Module-3	**Local Search and Adversarial Search** **~~Local Search algorithms – Hill-climbing search, Simulated annealing, Genetic Algorithm,~~ Adversarial Search: Game Trees and Minimax Evaluation, Elementary two-players games: tic-tac-toe, Minimax with Alpha-Beta Pruning.**

Adversarial Search

Adversarial search in artificial intelligence is a decision-making technique for multi-agent competitive environments where agents have opposing goals.

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.

Key Concepts in Adversarial Search

- Competitive Environment
- Conflicting Goals
- Game Theory
- Game Tree
- Minimax Algorithm
- Alpha-Beta Pruning

How Adversarial Search Works?

- **Modeling the Game**
- **Anticipating Opponent Moves**
- **Evaluating Outcomes**
- **Selecting a Strategy**

Applications

- **Game Playing AI**
- **Business and Trading**

Game Playing

Introduction

- Game playing is one of the oldest sub-filed of AI.
- Game playing involves **abstract and pure** form of competition that seems to require intelligence.
- It is easy to represent **the state and actions**.
- To implement game playing is required very little world knowledge.

Why Study Game Playing?

- Games allow us to experiment with easier versions of **real-world situations**
- **Hostile agents** act against our goals
- Games have a finite set of moves
- Games are fairly easy to represent
- Good idea to decide about what to think
- Perfection is unrealistic, must settle for good
- One of the earliest areas of AI
  - Claude Shannon and Alan Turing wrote chess programs in 1950s
- The opponent introduces **uncertainty**
- The environment may contain uncertainty (backgammon)
- Search space too hard to consider exhaustively
  - **Chess has about 10^{40} legal positions**
  - Efficient and effective search strategies even more critical
- Games are fun to target!

Assumptions

- Static or dynamic?
- Fully or partially observable?
- Discrete or continuous?
- Deterministic or stochastic?
- Episodic or sequential?
- Single agent or multiple agent?

Zero-Sum Games

- Focus primarily on "adversarial games"
- Two-player, zero-sum games

As Player 1 gains strength

Player 2 loses strength

and vice versa

The sum of the two strengths is always 0.

- The most common used AI technique in game is search.
- Game playing research has contributed ideas on how **to make the best use of time to reach good decisions.**
- Game playing is a search problem defined by:
    - » **Initial state of the game**
    - » **Operators defining legal moves**
    - » **Successor function**
    - » **Terminal test defining end of game states**
    - » **Goal test**
    - » **Path cost/ utility/ payoff function**

Search Applied to Adversarial Games

- Initial state
  - Current board position (description of current game state)
- Operators
  - Legal moves a player can make
- Terminal nodes
  - Leaf nodes in the tree
  - Indicate the game is over
- Utility function
  - Payoff function
  - Value of the outcome of a game
  - Example: tic tac toe, utility is -1, 0, or 1

Using Search

- Search could be used to find a perfect sequence of moves except the following problems arise:
  - There exists an adversary who is trying to minimize your chance of winning every other move
    - You cannot control his/her move
  - **Search trees can be very large**, but you have finite time to move
    - Chess has 10^{40} nodes in search space
    - With single-agent search, can afford to wait
    - Some two-player games have time limits
    - Solution?
      - Search to n levels in the tree (n **ply**)
      - Evaluate the nodes at the nth level
      - Head for the best looking node

Characteristics of game playing

- **There is always an "Unpredictable" opponent**
  - Due to their uncertainty.
  - He/she also wants to win.
  - Solution for each problem is a strategy, which specifies a move for every possible opponent reply.
- **Time Limits**
  - Games are often played under strict time constraints and therefore must be very effectively handled.
  - There are special games where the two players have exactly opposite goals.
  - It can be divided into two types.
    - **Perfect Information games (where both players have access to the same information**
    - **Ex: Chess**
    - **Imperfect Information games (different information can be accessed)**
    - **Ex:**tic-tac-toe, Battleship, blind, Bridge

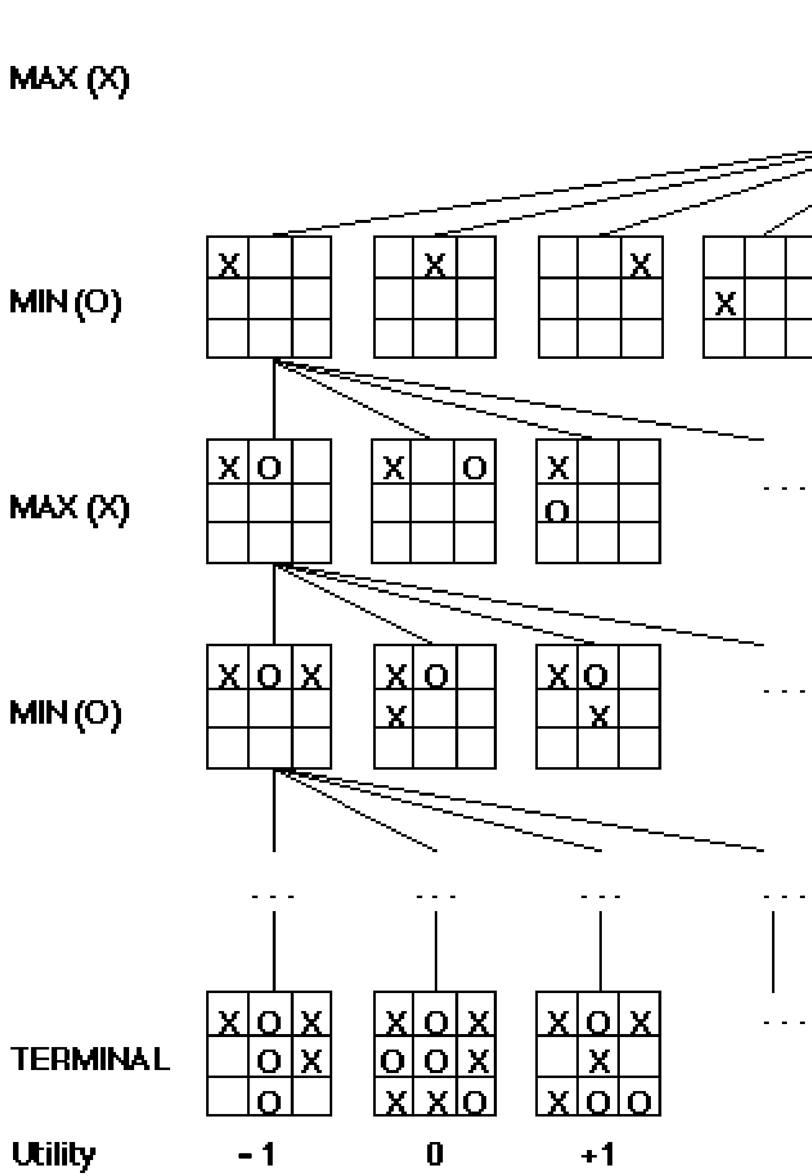
Optimal Decisions in Games

- MIN and MAX are two game players
- First move performed by MAX then turn is given to MIN and so on until the game is over
- A loser need to pay penalties and game points a credits are gifted to the winning player, at end of the game
- A game as search problem and broadly defined by following components
    1. **The Initial State:** It is start state position on the board, which leads to further move.
    2. **A successor function:** It is collection of (move, state) pairs, each specifies a legal move and output state. Here actions are considered.
    3. **Terminal Test:** It is final state which declares that the game is over or ended or terminal states.

Game Trees

- **Game Tree:** Game tree is graphical representation of the initial state and legal moves for each side (two players – one by one) for a specific game
- MAX has nine possible move from initial state(Tic tac toe).
- MAX with X and MIN with O symbol. The leaf nodes are entitled by utility value
- The MAX is assumed with high values and MIN is with low values. The best move is always determined by MAX by using search tree
- **Optimal Strategies**
  - A sequence of moves which achieves Win state/ Terminal state/ Goal state in search problem
- **Optimal Strategy in Game**
  - It is techniques which always leads to superior that any other strategy as opponent is playing in perfect manner

Game tree for Tic-Tac-Toe: 2-player, deterministic



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

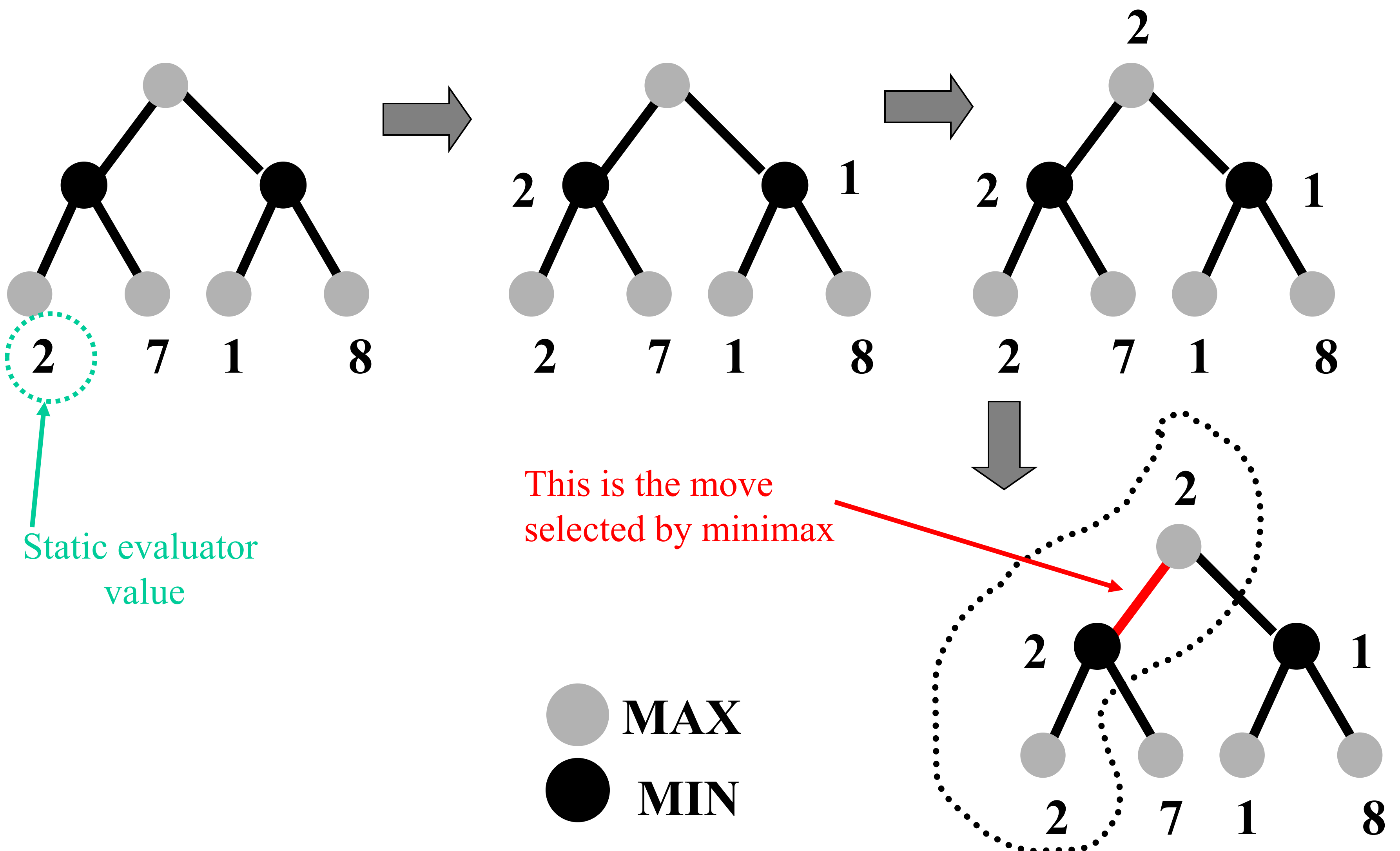
Minimax Rule

- Goal of game tree search: to determine **one move** for Max player that **maximizes** the **guaranteed payoff** for a given game tree for MAX
  - Regardless of the moves the MIN will take
- The value of each node (Max and MIN) is determined by (back up from) the values of its children
- MAX plays the worst case scenario:
  - Always assume MIN to take moves to maximize his pay-off (i.e., to minimize the pay-off of MAX)
- For a MAX node, the backed up value is the **maximum** of the values associated with its children
- For a MIN node, the backed up value is the **minimum** of the values associated with its children

MINIMAX Procedure

- Starting from the leaves of the tree (with final scores with respect to one player, MAX), and go backwards towards the root.
- At each step, one player (MAX) takes the action that leads to the highest score, while the other player (MIN) takes the action that leads to the lowest score.
- All nodes in the tree will be scored, and the path from root to the actual result is the one on which all nodes have the same score.
- We are MAX - trying to maximise our score / move to best state. Opponent is MIN - tries to minimise our score / move to worst state for us.

Minimax Search



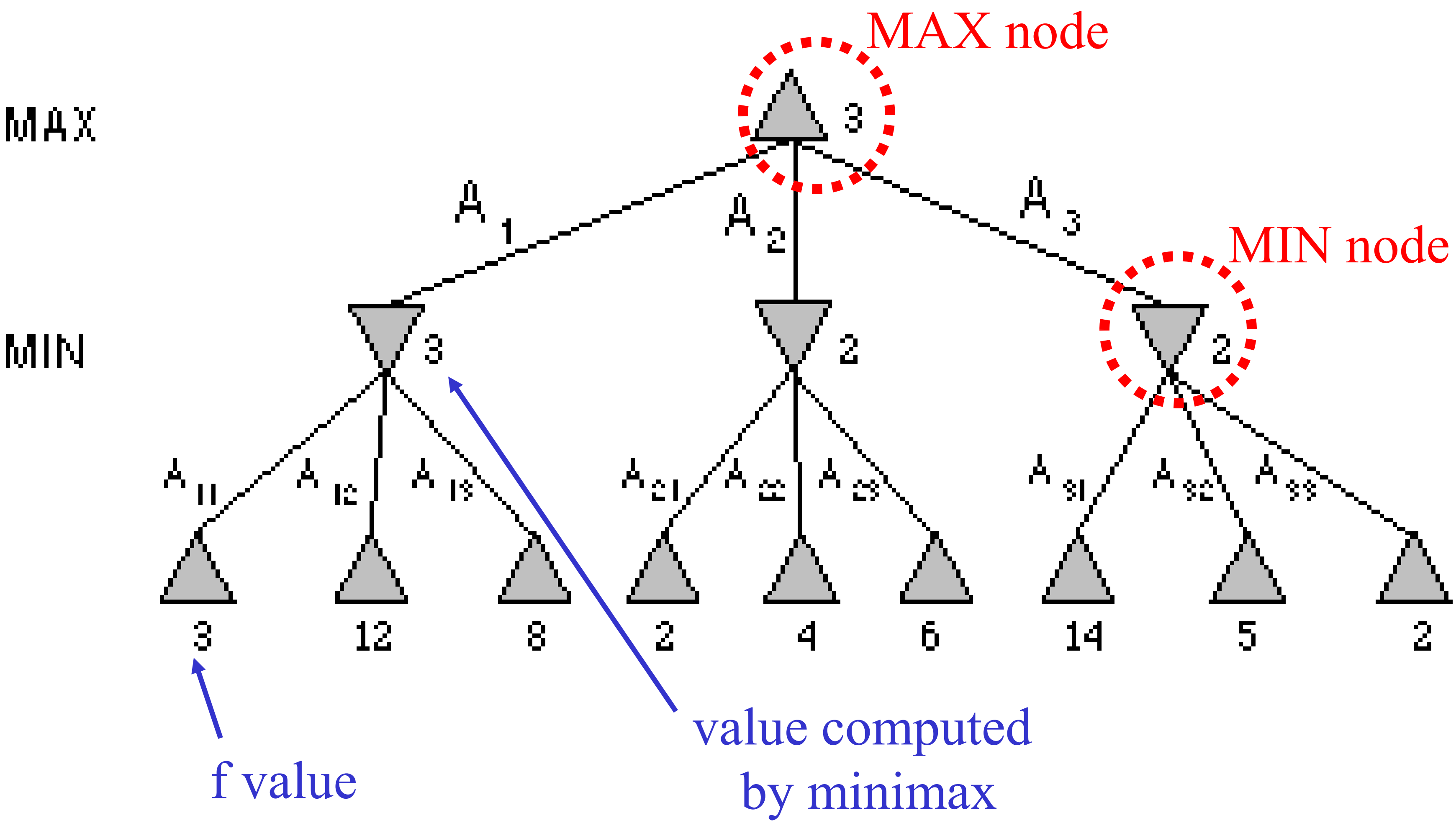
Comments on Minimax search

- The search is **depth-first** with the given depth (ply) as the limit
  - Time complexity: O(b^d)
  - Linear space complexity
- Performance depends on
  - Quality of evaluation functions (domain knowledge)
  - Depth of the search (computer power and search algorithm)
- Different from ordinary state space search
  - Not to search for a solution but for one move only
  - No cost is associated with each arc
  - MAX does not know how MIN is going to counter each of his moves
- Minimax rule is a basis for other game tree search algorithms

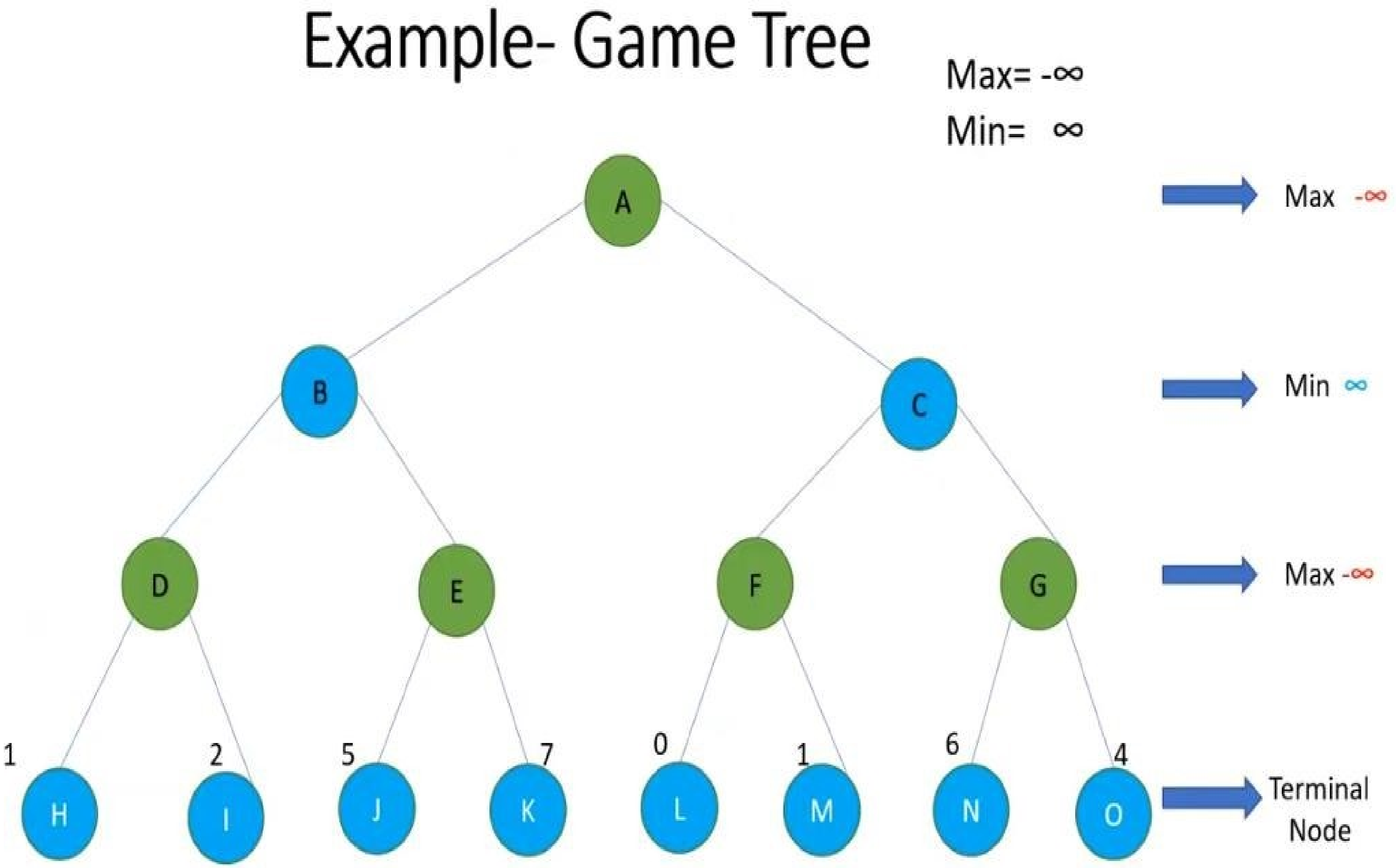
Minimax

- 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:

Minimax Tree

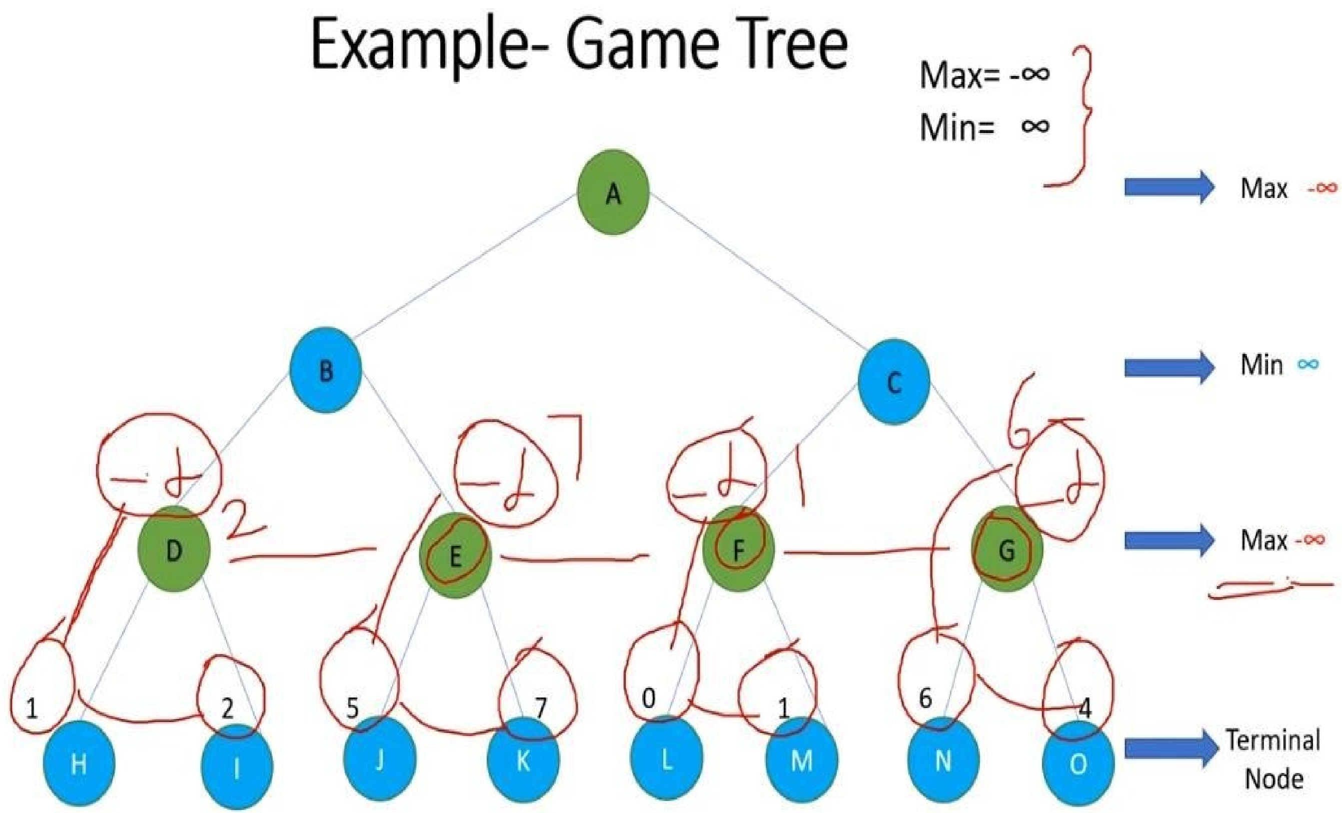


Minimax


Example- Game Tree
Max= -∞
Min= ∞
A
B
C
D
E
F
G
1
2
5
7
0
1
6
4
H
I
J
K
L
M
N
O
Max -∞
Min ∞
Max -∞
Terminal
Node

Minimax

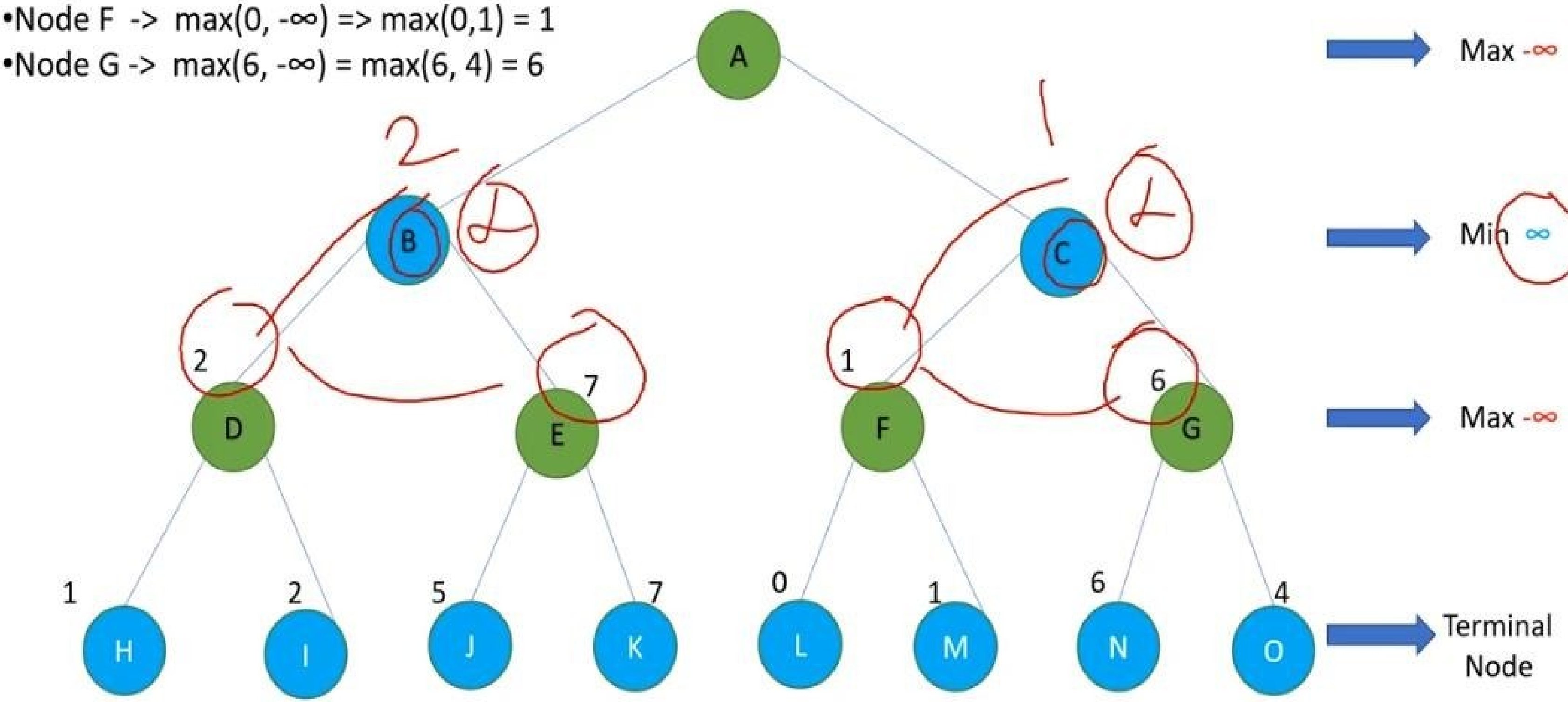
- Node D -> max(1,-∞) => max(1,2)= 2
- Node E -> max(5, -∞) => max(5, 7)= 7
- Node F -> max(0, -∞) => max(0,1) = 1
- Node G -> max(6, -∞) = max(6, 4) = 6



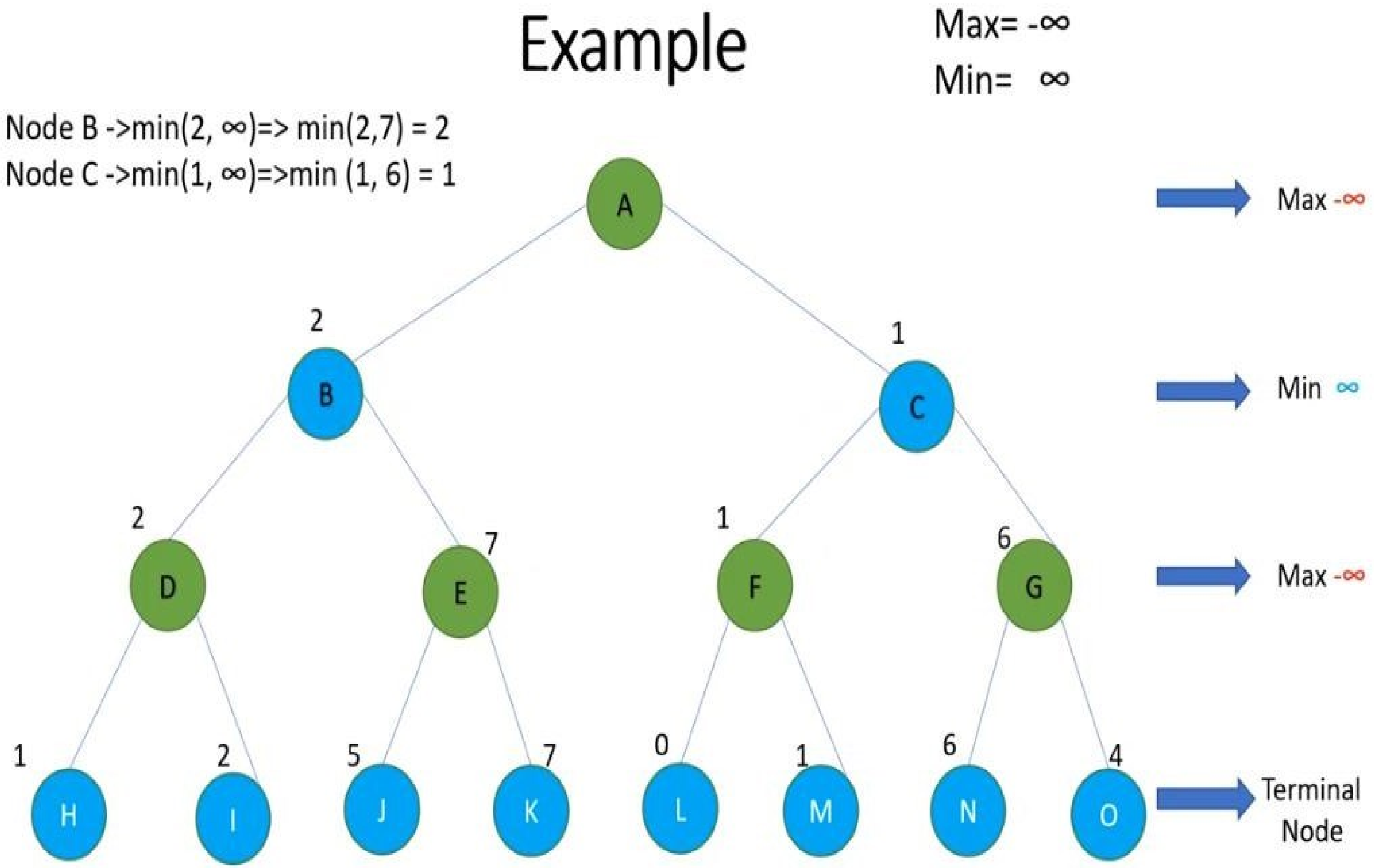
Minimax

Example

- Node D -> max(1,-∞) => max(1,2)= 2
- Node E -> max(5, -∞) => max(5, 7)= 7
- Node F -> max(0, -∞) => max(0,1) = 1
- Node G -> max(6, -∞) = max(6, 4) = 6

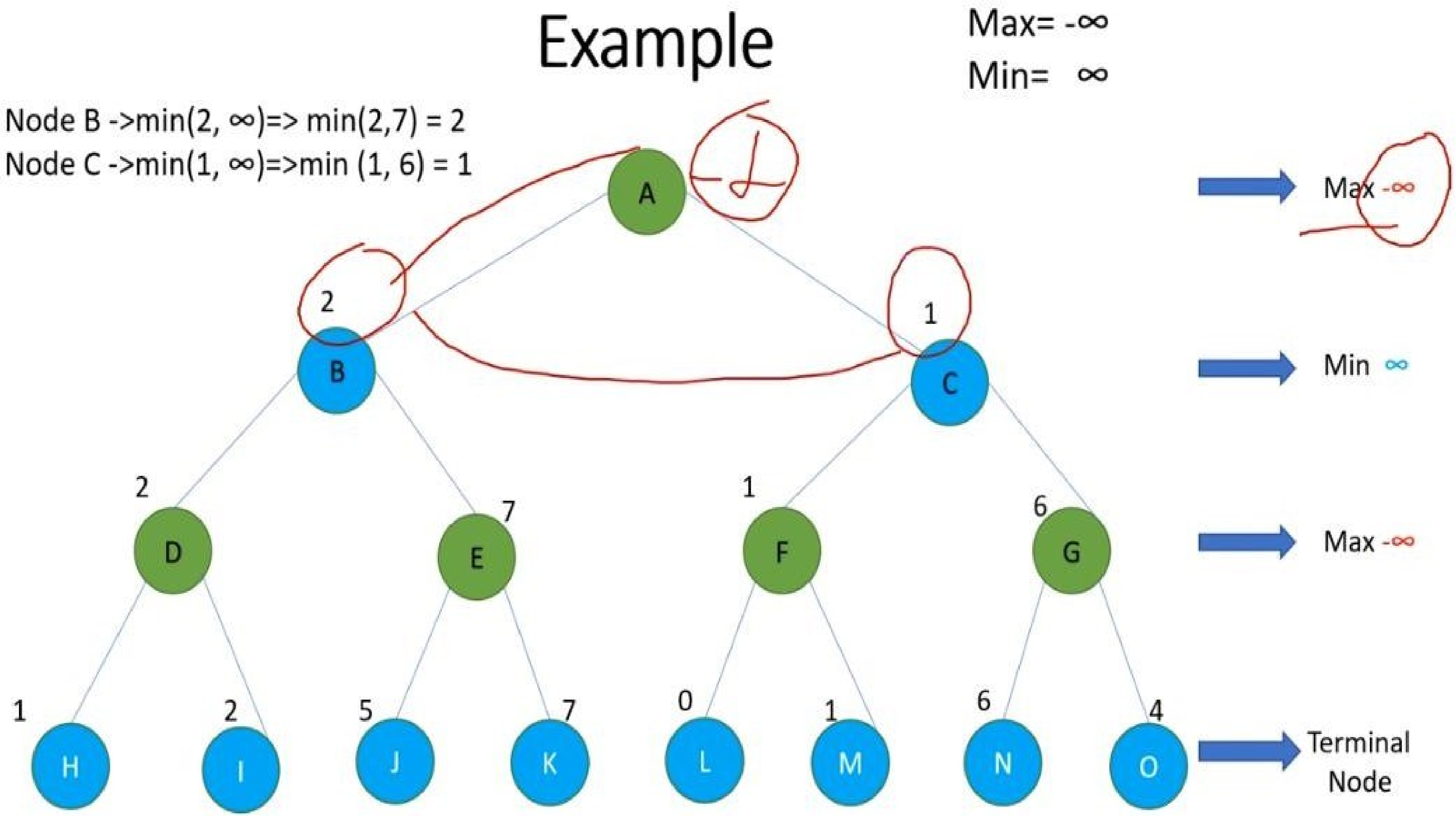


Minimax



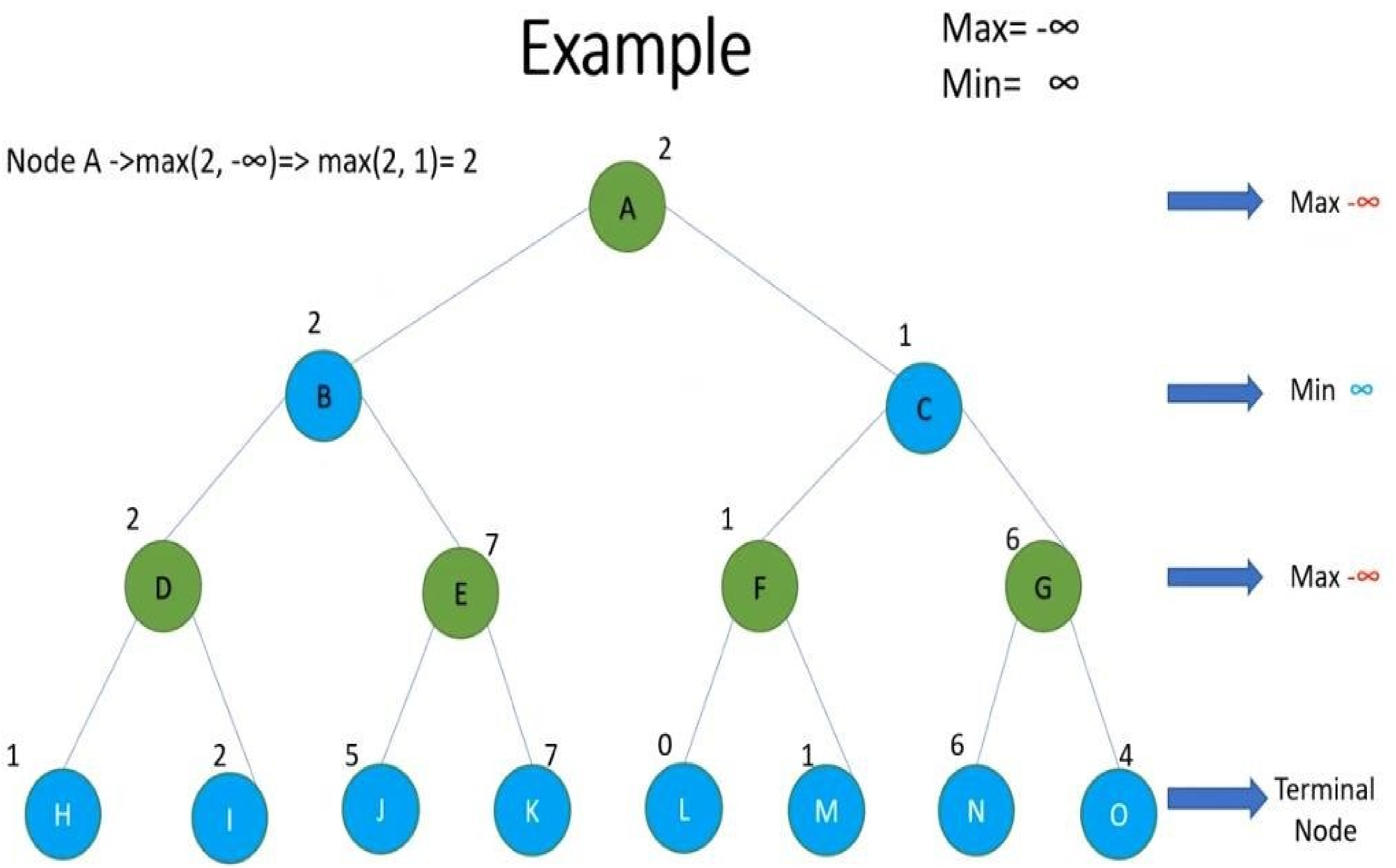
Example
Max= -∞
Min= ∞
Node B ->min(2, ∞)=> min(2,7) = 2
Node C ->min(1, ∞)=>min (1, 6) = 1
A
2
B
1
C
2
D
7
E
1
F
6
G
1
H
2
I
5
J
7
K
0
L
1
M
6
N
4
O
Max -∞
Min ∞
Max -∞
Terminal Node

Minimax

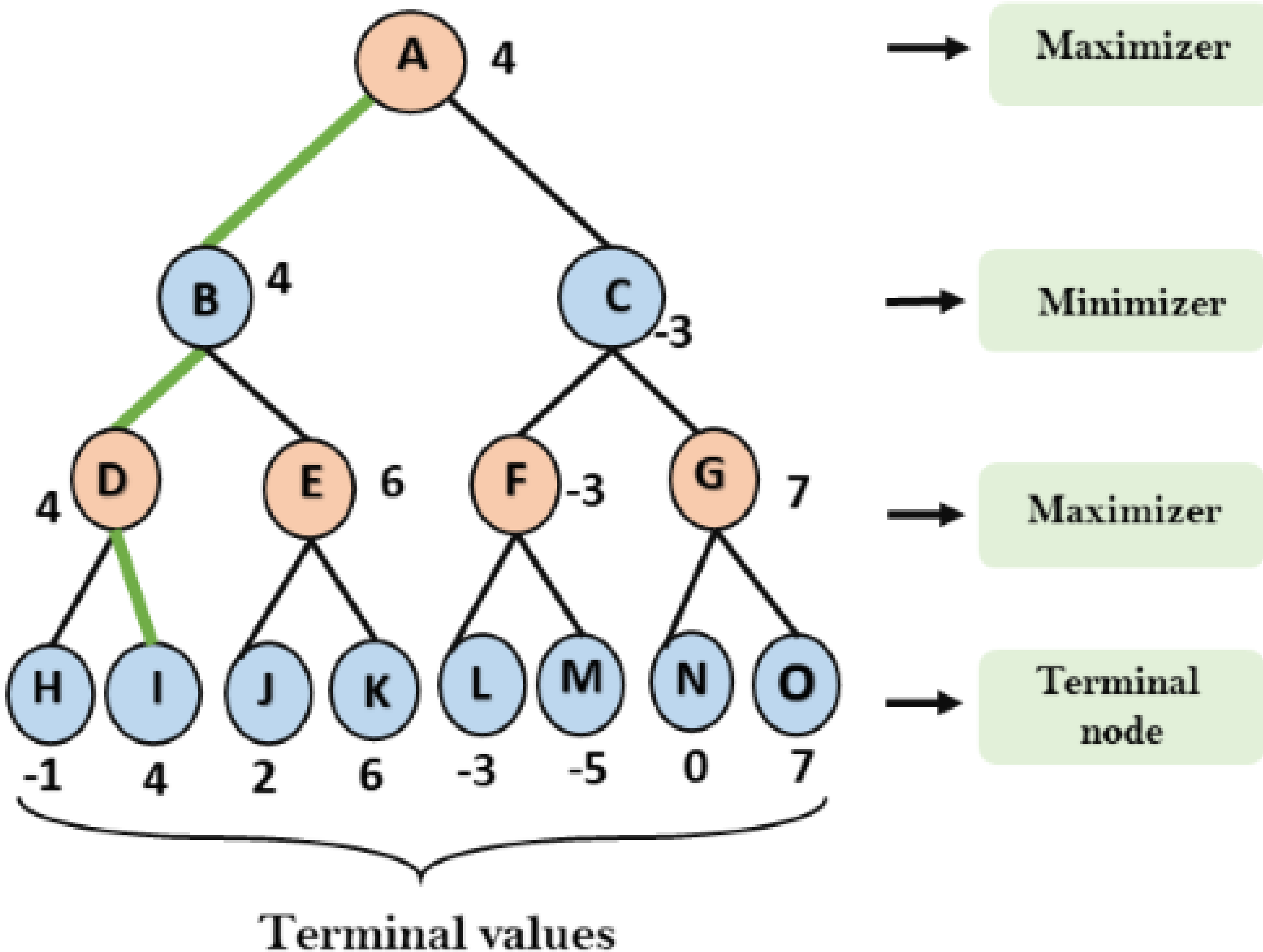


Example
Max= -∞
Min= ∞
Node B ->min(2, ∞)=> min(2,7) = 2
Node C ->min(1, ∞)=>min (1, 6) = 1
A
2
B
1
C
2
D
7
E
1
F
6
G
1
H
2
I
5
J
7
K
0
L
1
M
6
N
4
O
Max -∞
Min ∞
Max -∞
Terminal Node

Minimax



Example
Max= -∞
Min= ∞
Node A ->max(2, -∞)=> max(2, 1)= 2
2
A
2
B
1
C
2
D
7
E
1
F
6
G
1
H
2
I
5
J
7
K
0
L
1
M
6
N
4
O
Max -∞
Min ∞
Max -∞
Terminal
Node



A 4
B 4
C -3
D 4
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

Minimax algorithm

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)
- Time complexity? $O(b^m)$
- Space complexity? $O(bm)$ (**depth-first exploration**)

Alpha-Beta pruning

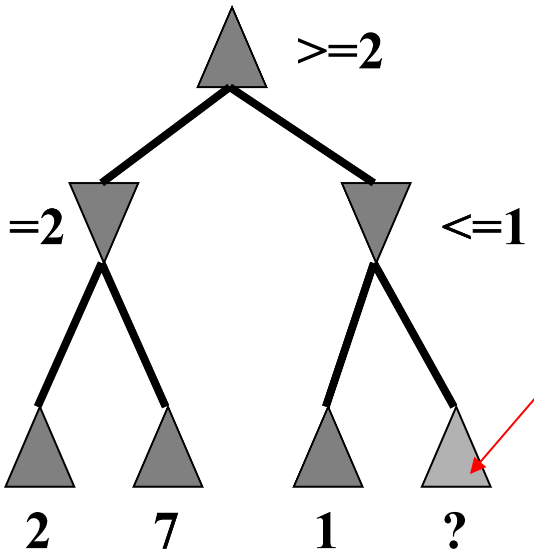
Why do we need **Alpha-beta pruning with MiniMax algorithm?**

- Alpha-beta pruning is a crucial optimization technique used in **conjunction with the minimax algorithm**, rather than a replacement for it.
- The core reason for its necessity lies in **the computational efficiency it provides, especially in games with large search spaces**.

> - **Reduces Computation Time**
> - **Allows Deeper Search**
> - **Maintains Optimality**

Alpha-Beta pruning

- We can **improve on the performance** of the minimax algorithm through alpha-beta pruning.
- Basic idea: *"If you have an idea that is surely bad, don't take the time to see how truly awful it is."* -- Pat Winston



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

Alpha-Beta pruning

- Traverse the search tree in **depth-first order**
- At each **Max** node n, **alpha(n)** = maximum value found so far
  - **Start with -infinity and only increase**
  - Increases if a child of n returns a value greater than the current alpha
  - Serve as a tentative lower bound of the final pay-off
- At each **Min** node n, **beta(n)** = minimum value found so far
  - **Start with infinity and only decrease**
  - Decreases if a child of n returns a value less than the current beta
  - Serve as a tentative upper bound of the final pay-off

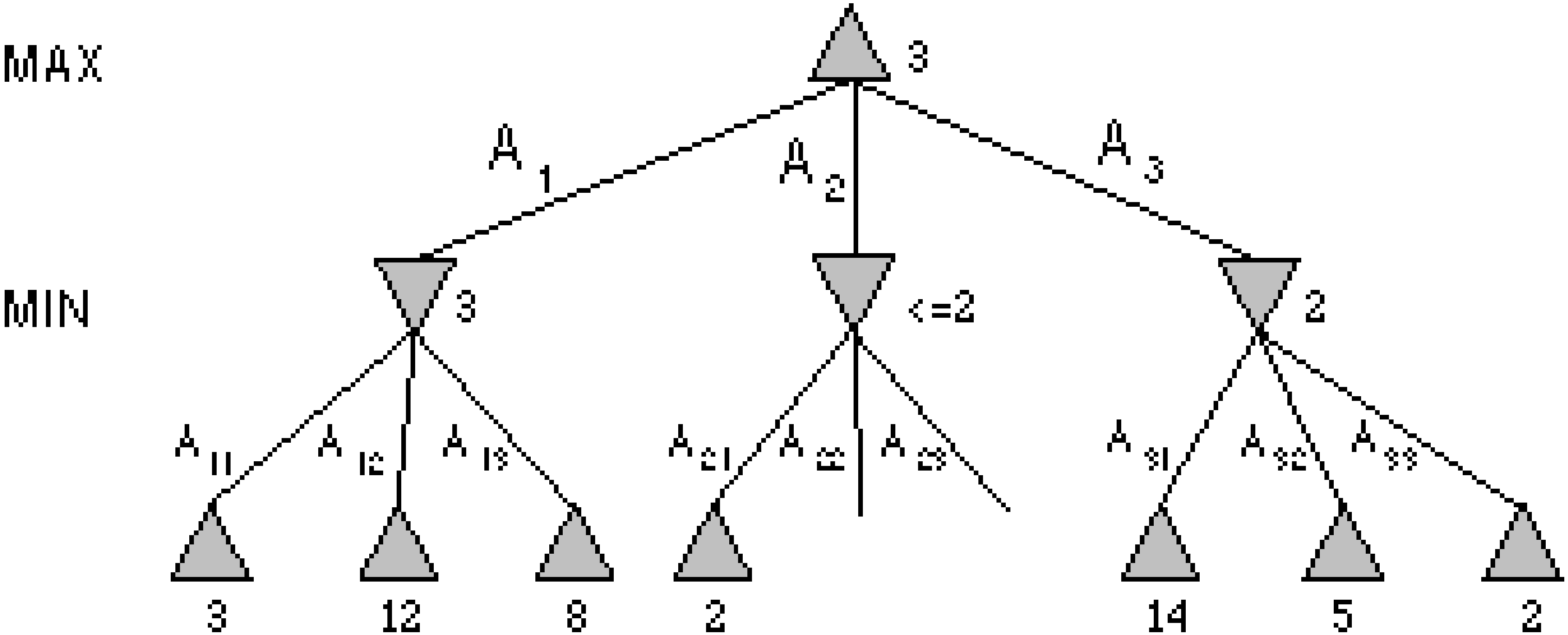
Alpha-Beta pruning

- **Alpha cutoff**: Given a Max node n, cutoff the search below n (i.e., don't generate or examine any more of n's children) if **alpha(n) >= beta(n)**
  (alpha increases and passes beta from below)

- **Beta cutoff.:** Given a Min node n, cutoff the search below n (i.e., don't generate or examine any more of n's children) if **beta(n) <= alpha(n)**
  (beta decreases and passes alpha from above)

- Carry alpha and beta values down during search
  Pruning occurs whenever alpha >= beta

Alpha-Beta pruning



Alpha-Beta pruning

initiation :
α :=-∞
β := ∞

- Two functions recursively call each other

function MAX-value (n, alpha, beta)
  if n is a leaf node **then return** f(n);
  for each child n' of n **do**
    alpha :=max{alpha, MIN-value(n', alpha, beta)};
    if alpha >= beta **then return** beta /* pruning */
  end{do}
  return alpha

function MIN-value (n, alpha, beta)
  if n is a leaf node **then return** f(n);
  for each child n' of n **do**
    beta :=min{beta, MAX-value(n', alpha, beta)};
    if beta <= alpha **then return** alpha /* pruning */
  end{do}
  return beta

Effectiveness of Alpha-beta pruning

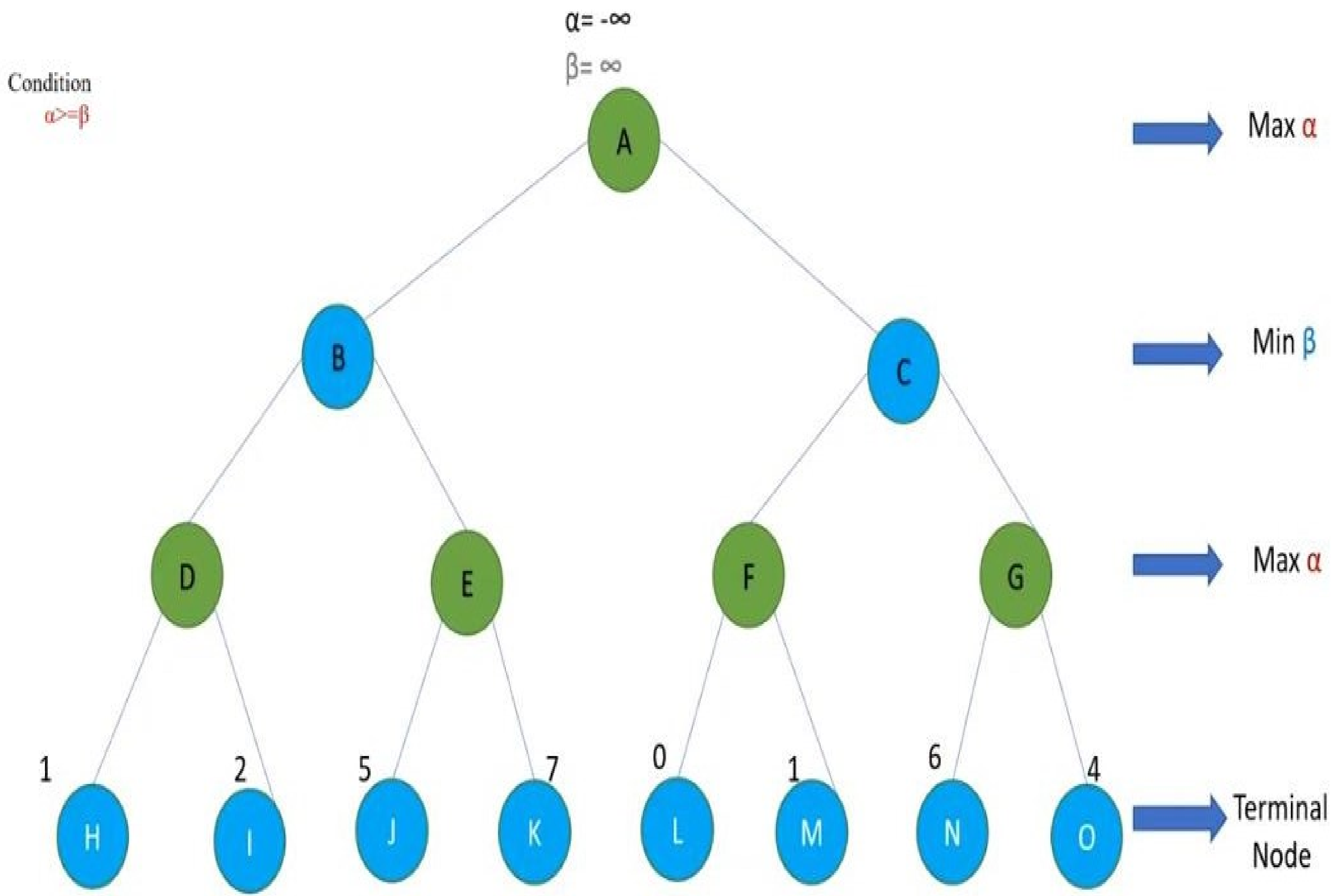
- Alpha-Beta is **guaranteed to compute the same value** for the root node as computed by Minimax.
- **Worst case: NO pruning, examining O(b^d) leaf nodes**, where each node has b children and a d-ply search is performed
- **Best case: examine only O(b^(d/2)) leaf nodes.**
  - You can **search twice as deep as Minimax**! Or the branch factor is b^(1/2) rather than b.
- **Best case** is when each player's best move is the leftmost alternative, i.e. at MAX nodes the child with the largest value generated first, and at MIN nodes the child with the smallest value generated first.
- In Deep Blue, they found empirically that Alpha-Beta pruning meant that **the average branching factor** at each node was about 6 instead of about 35-40

Alpha-Beta pruning

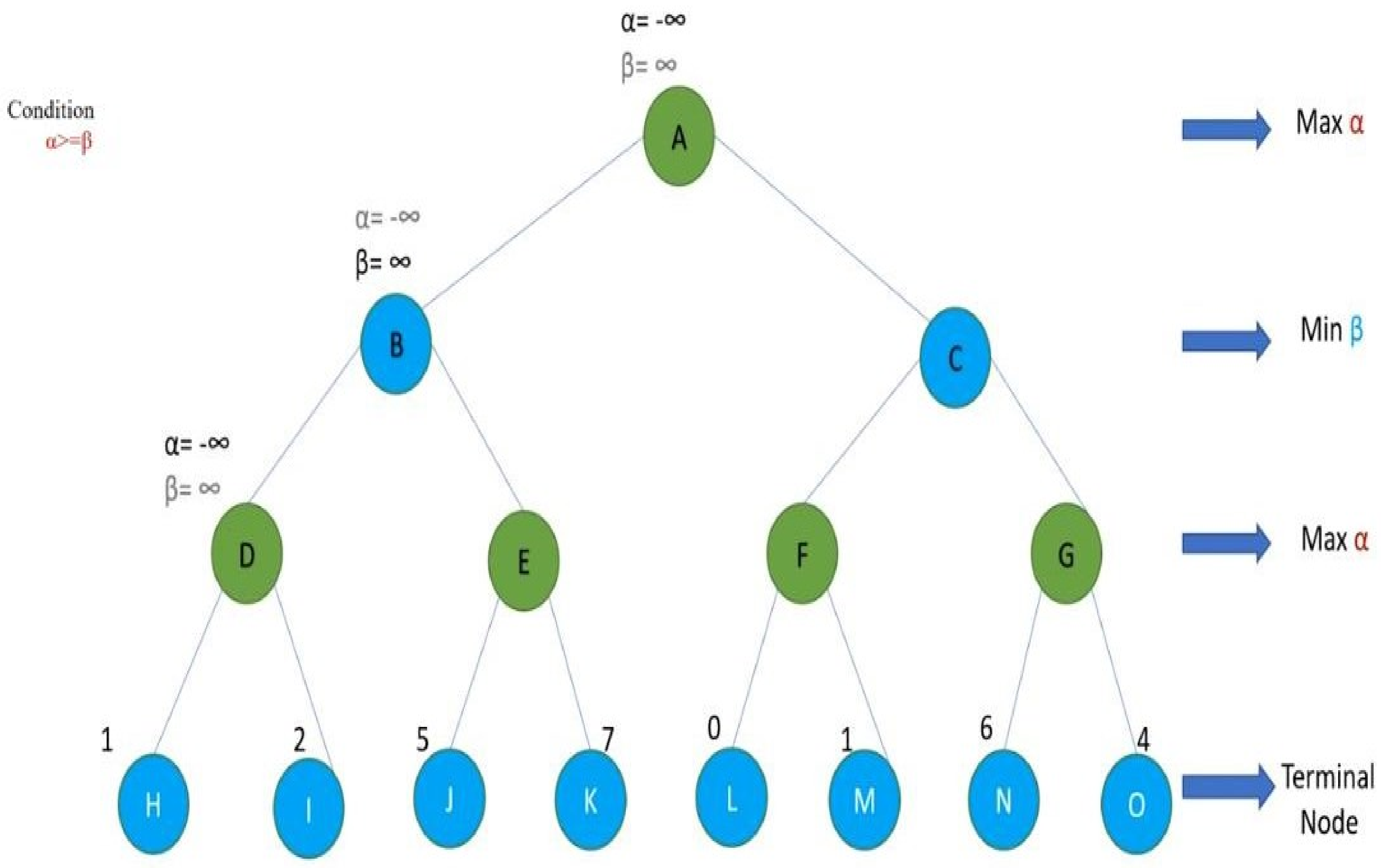
- The basic idea of alpha-beta cutoffs is "**It is possible to compute the correct minimax decision without looking at every node in the search tree**"
- Allow the search process to ignore the portion of the search tree that makes no difference to the final choice
- General principle of α-β pruning is
  - Consider a **node n somewhere** in the tree, such that a player has a chance to move to this node.
  - If the player has a better chance m either at the parent node of n then n will never be reached in actual play.

- **Alpha:** The highest value that the maximizer can guarantee by making some move at the current node OR at some node earlier on the path.
- **Beta:** The lowest value that the minimizer can guarantee by making some move at the current node OR at some node earlier on the path to this node.

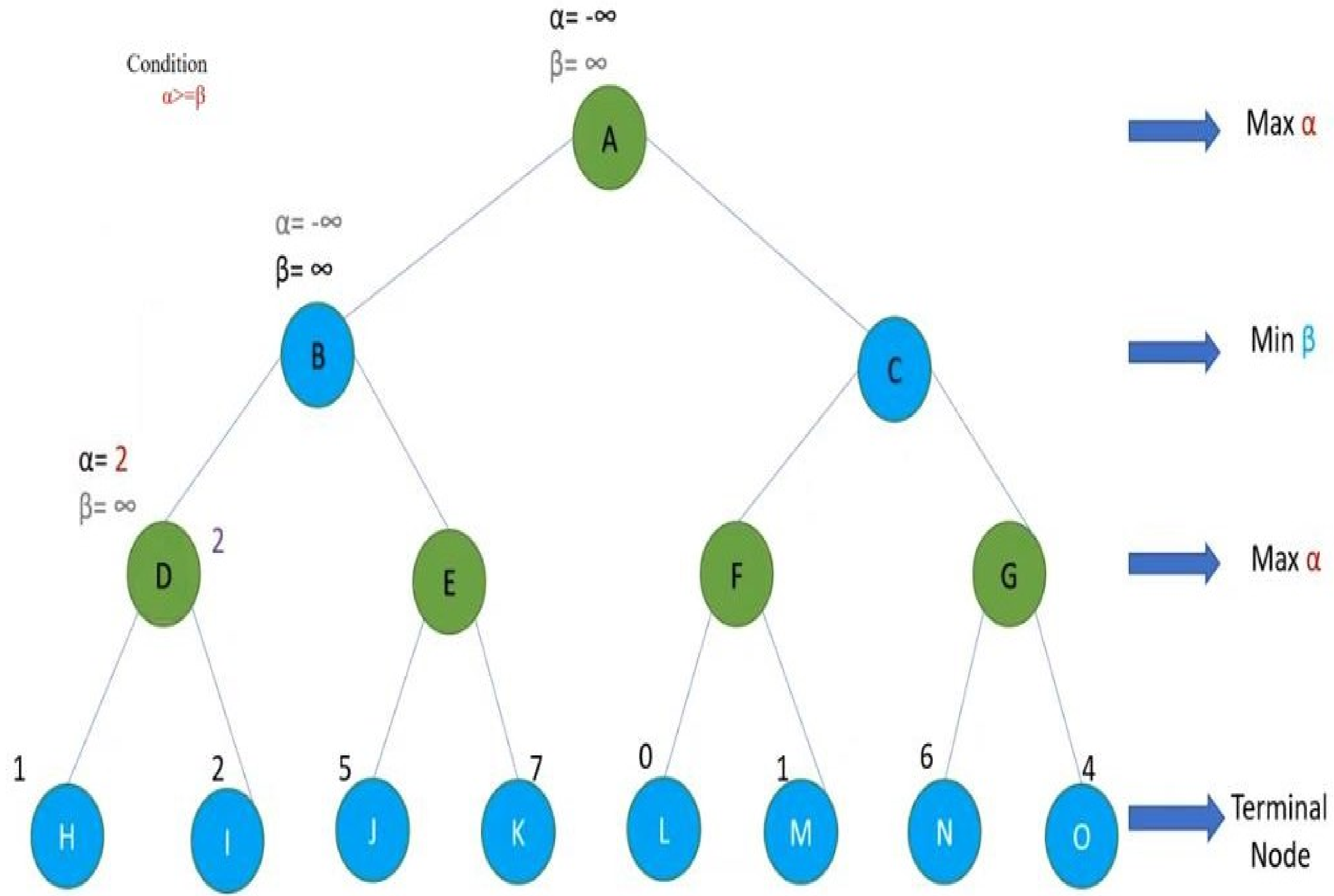
Alpha-Beta pruning: Example



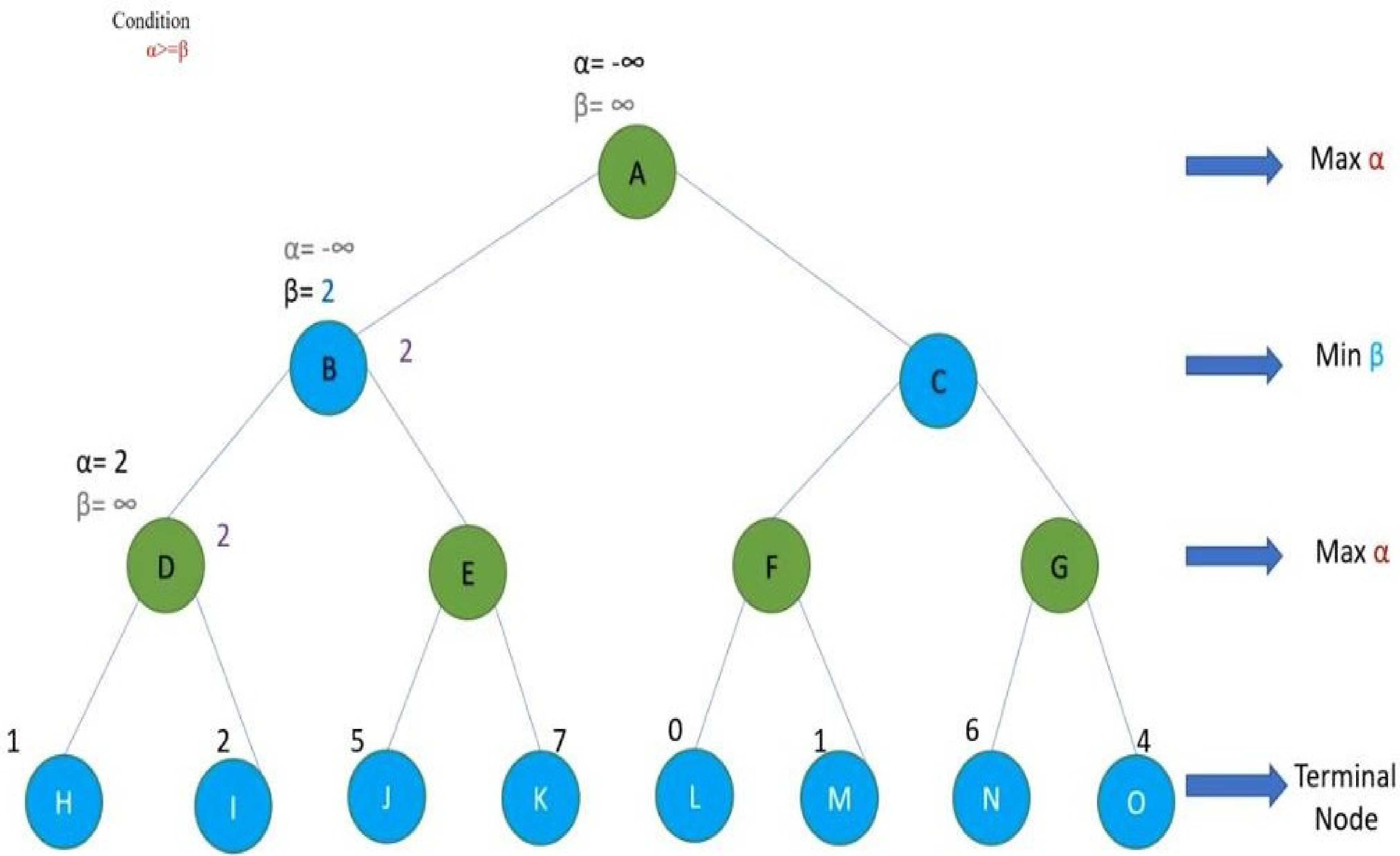
Alpha-Beta pruning: Example



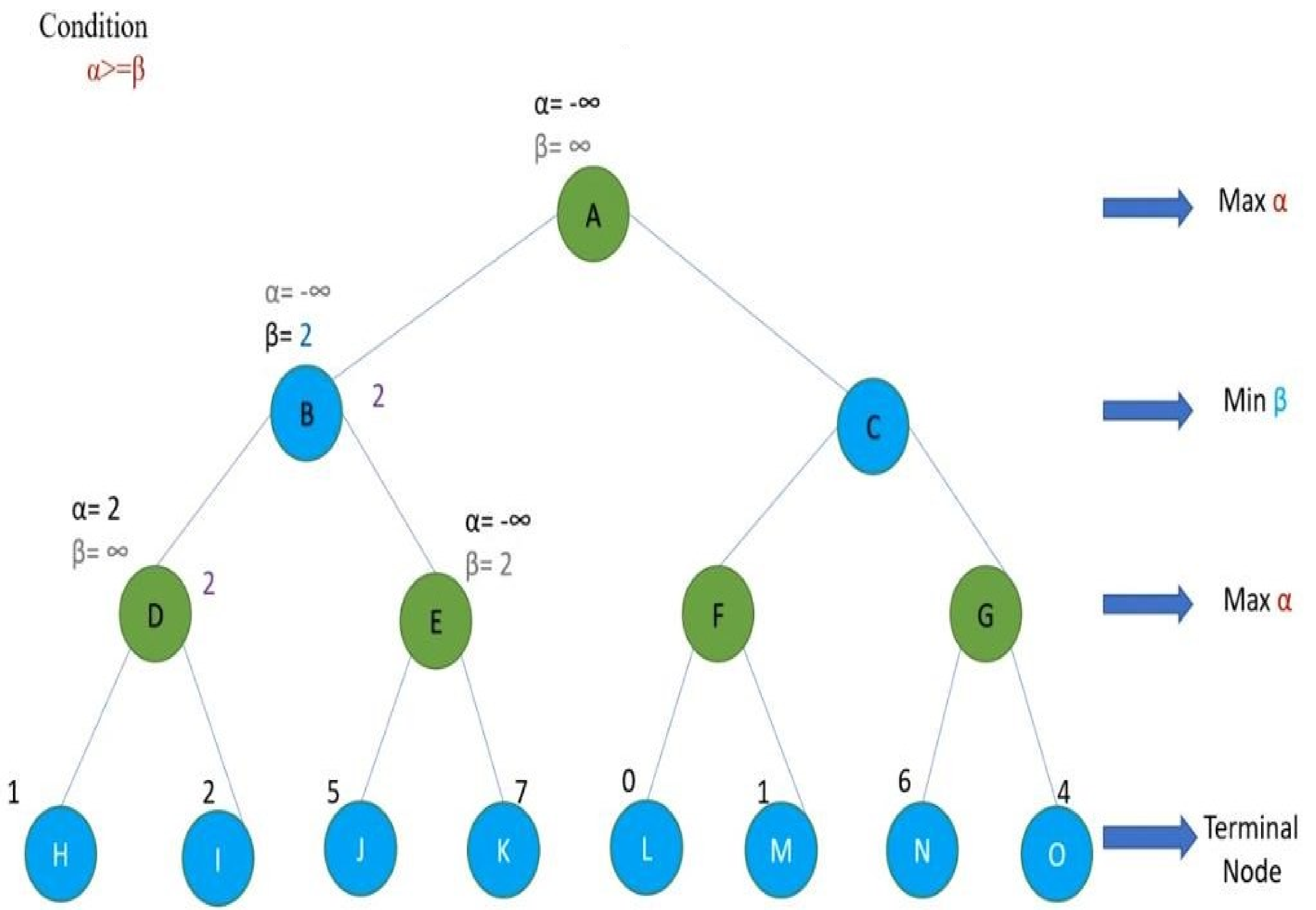
Alpha-Beta pruning: Example



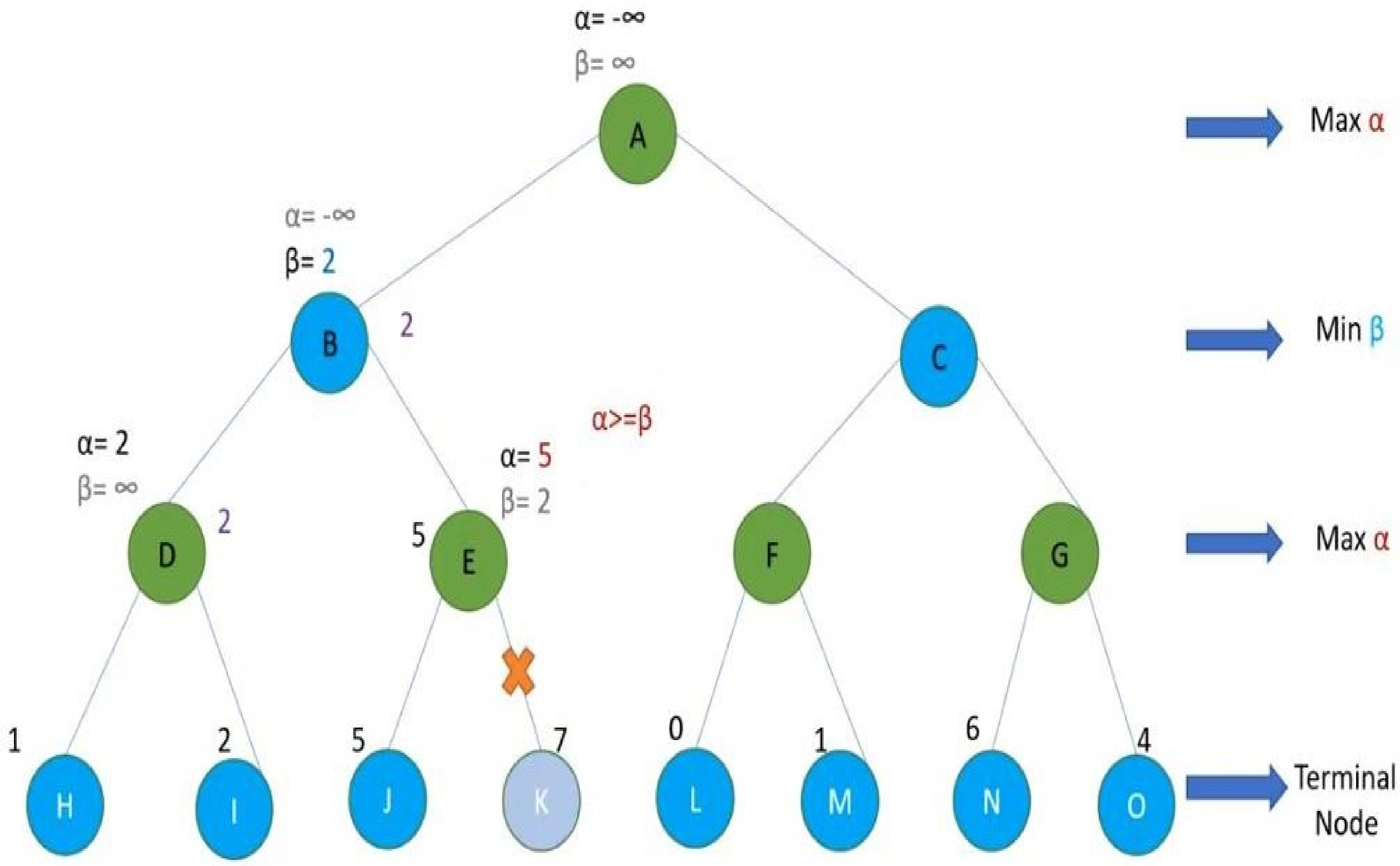
Alpha-Beta pruning: Example



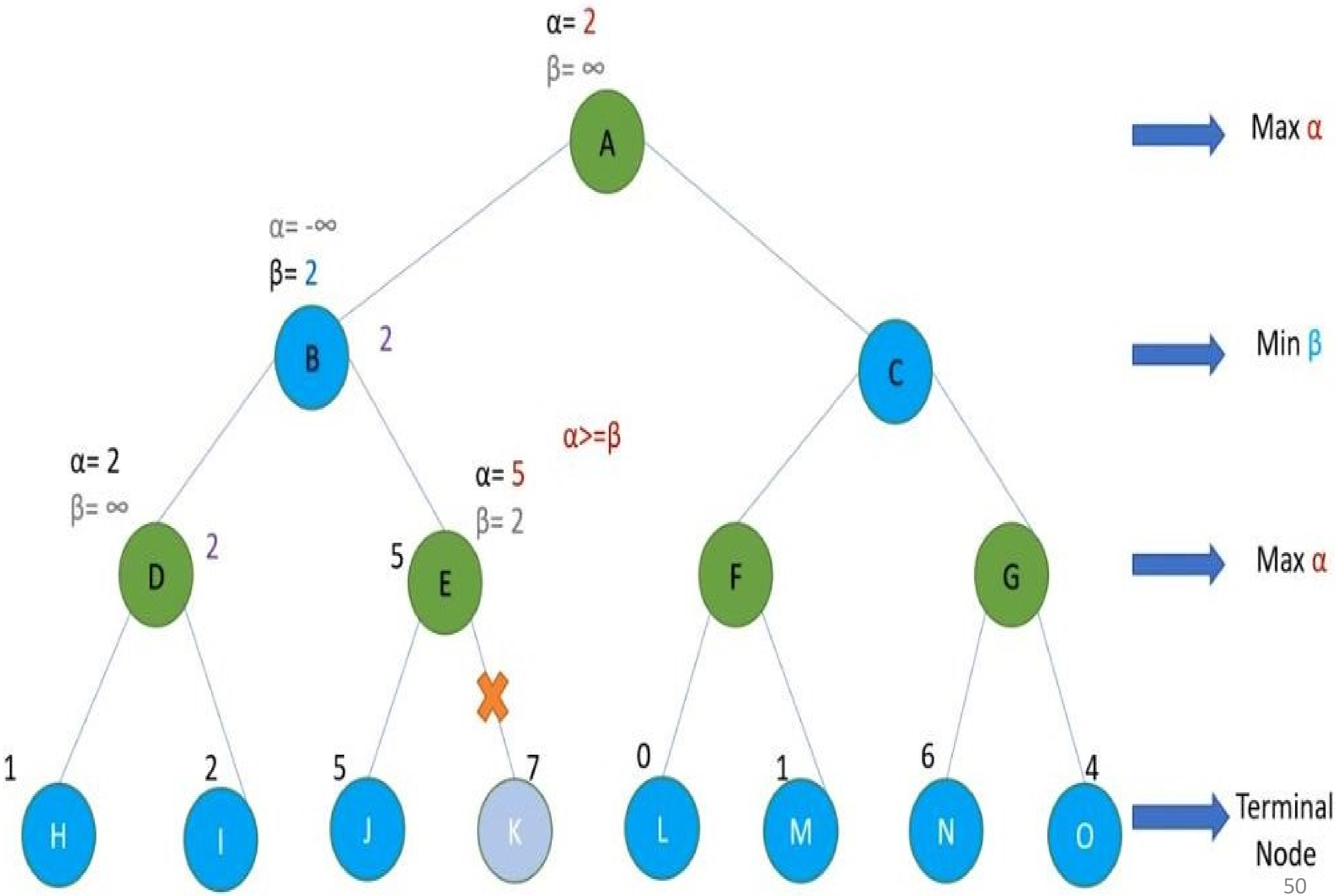
Alpha-Beta pruning: Example



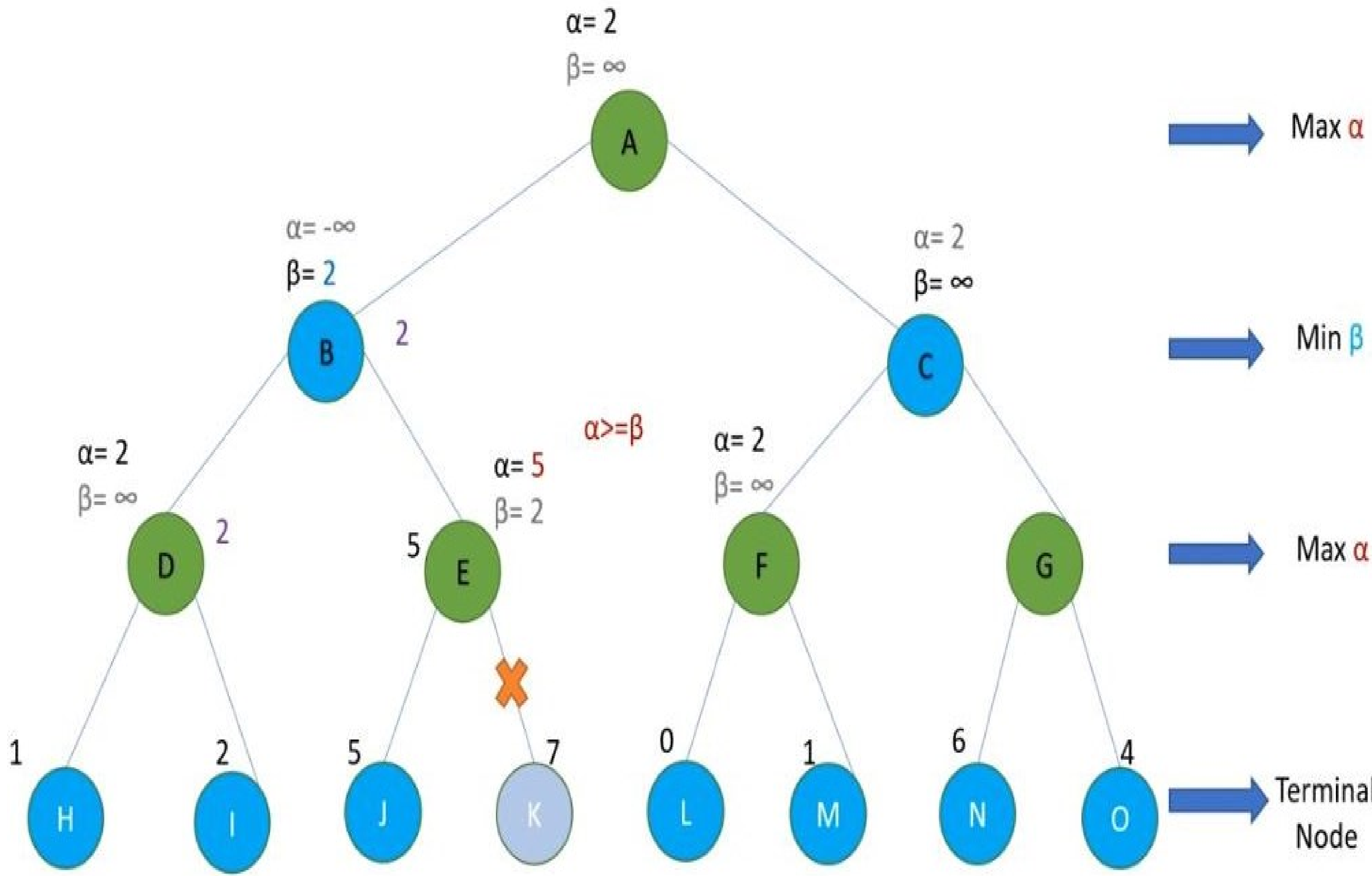
Alpha-Beta pruning: Example



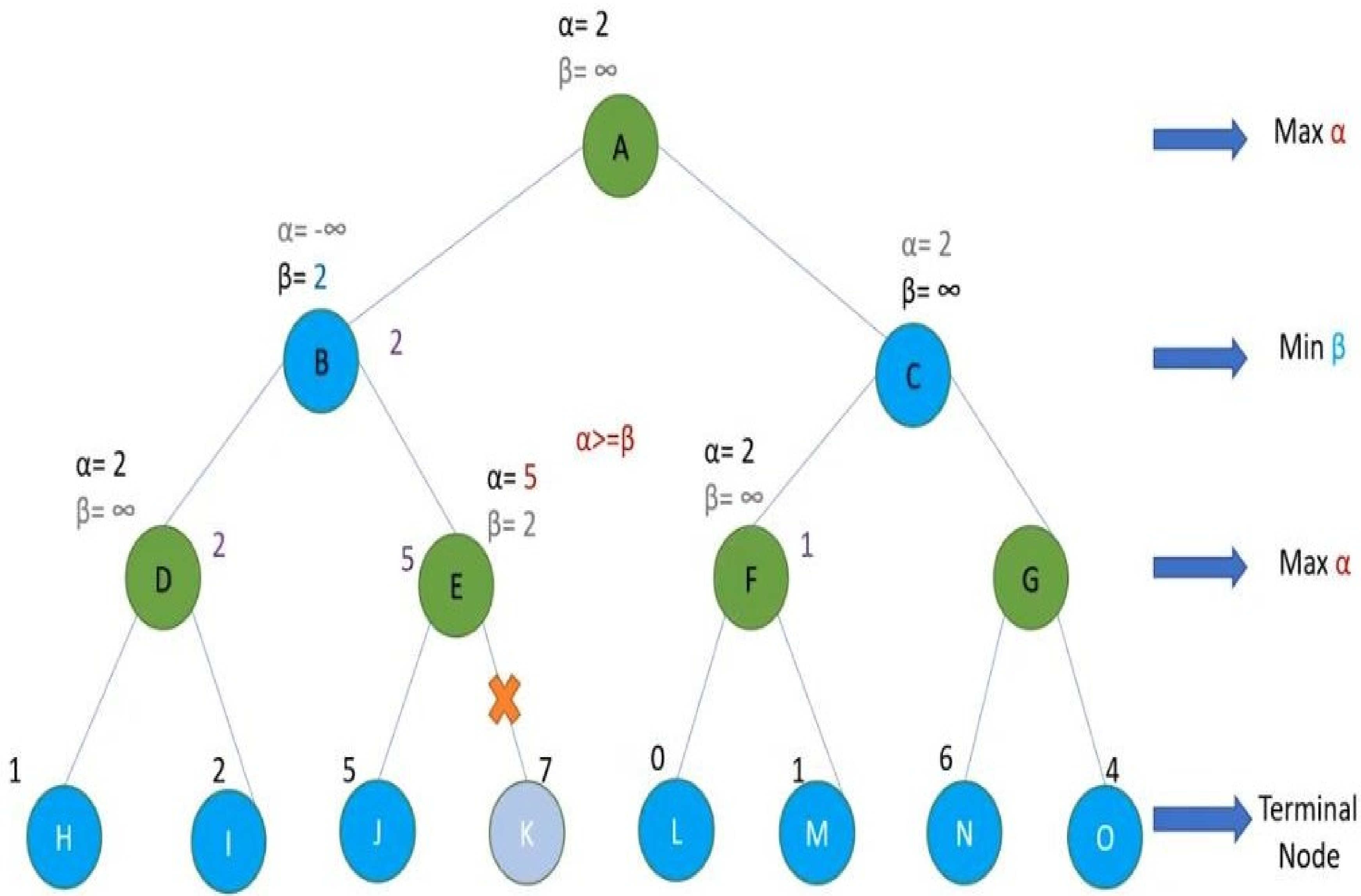
Alpha-Beta pruning: Example



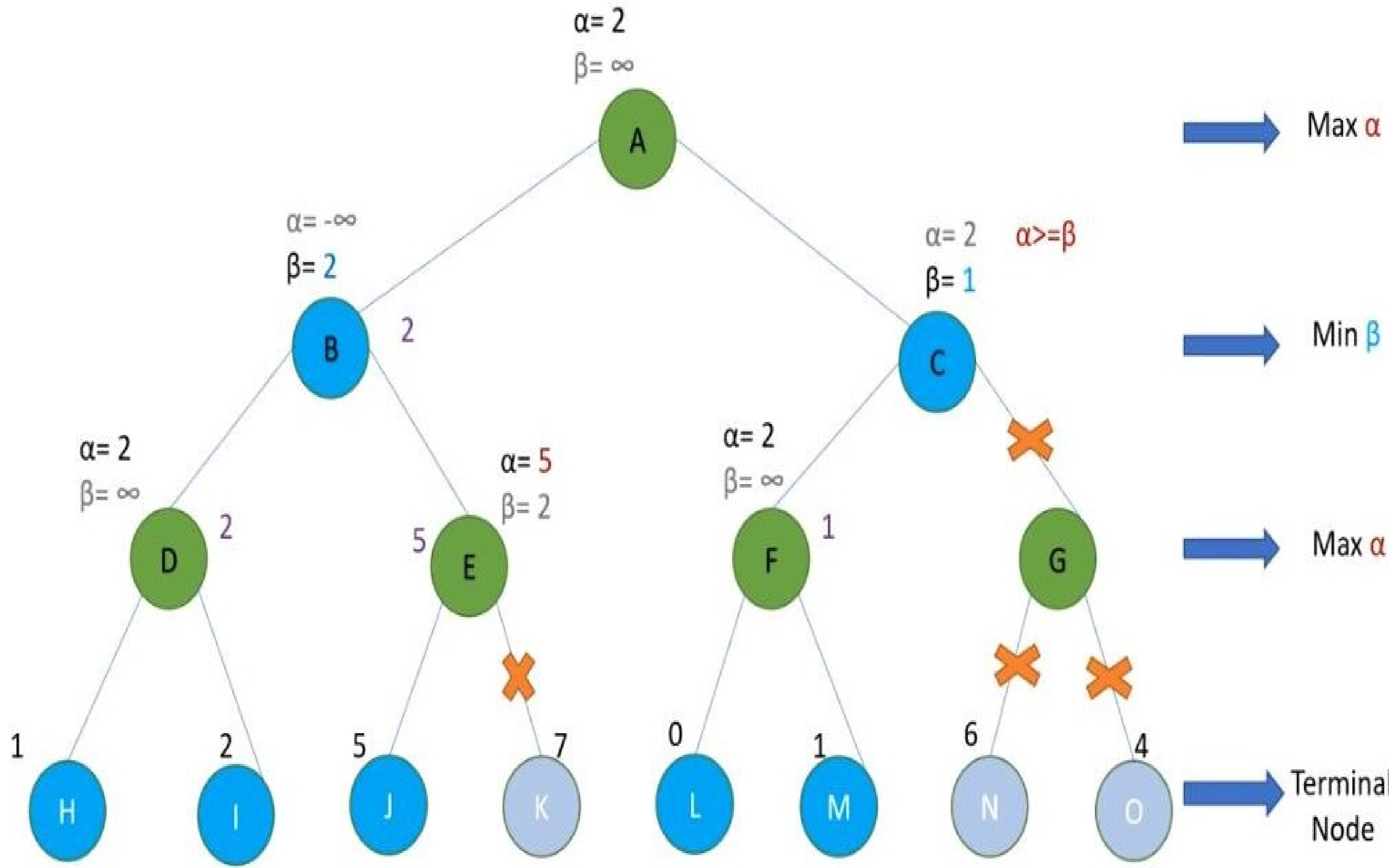
Alpha-Beta pruning: Example



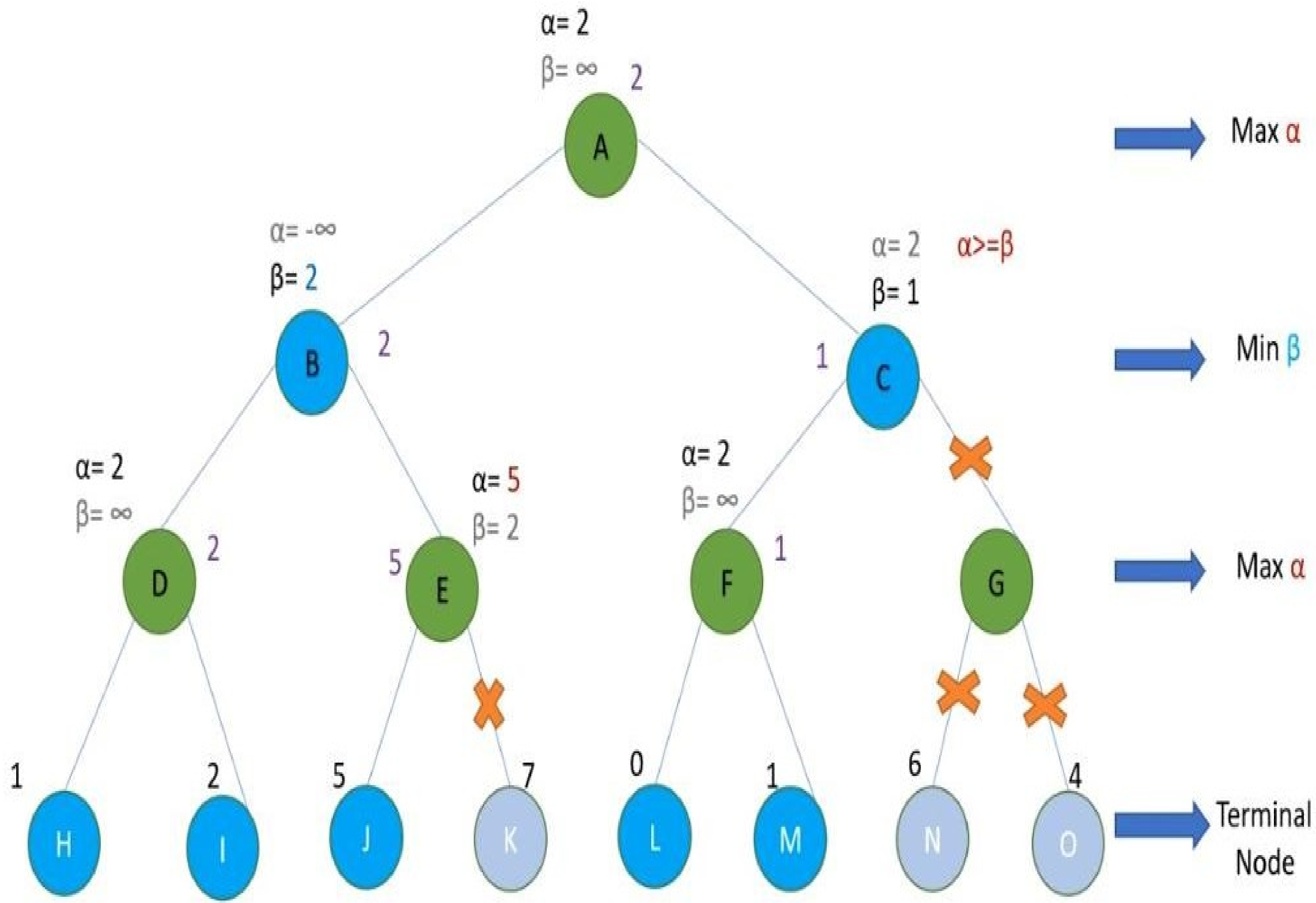
Alpha-Beta pruning: Example



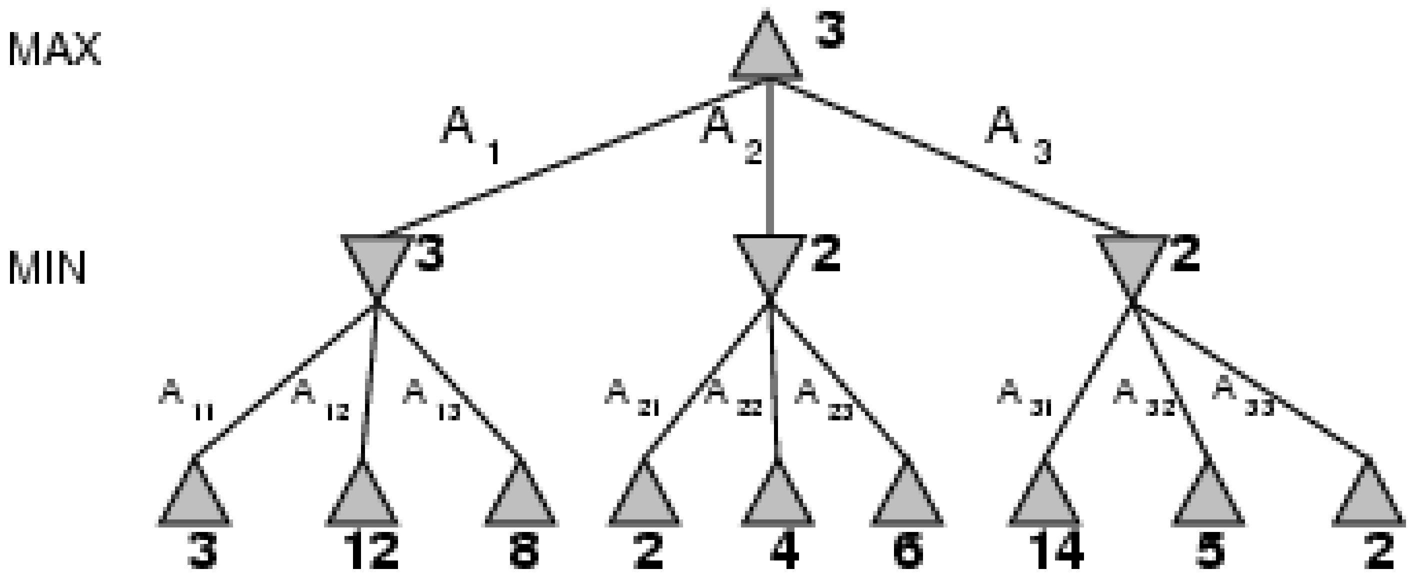
Alpha-Beta pruning: Example



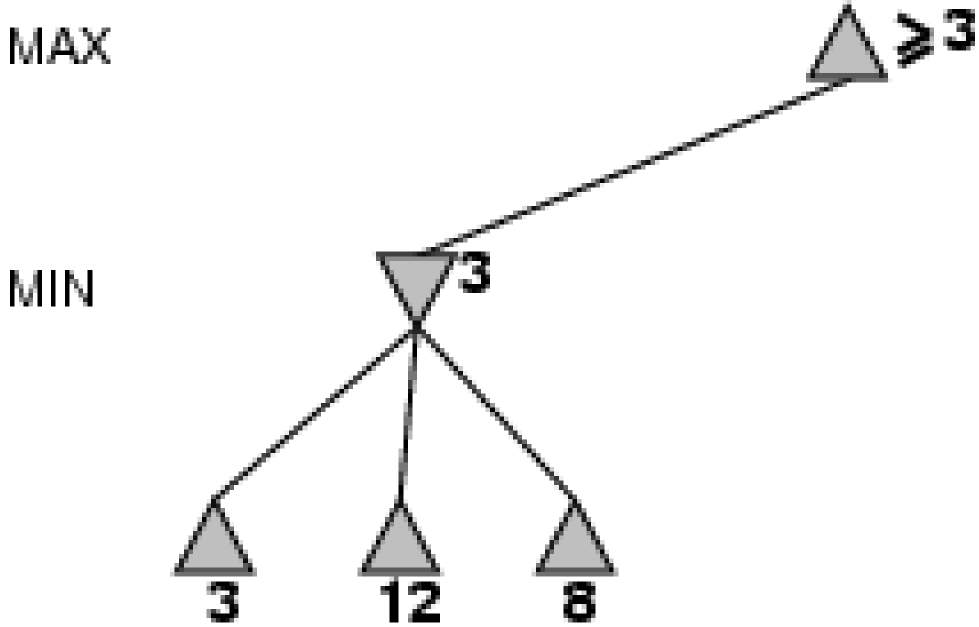
Alpha-Beta pruning: Example



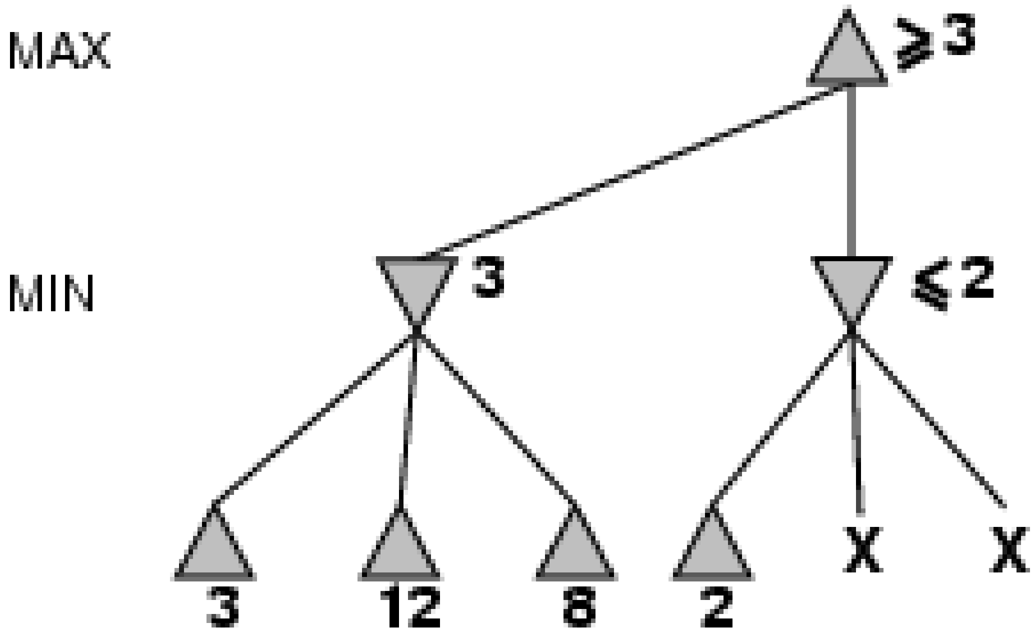
Alpha-Beta pruning: Example



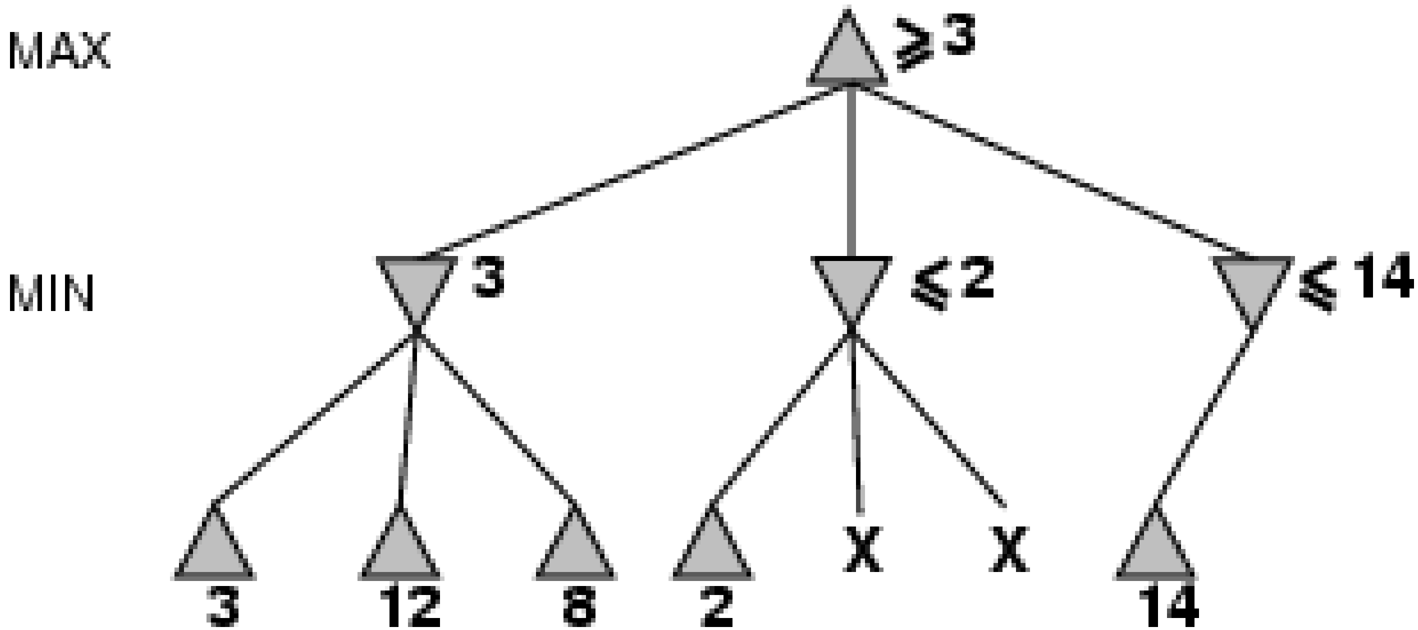
Alpha-Beta pruning: Example



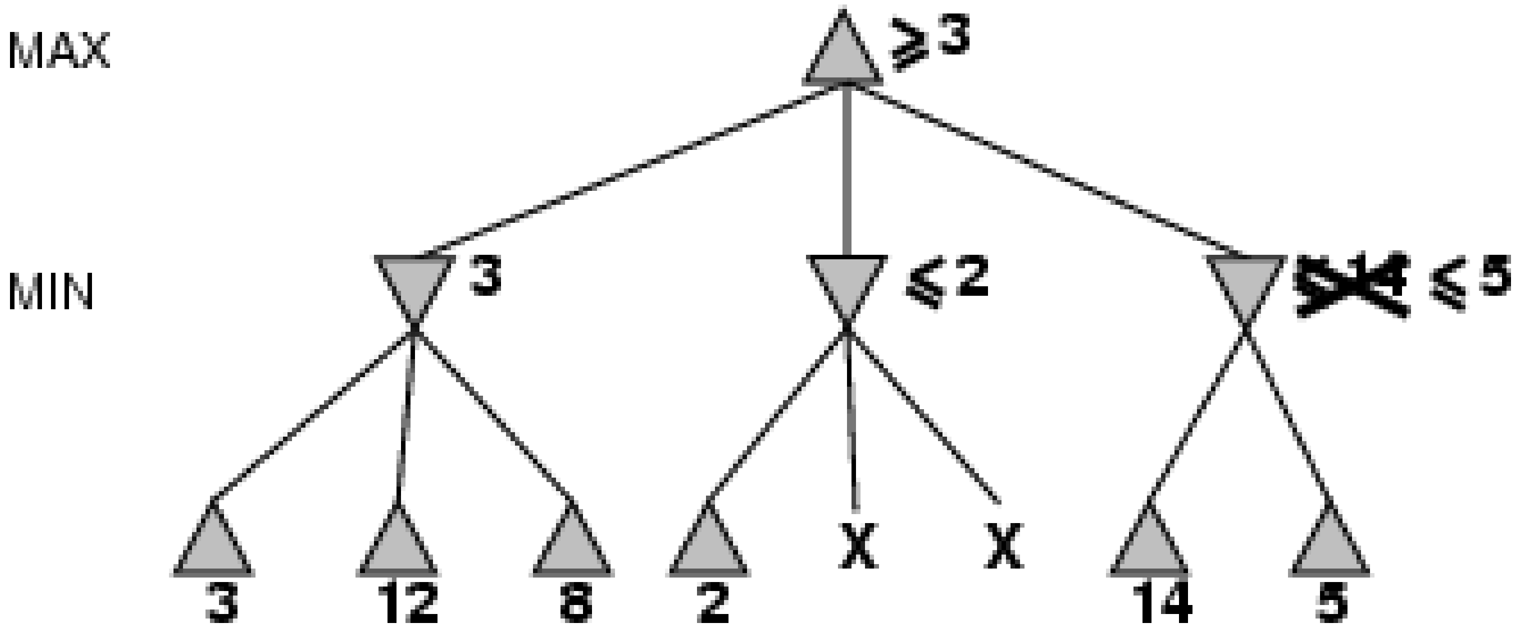
Alpha-Beta pruning: Example



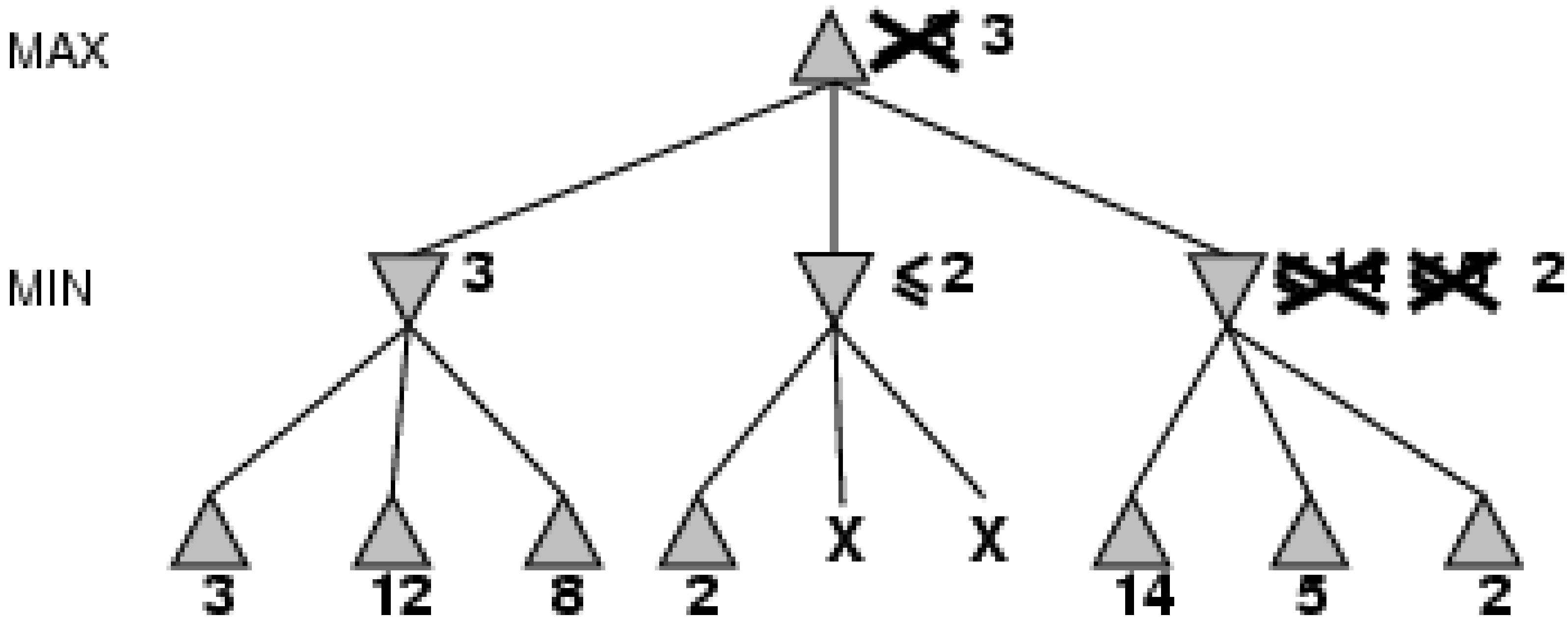
Alpha-Beta pruning: Example



Alpha-Beta pruning: Example



Alpha-Beta pruning: Example



Properties of α-β

- Pruning does not affect the final result
- Good move ordering improves the effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{d/2})$

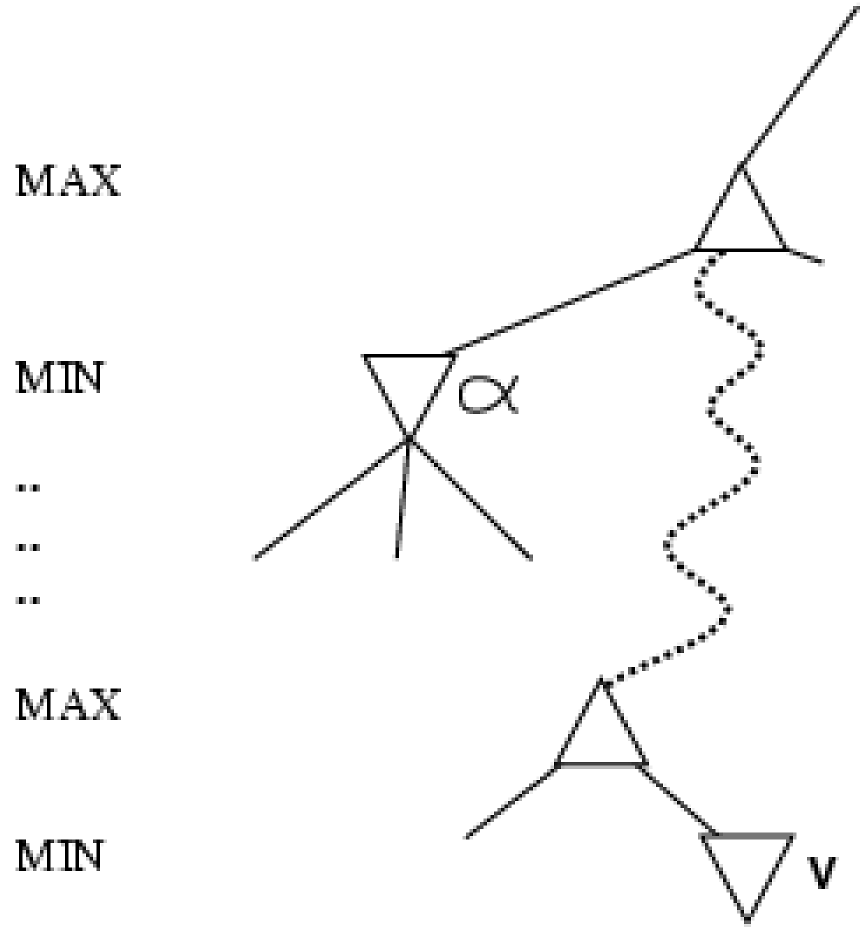
Why is it called α-β?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*

- If v is worse than α, *max* will avoid it

  → prune that branch

- Define β similarly for *min*



The α-β algorithm

function ALPHA-BETA-SEARCH(*state*) **returns** *an action*
  inputs: *state*, current state in game

  $v \leftarrow$ MAX-VALUE(*state*, $-\infty, +\infty$)
  return the *action* in SUCCESSORS(*state*) with value v

---

function MAX-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

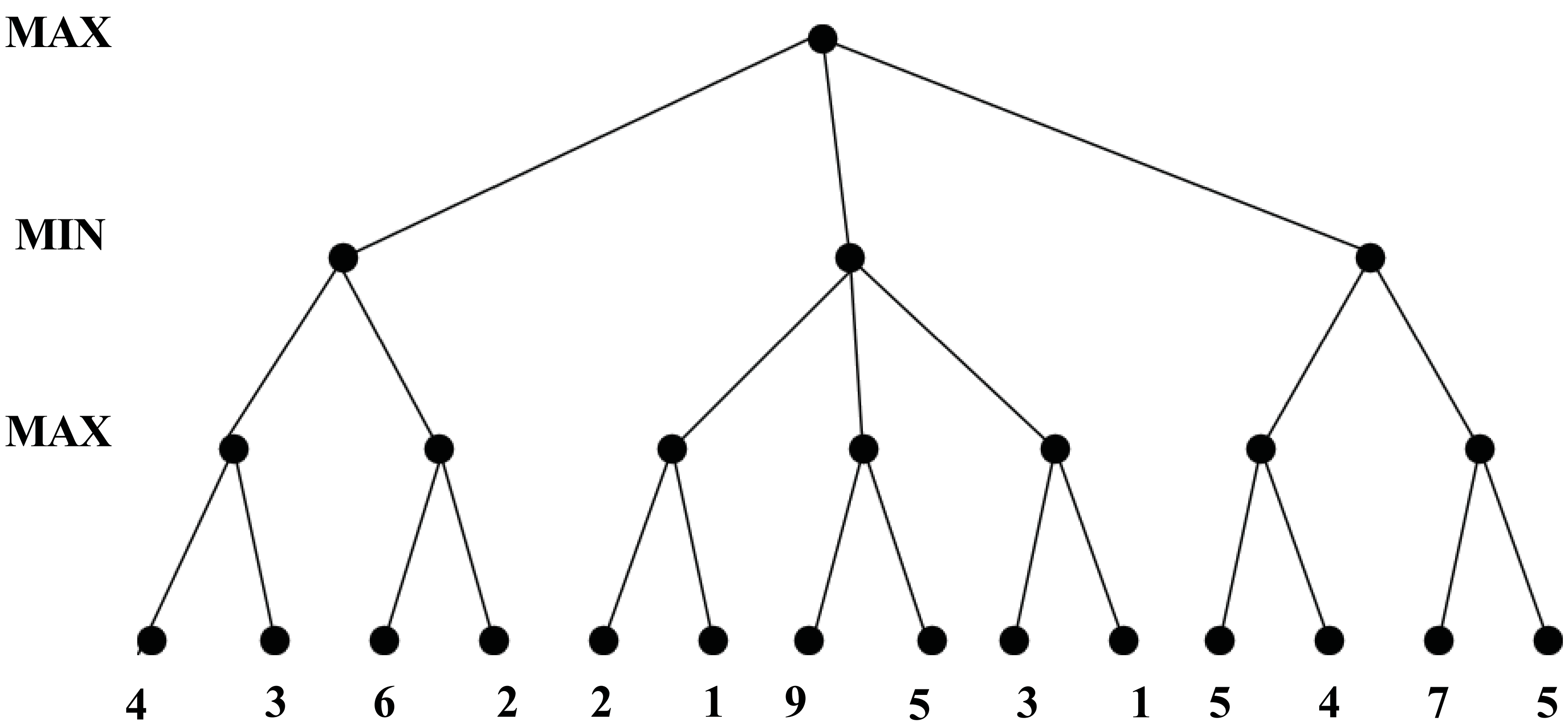
  if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

The α-β algorithm

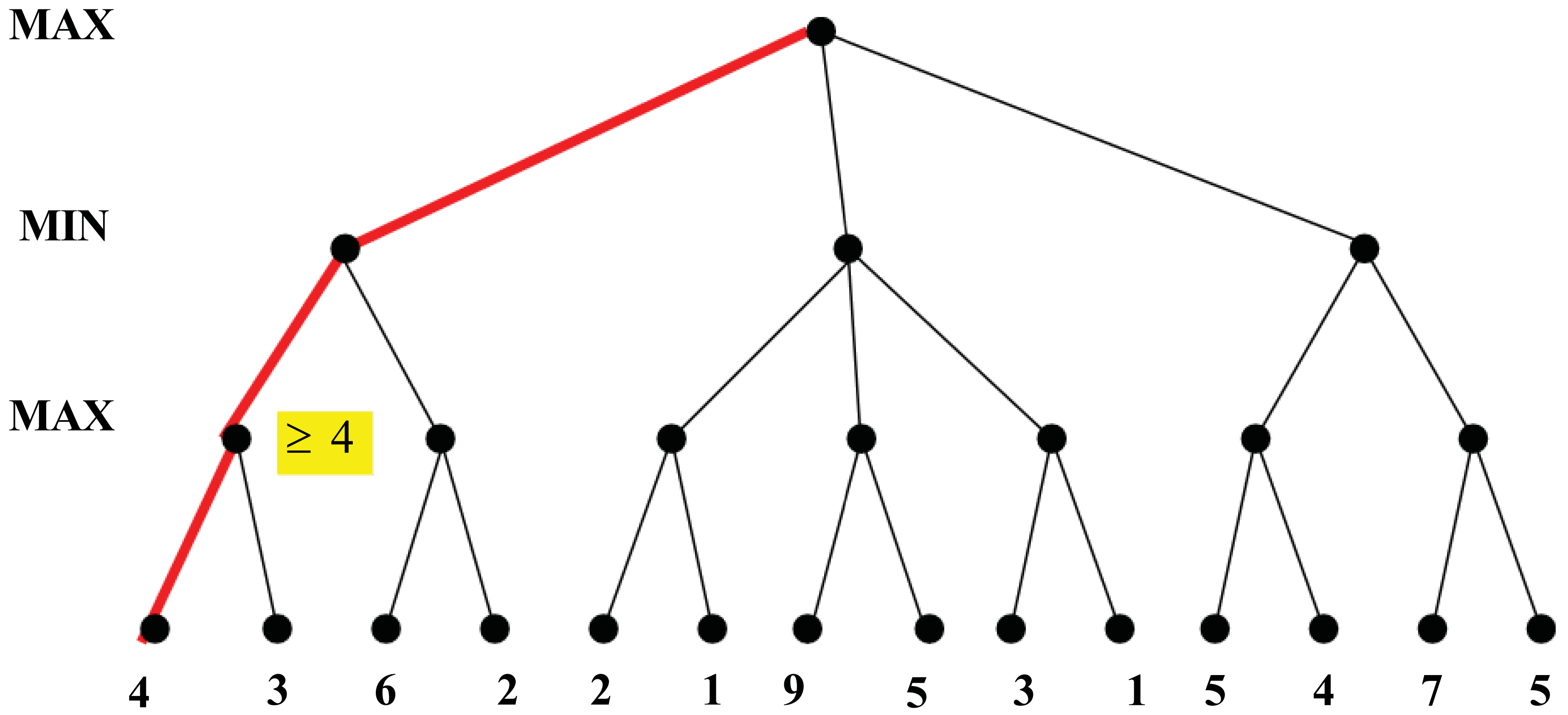
function MIN-VALUE(*state*, α, β) **returns** *a utility value*
  inputs: *state*, current state in game
    α, the value of the best alternative for MAX along the path to *state*
    β, the value of the best alternative for MIN along the path to *state*

  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, α, β))
    if $v \leq \alpha$ **then return** v
    $\beta \leftarrow$ MIN(β, v)
  return v

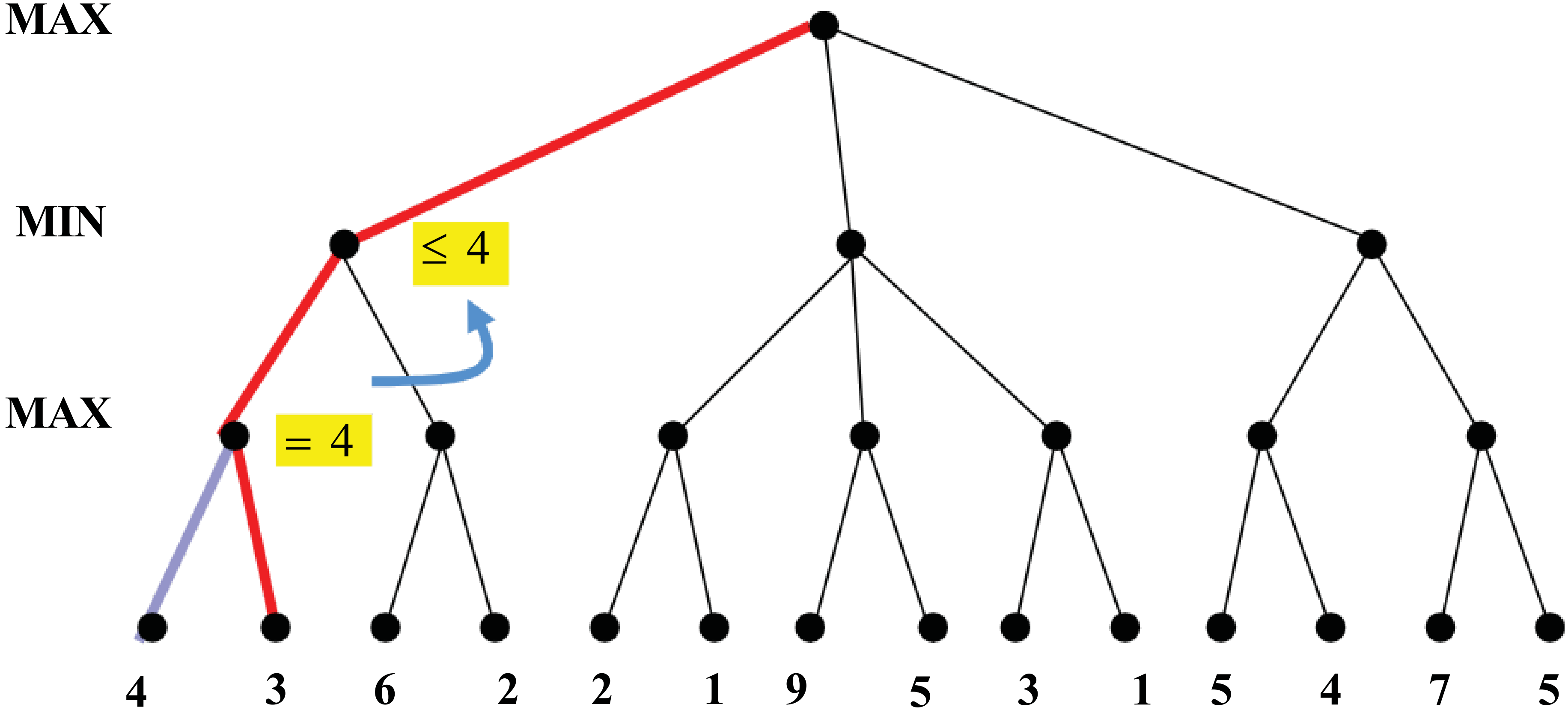
Alpha beta pruning. Example



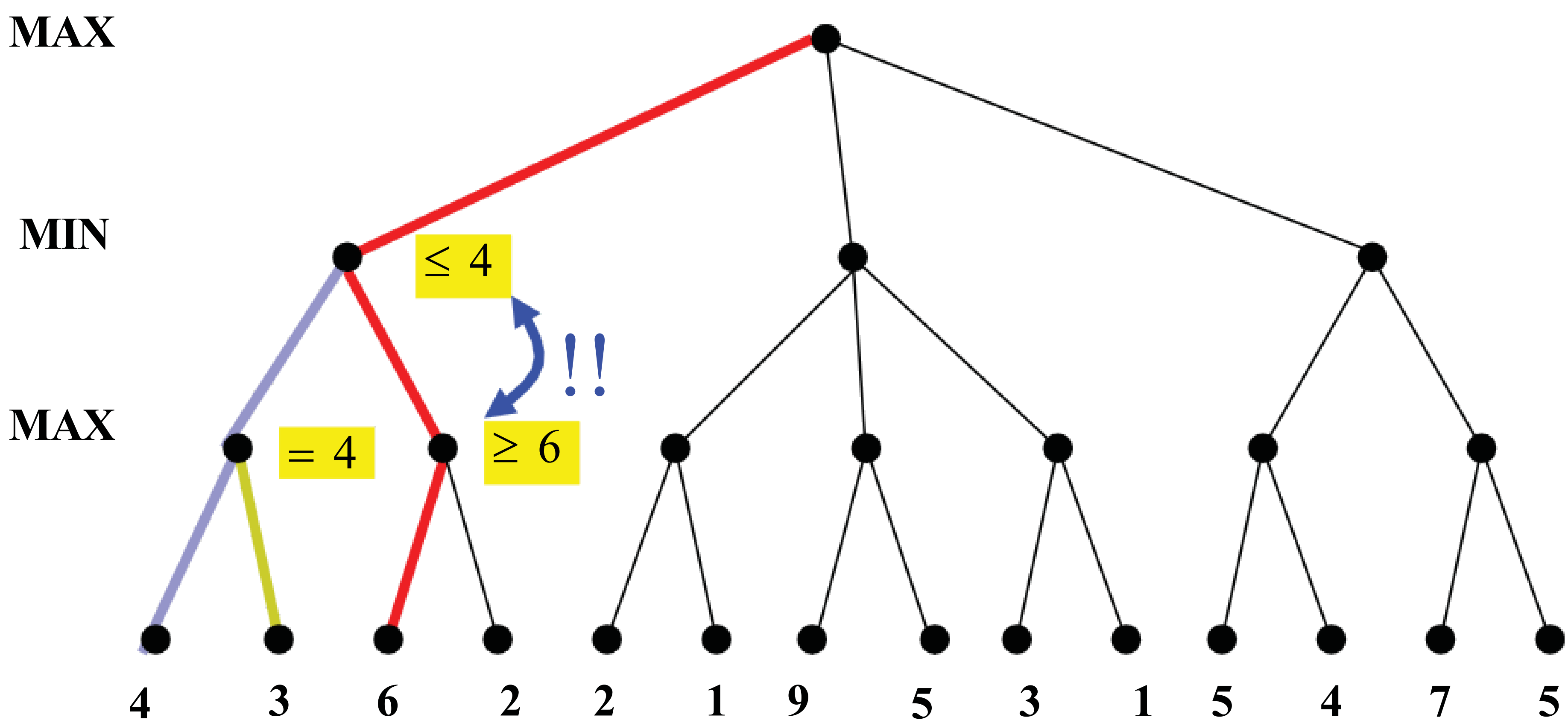
Alpha beta pruning. Example



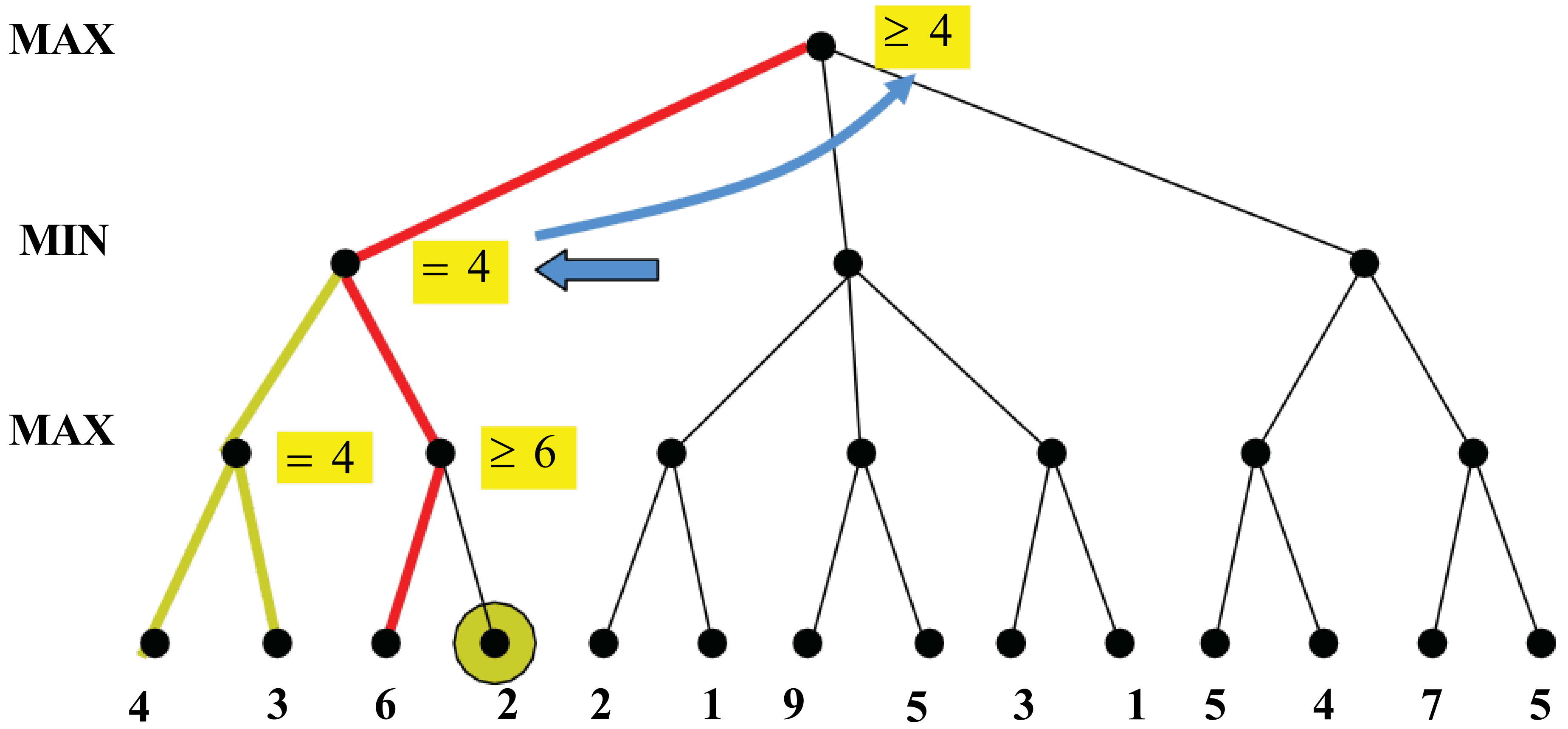
Alpha beta pruning. Example



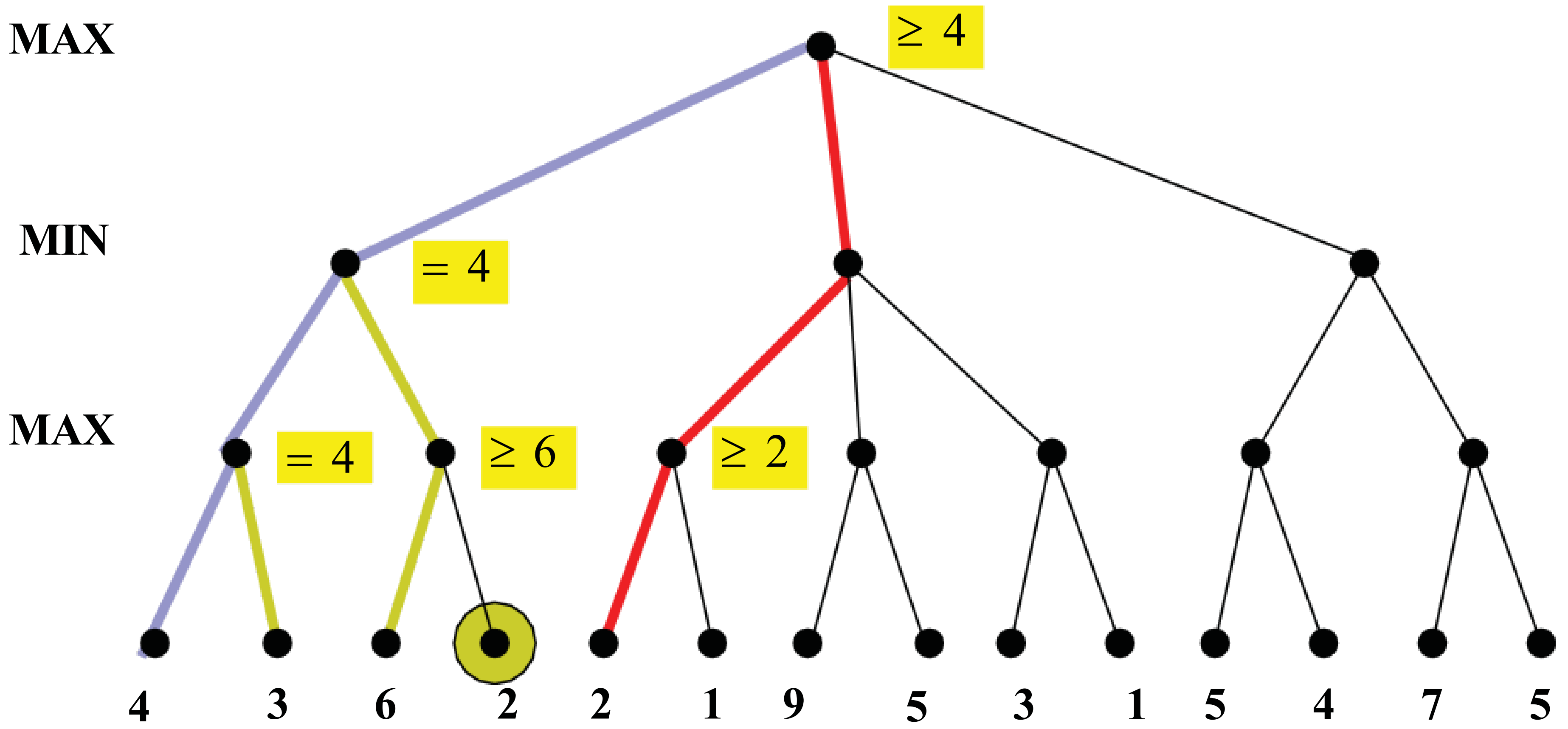
Alpha beta pruning. Example



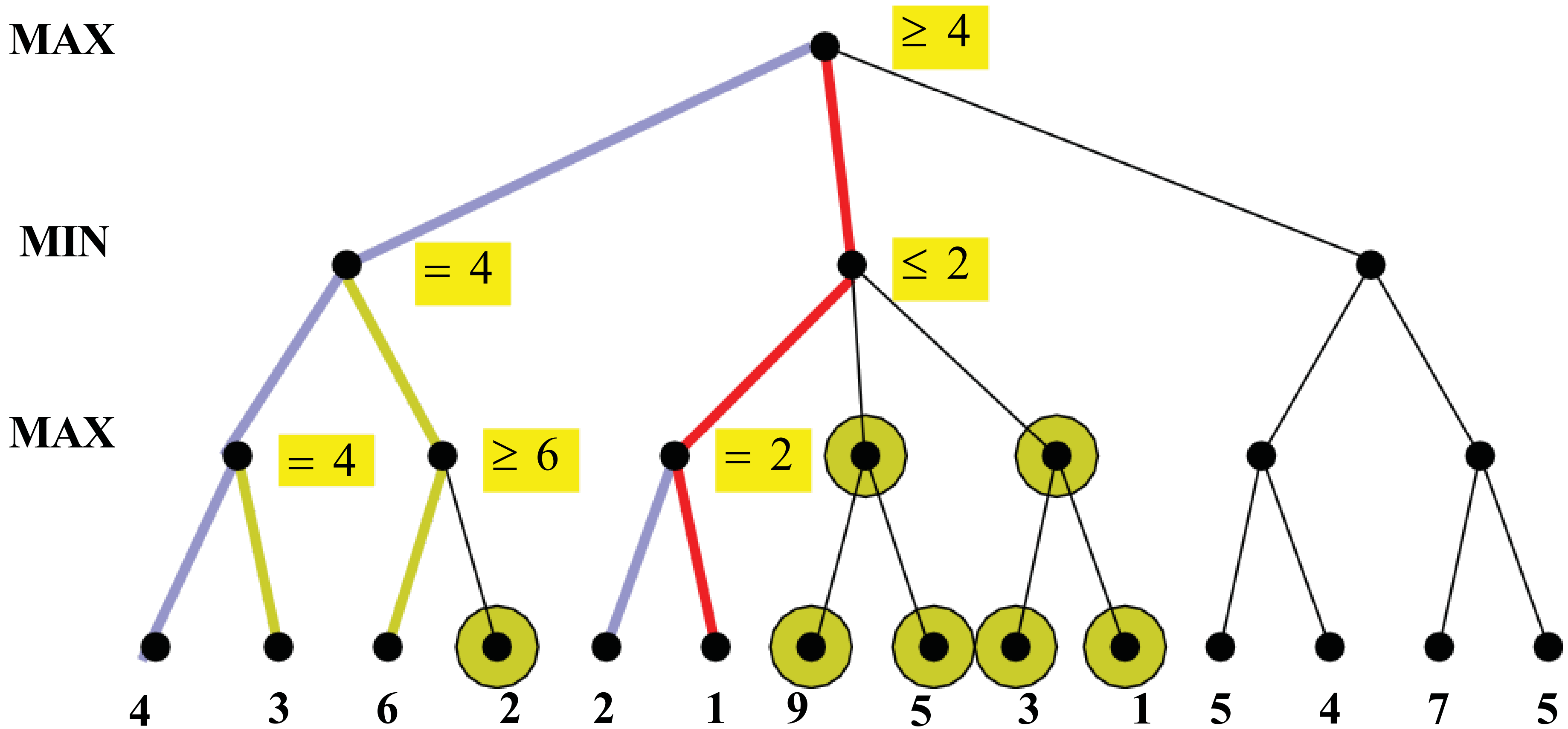
Alpha beta pruning. Example



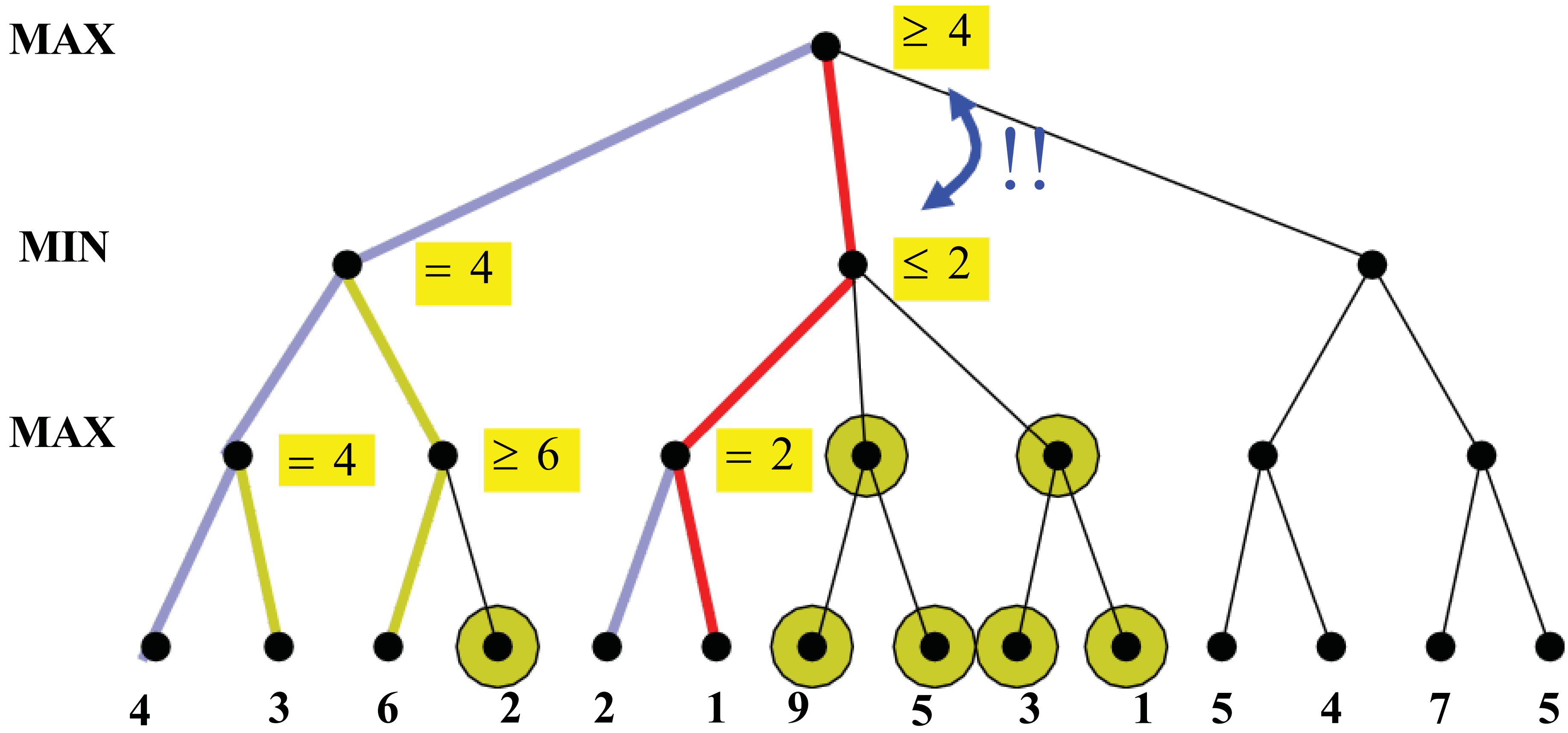
Alpha beta pruning. Example



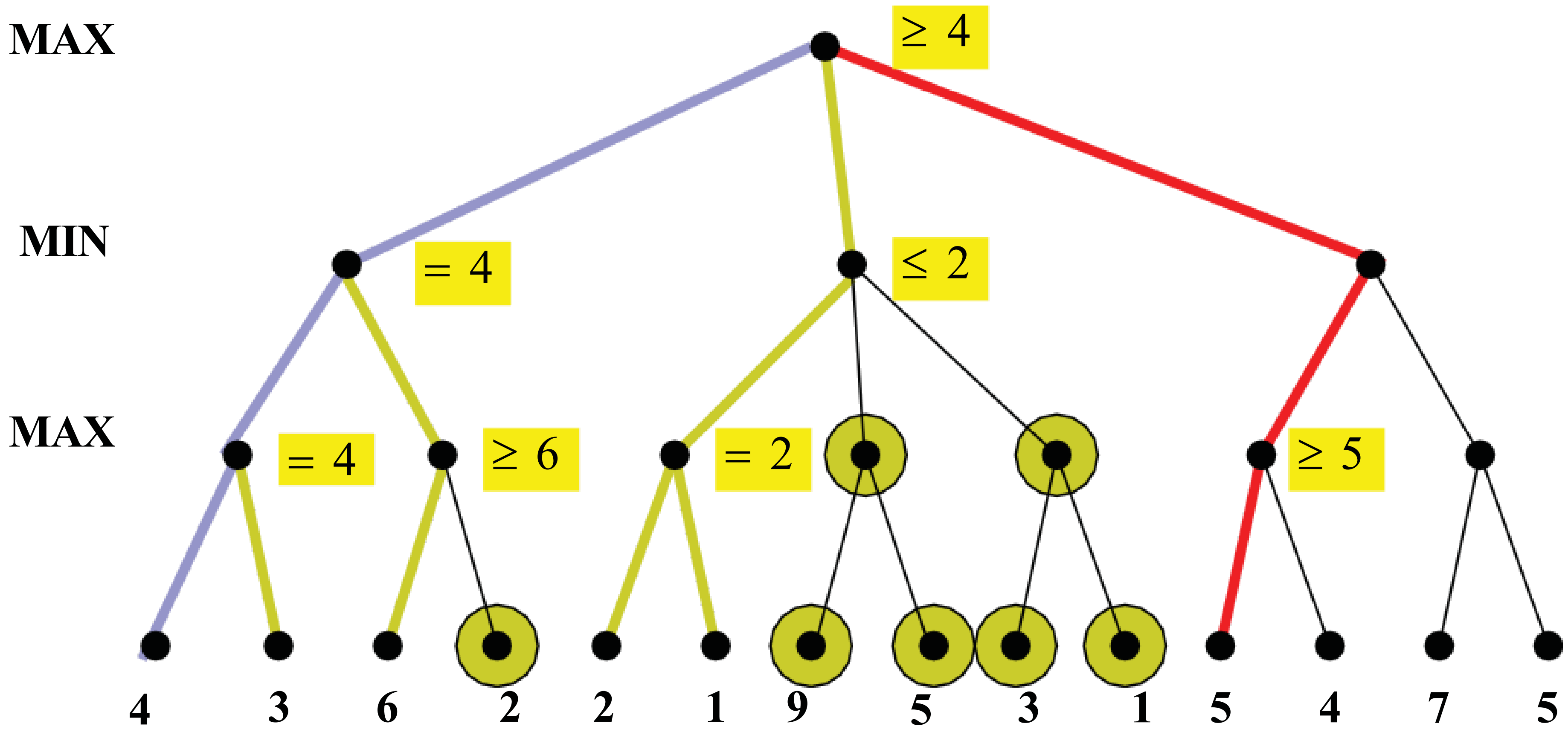
Alpha beta pruning. Example



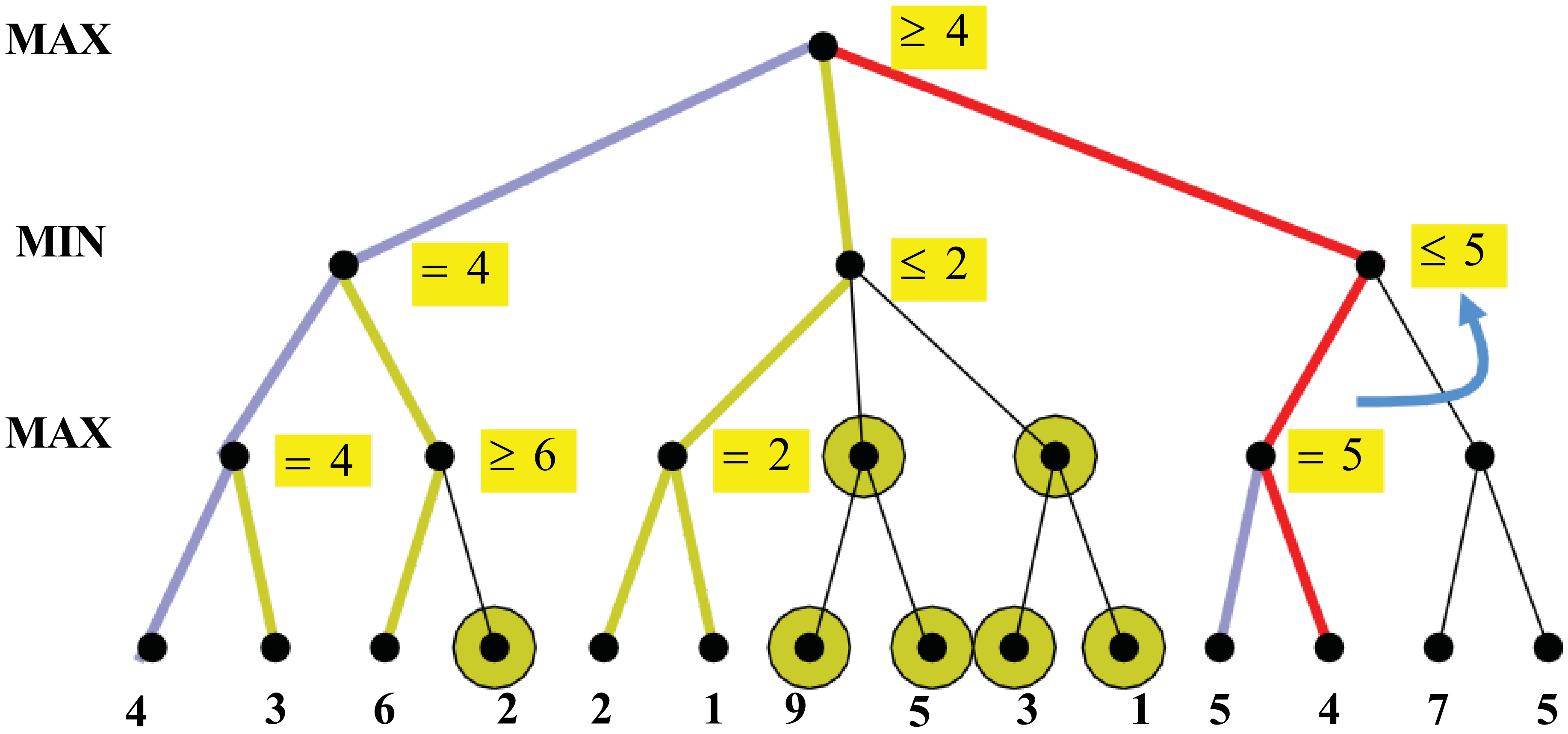
Alpha beta pruning. Example



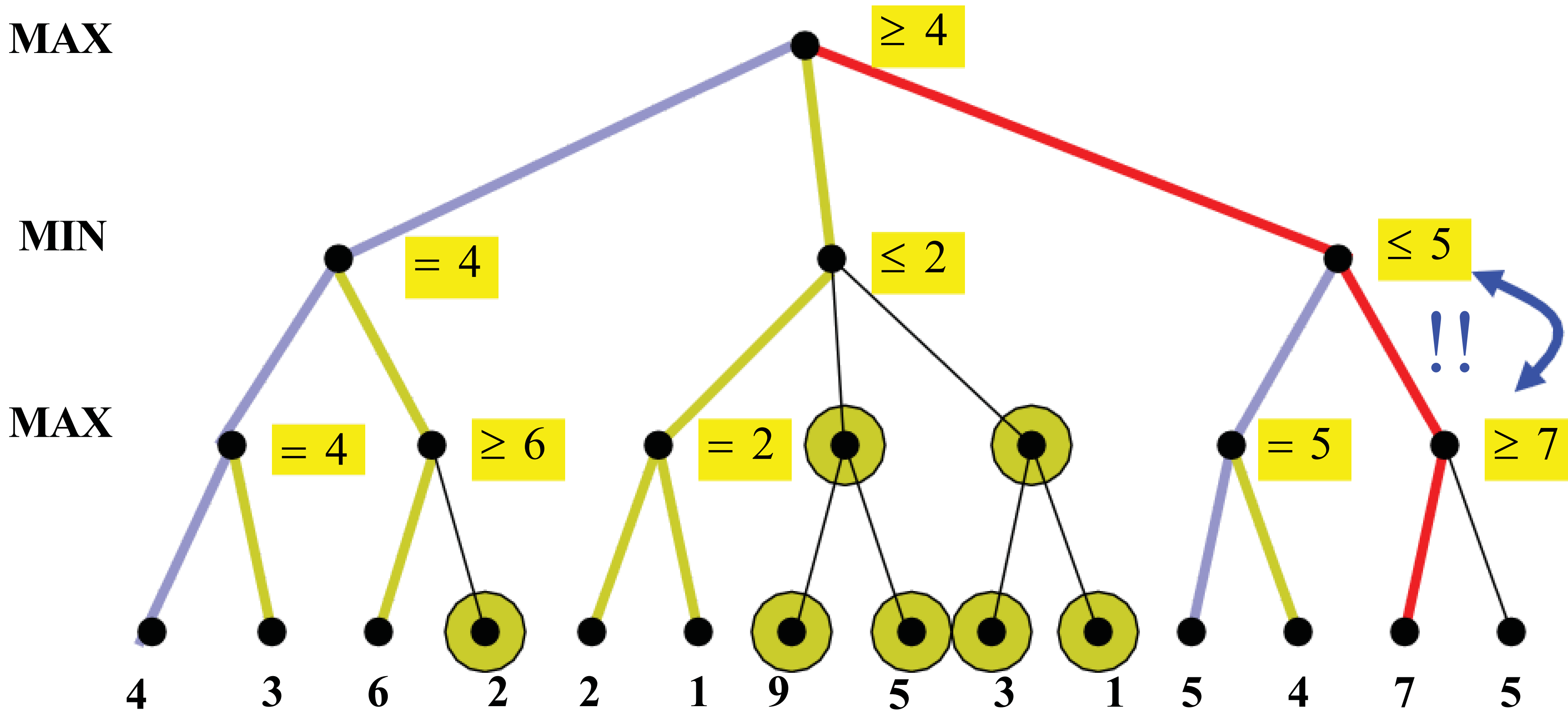
Alpha beta pruning. Example



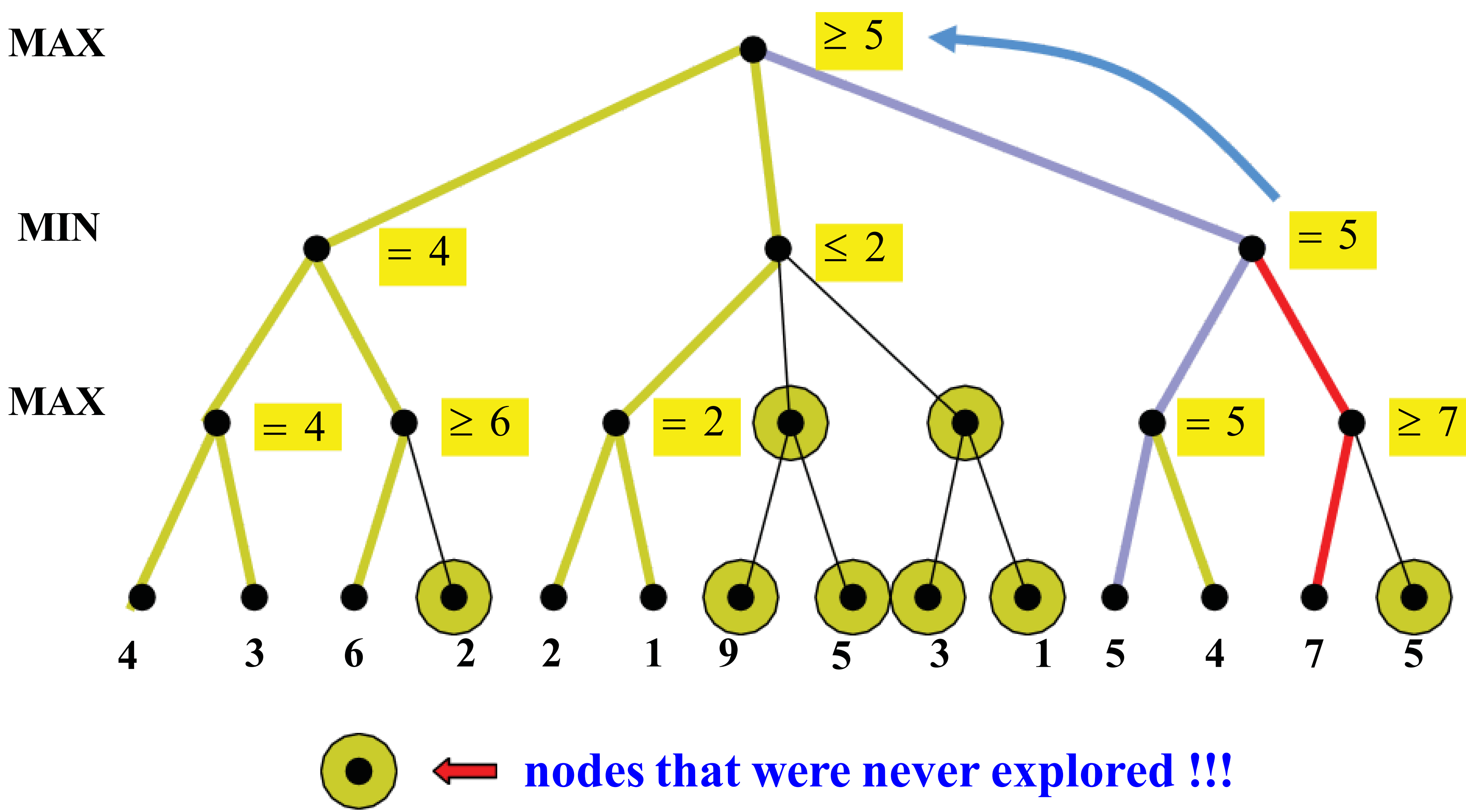
Alpha beta pruning. Example



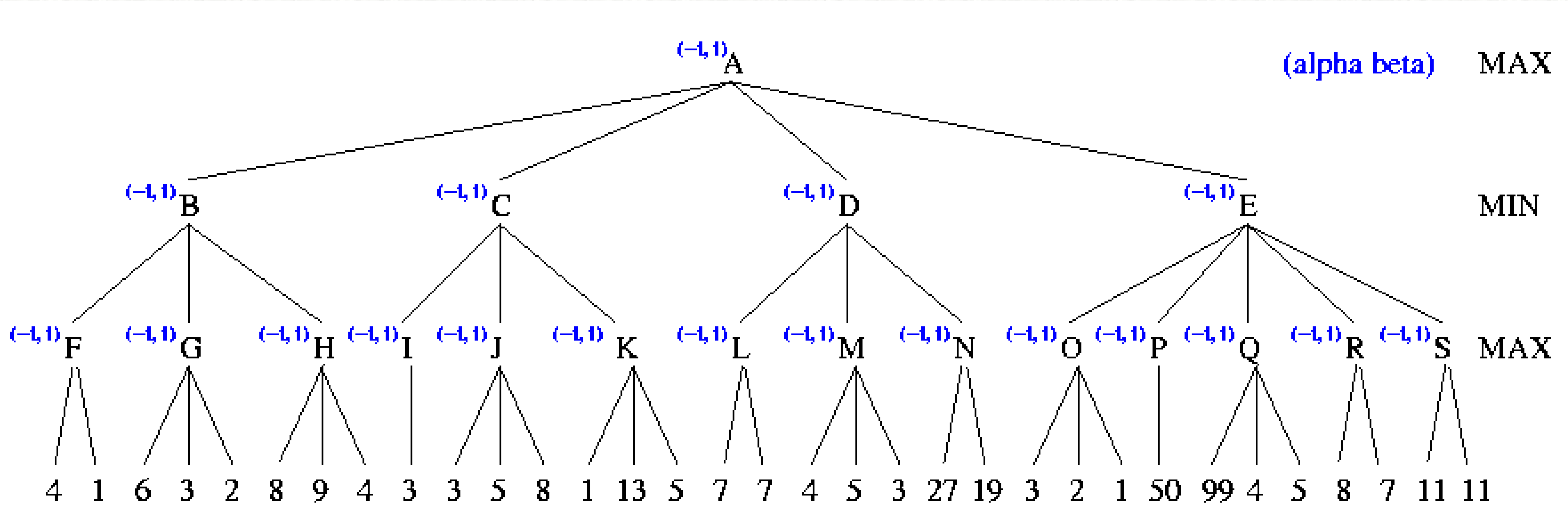
Alpha beta pruning. Example



Alpha beta pruning. Example

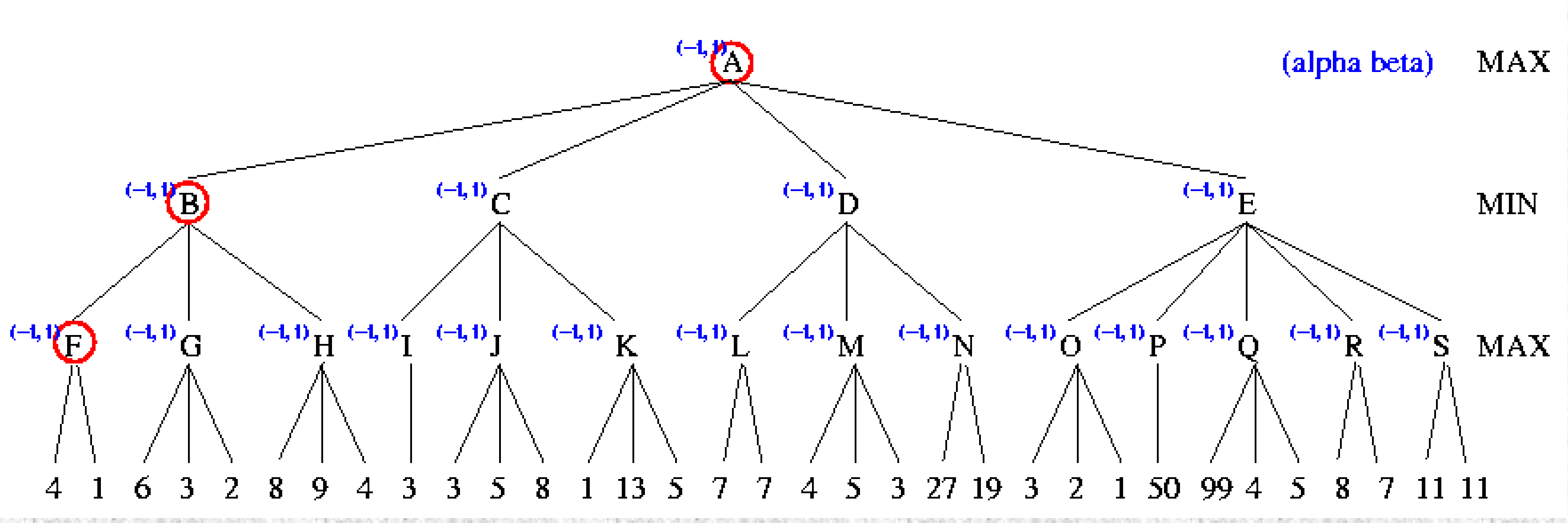


Example

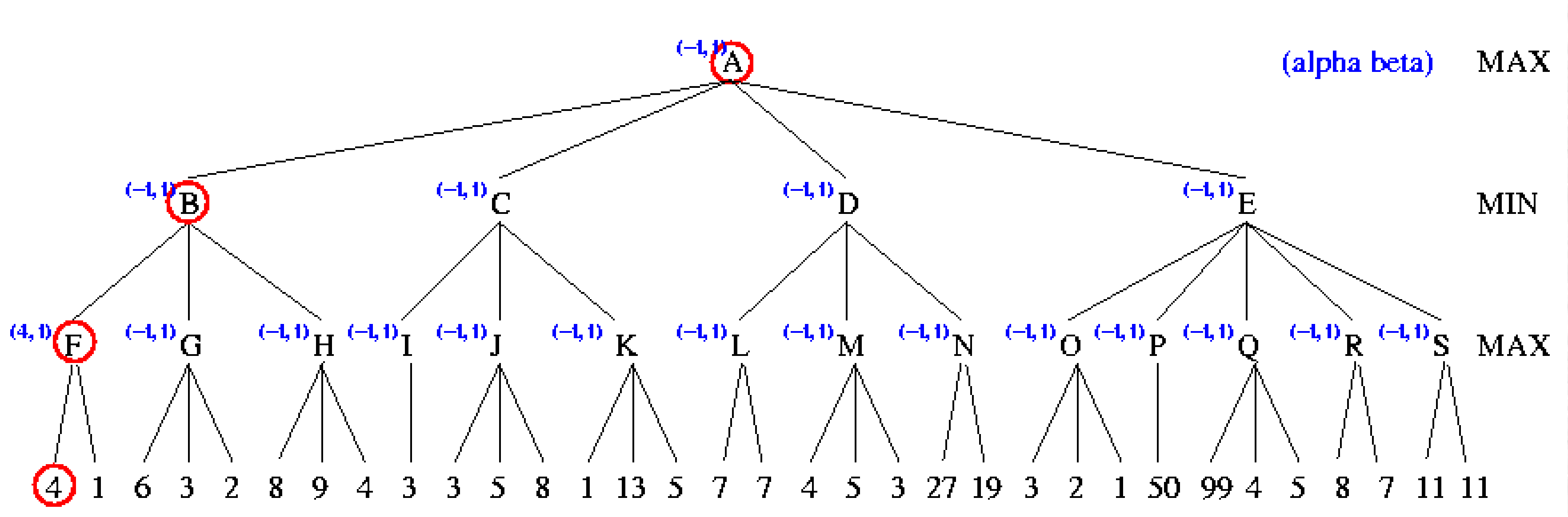


A
(alpha beta)
MAX
B
C
D
E
MIN
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example

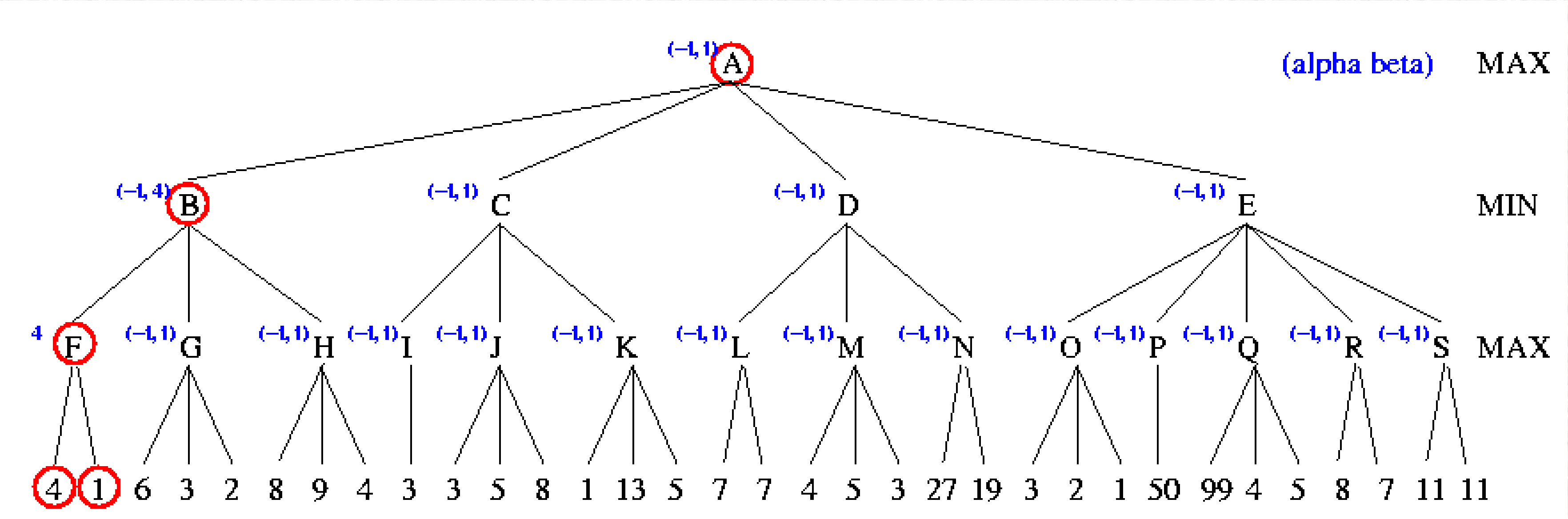

(−I, I) A
(alpha beta)
MAX
(−I, I) B
(−I, I) C
(−I, I) D
(−I, I) E
MIN
(−I, I) F
(−I, I) G
(−I, I) H
(−I, I) I
(−I, I) J
(−I, I) K
(−I, I) L
(−I, I) M
(−I, I) N
(−I, I) O
(−I, I) P
(−I, I) Q
(−I, I) R
(−I, I) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



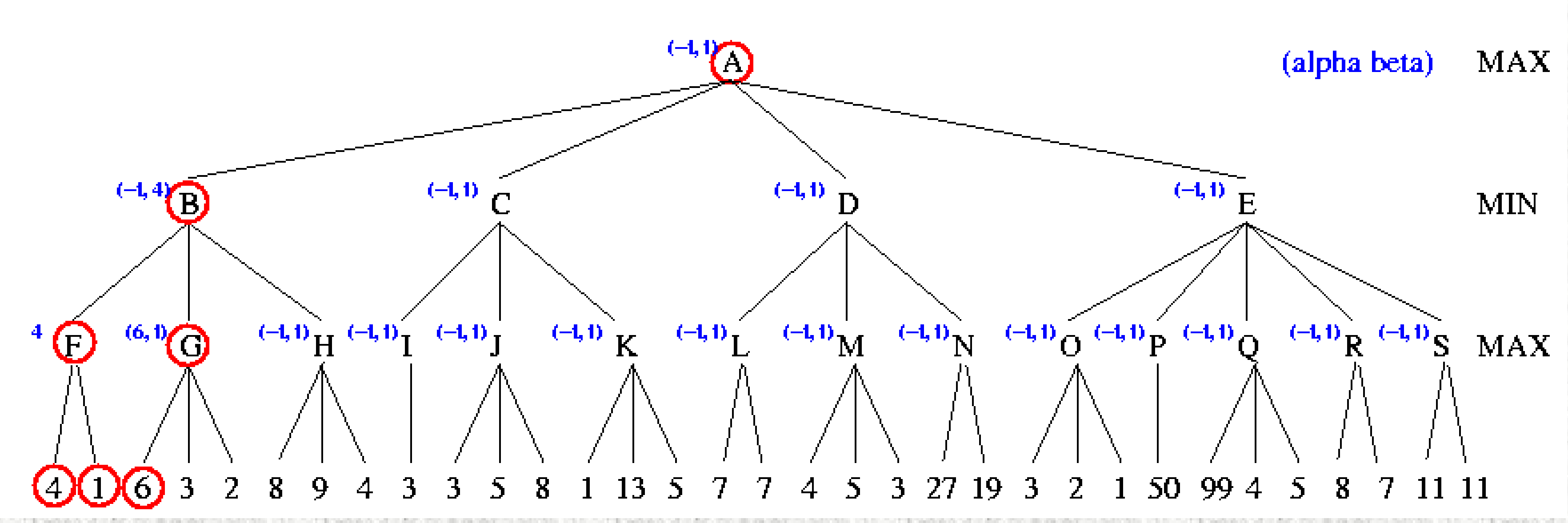
(alpha beta)
MAX
MIN
MAX
(−I, I) A
(−I, I) B
(−I, I) C
(−I, I) D
(−I, I) E
(4, I) F
(−I, I) G
(−I, I) H
(−I, I) I
(−I, I) J
(−I, I) K
(−I, I) L
(−I, I) M
(−I, I) N
(−I, I) O
(−I, I) P
(−I, I) Q
(−I, I) R
(−I, I) S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



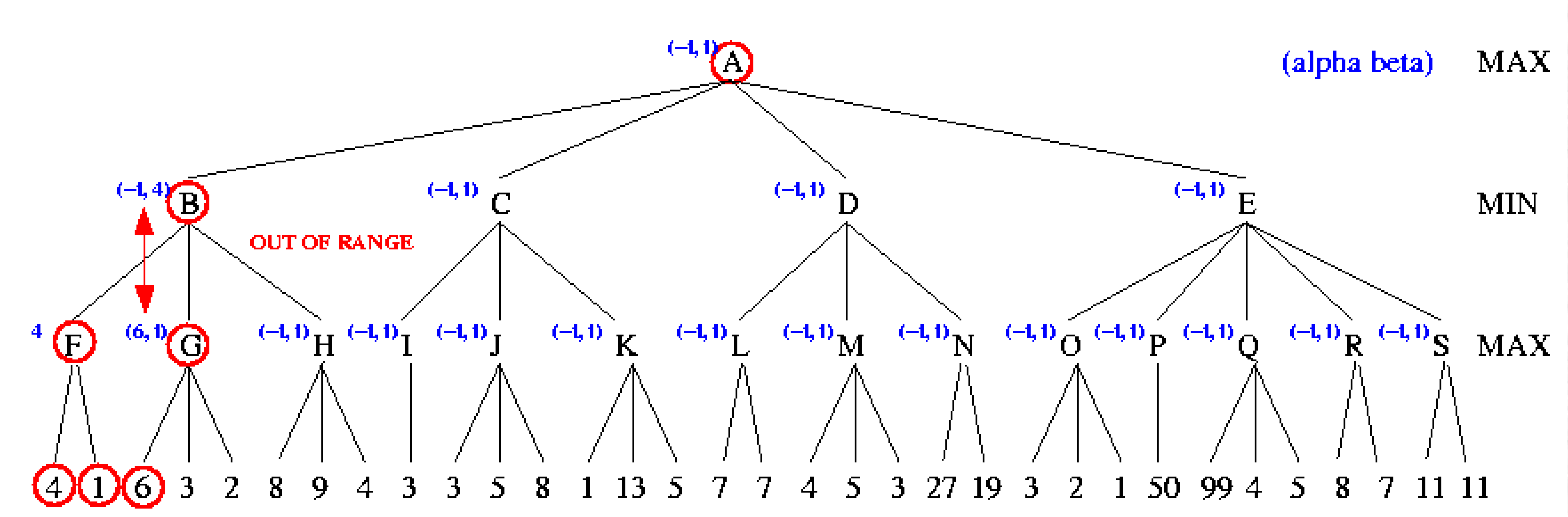
(alpha beta)
MAX
MIN
MAX
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



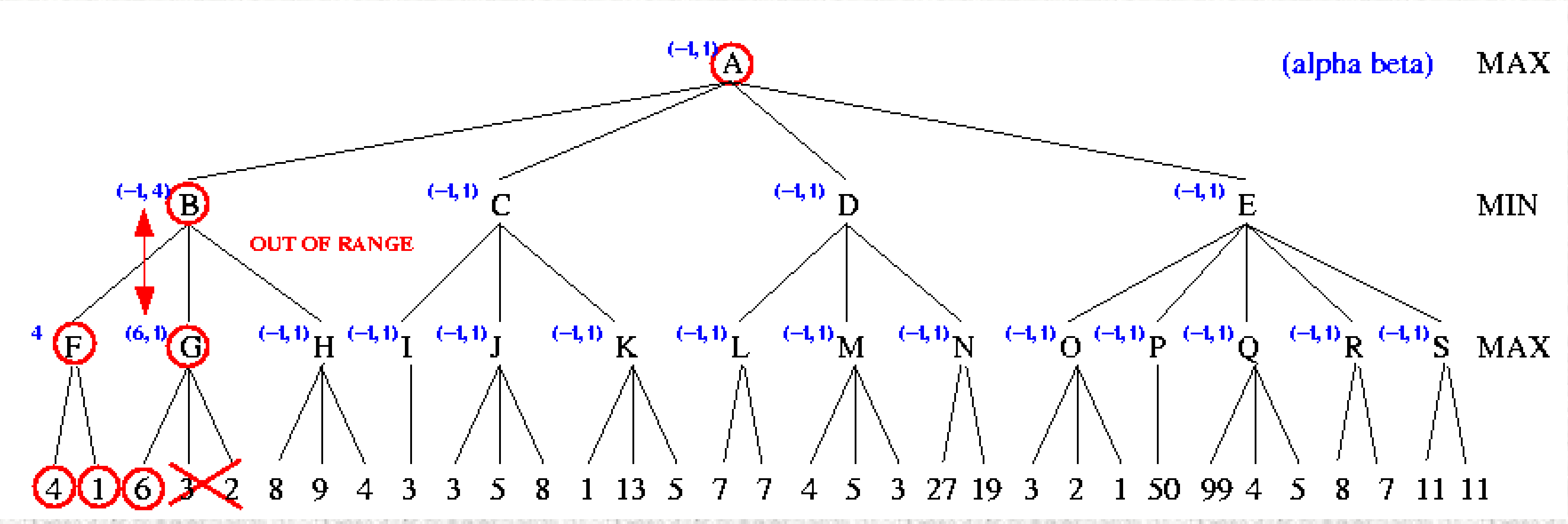
(alpha beta)
MAX
MIN
MAX
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
(−I, I)
(−I, 4)
(6, I)
4
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



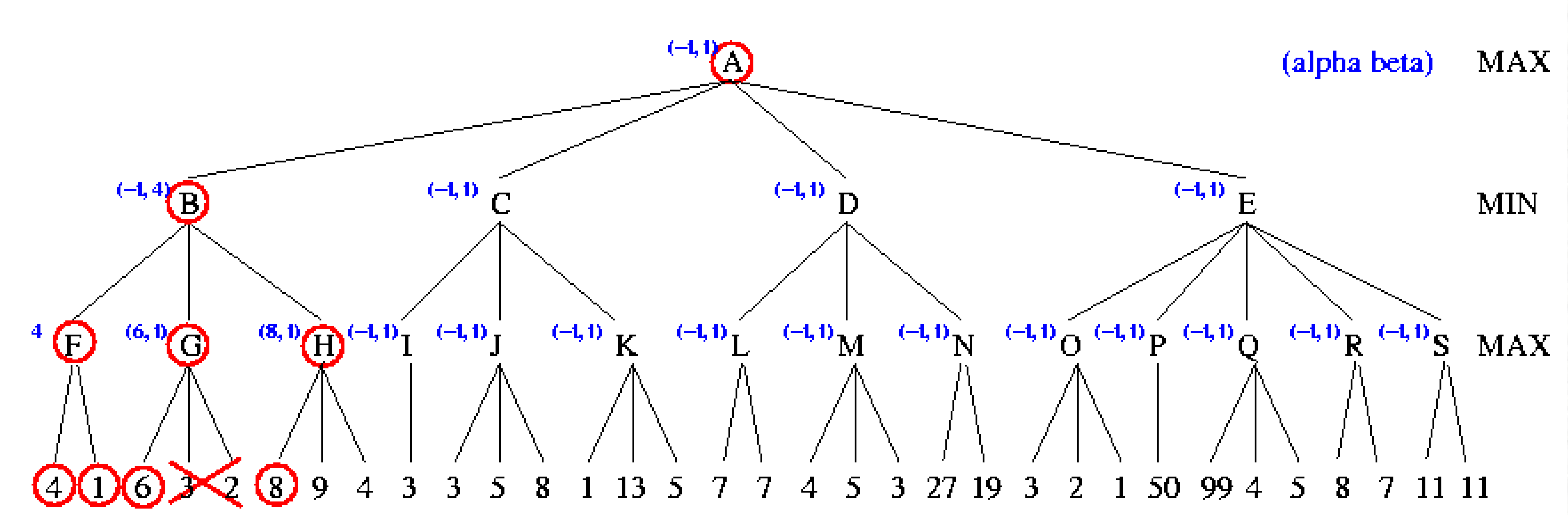
(alpha beta)
MAX
MIN
MAX
(−I, I)
A
(−I, 4)
B
(−I, I)
C
(−I, I)
D
(−I, I)
E
OUT OF RANGE
4
F
(6, I)
G
(−I, I)
H
(−I, I)
I
(−I, I)
J
(−I, I)
K
(−I, I)
L
(−I, I)
M
(−I, I)
N
(−I, I)
O
(−I, I)
P
(−I, I)
Q
(−I, I)
R
(−I, I)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example

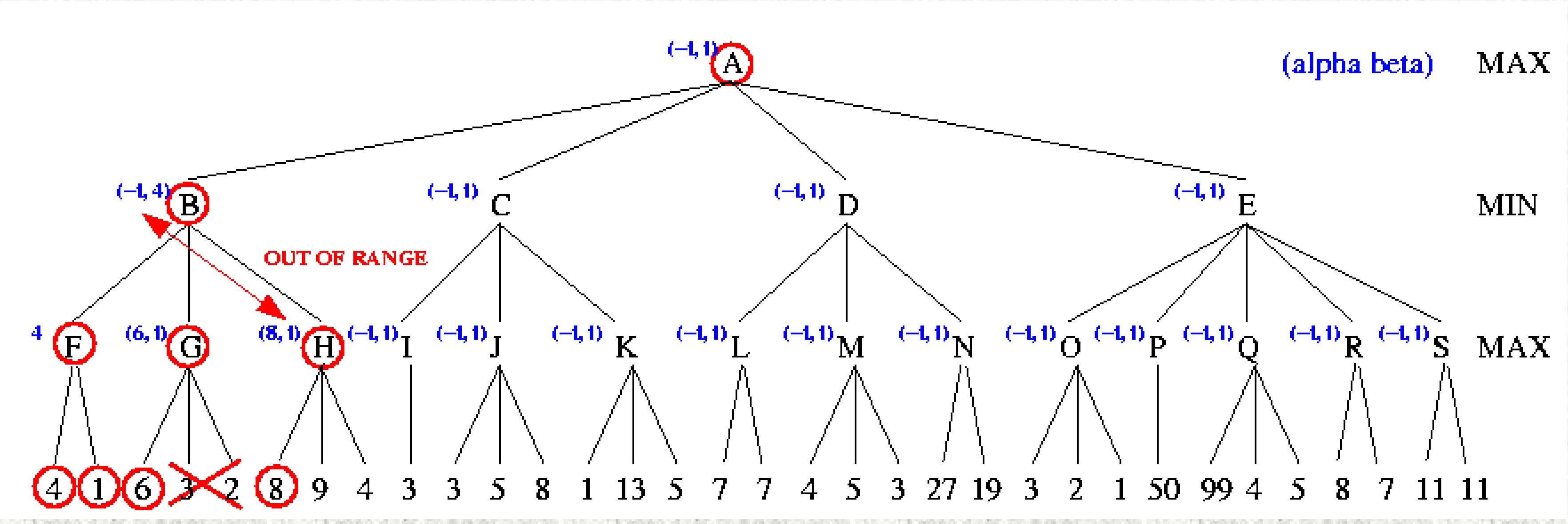


(alpha beta)
MAX
MIN
MAX
OUT OF RANGE
(−I, I)
A
(−I, 4)
B
(−I, I)
C
(−I, I)
D
(−I, I)
E
4
F
(6, I)
G
(−I, I)
H
(−I, I)
I
(−I, I)
J
(−I, I)
K
(−I, I)
L
(−I, I)
M
(−I, I)
N
(−I, I)
O
(−I, I)
P
(−I, I)
Q
(−I, I)
R
(−I, I)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example

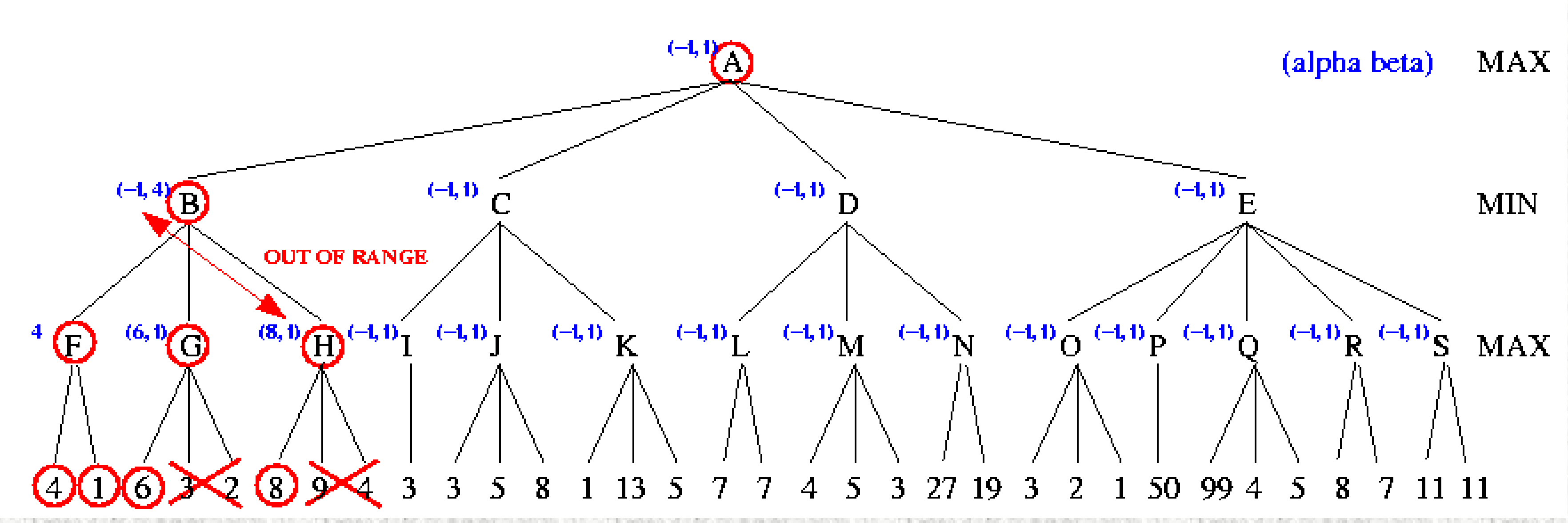

(alpha beta)
MAX
MIN
MAX
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



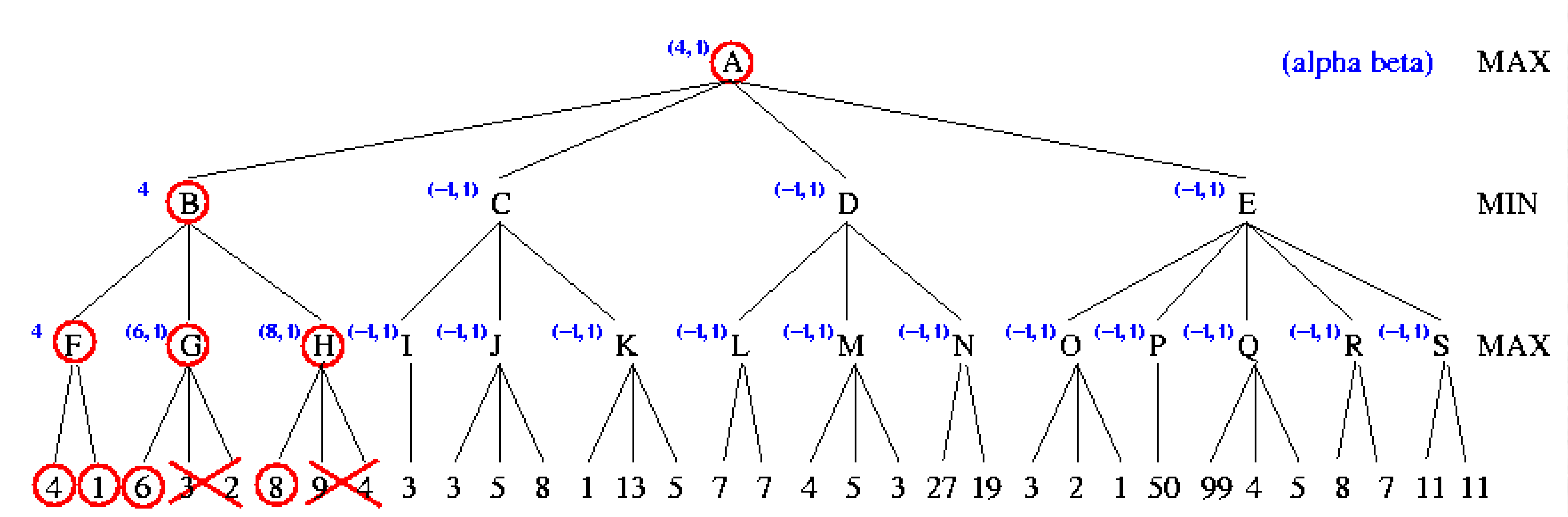
(alpha beta)
MAX
MIN
MAX
(−I, I)
A
(−I, 4)
B
(−I, I)
C
(−I, I)
D
(−I, I)
E
OUT OF RANGE
4
F
(6, I)
G
(8, I)
H
(−I, I)
I
(−I, I)
J
(−I, I)
K
(−I, I)
L
(−I, I)
M
(−I, I)
N
(−I, I)
O
(−I, I)
P
(−I, I)
Q
(−I, I)
R
(−I, I)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



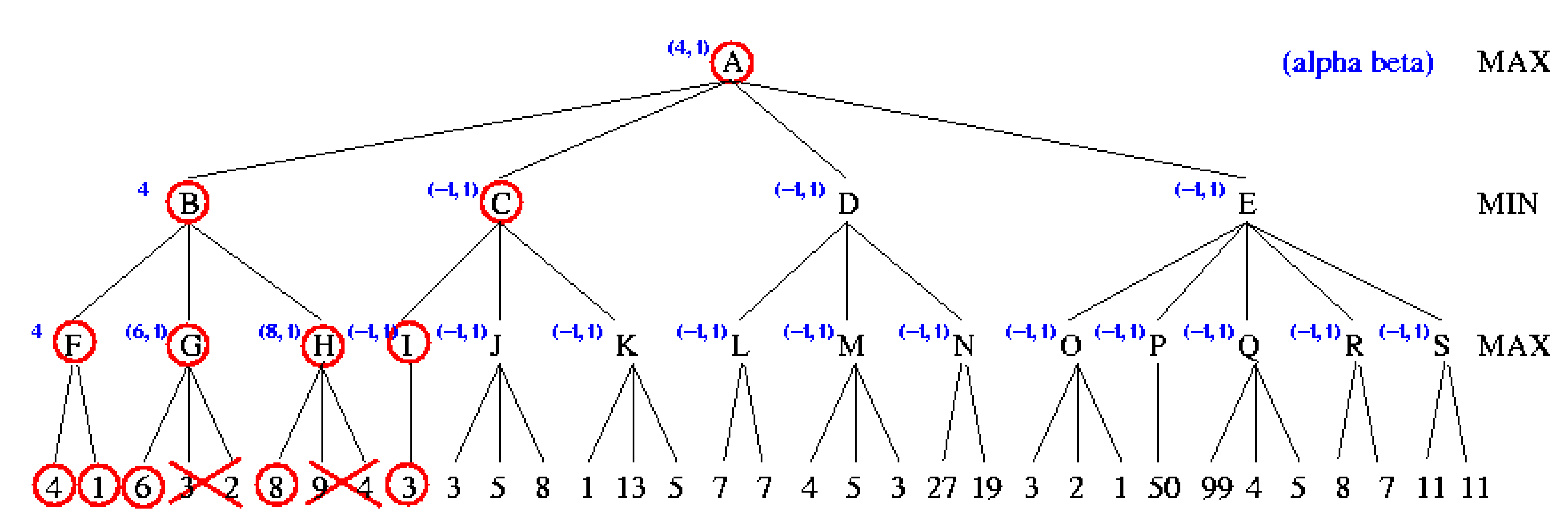
(−I, I)
A
(alpha beta)
MAX
(−I, 4)
B
(−I, I)
C
(−I, I)
D
(−I, I)
E
MIN
OUT OF RANGE
4
F
(6, I)
G
(8, I)
H
(−I, I)
I
(−I, I)
J
(−I, I)
K
(−I, I)
L
(−I, I)
M
(−I, I)
N
(−I, I)
O
(−I, I)
P
(−I, I)
Q
(−I, I)
R
(−I, I)
S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



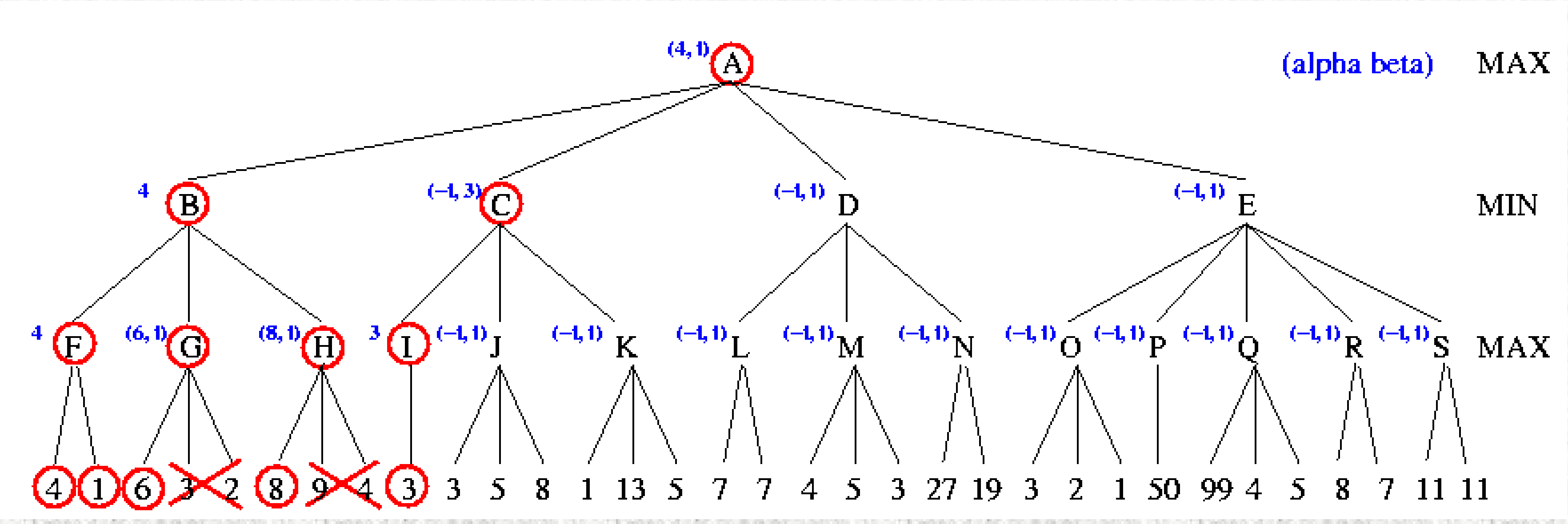
(alpha beta)
MAX
MIN
MAX
(4, 1)
A
4
B
(−I, 1)
C
(−I, 1)
D
(−I, 1)
E
4
F
(6, 1)
G
(8, 1)
H
(−I, 1)
I
(−I, 1)
J
(−I, 1)
K
(−I, 1)
L
(−I, 1)
M
(−I, 1)
N
(−I, 1)
O
(−I, 1)
P
(−I, 1)
Q
(−I, 1)
R
(−I, 1)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



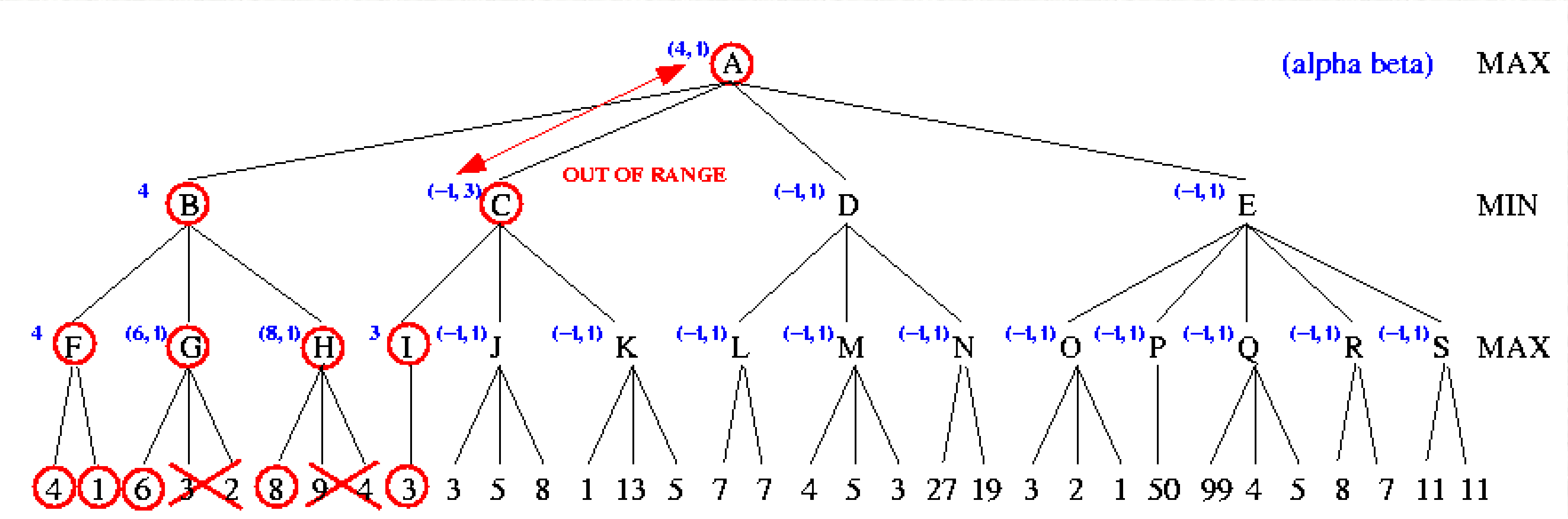
(4, 1) A
(alpha beta)
MAX
4 B
(−1, 1) C
(−1, 1) D
(−1, 1) E
MIN
4 F
(6, 1) G
(8, 1) H
(−1, 1) I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



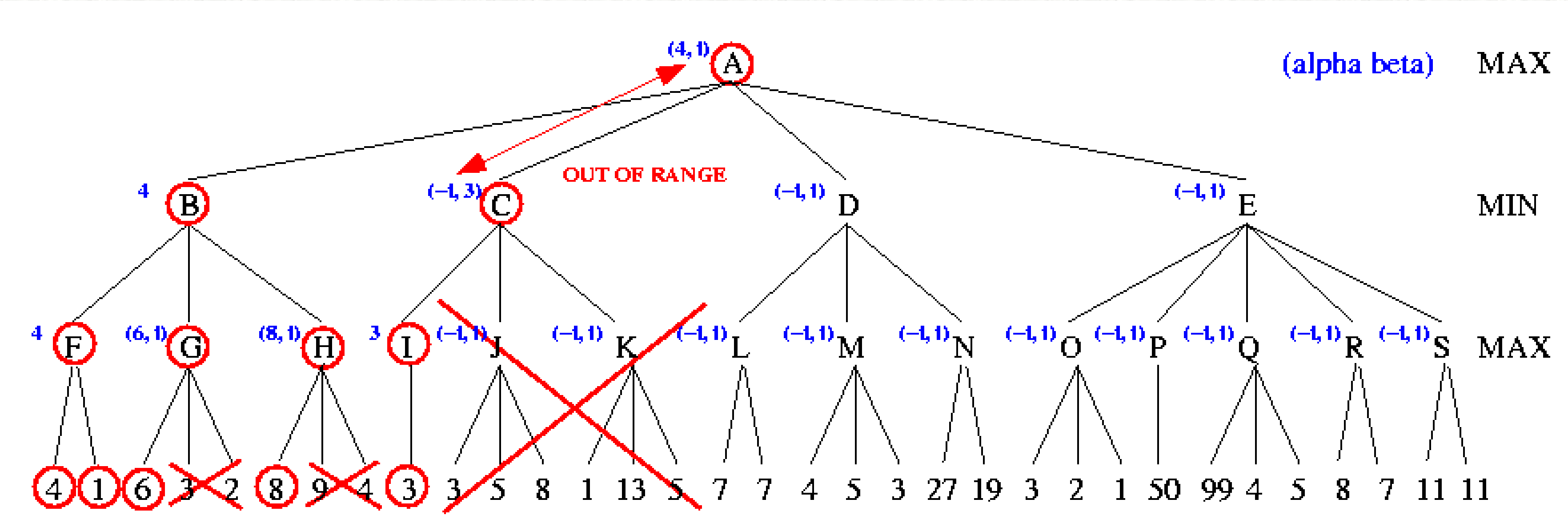
(alpha beta)
MAX
MIN
MAX
A
B C D E
F G H I J K L M N O P Q R S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



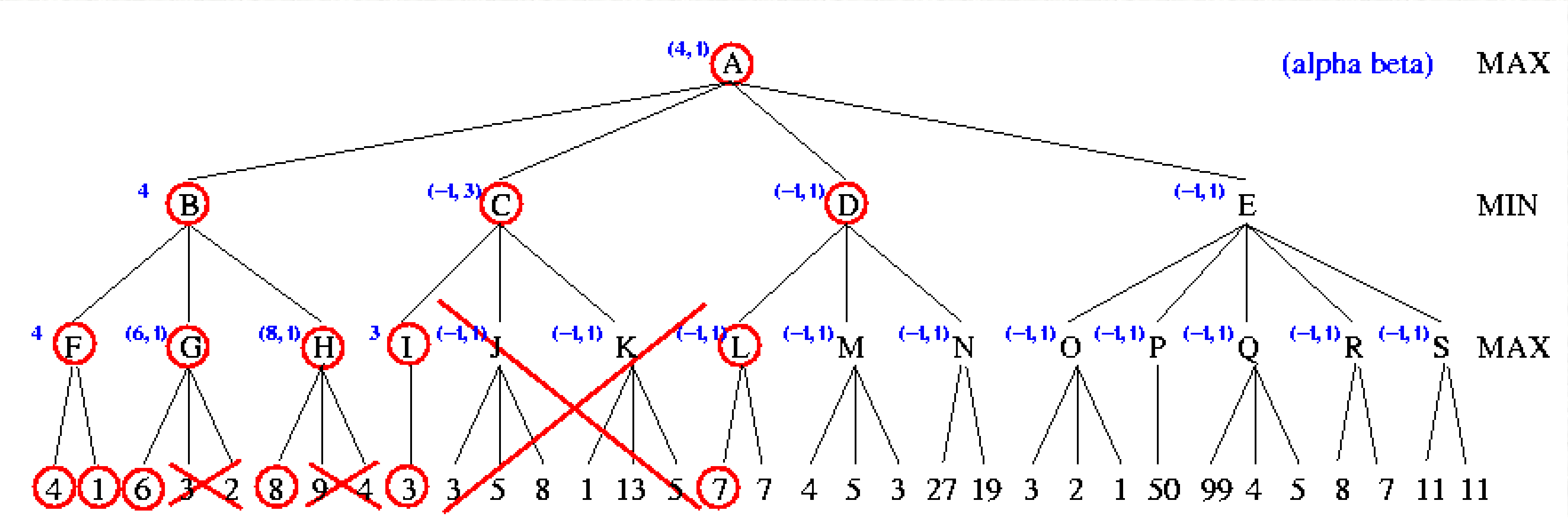
(alpha beta)
MAX
MIN
MAX
(4, 1)
A
OUT OF RANGE
4
B
(−1, 3)
C
(−1, 1)
D
(−1, 1)
E
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
(−1, 1)
L
(−1, 1)
M
(−1, 1)
N
(−1, 1)
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



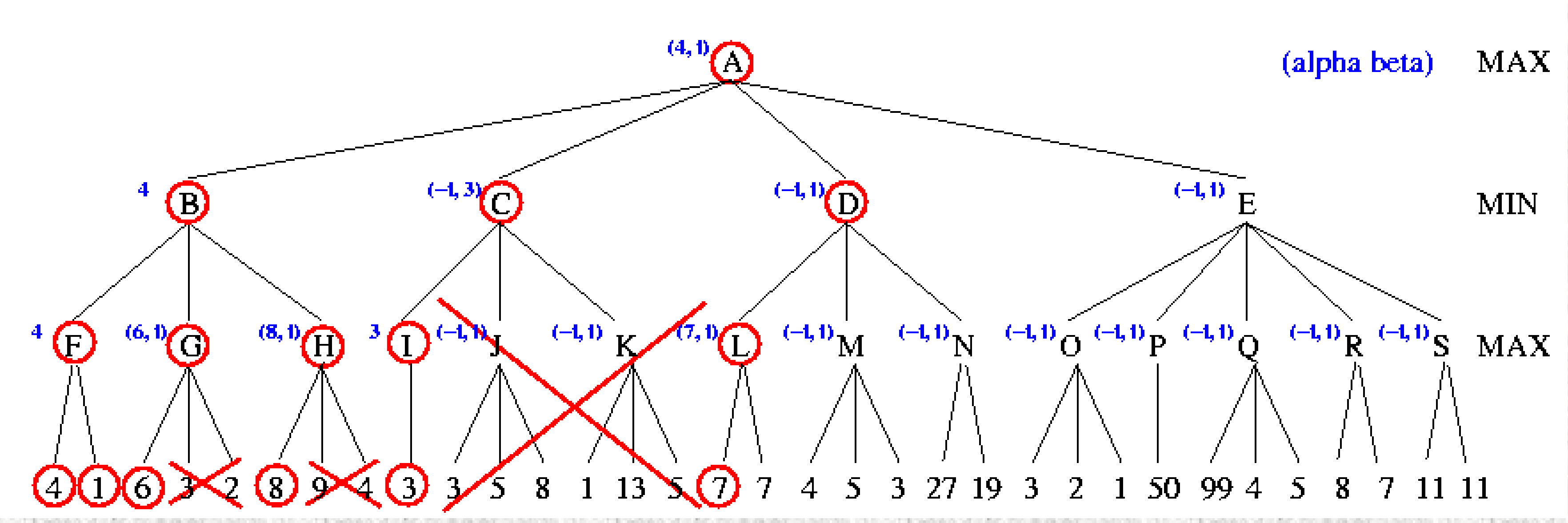
(alpha beta)
MAX
MIN
MAX
(4, 1)
A
OUT OF RANGE
4
B
(−1, 3)
C
(−1, 1)
D
(−1, 1)
E
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



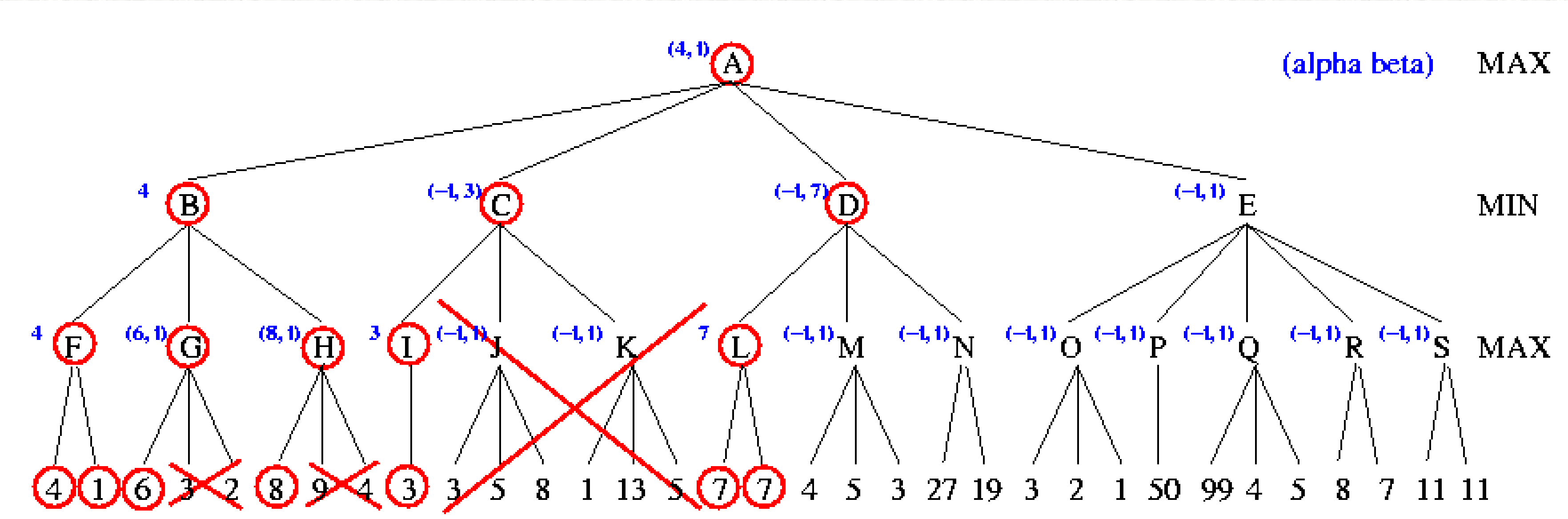
(alpha beta)
MAX
MIN
MAX
(4, 1) A
4 B
(−1, 3) C
(−1, 1) D
(−1, 1) E
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
(−1, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



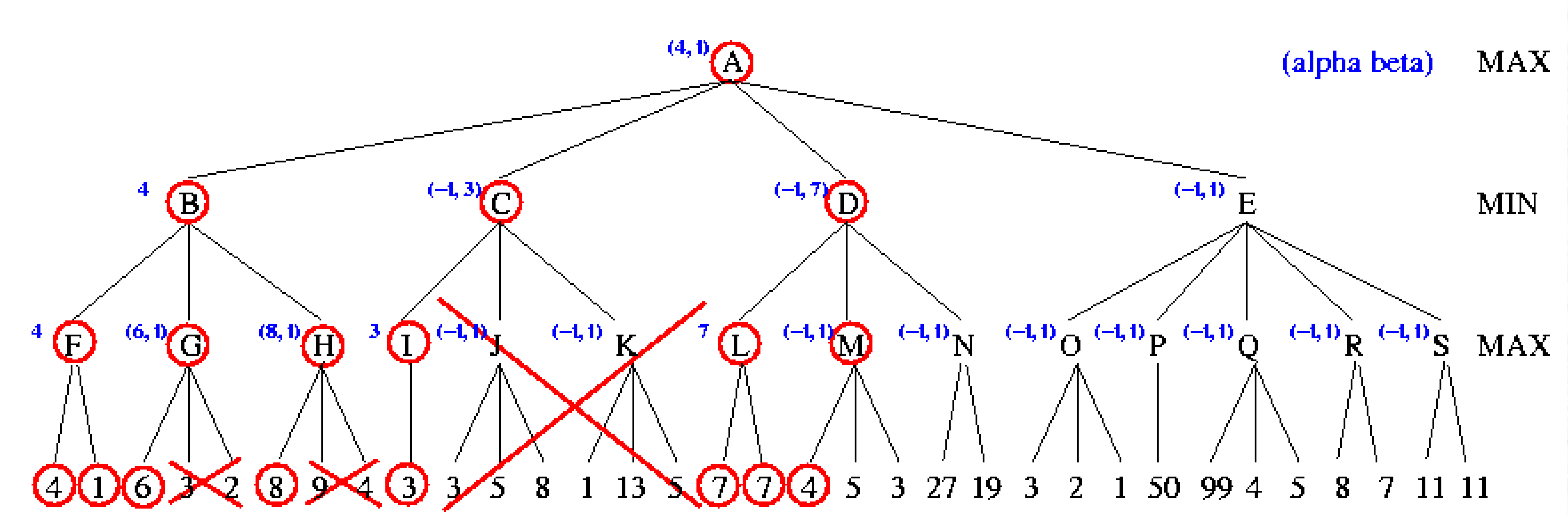
(alpha beta)
MAX
MIN
MAX
(4, 1) A
4 B
(−1, 3) C
(−1, 1) D
(−1, 1) E
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
(7, 1) L
(−1, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



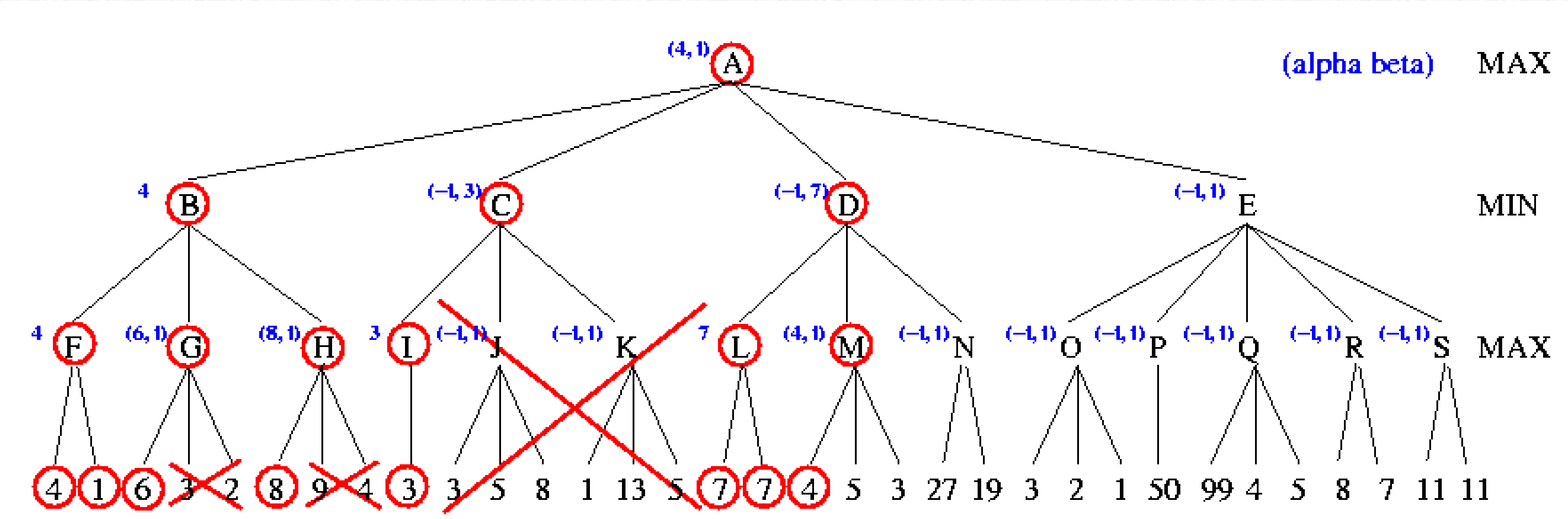
(4, 1)
A
(alpha beta)
MAX
4
B
(−1, 3)
C
(−1, 7)
D
(−1, 1)
E
MIN
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
(−1, 1)
M
(−1, 1)
N
(−1, 1)
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



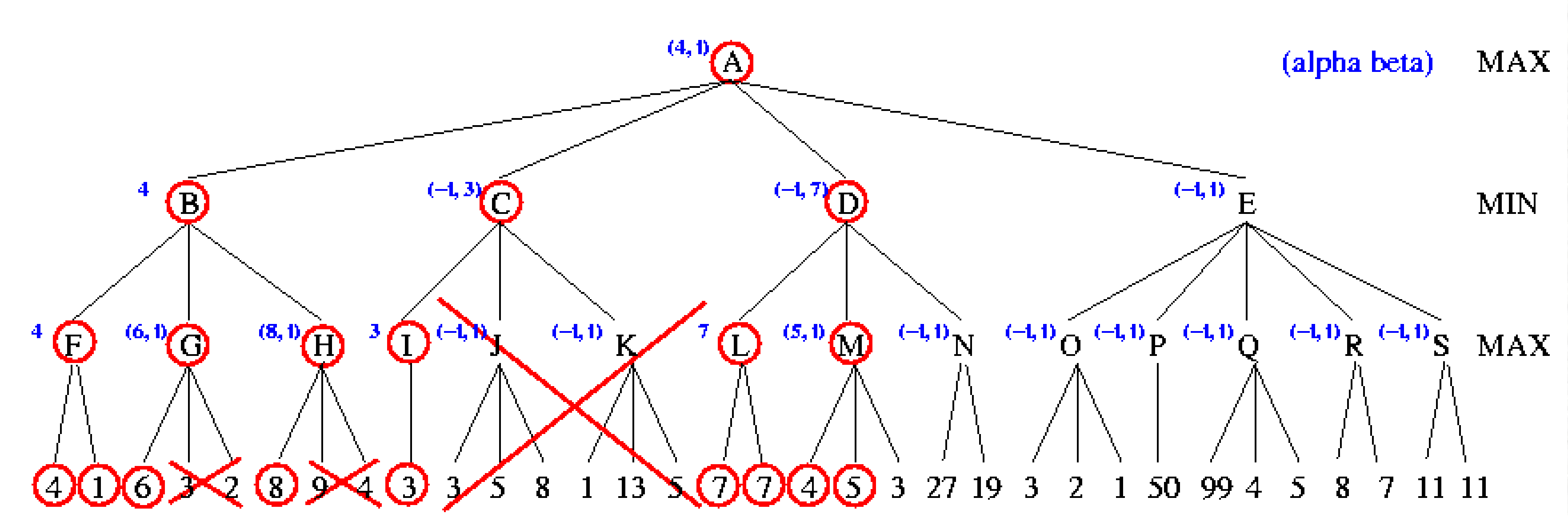
(4, 1)
A
(alpha beta)
MAX
4
B
(−1, 3)
C
(−1, 7)
D
(−1, 1)
E
MIN
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
(−1, 1)
M
(−1, 1)
N
(−1, 1)
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



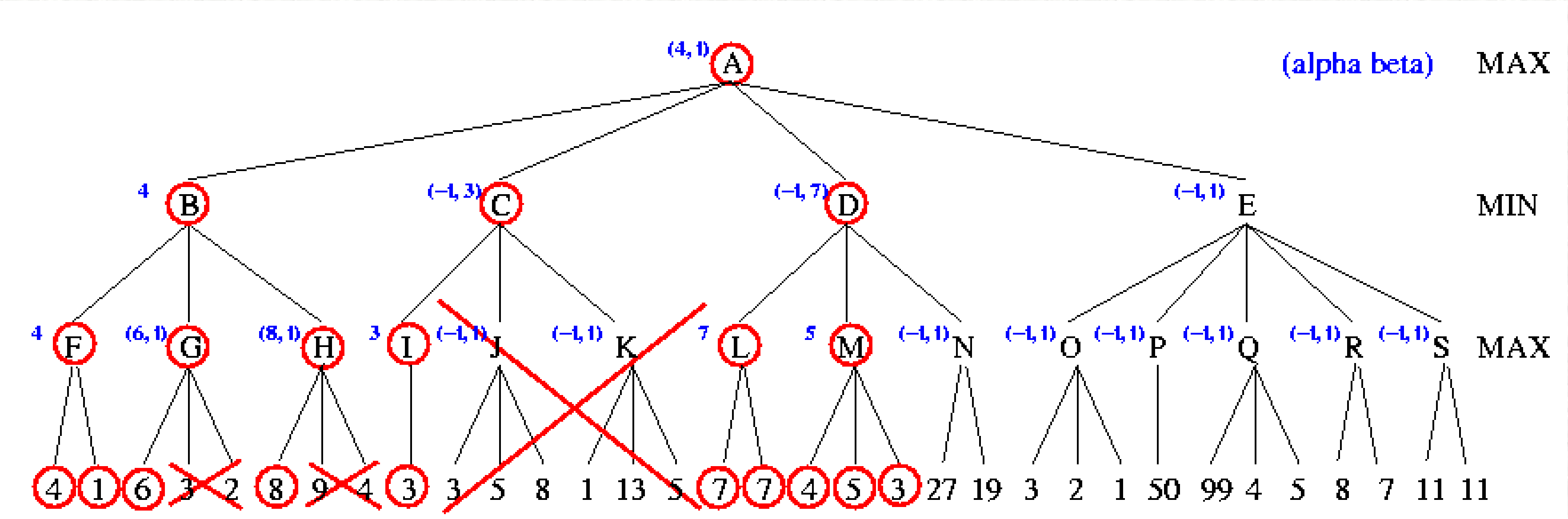
(alpha beta)
MAX
MIN
MAX
(4, 1)
A
4
B
(−1, 3)
C
(−1, 7)
D
(−1, 1)
E
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
(4, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



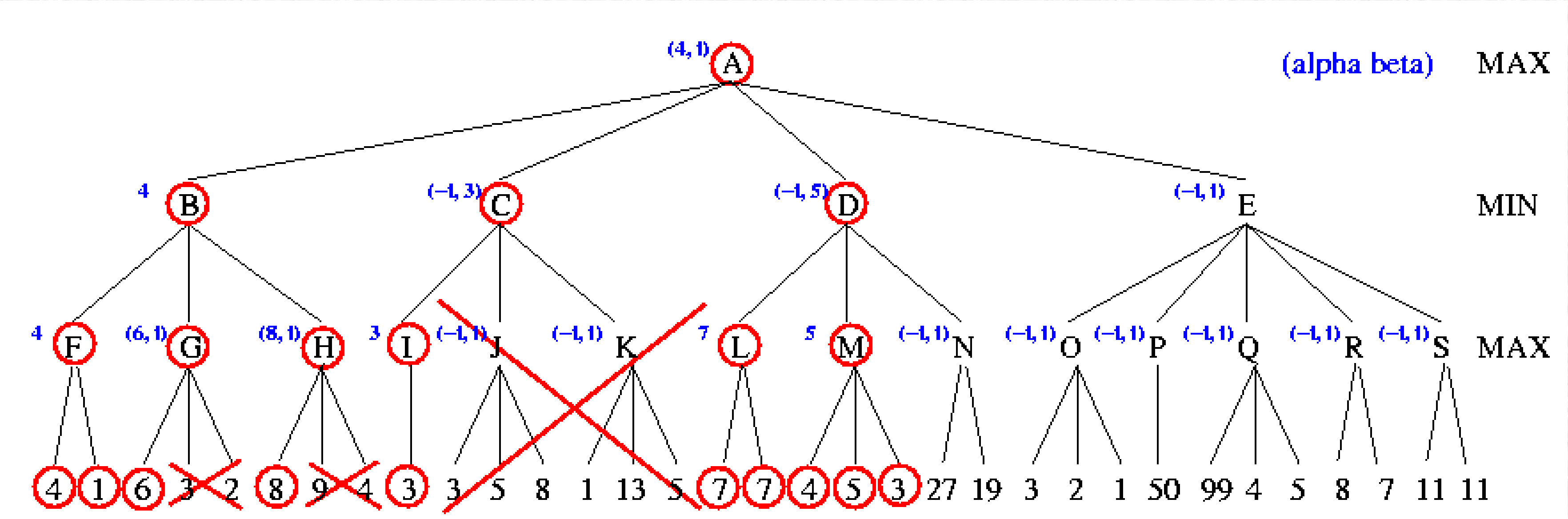
(alpha beta)
MAX
MIN
MAX
(4, 1) A
4 B
(−1, 3) C
(−1, 7) D
(−1, 1) E
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
(5, 1) M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



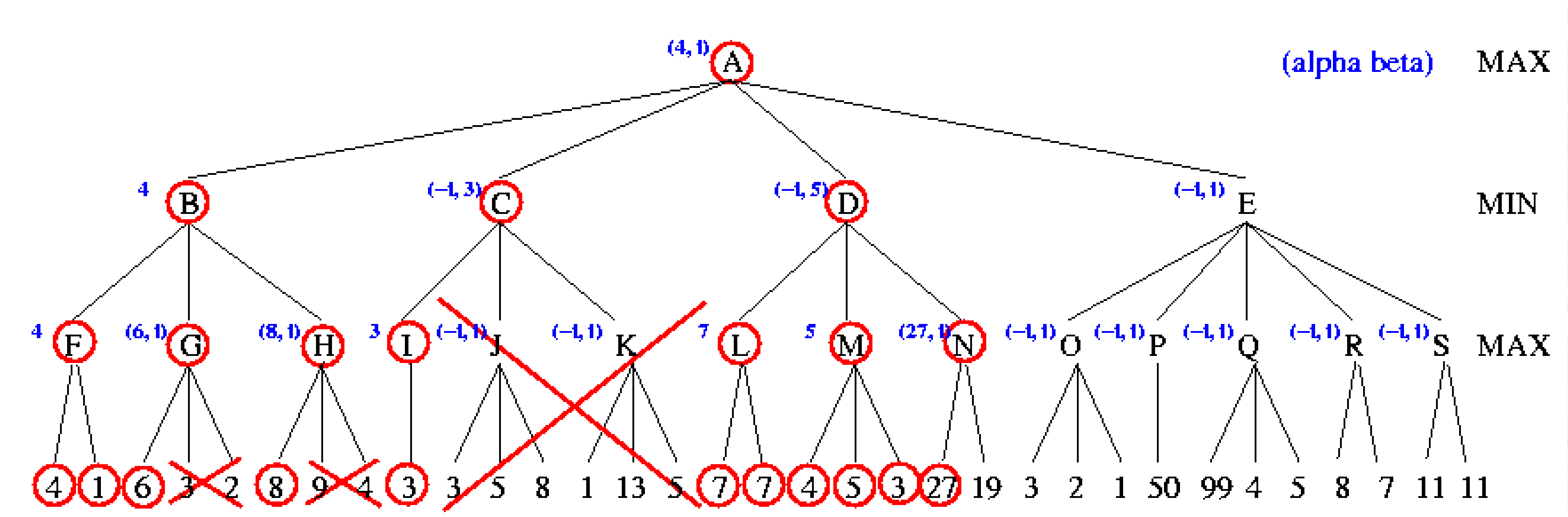
(alpha beta)
MAX
MIN
MAX
(4, 1) A
4 B
(−1, 3) C
(−1, 7) D
(−1, 1) E
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(−1, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



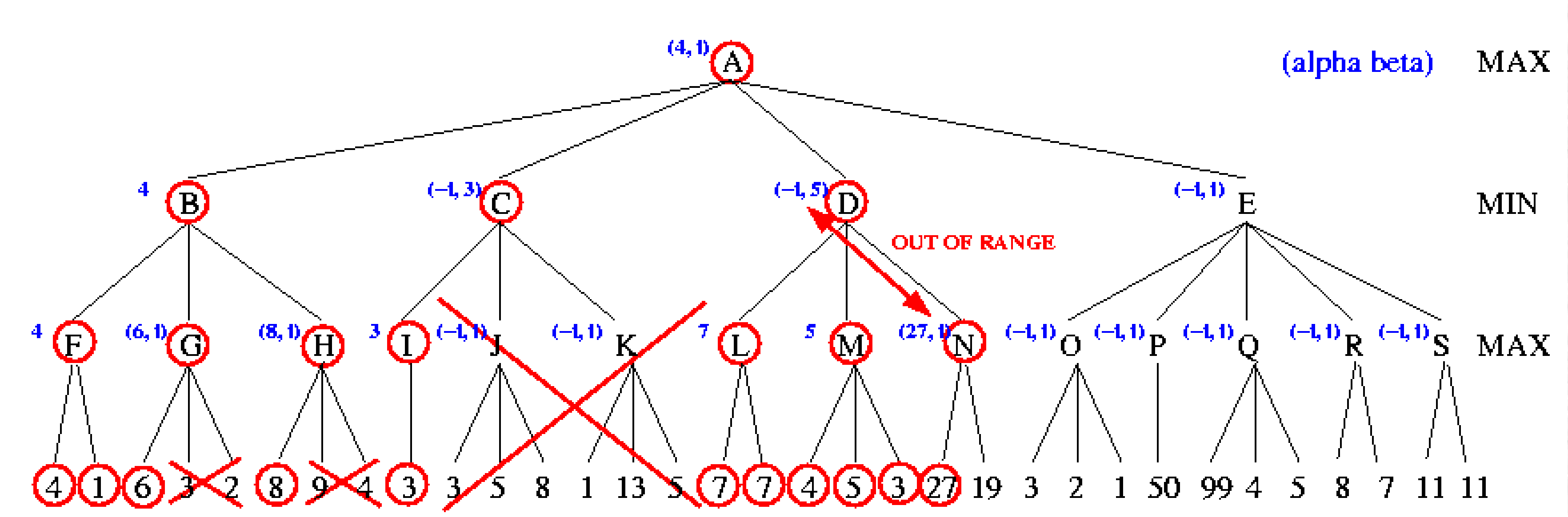
(alpha beta)
MAX
MIN
MAX
(4, 1)
A
4
B
(−1, 3)
C
(−1, 5)
D
(−1, 1)
E
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
5
M
(−1, 1)
N
(−1, 1)
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



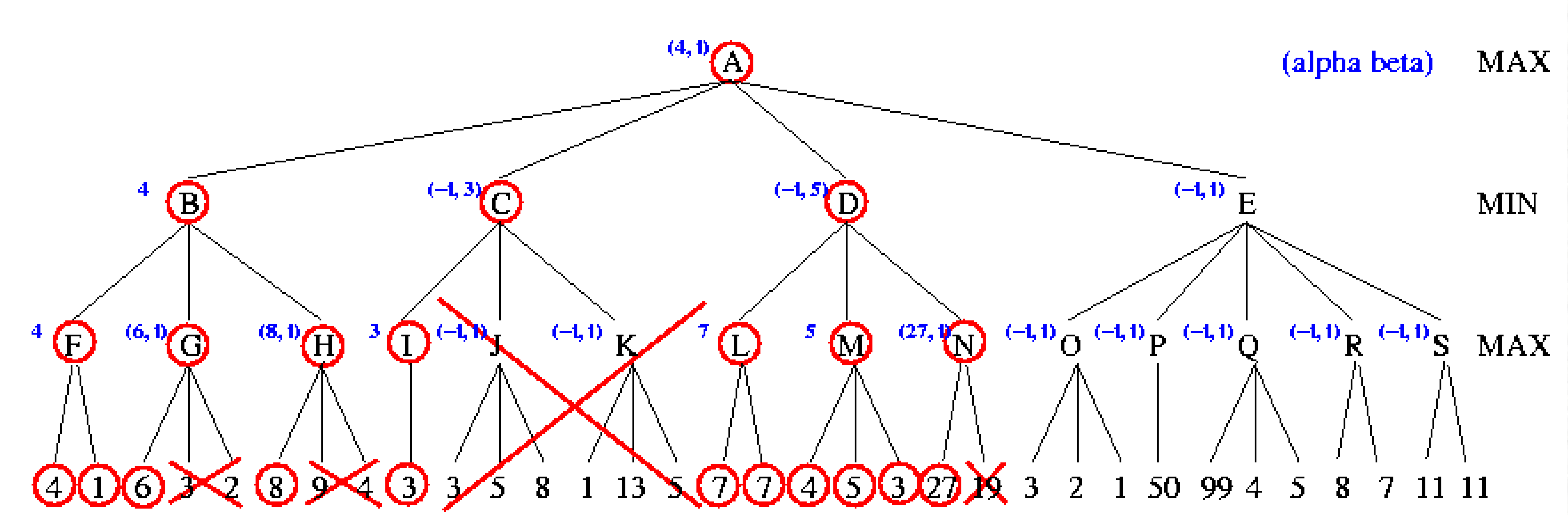
(alpha beta)
MAX
MIN
MAX
(4, 1) A
4 B
(−1, 3) C
(−1, 5) D
(−1, 1) E
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(27, 1) N
(−1, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



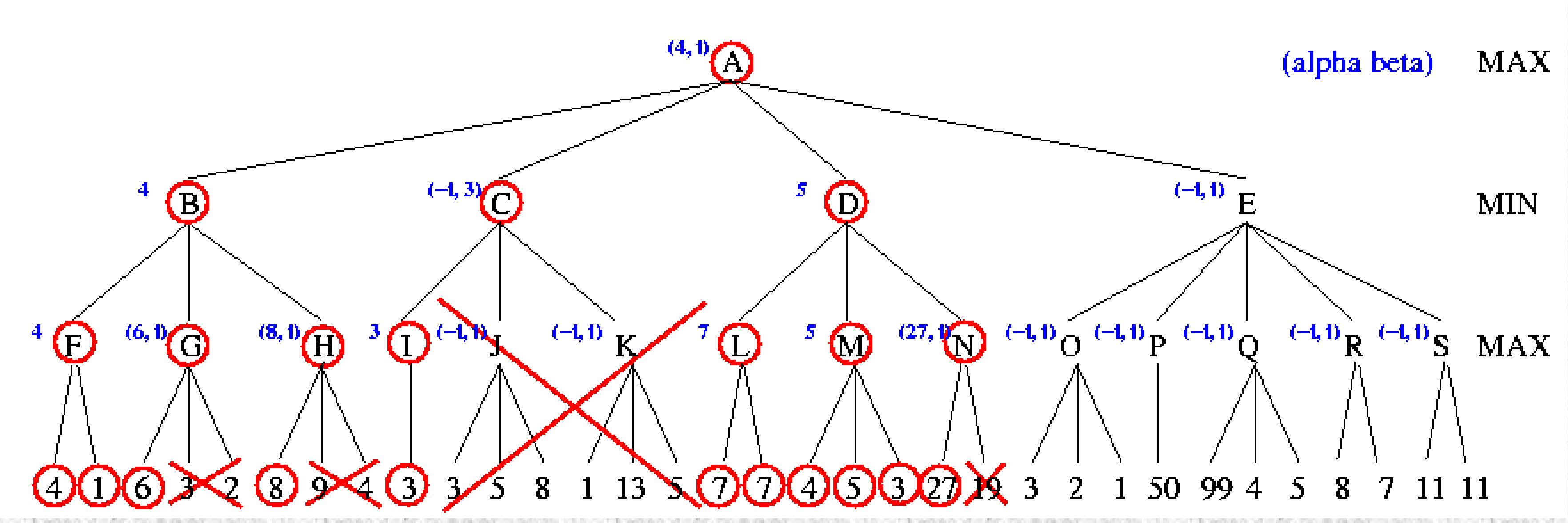
(alpha beta)
MAX
MIN
MAX
(4, 1)
A
4
B
(−1, 3)
C
(−1, 5)
D
(−1, 1)
E
OUT OF RANGE
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
5
M
(27, 1)
N
(−1, 1)
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



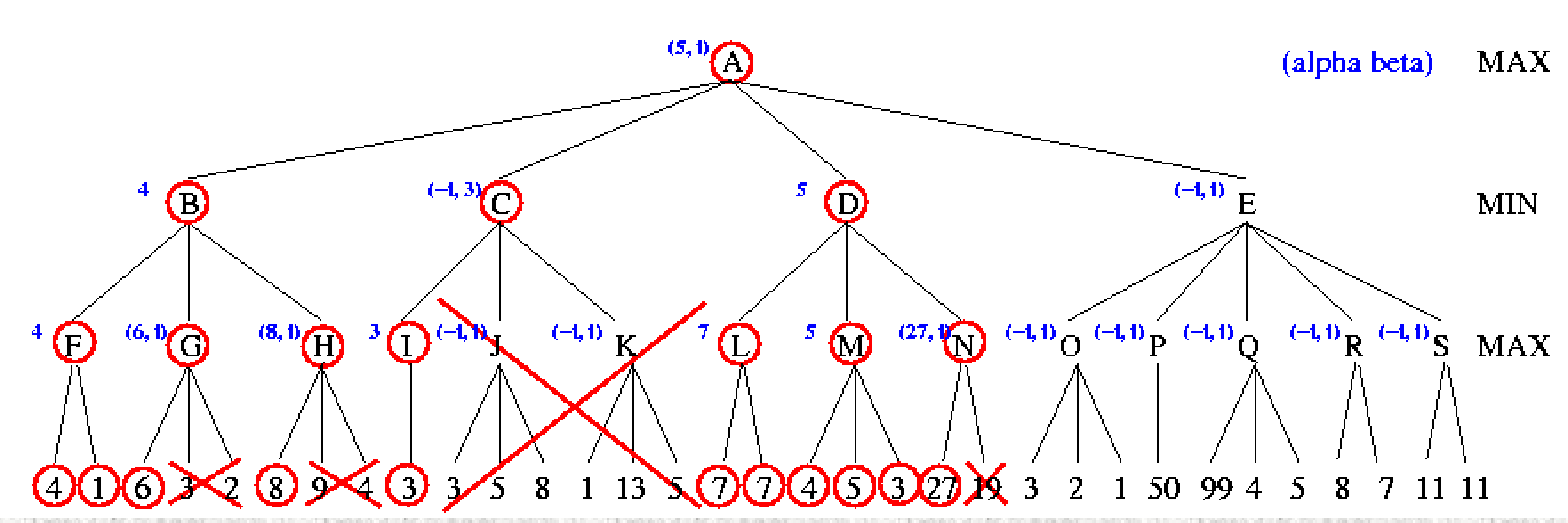
(alpha beta)
MAX
MIN
MAX
(4, 1)
A
4
B
(−1, 3)
C
(−1, 5)
D
(−1, 1)
E
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
5
M
(27, 1)
N
(−1, 1)
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



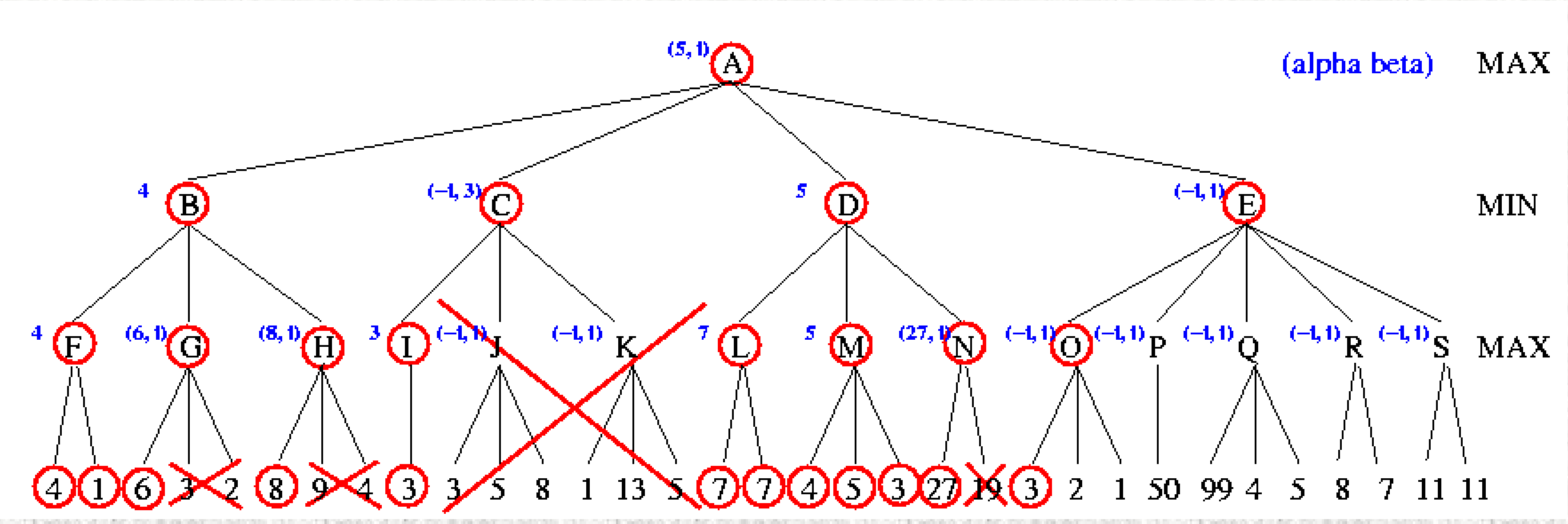
(4, 1)
A
(alpha beta)
MAX
4
B
(−1, 3)
C
5
D
(−1, 1)
E
MIN
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
5
M
(27, 1)
N
(−1, 1)
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example

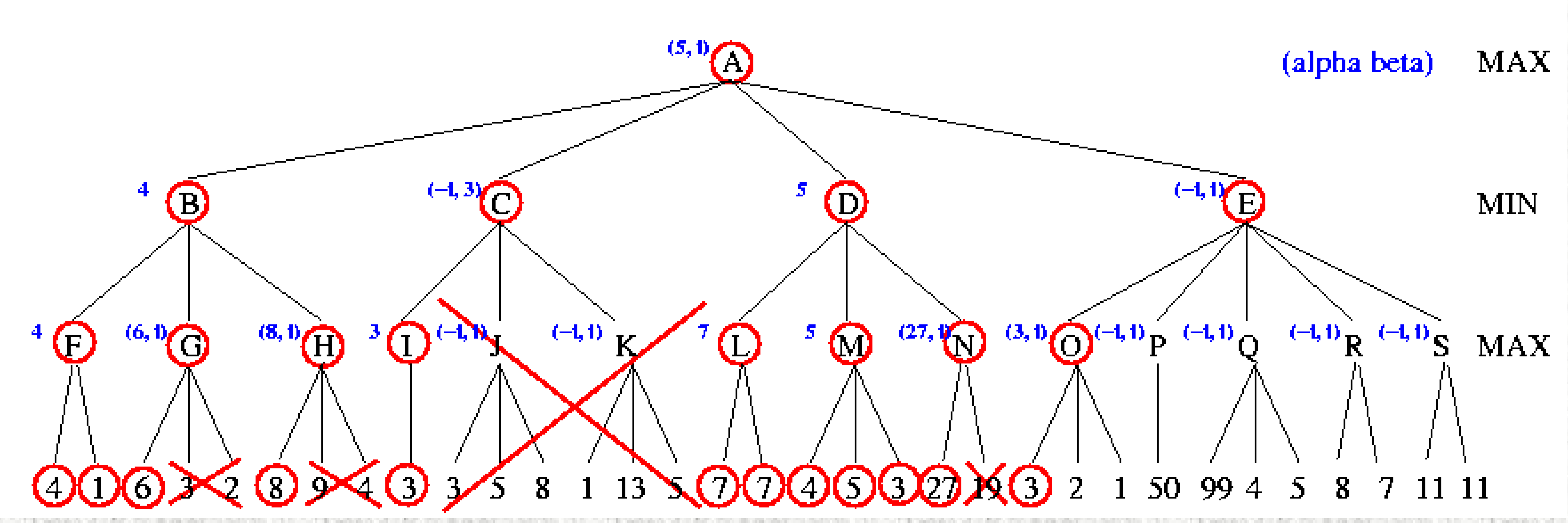


(5, 1)
A
(alpha beta)
MAX
4
B
(−1, 3)
C
5
D
(−1, 1)
E
MIN
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
5
M
(27, 1)
N
(−1, 1)
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example

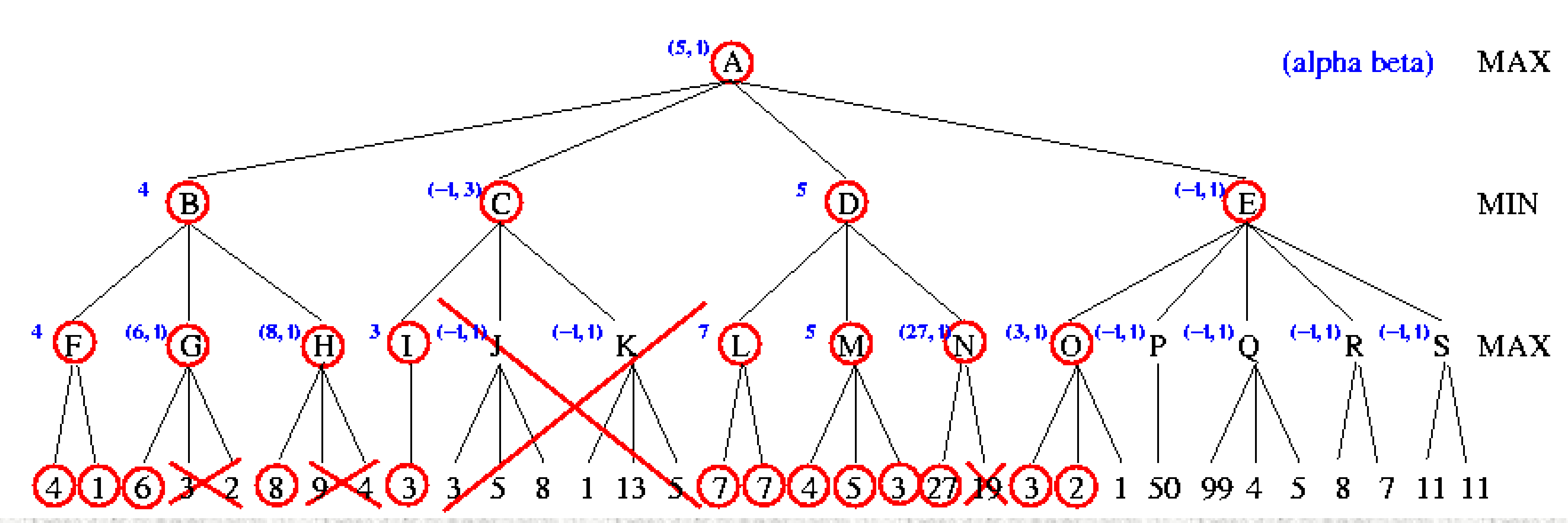

(alpha beta)
MAX
MIN
MAX
(5, 1)
A
4
B
(−1, 3)
C
5
D
(−1, 1)
E
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
5
M
(27, 1)
N
(−1, 1)
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



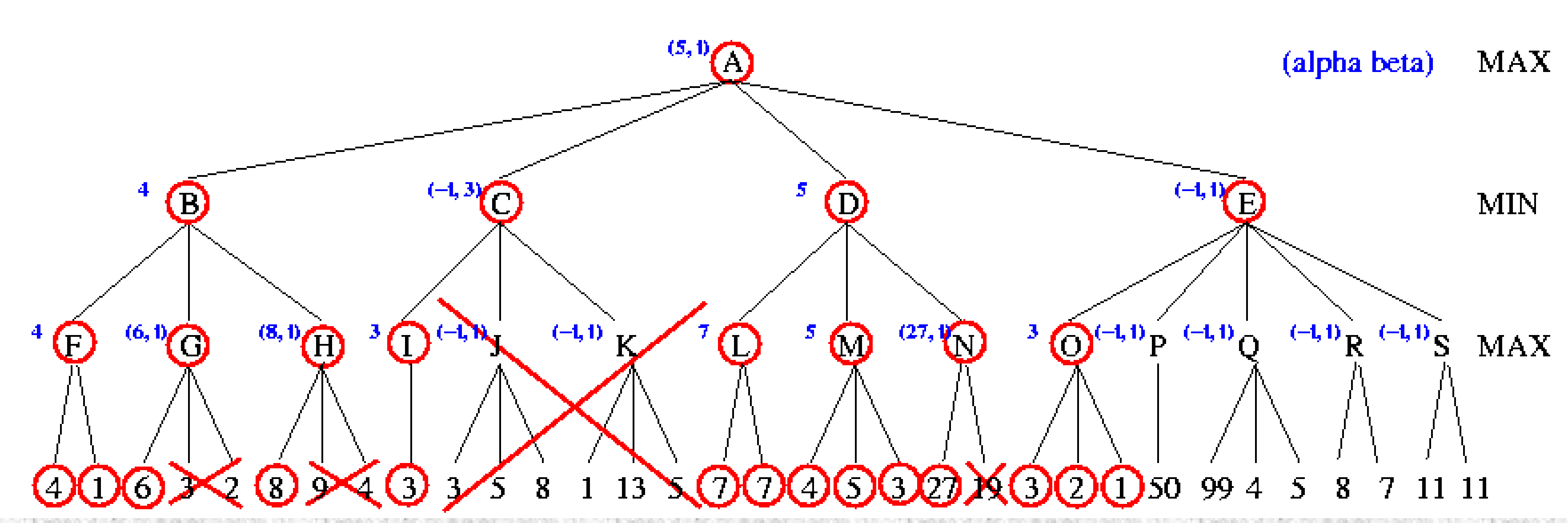
(alpha beta)
MAX
MIN
MAX
(5, 1)
A
4
B
(−1, 3)
C
5
D
(−1, 1)
E
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
5
M
(27, 1)
N
(3, 1)
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



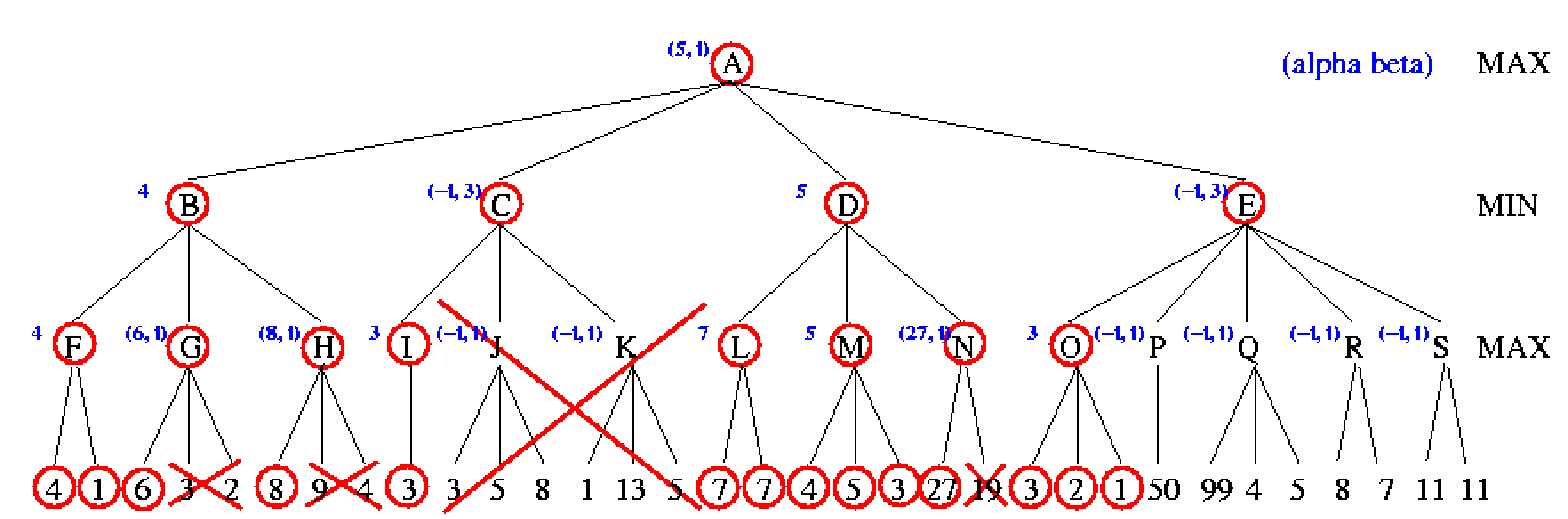
(alpha beta)
MAX
MIN
MAX
(5, 1) A
4 B
(−1, 3) C
5 D
(−1, 1) E
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(27, 1) N
(3, 1) O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



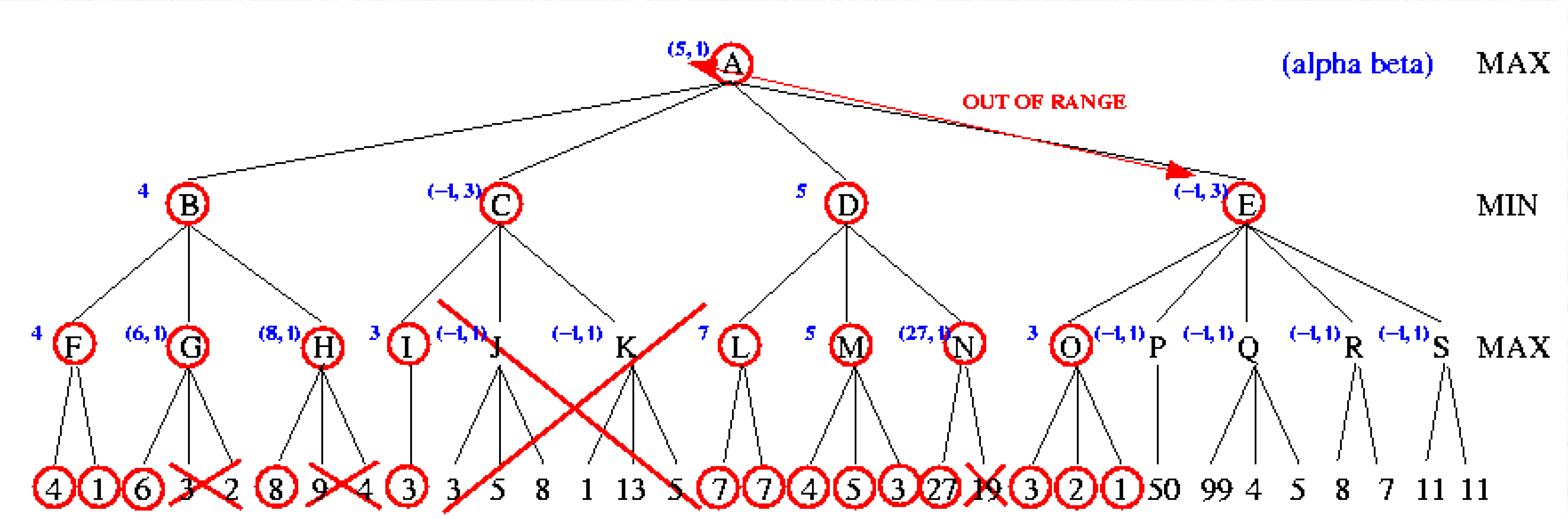
(alpha beta)
MAX
MIN
MAX
(5, 1)
A
4
B
(−1, 3)
C
5
D
(−1, 1)
E
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
5
M
(27, 1)
N
3
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



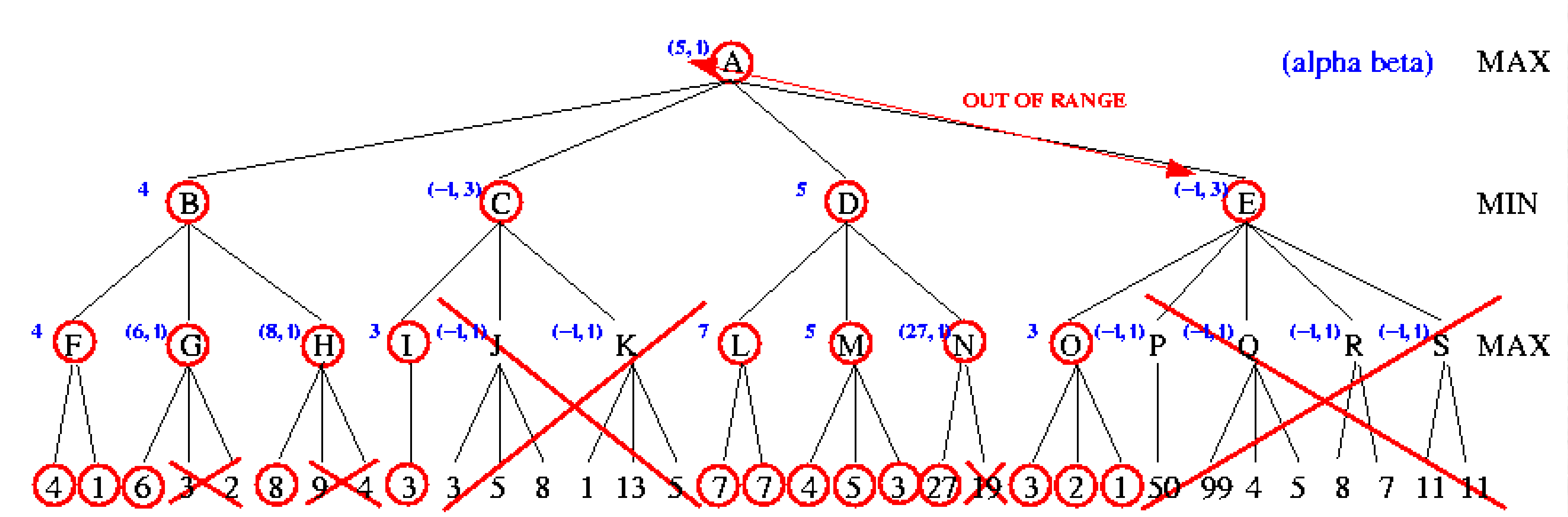
(5, 1)
A
(alpha beta)
MAX
4
B
(-1, 3)
C
5
D
(-1, 3)
E
MIN
4
F
(6, 1)
G
(8, 1)
H
3
I
(-1, 1)
J
(-1, 1)
K
7
L
5
M
(27, 1)
N
3
O
(-1, 1)
P
(-1, 1)
Q
(-1, 1)
R
(-1, 1)
S
MAX
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



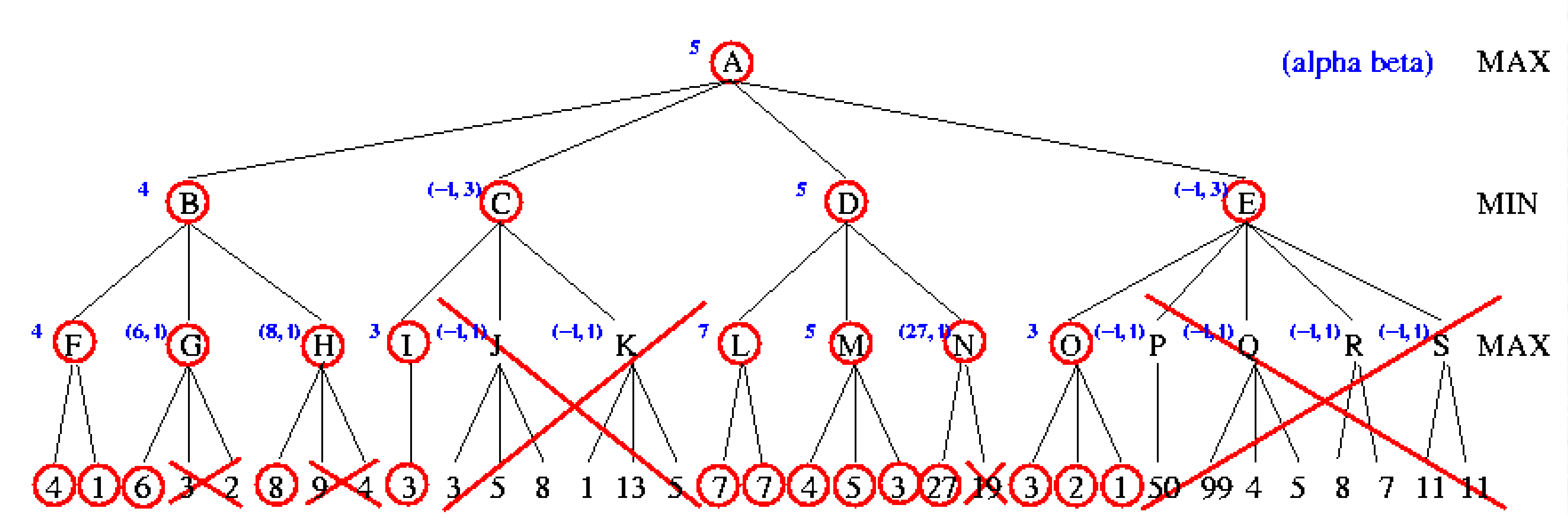
(alpha beta)
MAX
MIN
MAX
OUT OF RANGE
(5, 1)
A
4
B
(−1, 3)
C
5
D
(−1, 3)
E
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
5
M
(27, 1)
N
3
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



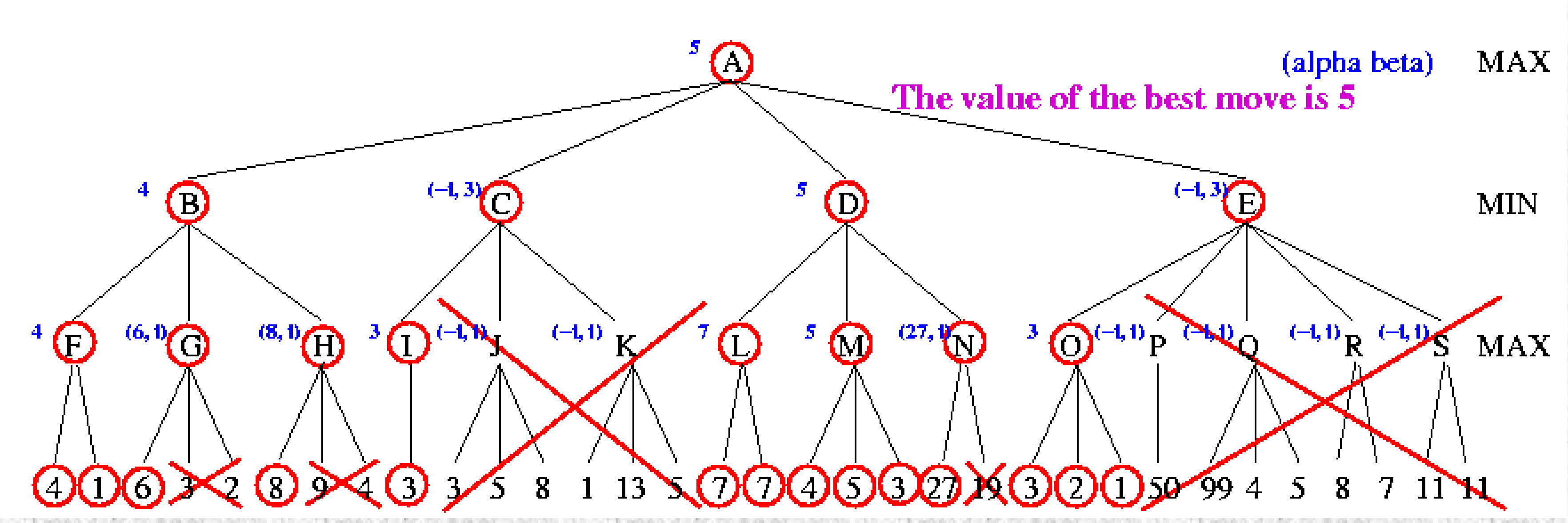
(alpha beta)
MAX
MIN
MAX
OUT OF RANGE
(5, 1)
A
4
B
(−1, 3)
C
5
D
(−1, 3)
E
4
F
(6, 1)
G
(8, 1)
H
3
I
(−1, 1)
J
(−1, 1)
K
7
L
5
M
(27, 1)
N
3
O
(−1, 1)
P
(−1, 1)
Q
(−1, 1)
R
(−1, 1)
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example

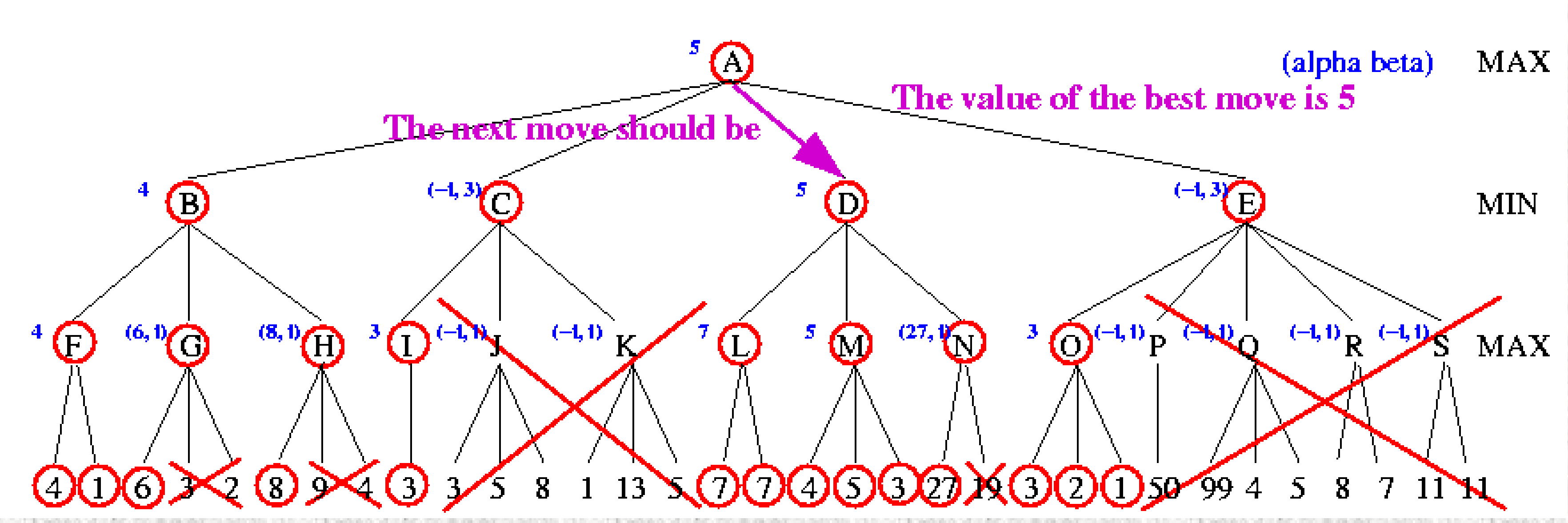


(alpha beta)
MAX
MIN
MAX
5 A
4 B
(−1, 3) C
5 D
(−1, 3) E
4 F
(6, 1) G
(8, 1) H
3 I
(−1, 1) J
(−1, 1) K
7 L
5 M
(27, 1) N
3 O
(−1, 1) P
(−1, 1) Q
(−1, 1) R
(−1, 1) S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example


(alpha beta)
MAX
MIN
MAX
The value of the best move is 5
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11

Example



(alpha beta)
MAX
MIN
MAX
The next move should be
The value of the best move is 5
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
P
Q
R
S
5
4
(−1, 3)
5
(−1, 3)
4
(6, 1)
(8, 1)
3
(−1, 1)
(−1, 1)
7
5
(27, 1)
3
(−1, 1)
(−1, 1)
(−1, 1)
(−1, 1)
4 1 6 3 2 8 9 4 3 3 5 8 1 13 5 7 7 4 5 3 27 19 3 2 1 50 99 4 5 8 7 11 11