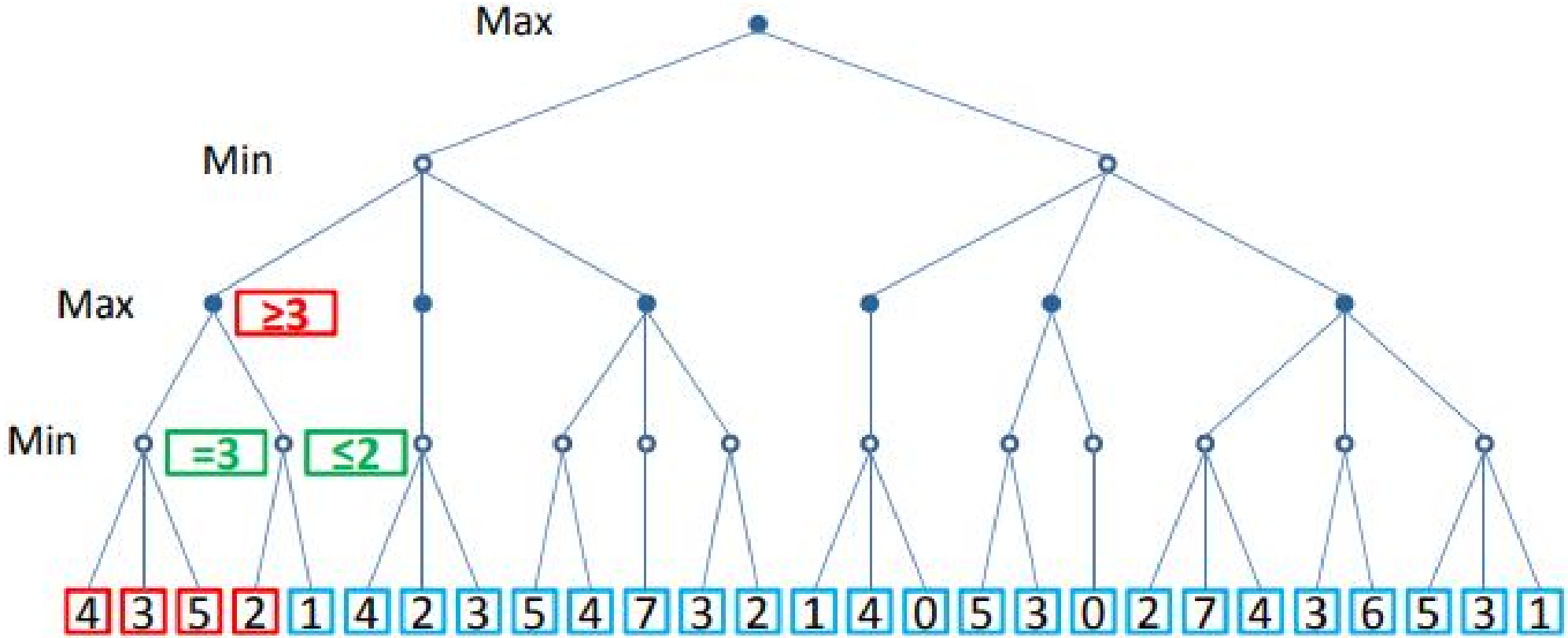

Max
Min
Max
Min
≤3
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1


Max
Min
Max
Min
=3
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

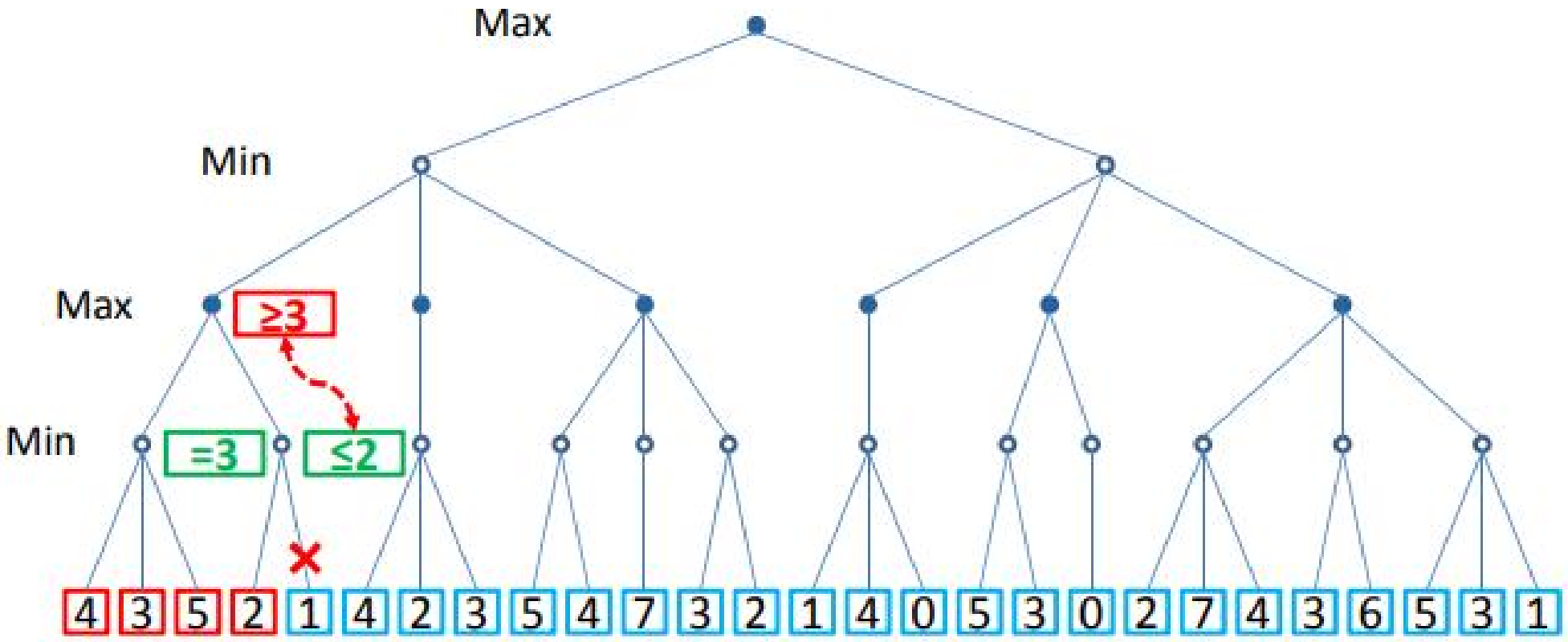
β-nodes: At MAX-nodes has maximum infinity number

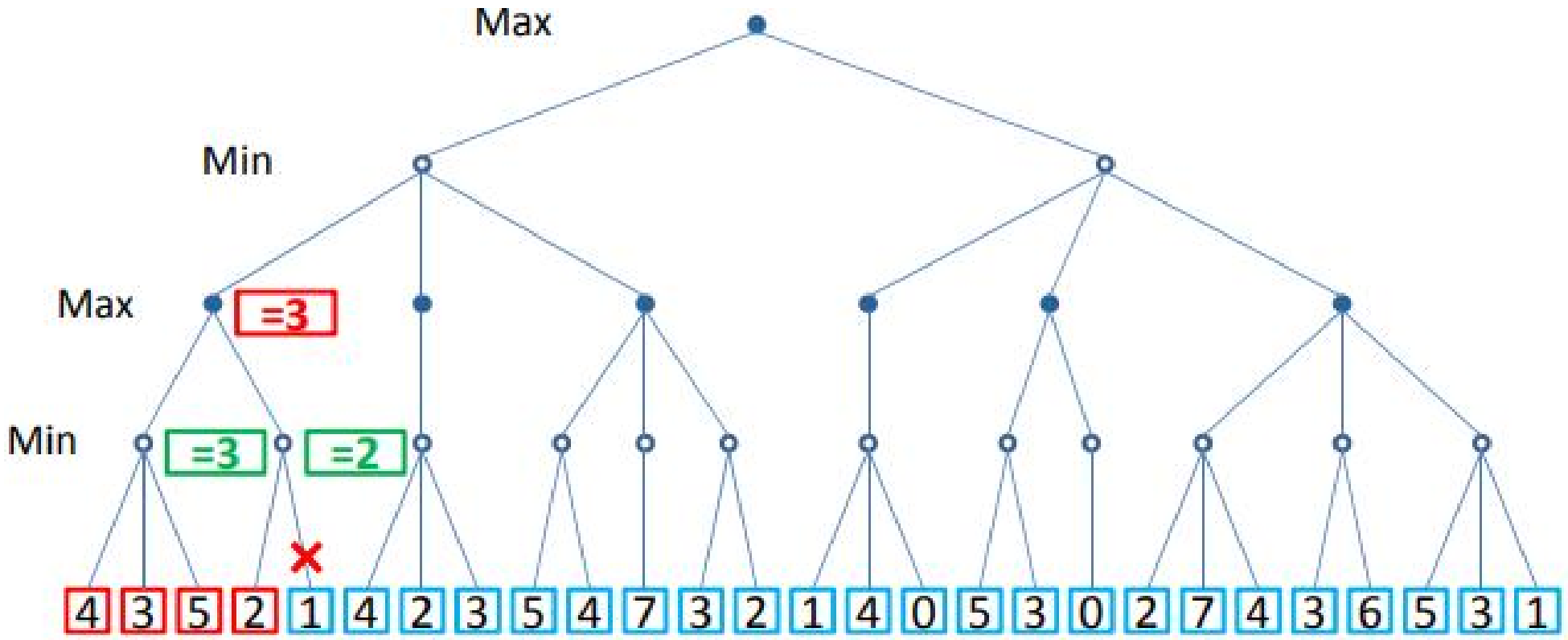


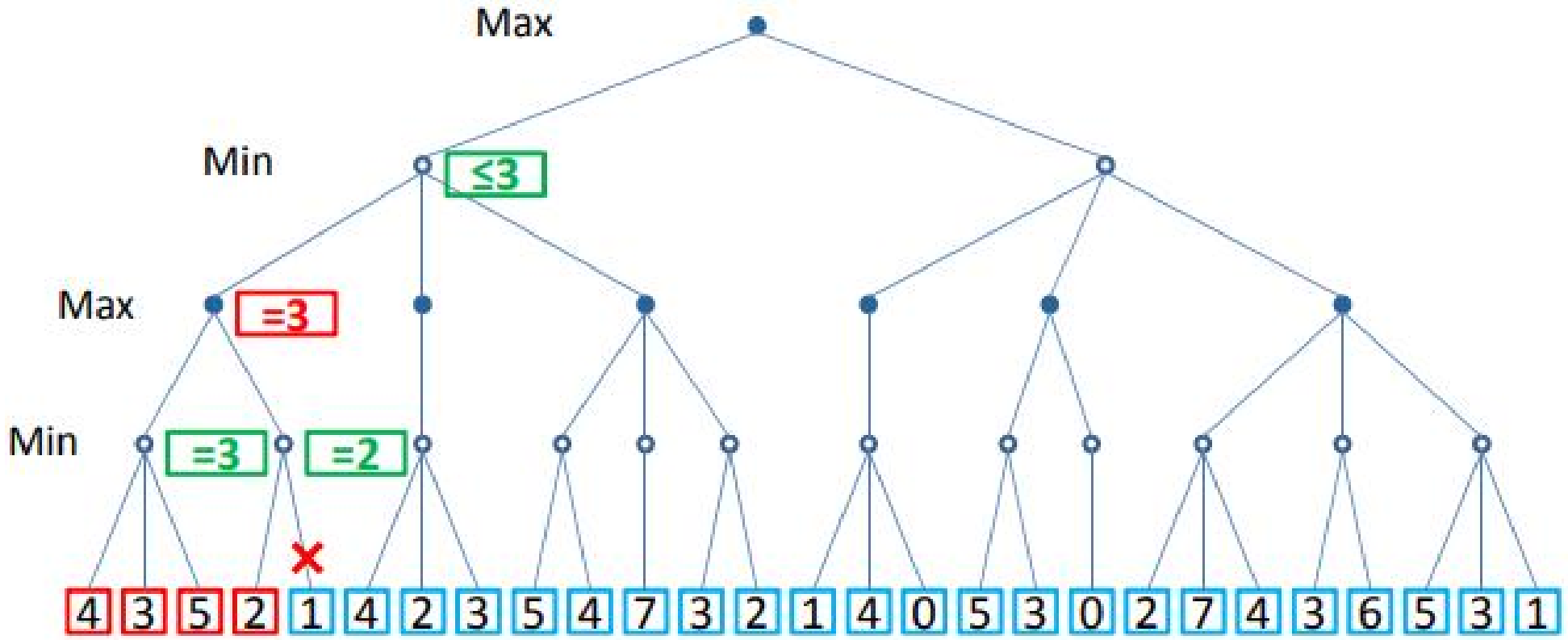

Max
Min
Max
≥3
Min
=3
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

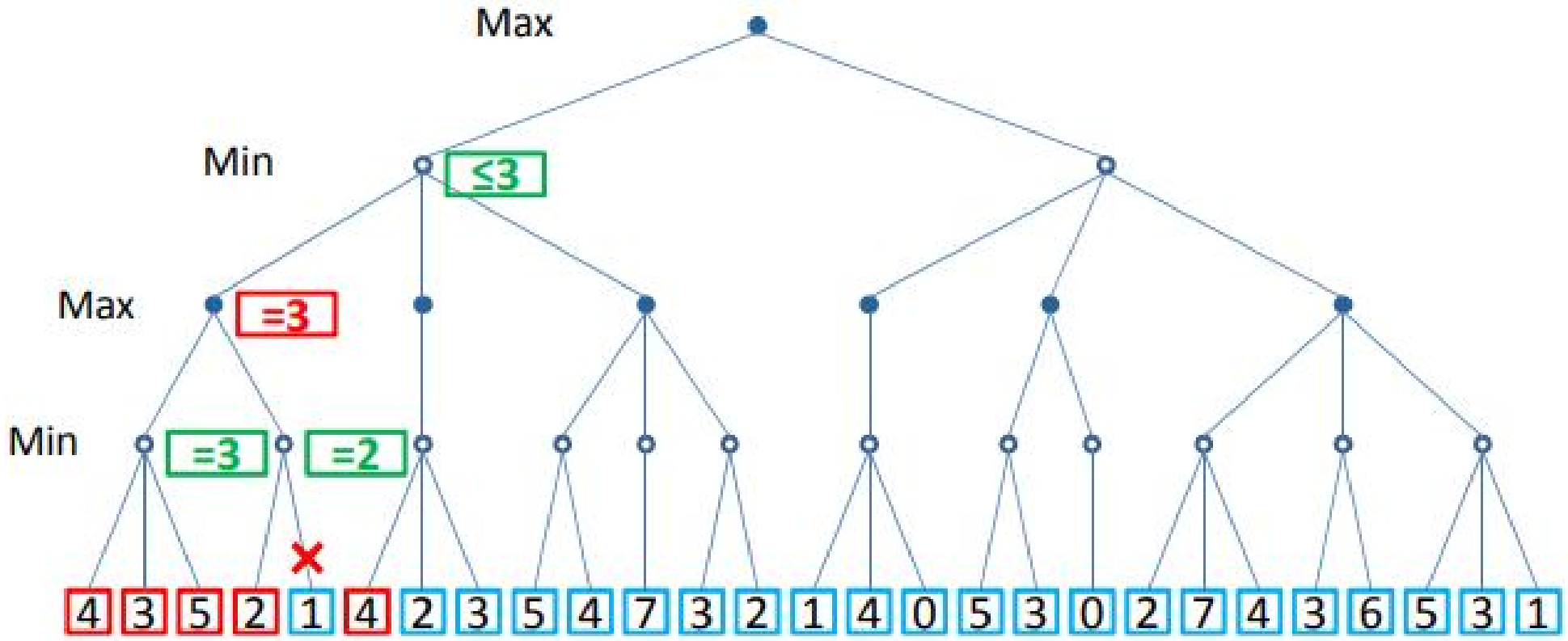

Max
Min
Max
≥3
Min
=3
≤2
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

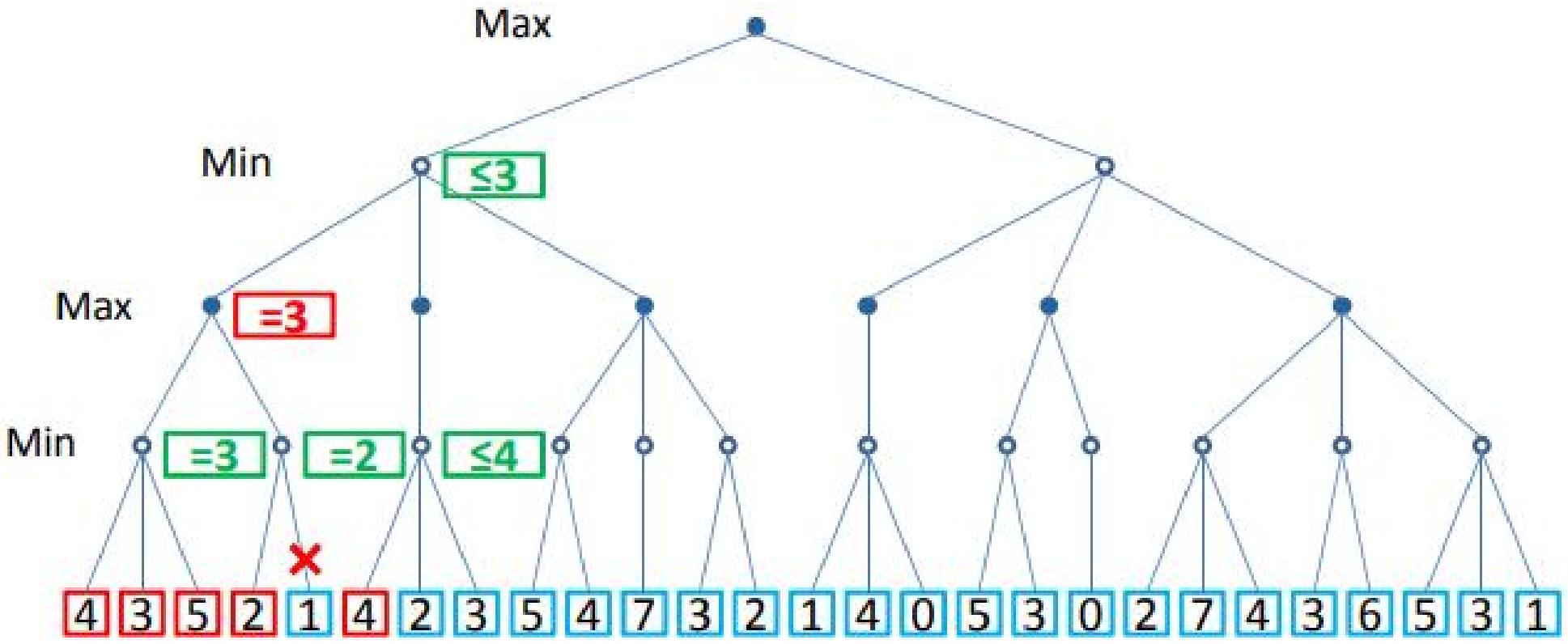
Prune: Parent α-nodes ≥ Child β-nodes

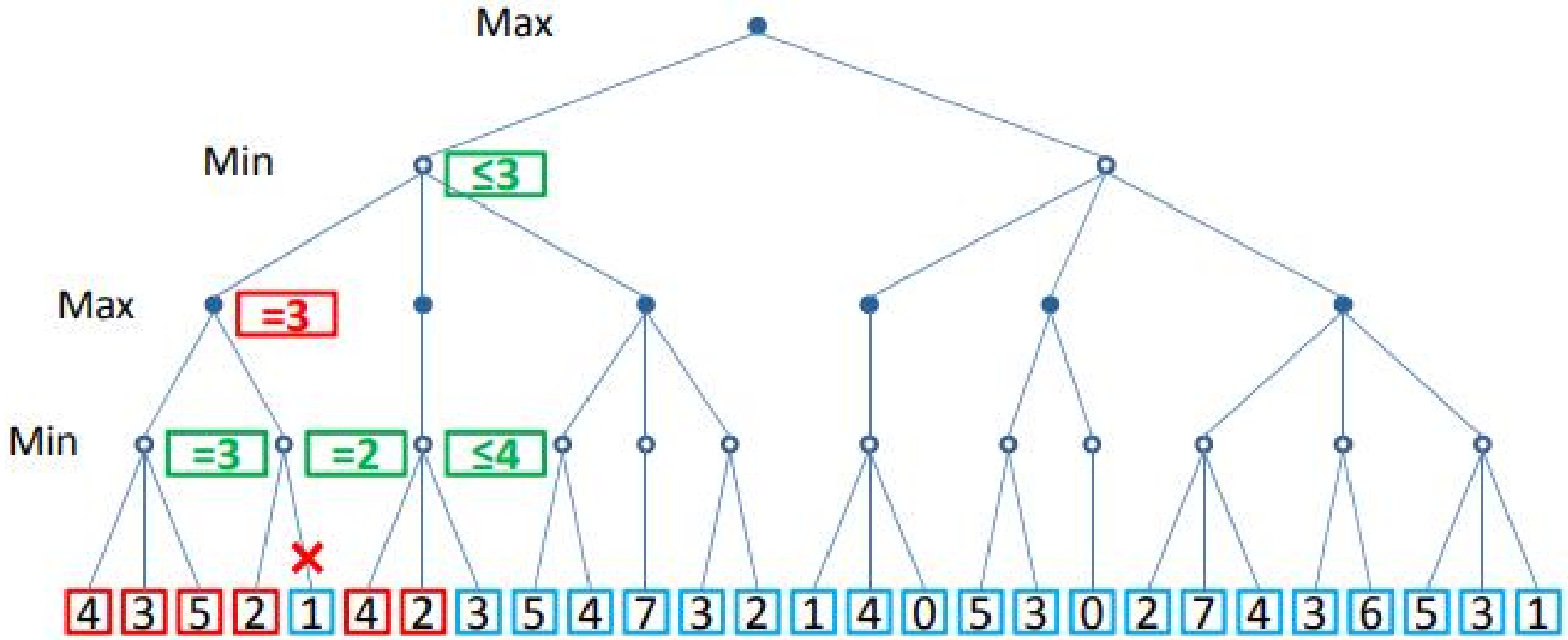


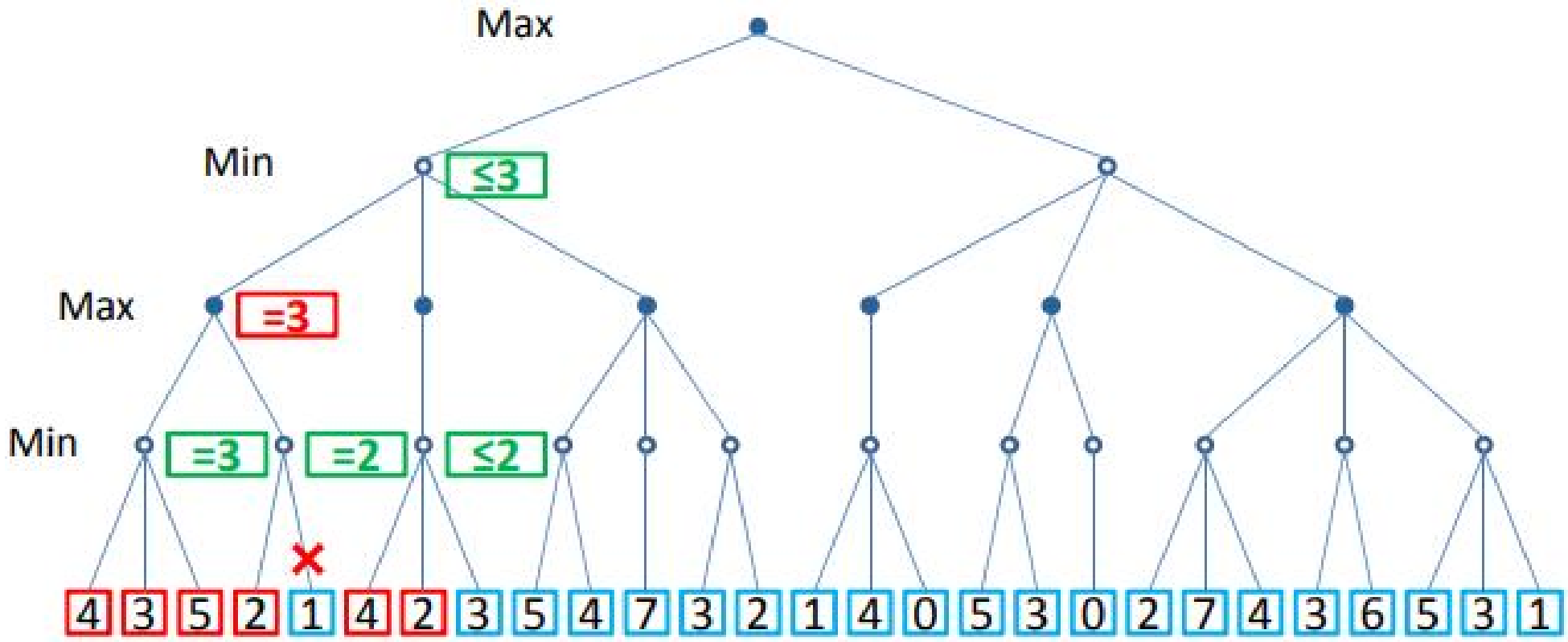

Max
Min
Max
=3
Min
=3
=2
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

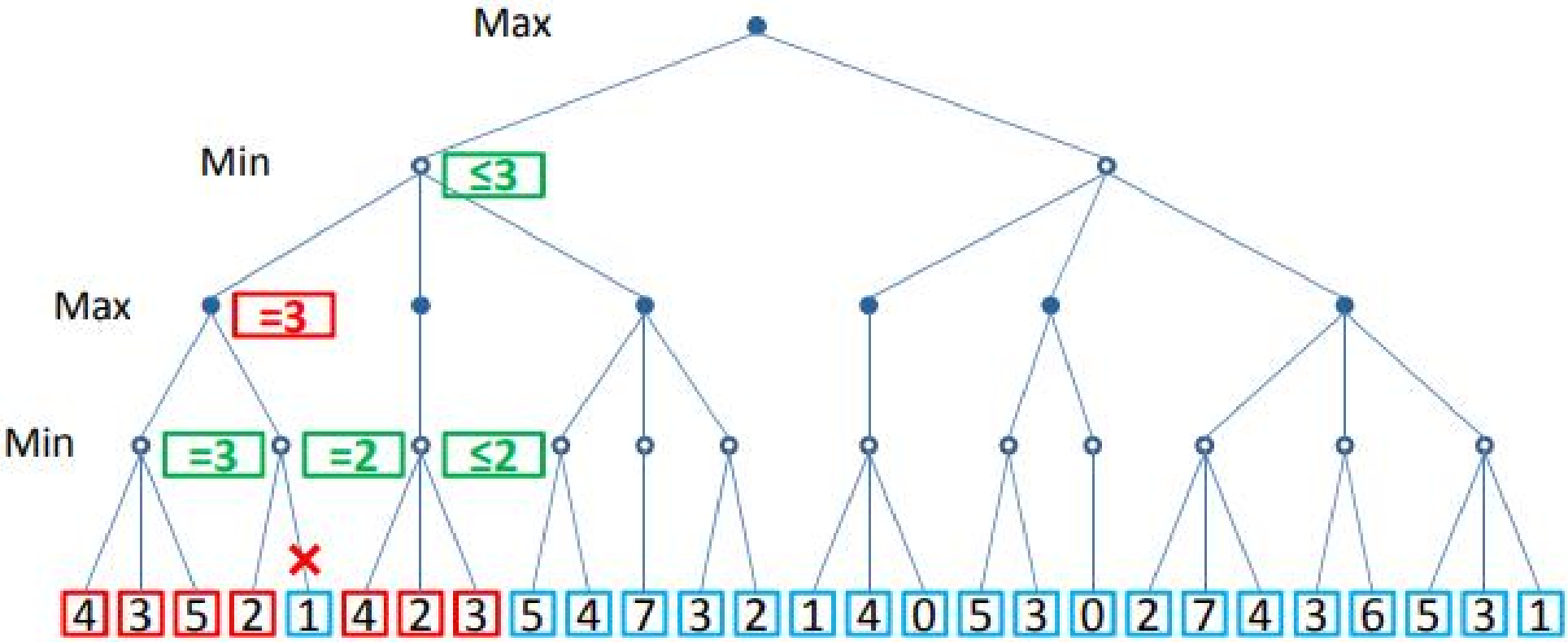

Max
Min
≤3
Max
=3
Min
=3
=2
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

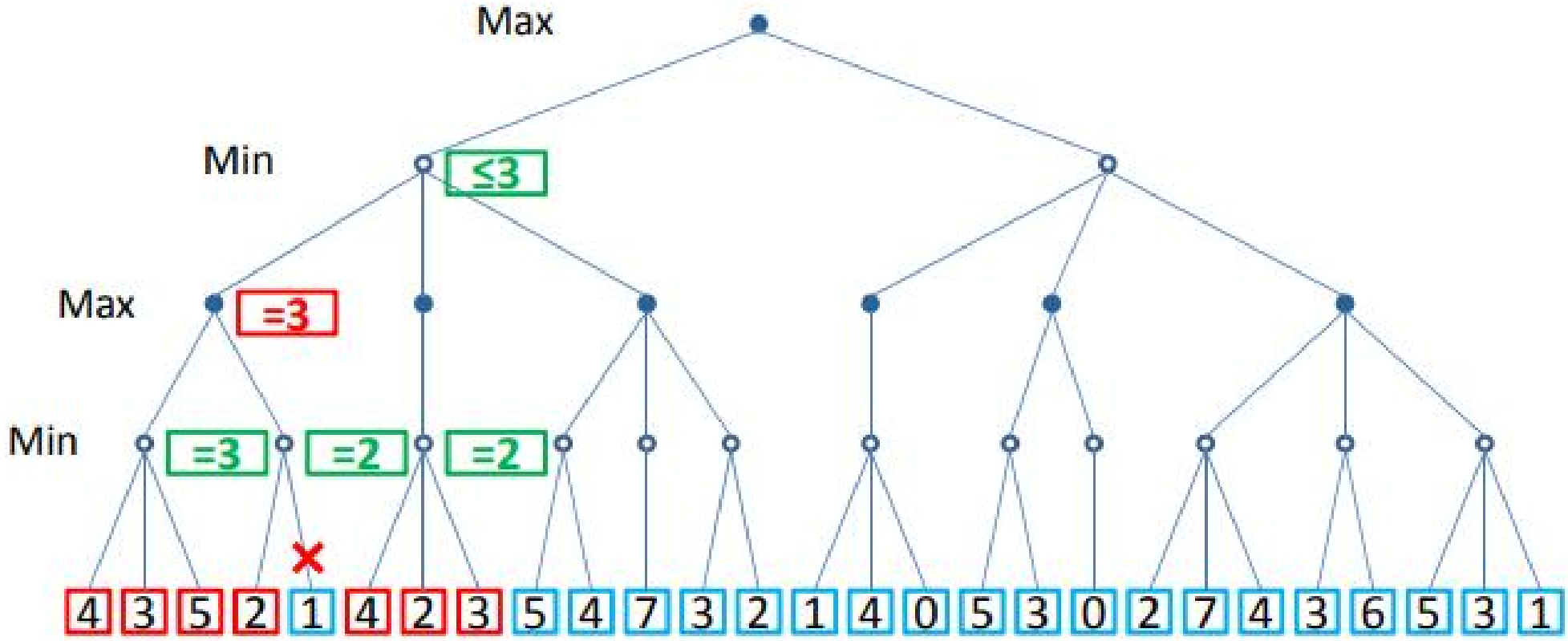

Max
Min
≤3
Max
=3
Min
=3
=2
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

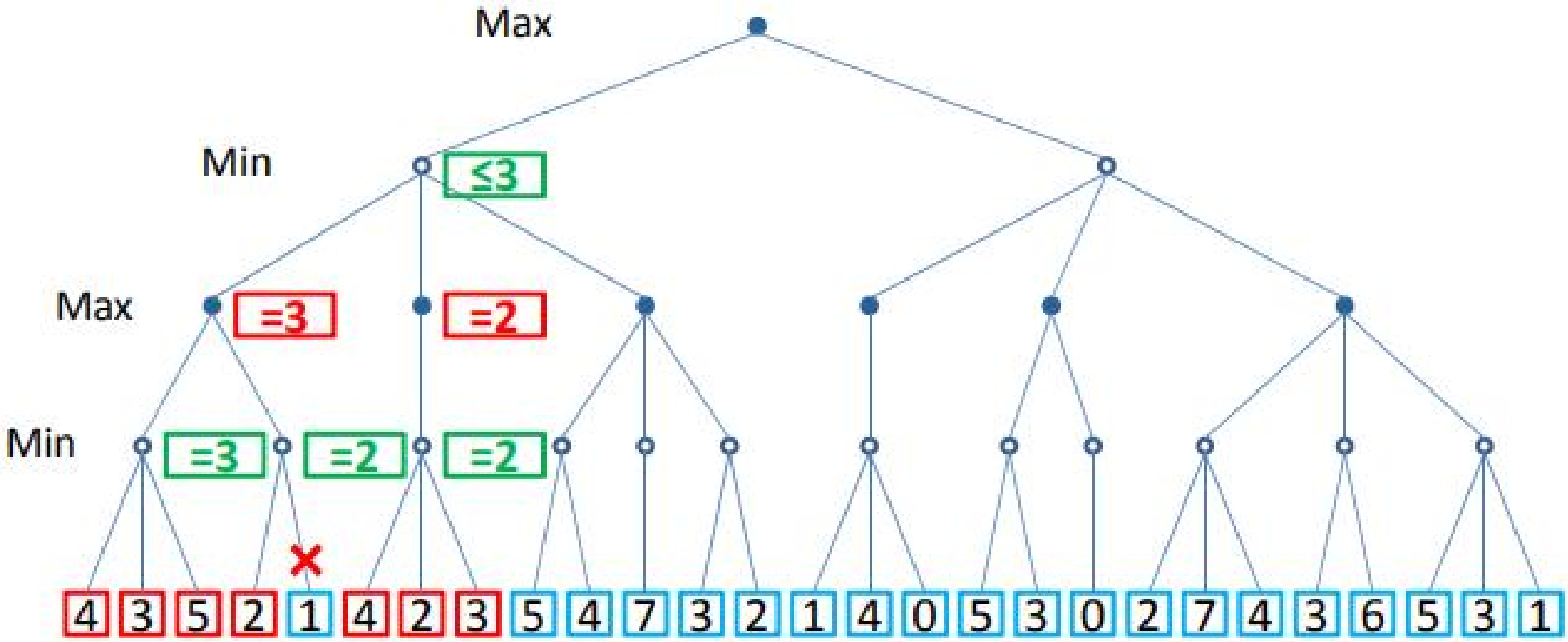

Max
Min
≤3
Max
=3
Min
=3
=2
≤4
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

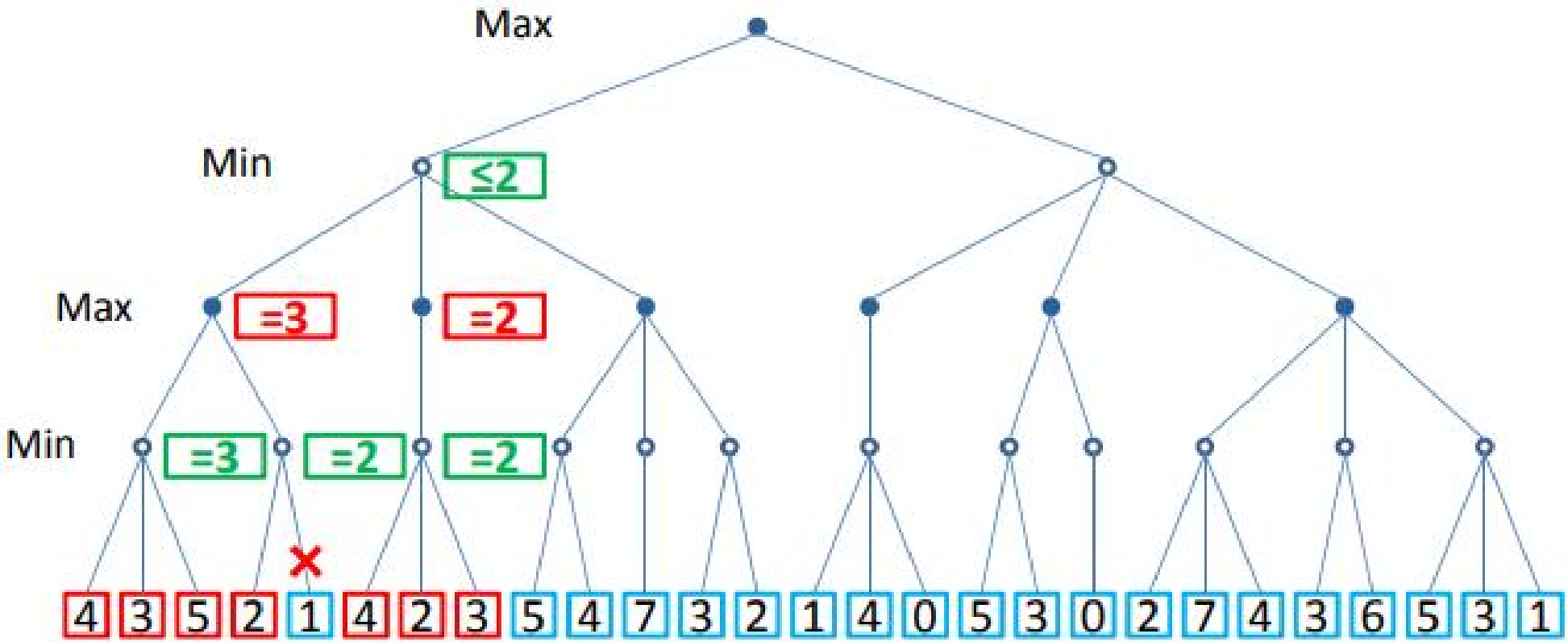

Max
Min
≤3
Max
=3
Min
=3
=2
≤4
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

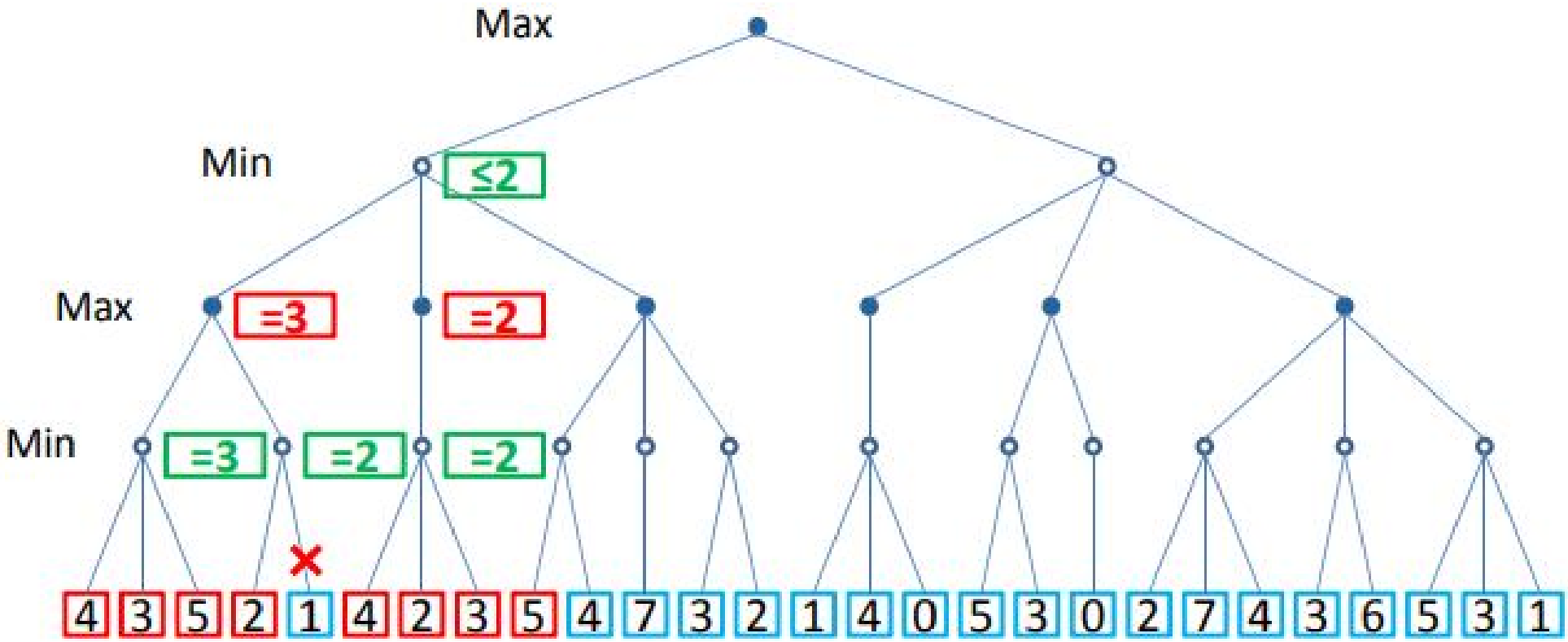

Max
Min
≤3
Max
=3
Min
=3
=2
≤2
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

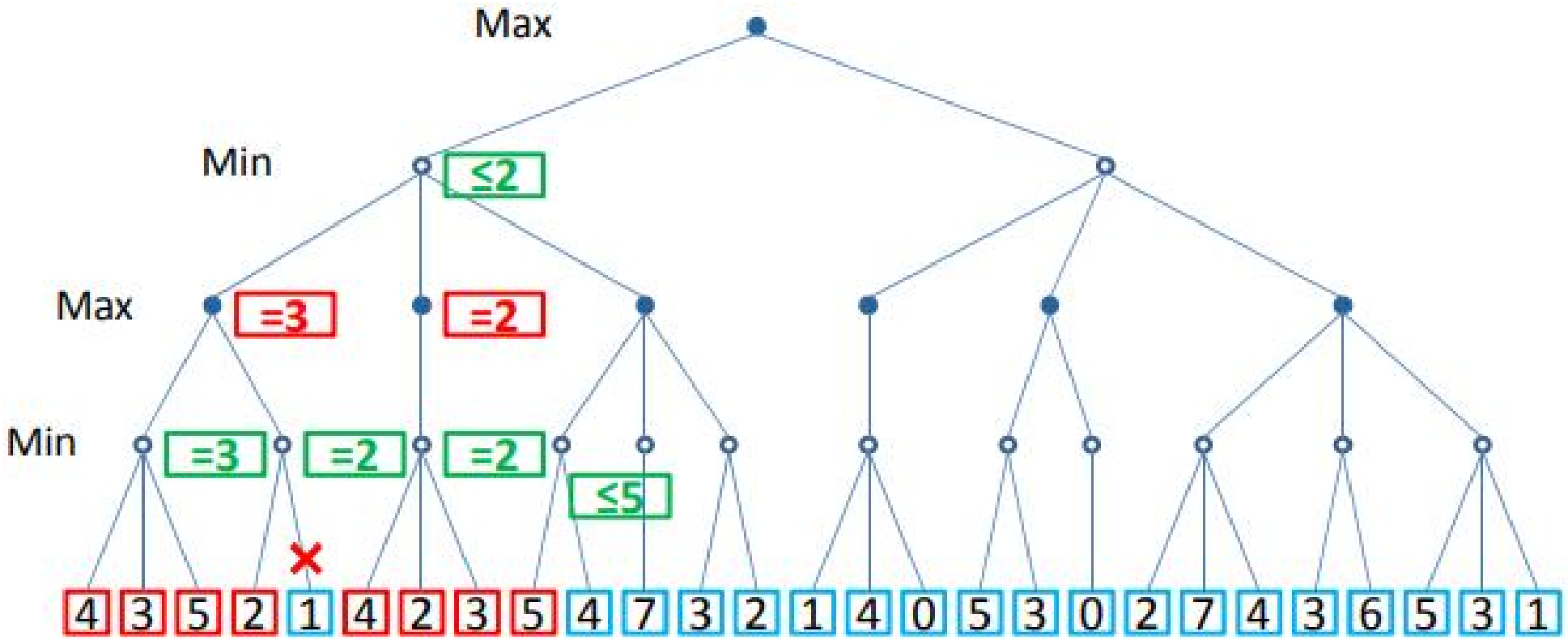

Max
Min
≤3
Max
=3
Min
=3
=2
≤2
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

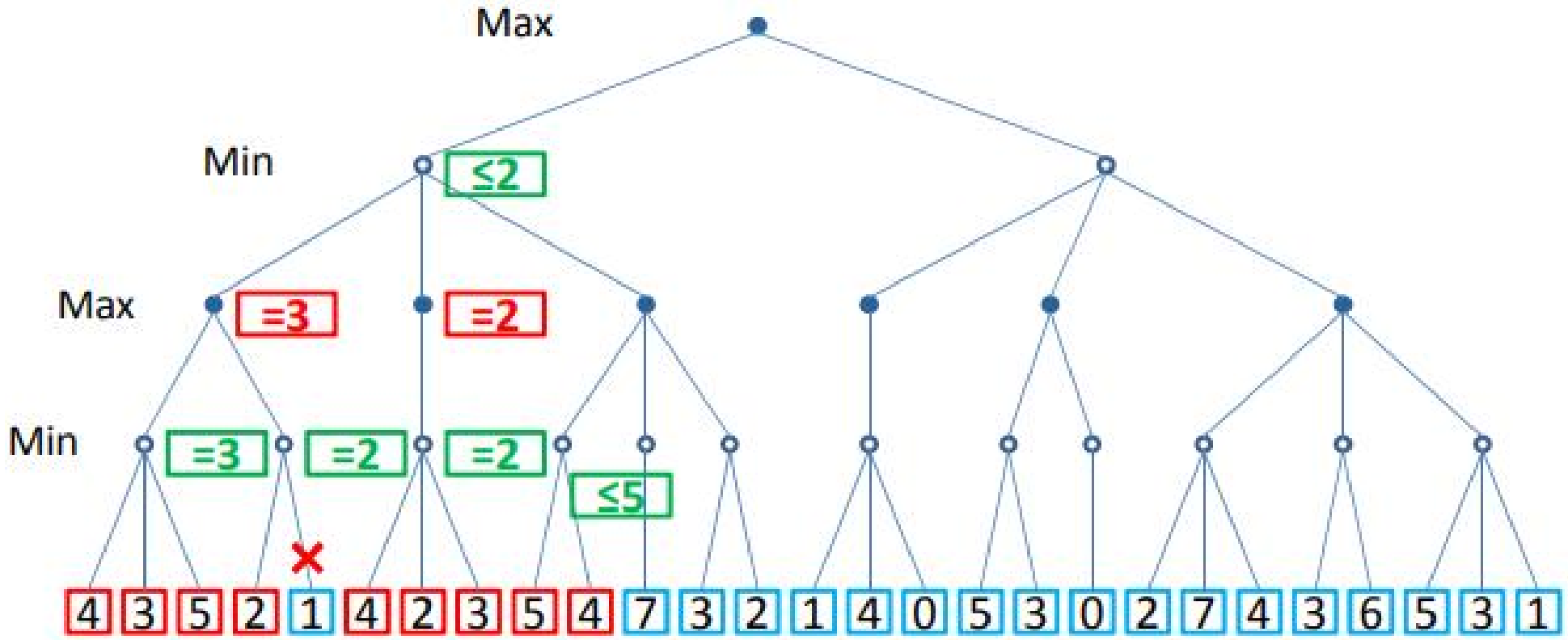

Max
Min
≤3
Max
=3
Min
=3
=2
=2
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

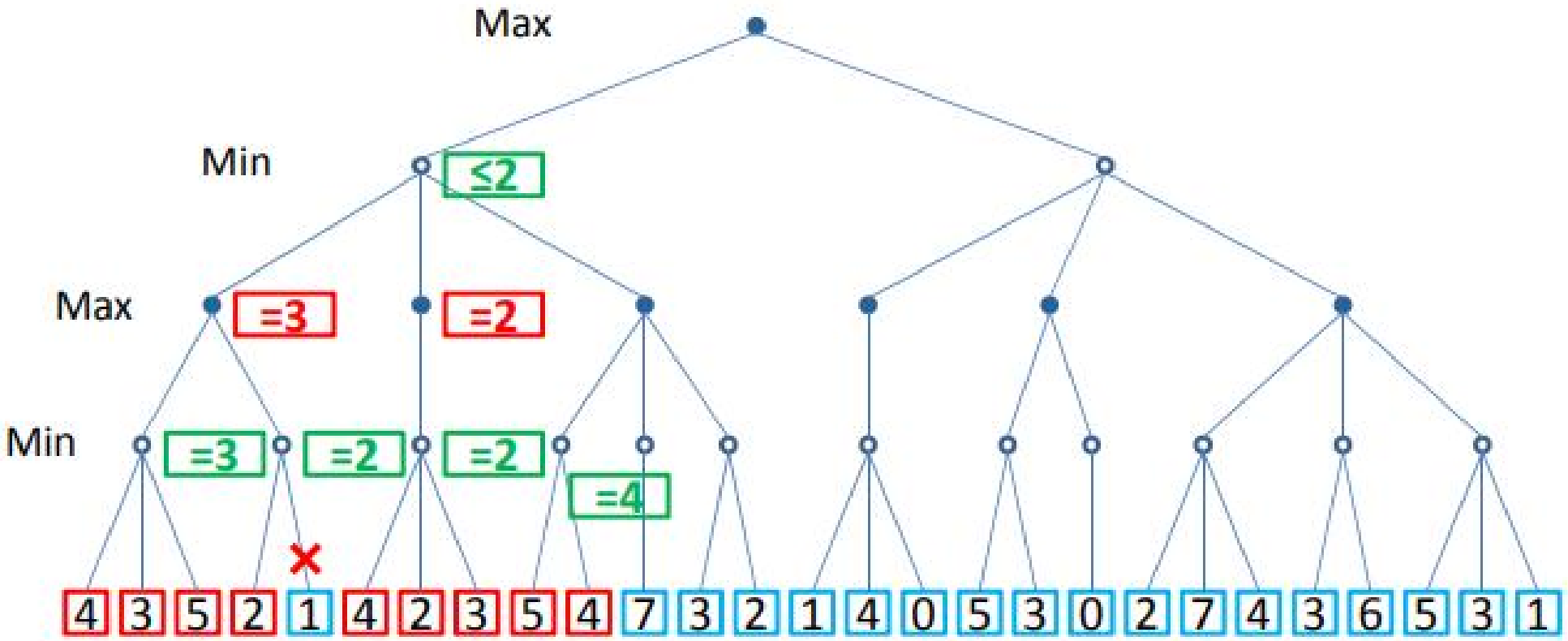

Max
Min
≤3
Max
=3
=2
Min
=3
=2
=2
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

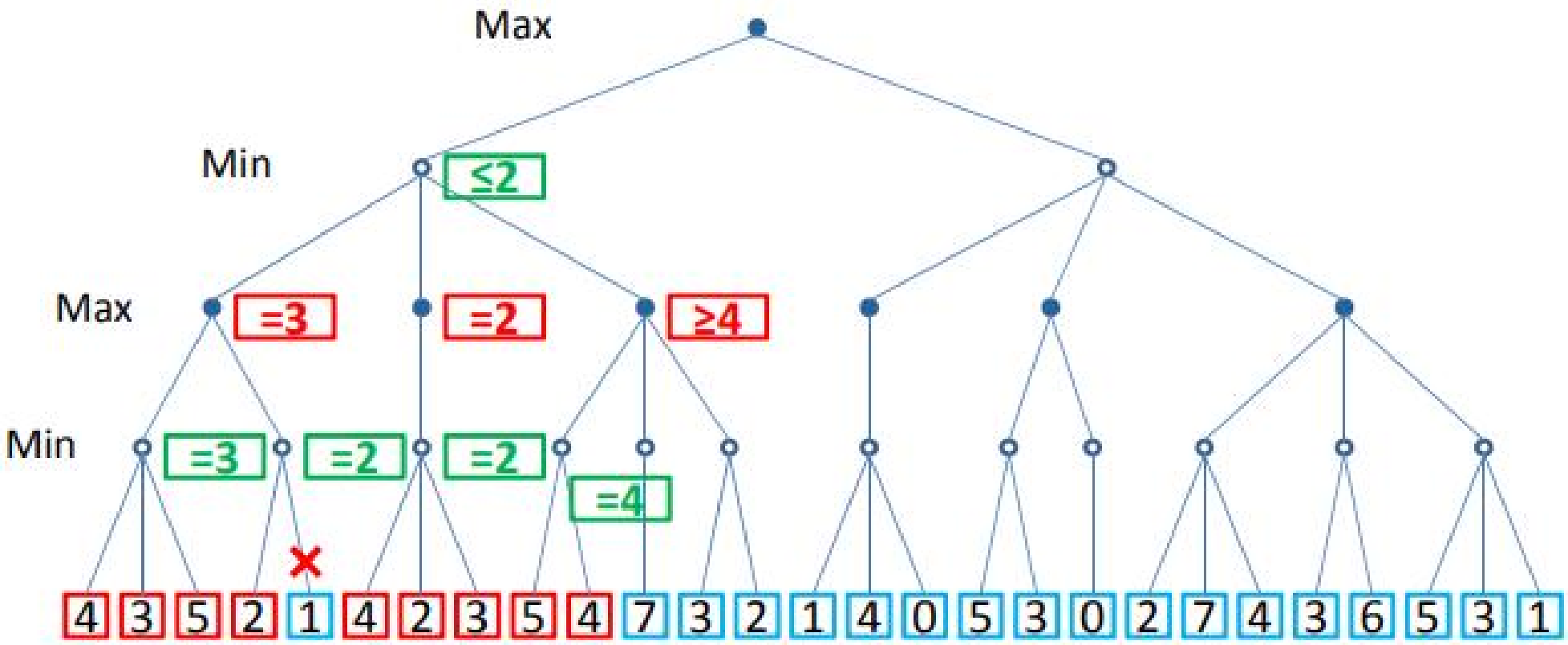

Max
Min
≤2
Max
=3
=2
Min
=3
=2
=2
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1


Max
Min
≤2
Max
=3
=2
Min
=3
=2
=2
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

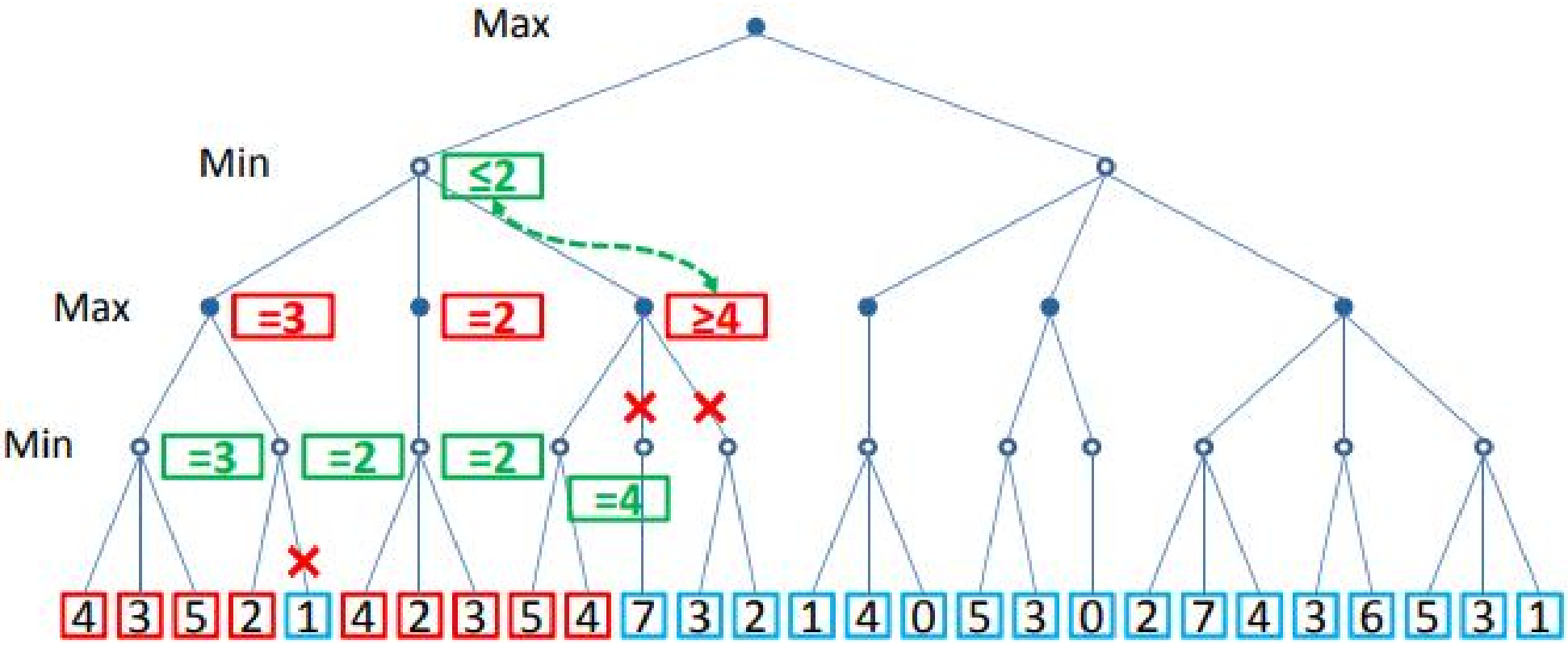

Max
Min
≤2
Max
=3
=2
Min
=3
=2
=2
≤5
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

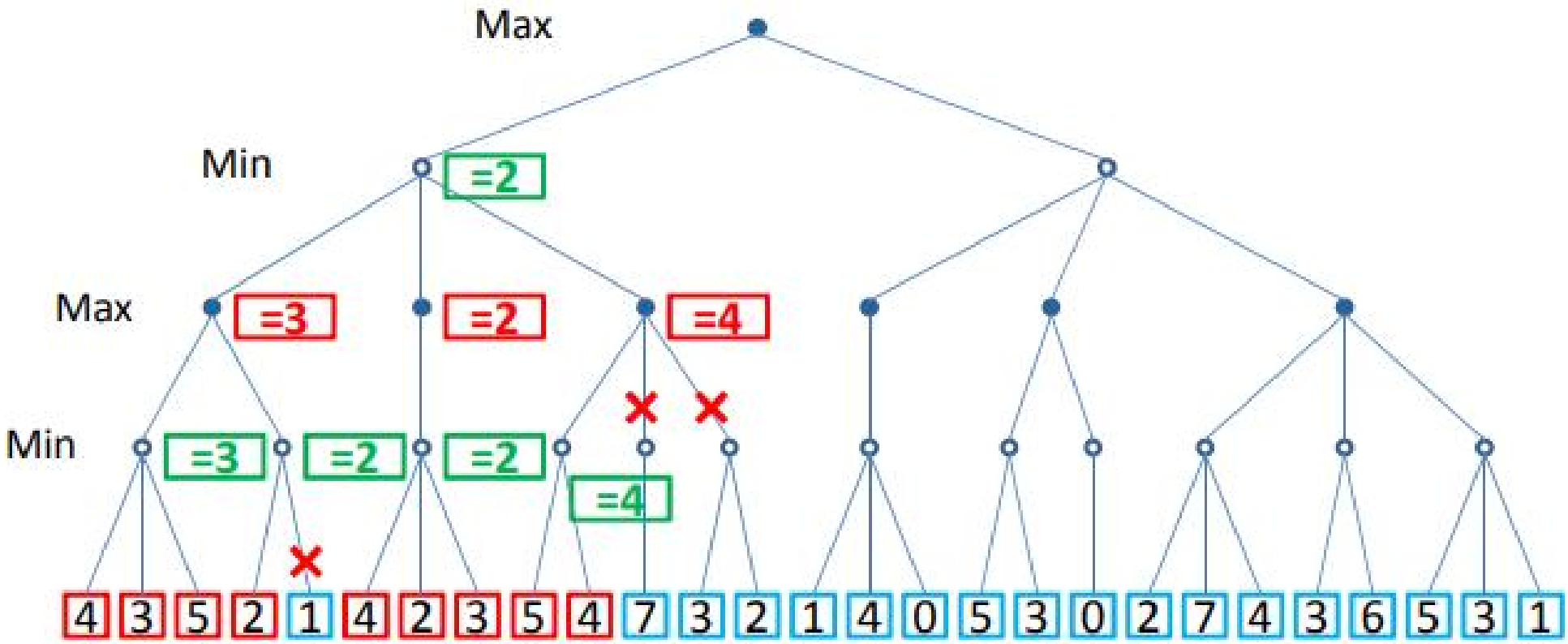

Max
Min
≤2
Max
=3
=2
Min
=3
=2
=2
≤5
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

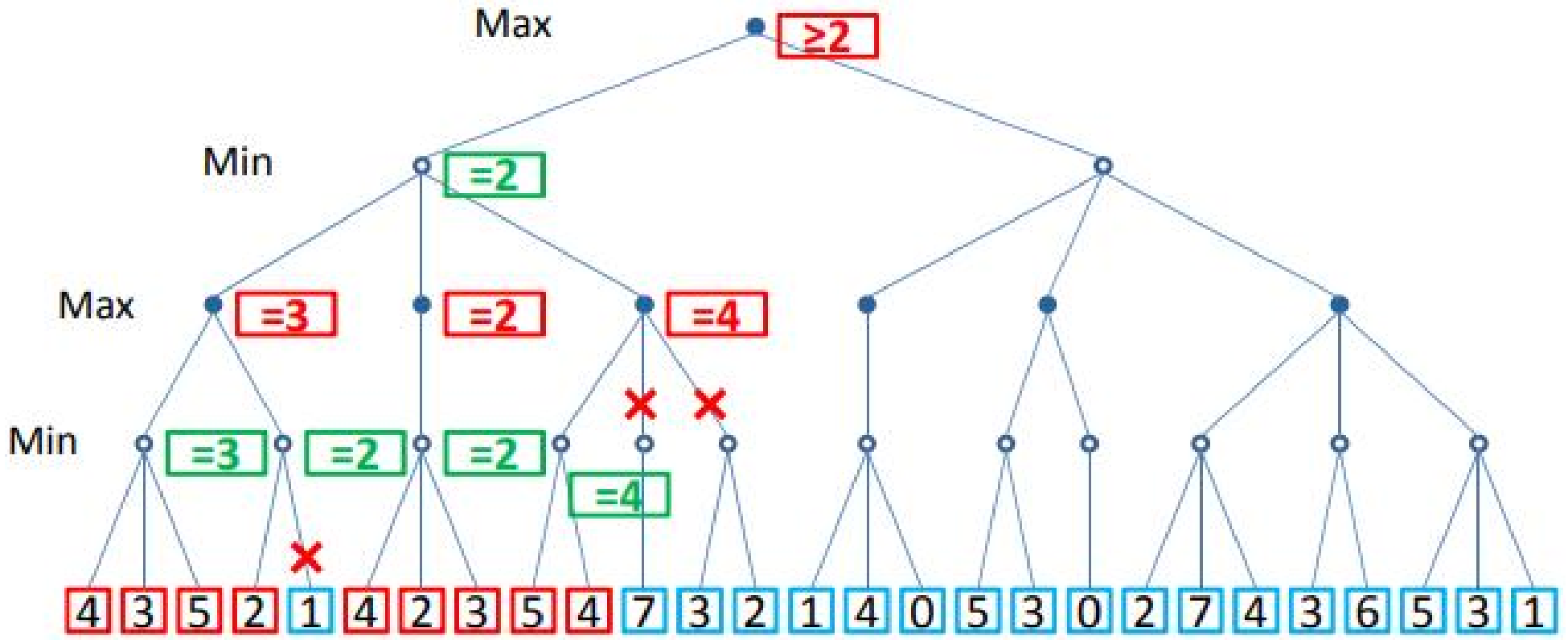

Max
Min
≤2
Max
=3
=2
Min
=3
=2
=2
=4
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

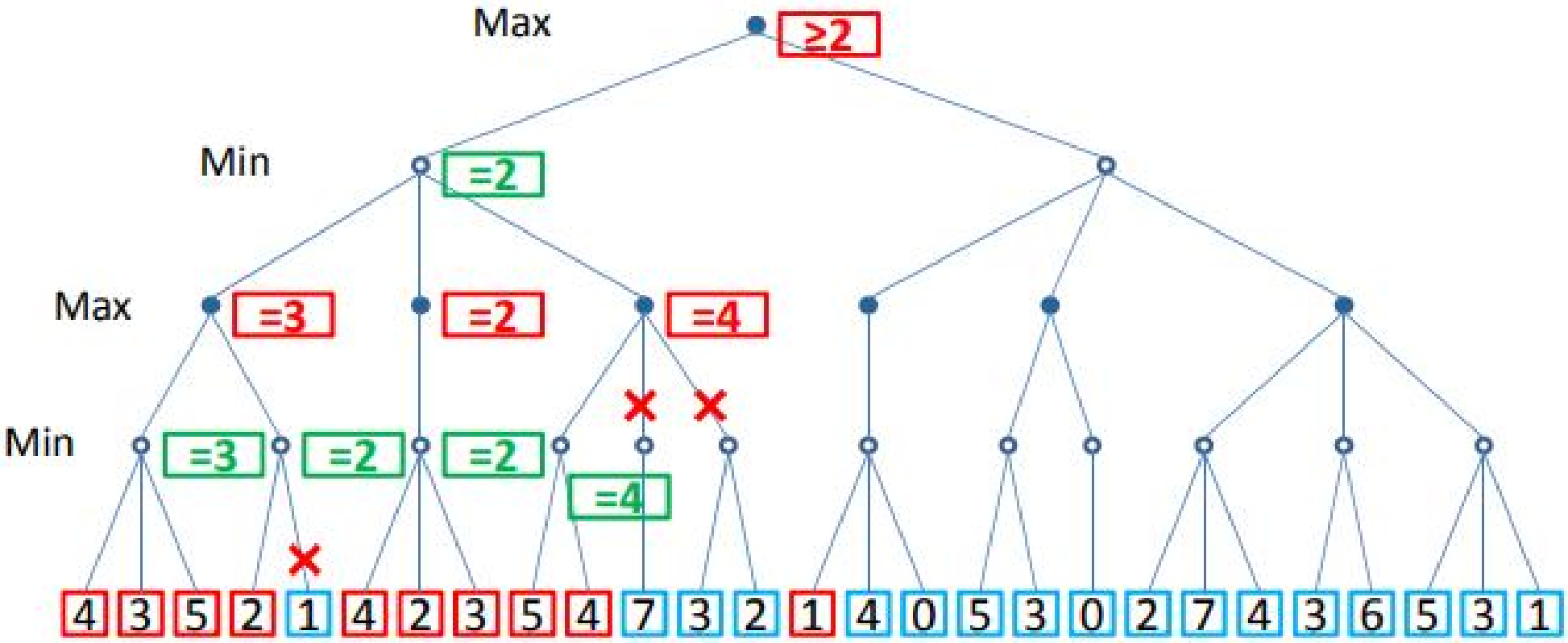

Max
Min
≤2
Max
=3
=2
≥4
Min
=3
=2
=2
=4
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

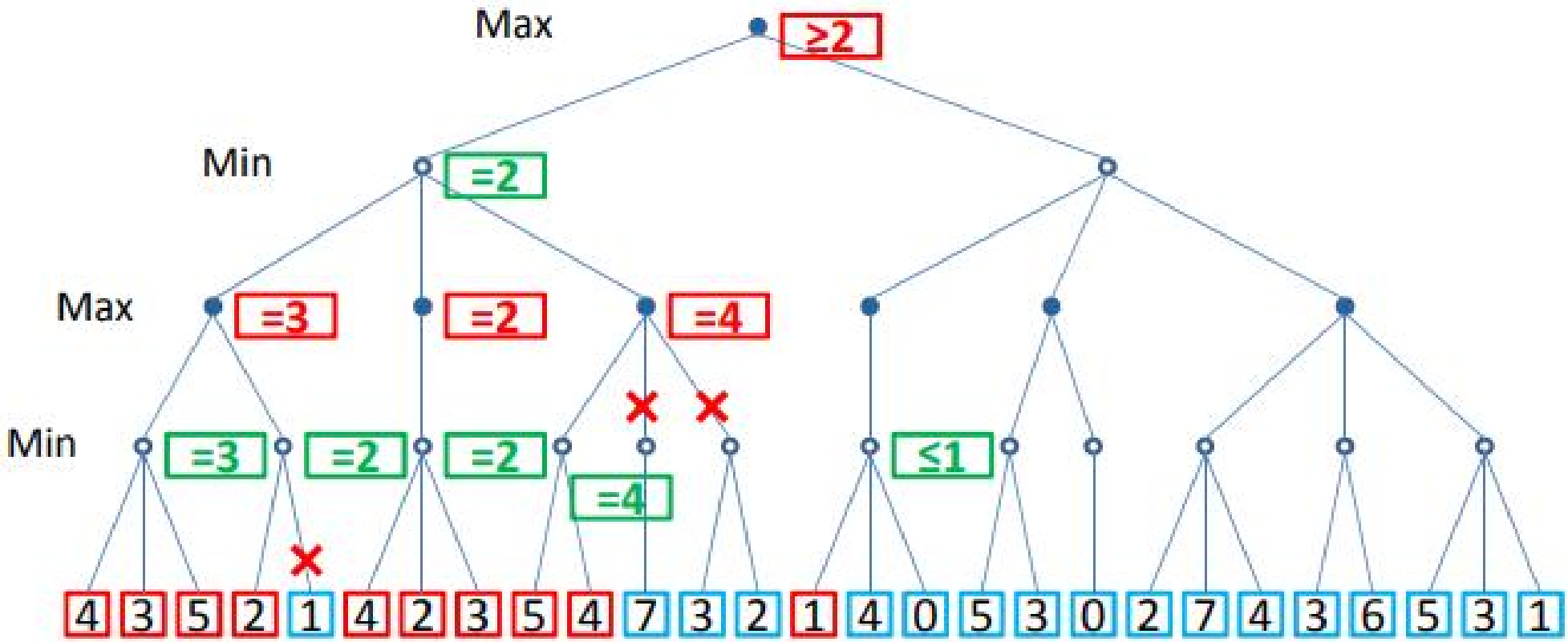
Prune: α-node ≥ β-node



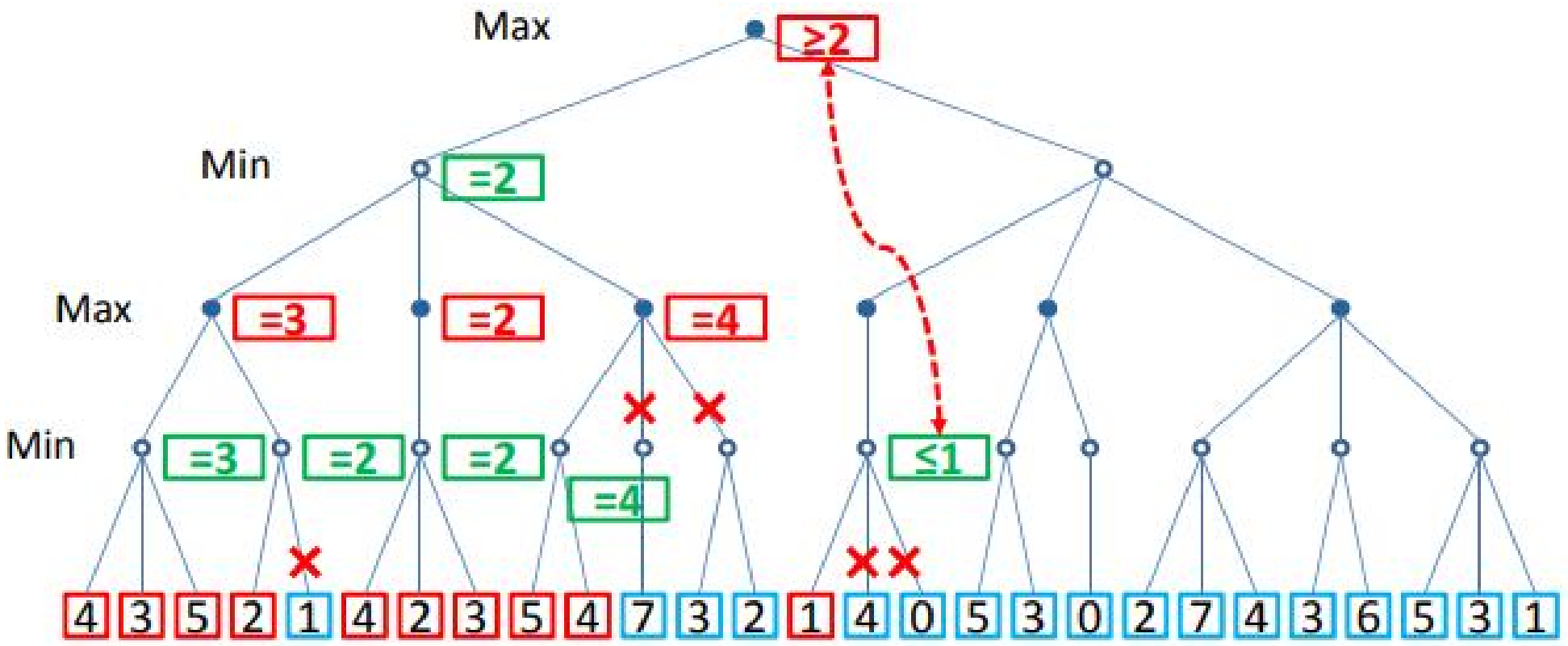

Max
Min
=2
Max
=3
=2
=4
Min
=3
=2
=2
=4
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

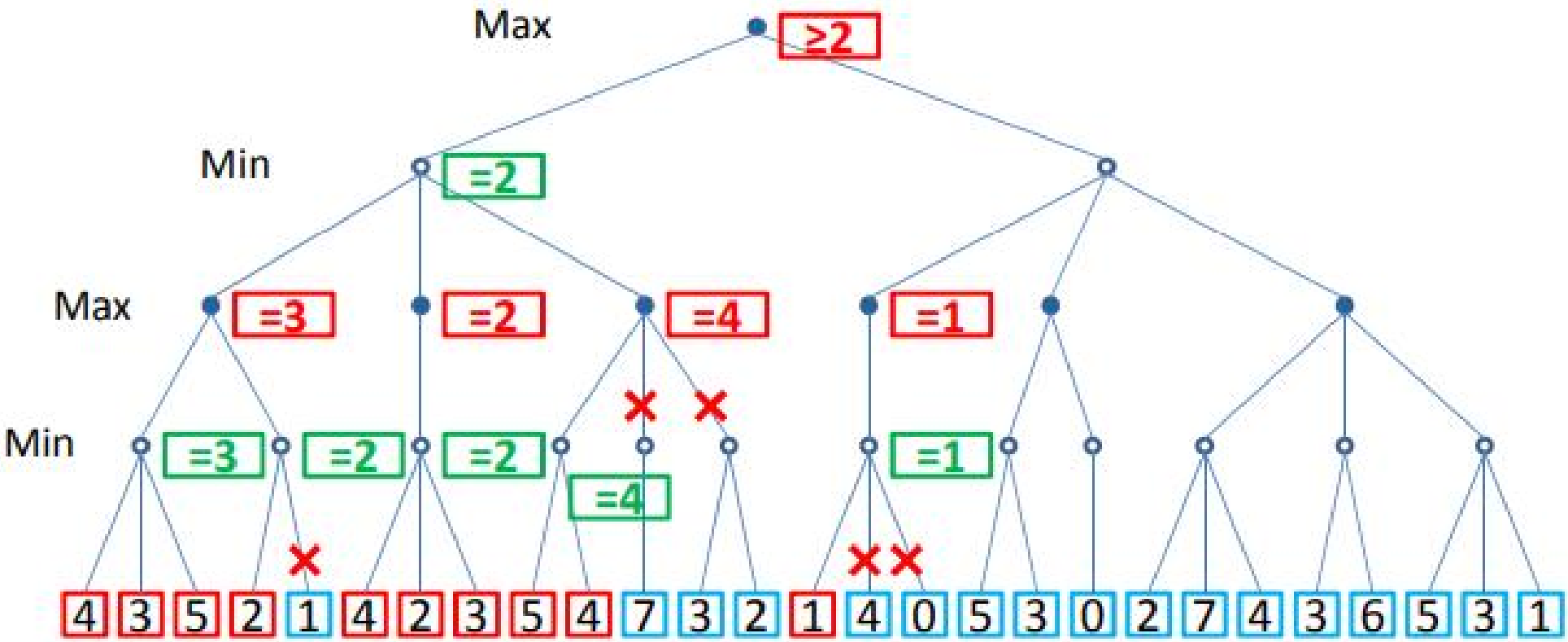

Max
≥2
Min
=2
Max
=3
=2
=4
Min
=3
=2
=2
=4
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

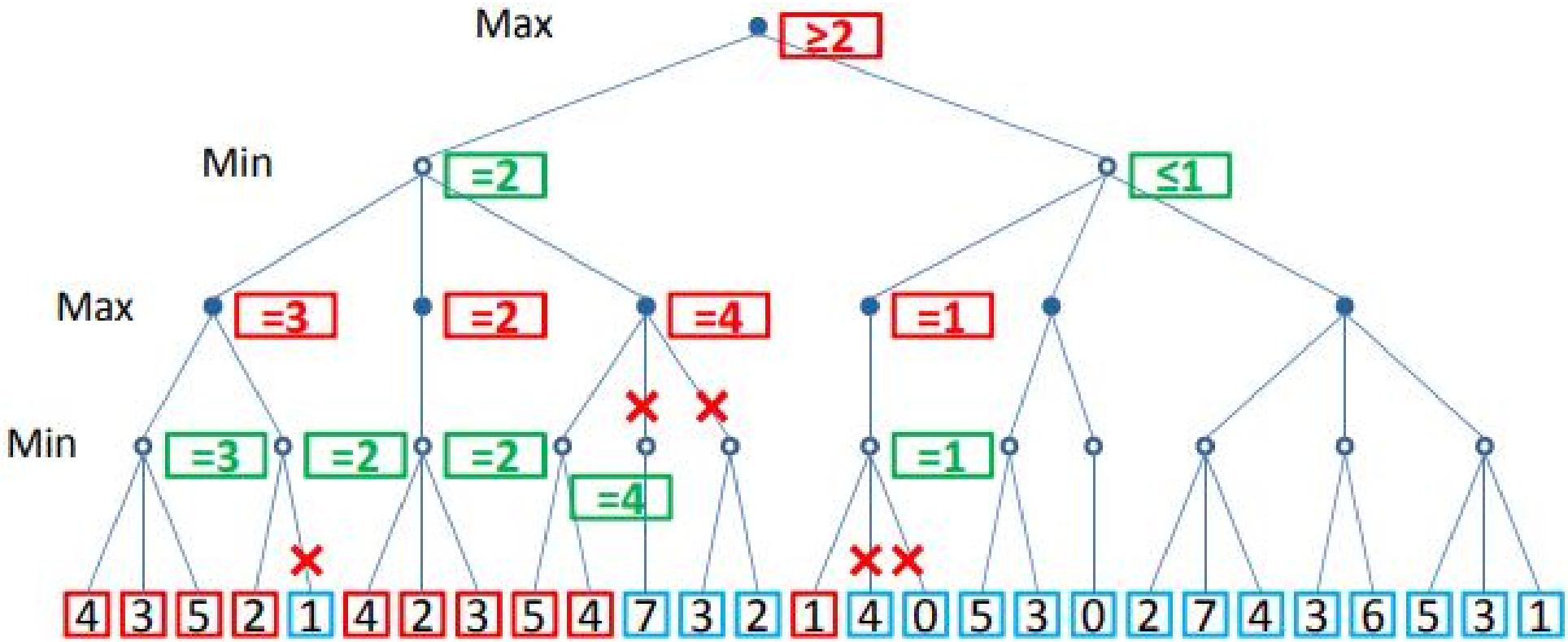

Max
≥2
Min
=2
Max
=3
=2
=4
Min
=3
=2
=2
=4
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1


Max
≥2
Min
=2
Max
=3
=2
=4
Min
=3
=2
=2
=4
≤1
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

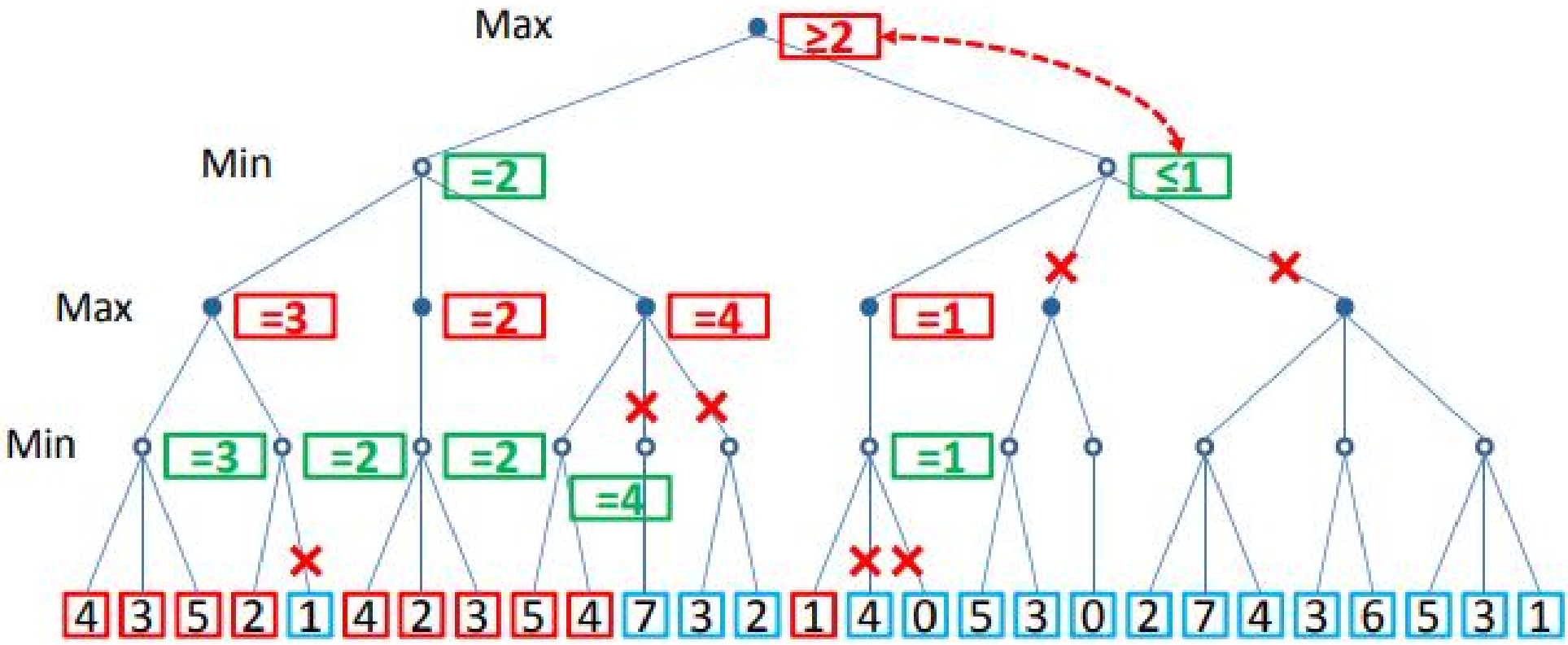
“Deep” cut-off: Parent α-nodes ≥ Child β-nodes

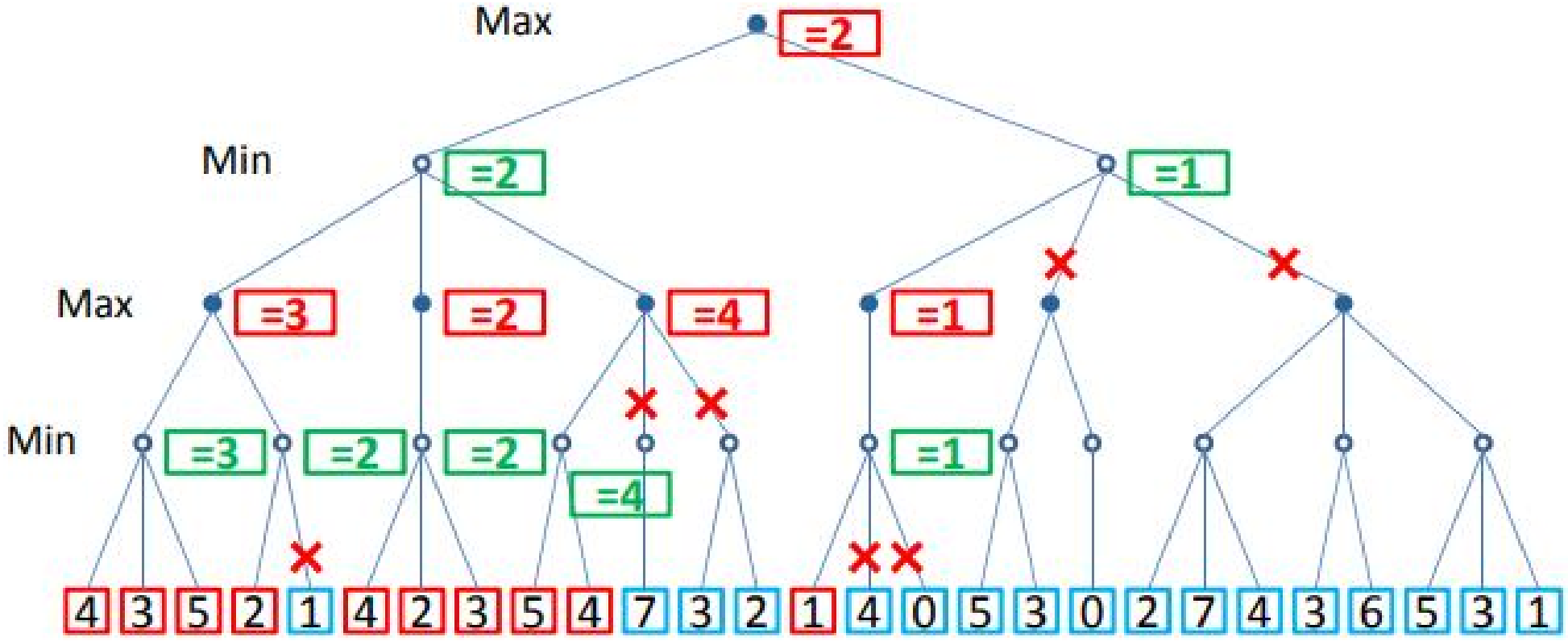



Max
≥2
Min
=2
Max
=3
=2
=4
=1
Min
=3
=2
=2
=4
=1
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1


Max
≥2
Min
=2
≤1
Max
=3
=2
=4
=1
Min
=3
=2
=2
=4
=1
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

Prune: Parent α-nodes ≥ Child β-nodes




Max
=2
Min
=2
=1
Max
=3
=2
=4
=1
Min
=3
=2
=2
=4
=1
4 3 5 2 1 4 2 3 5 4 7 3 2 1 4 0 5 3 0 2 7 4 3 6 5 3 1

References

- https://www.javatpoint.com/ai-adversarial-search
- https://www.javatpoint.com/mini-max-algorithm-in-ai
- https://www.javatpoint.com/ai-alpha-beta-pruning
- https://dtai.cs.kuleuven.be/education/fai/