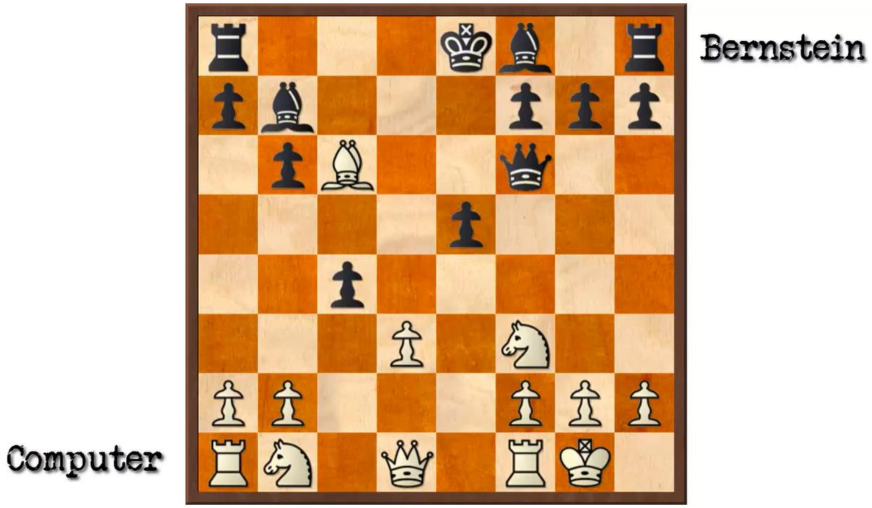
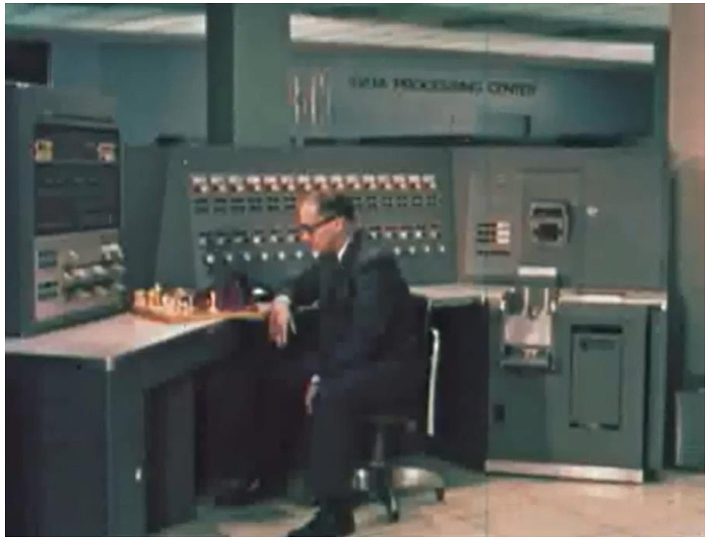
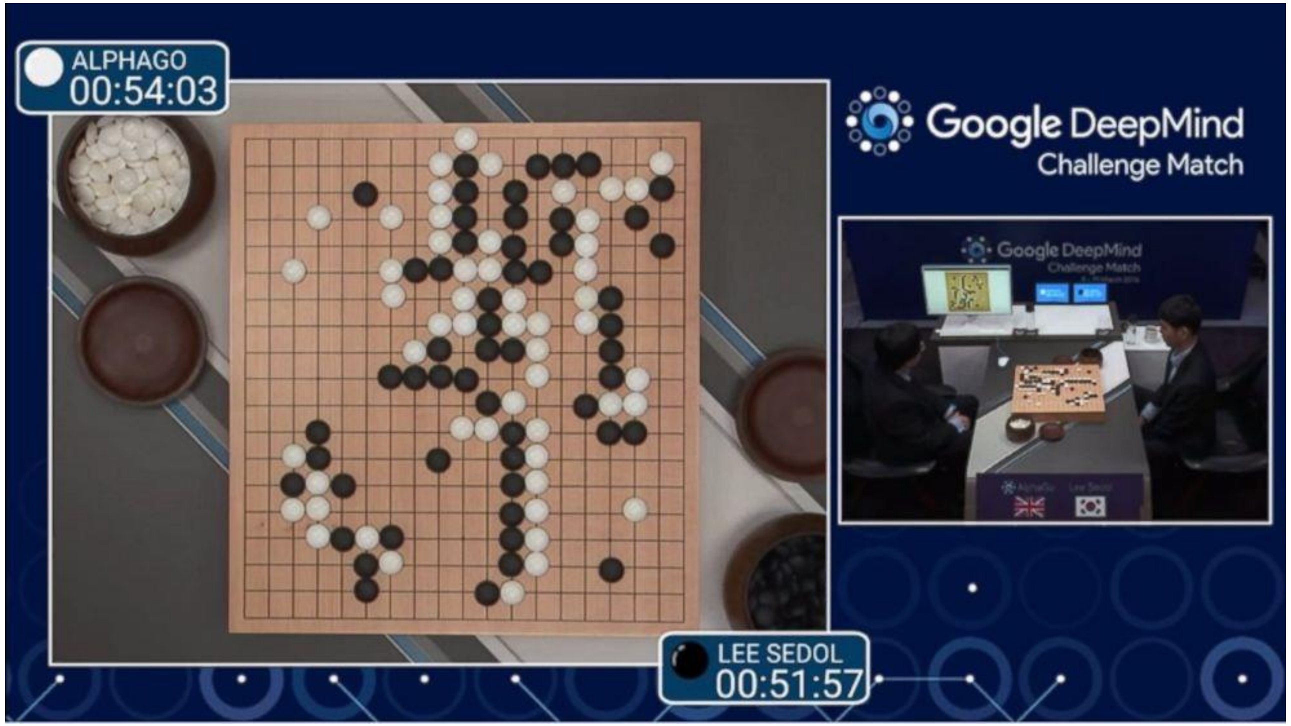
Search in Games

Chen-Yu Wei


Bernstein
Computer




ALPHAGO
00:54:03
Google DeepMind
Challenge Match
LEE SEDOL
00:51:57

Turn-Based Two-Player Game

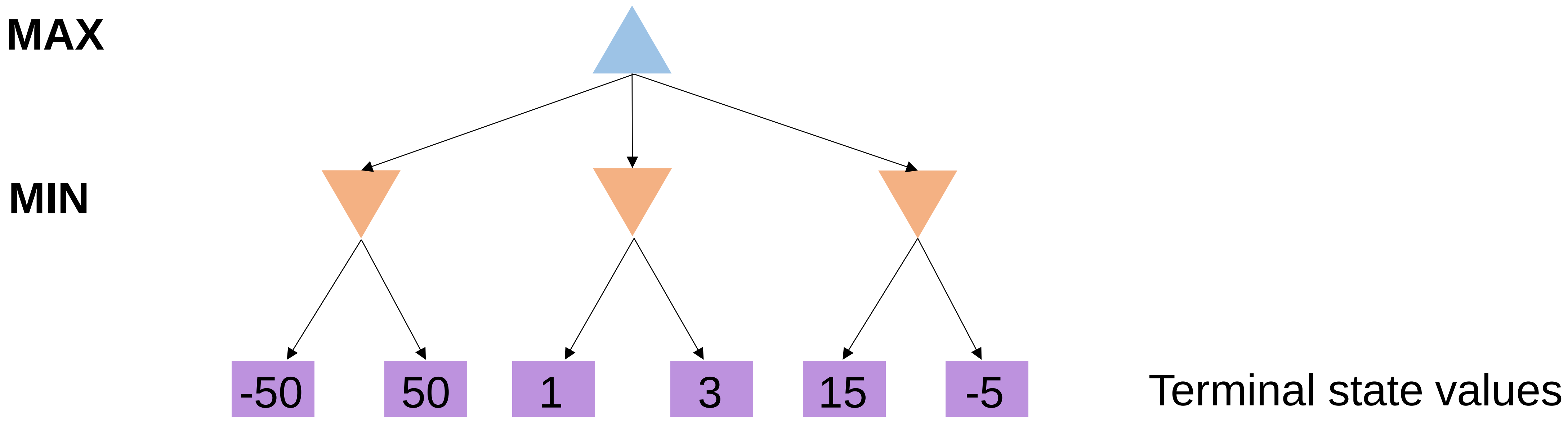
You choose one of the three bins. I choose a number from that bin. Your goal is to maximize the chosen number.

A	B	C
-50 50	1 3	15 -5

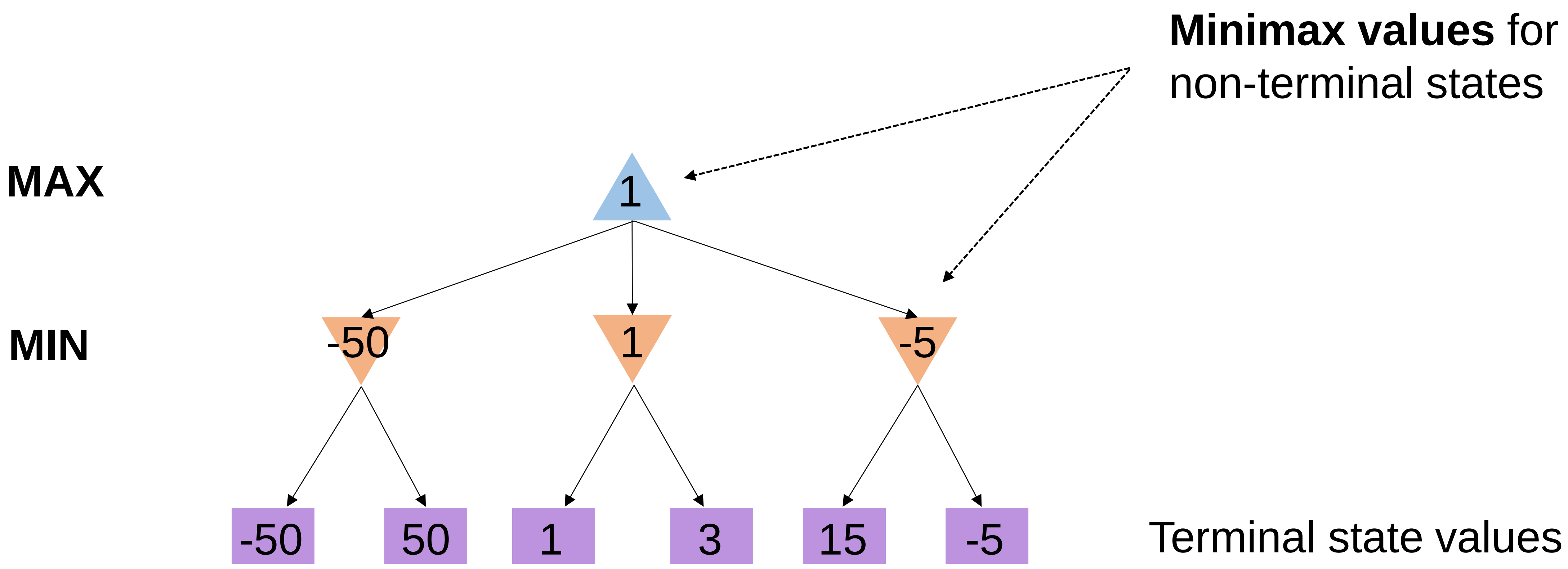
If I am

- adversarial
- random
- benign/cooperative

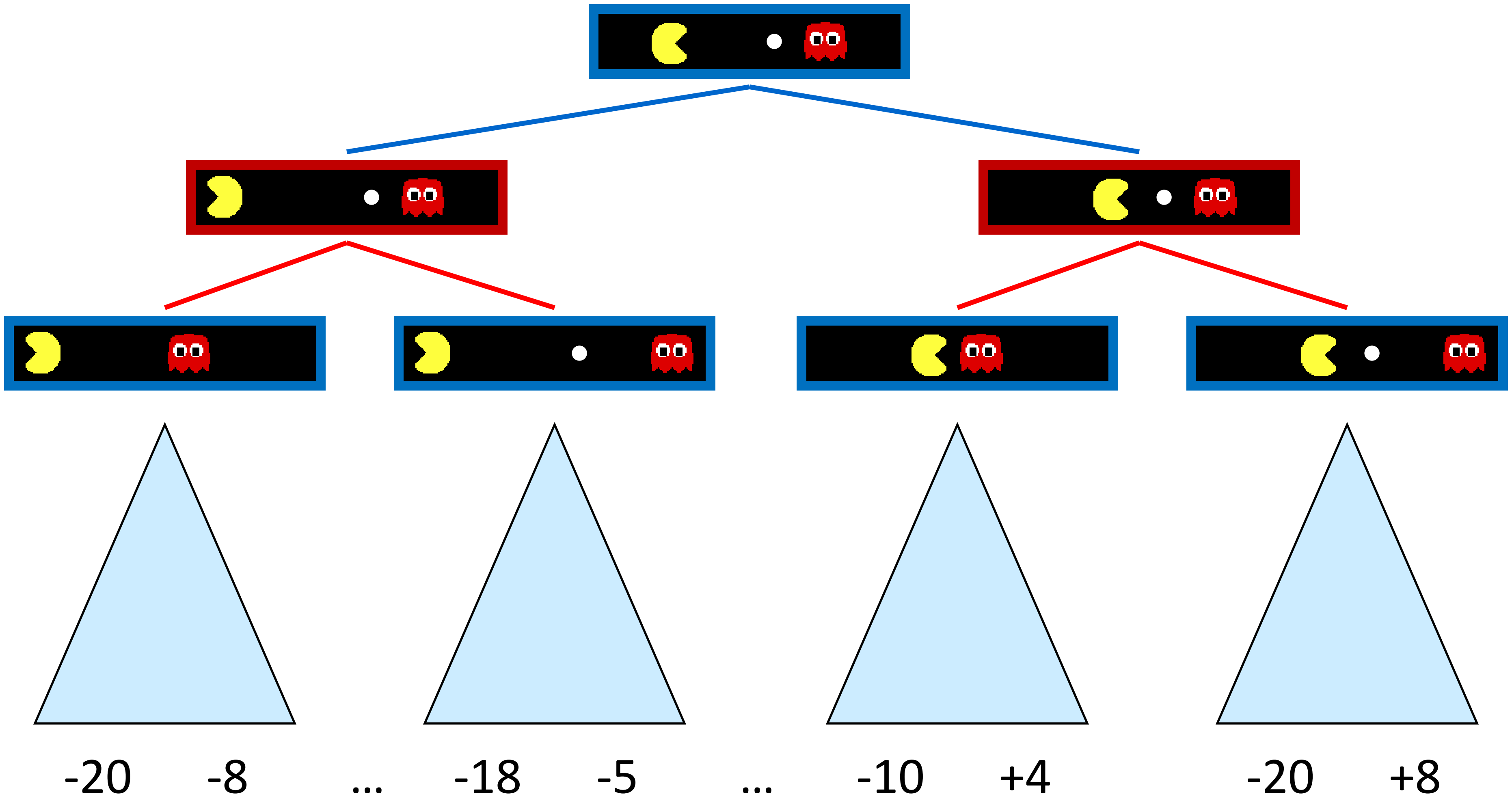
Turn-Based Two-Player Zero-Sum Games



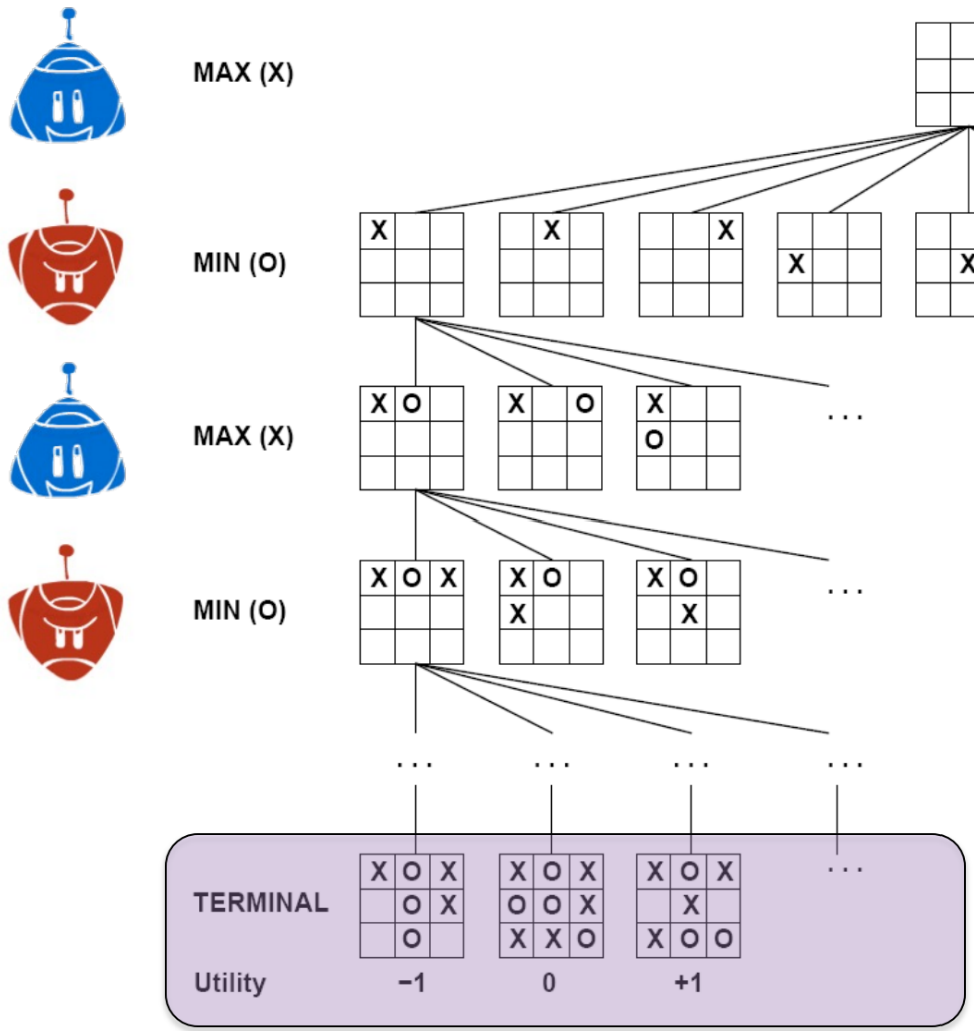
Turn-Based Two-Player Zero-Sum Games

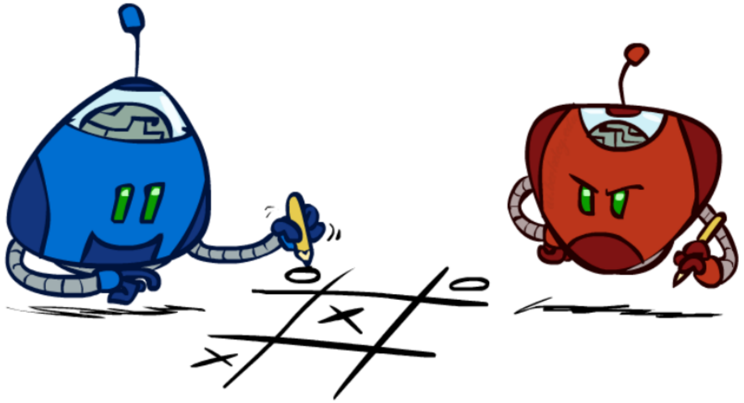


Example: PACMAN



Example: Tic-Tac-Toe





Calculating Minimax Values

```
def value(state):
    if the state is a terminal state: return the state's utility
    if the next agent is MAX: return max-value(state)
    if the next agent is MIN: return min-value(state)
```

```
def max-value(state):
    initialize v = −∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def min-value(state):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor))
    return v
```

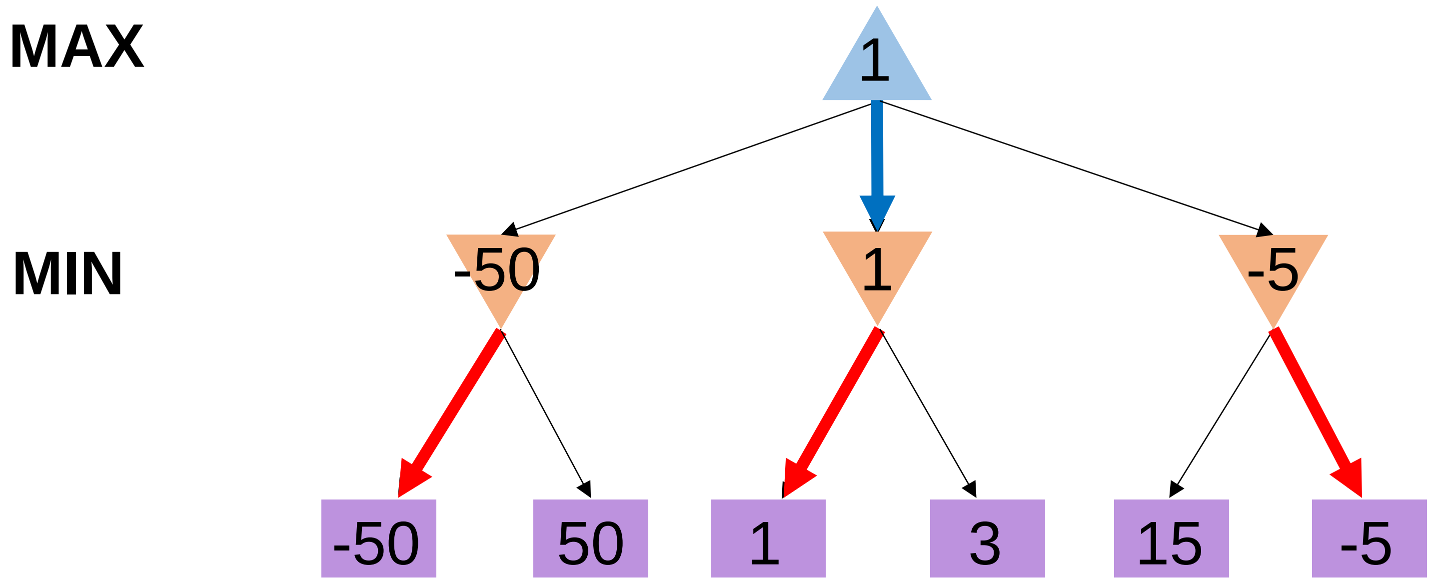
The Minimax Policy

"**Policy**" is mapping from state to action.

"**Minimax policy**" is the optimal policy against the most adversarial opponent.

MAX Player's minimax policy

MIN Player's minimax policy

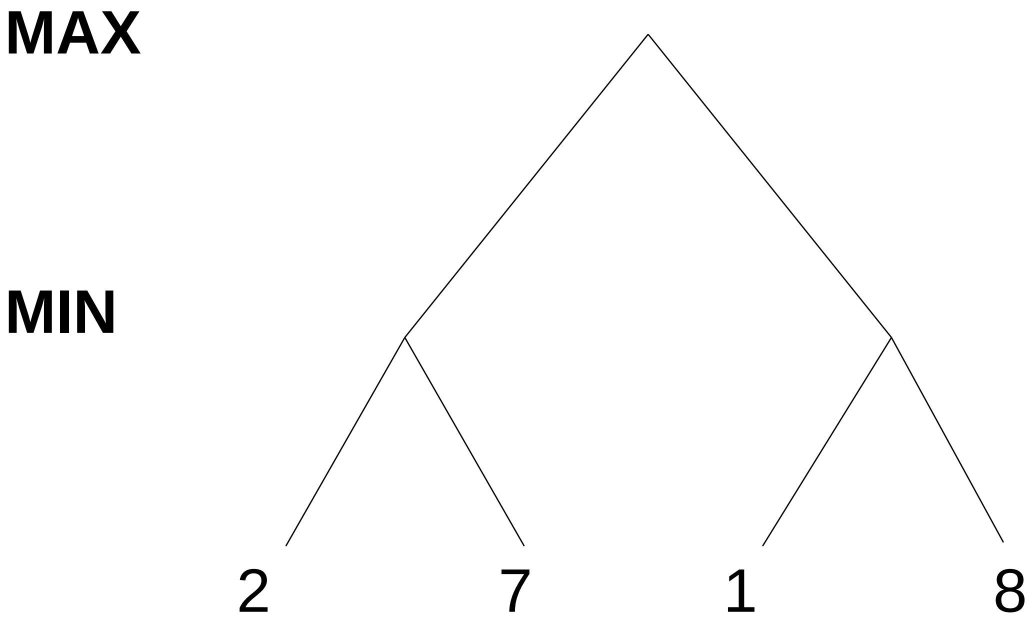


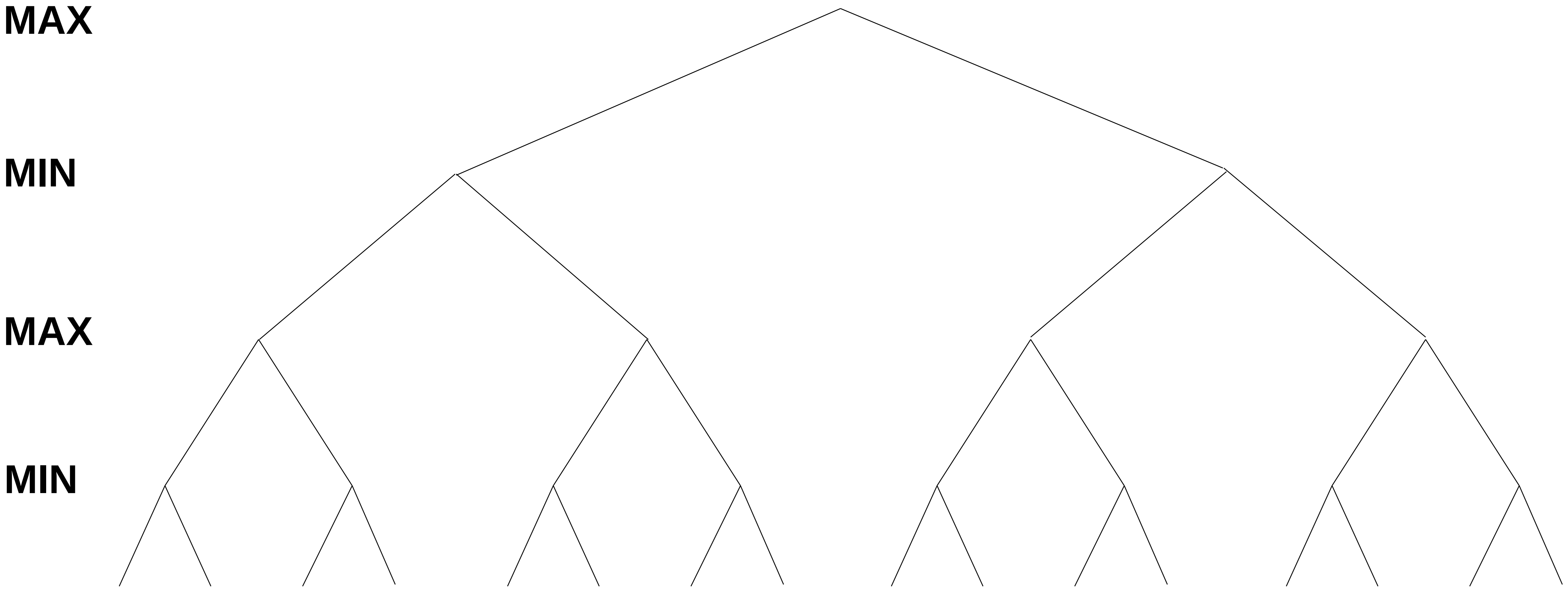
Time / Space Complexity

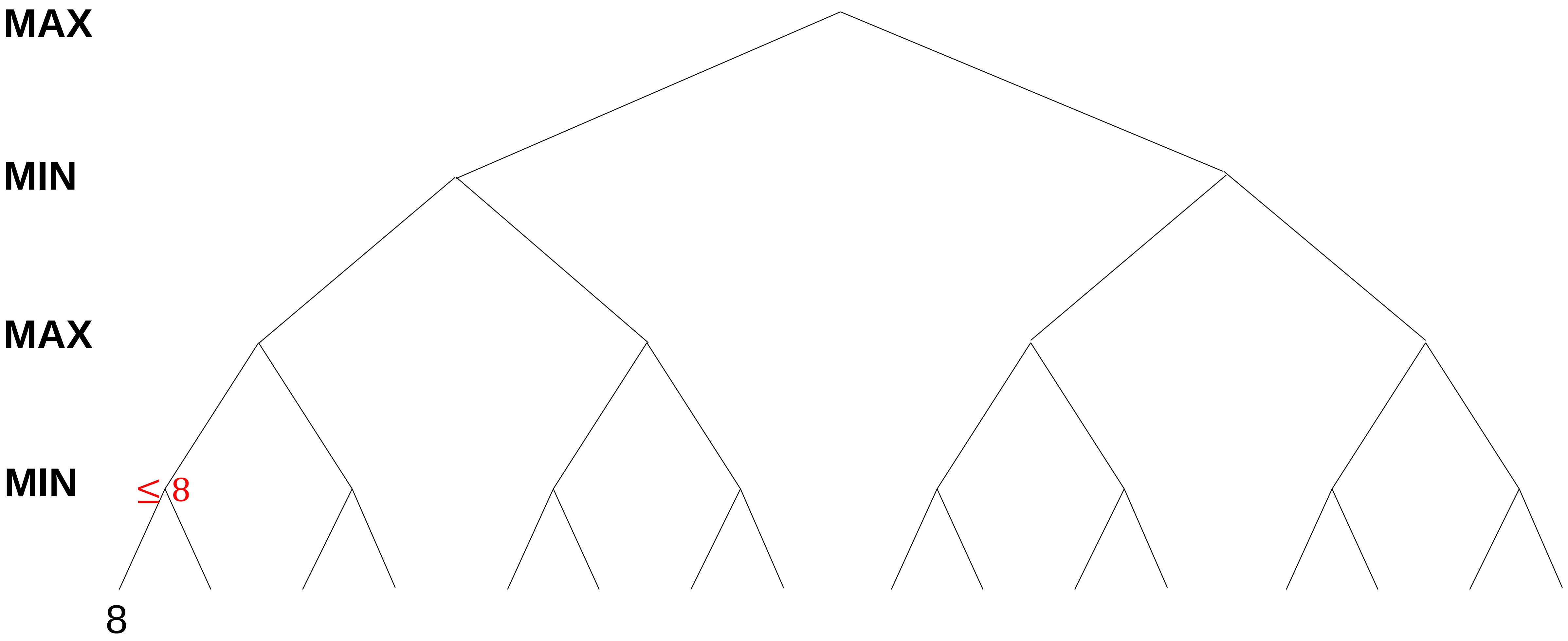
- Same as DFS
  - Time: $O(b^m)$
  - Space: $O(bm)$

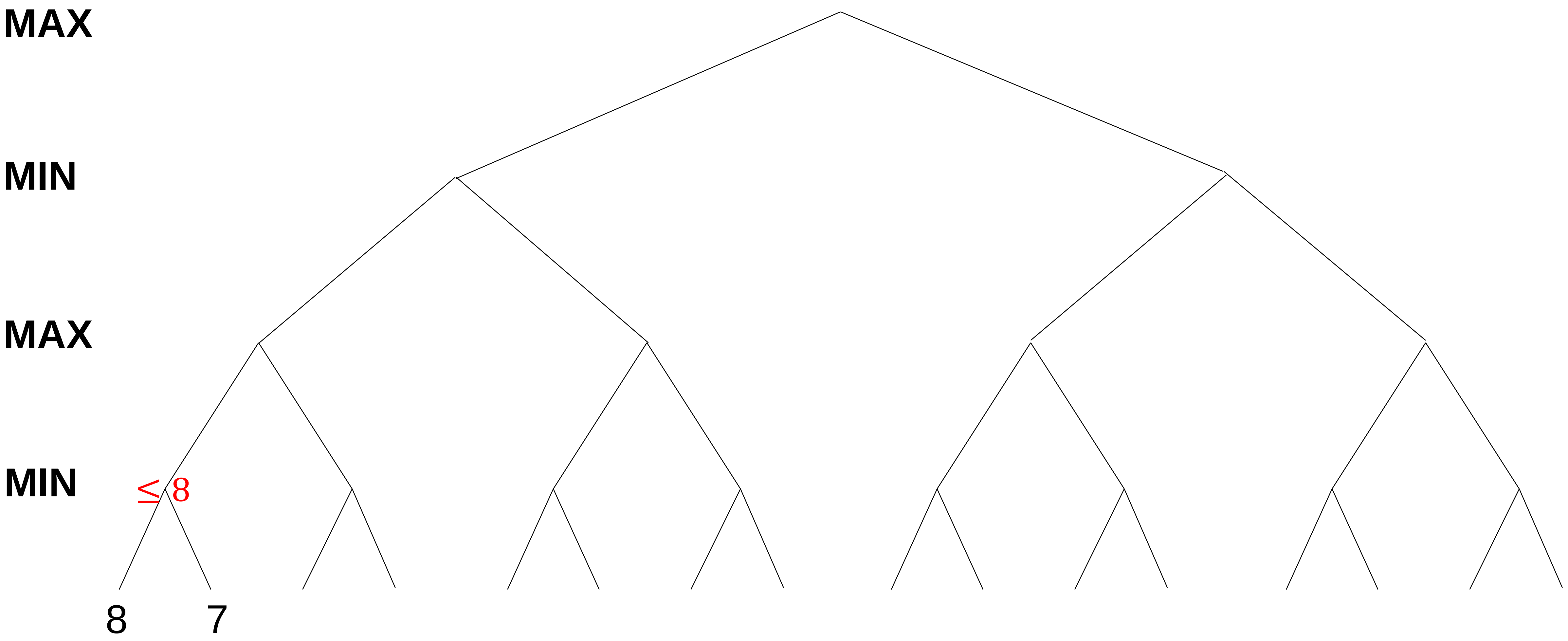
- For chess
  - $b \approx 35$, $m \approx 100$
  - Too large to find the true minimax value/policy

Alpha-Beta Pruning and Evaluation Functions

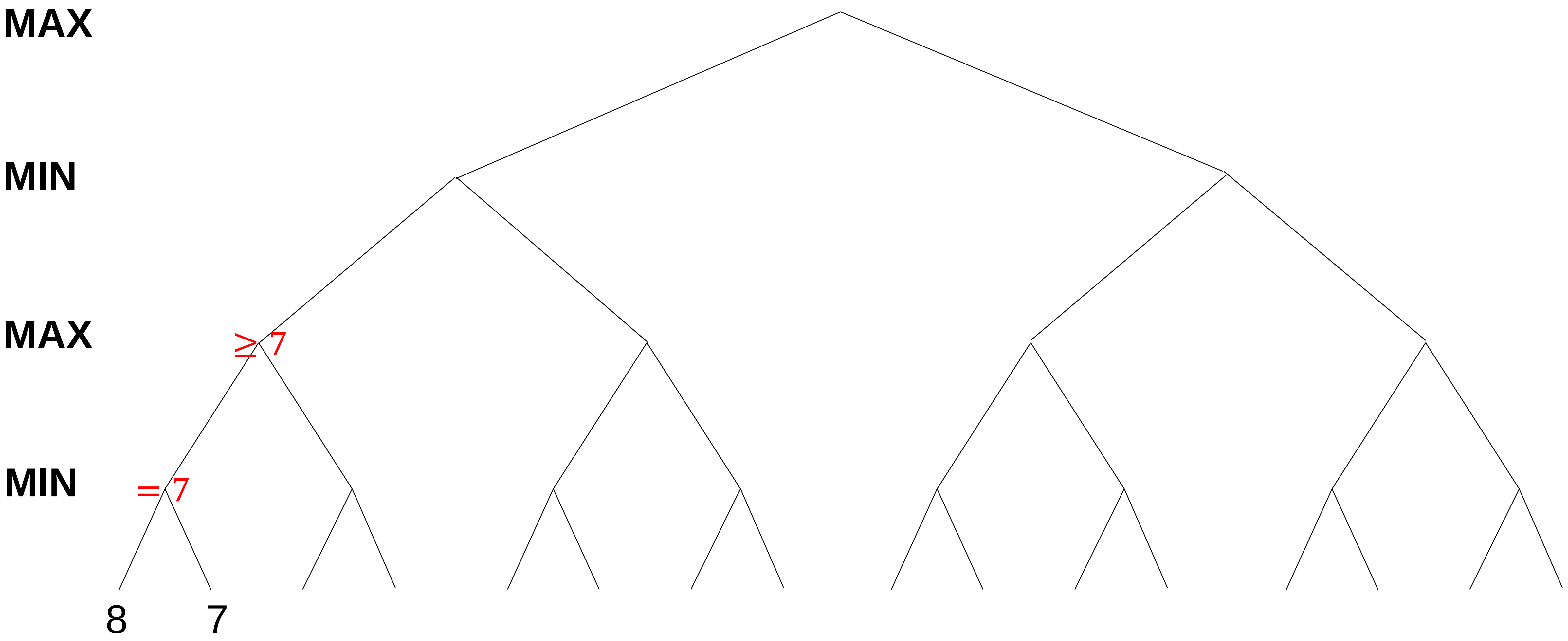

MAX
MIN
2
7
1
8

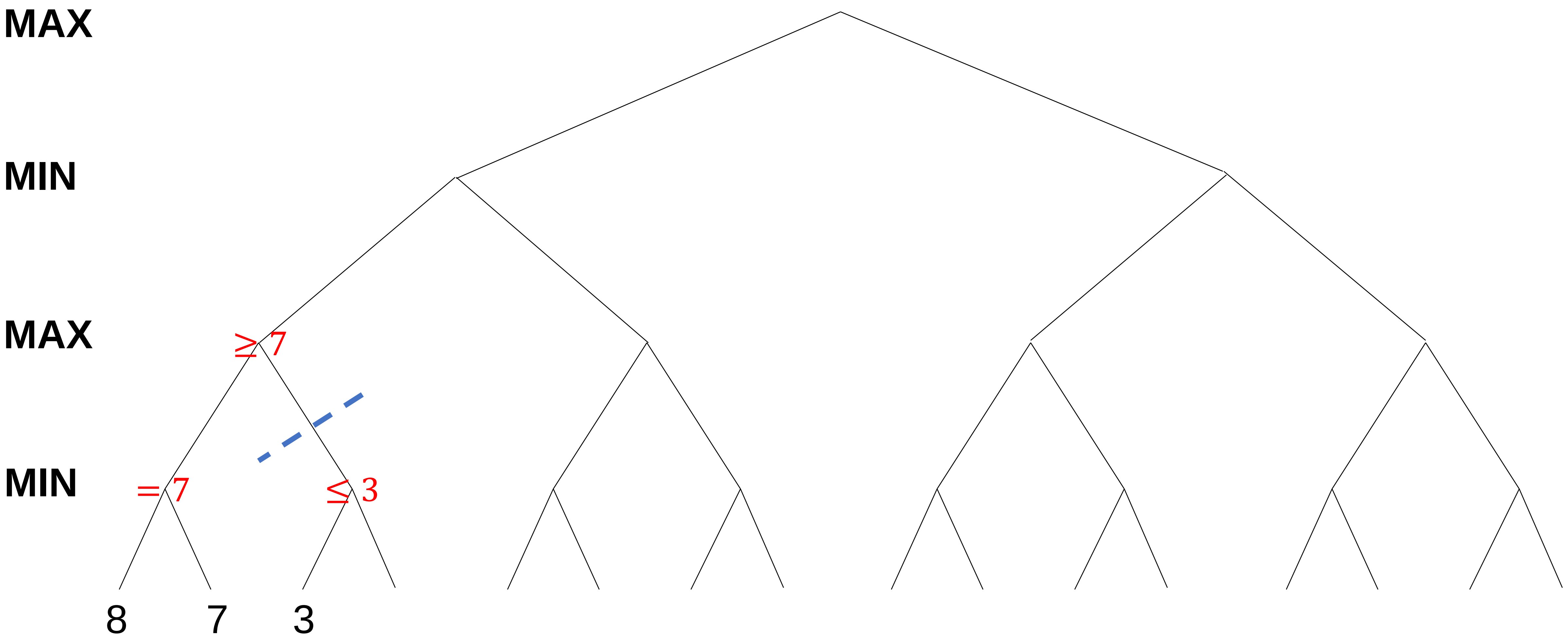

MAX
MIN
MAX
MIN


MAX
MIN
MAX
MIN
≤ 8
8

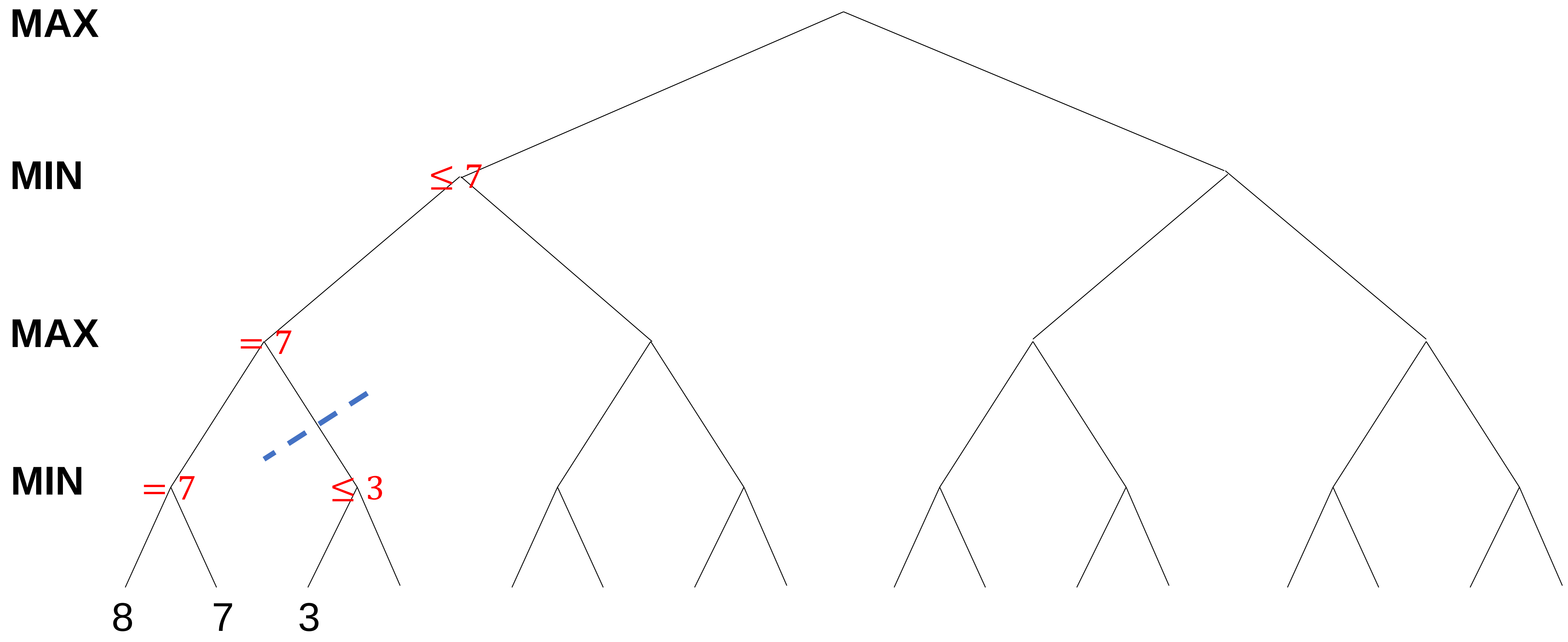


MAX
MIN
MAX
MIN
≤ 8
8
7

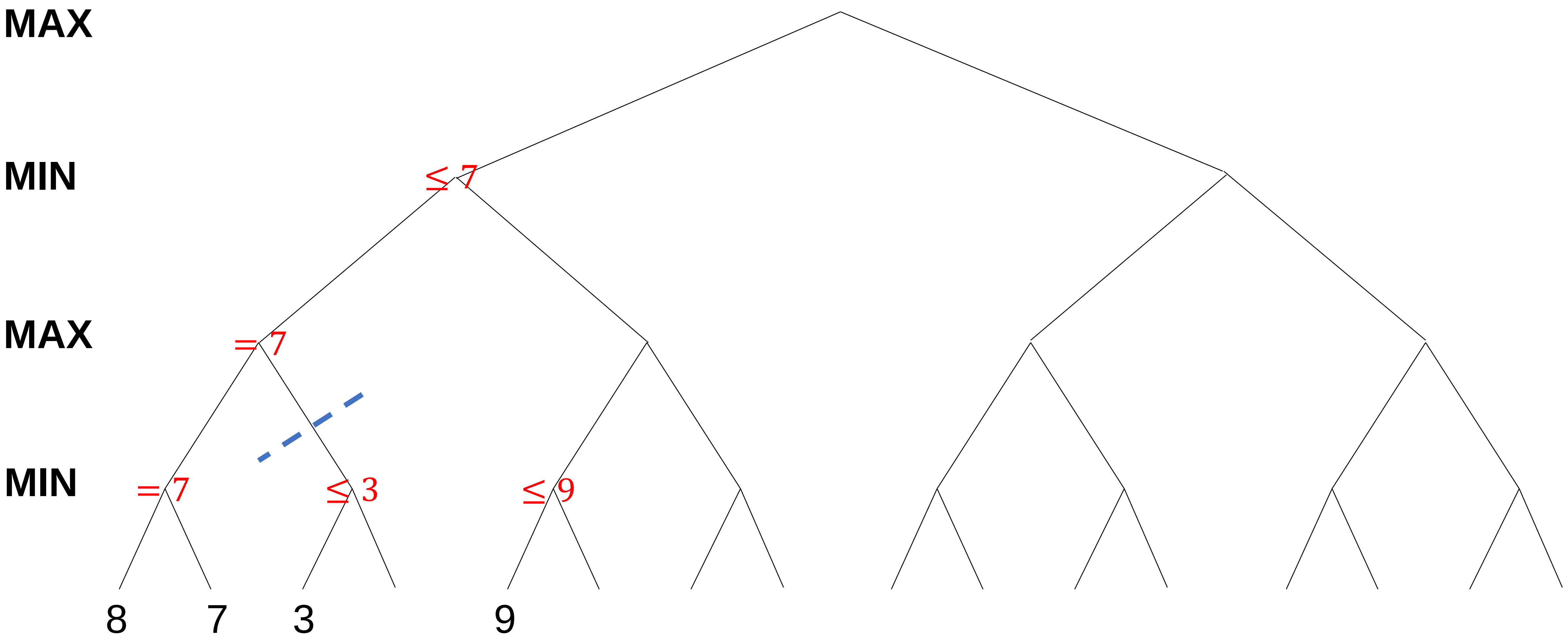

MAX
MIN
MAX
≥ 7
MIN
= 7
8
7



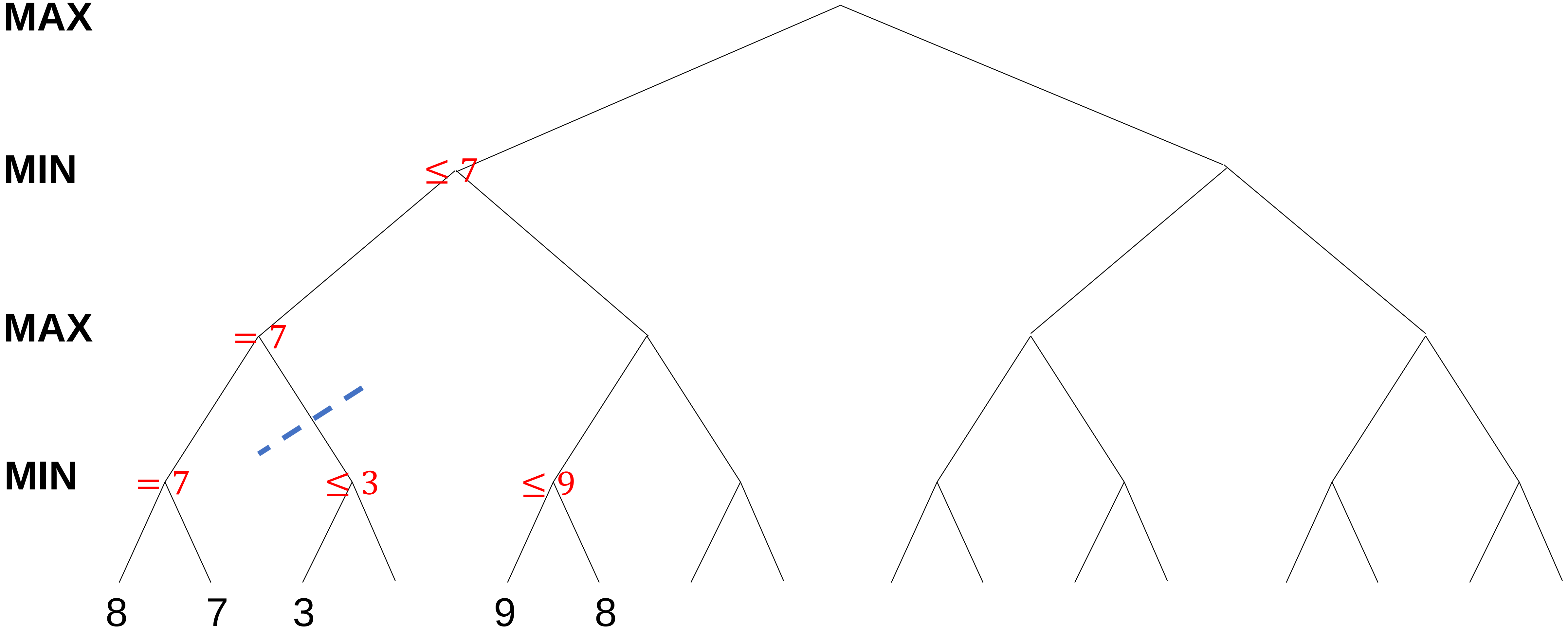
MAX
MIN
MAX
MIN
≥ 7
= 7
≤ 3
8
7
3

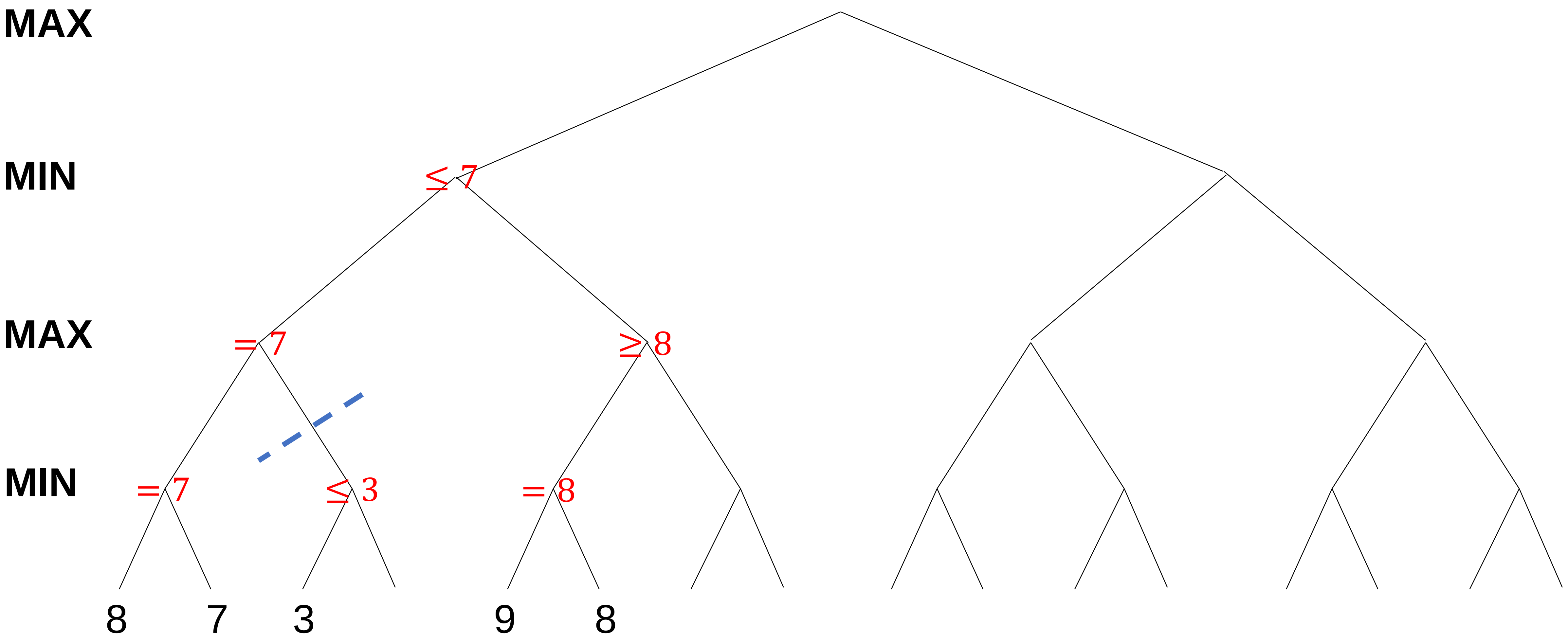


MAX
MIN
MAX
MIN
≤ 7
= 7
= 7
≤ 3
8
7
3

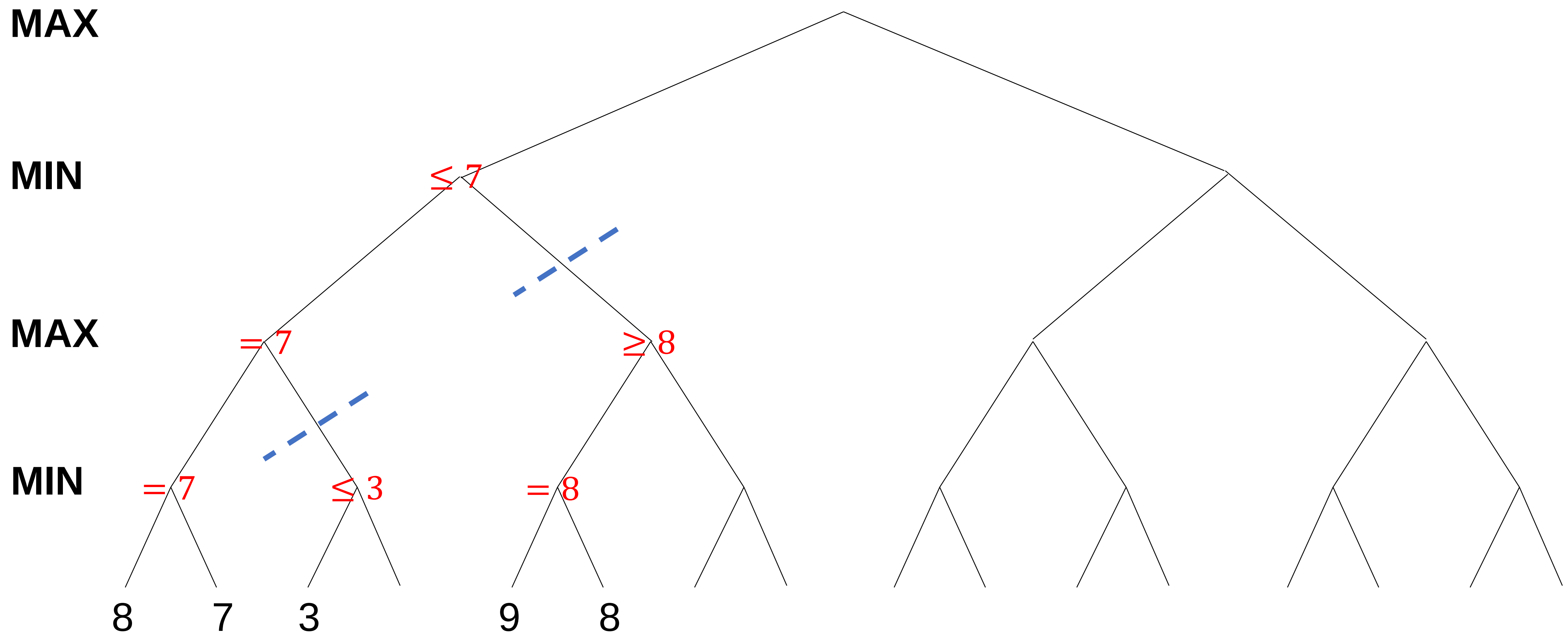


MAX
MIN
MAX
MIN
≤ 7
= 7
= 7
≤ 3
≤ 9
8
7
3
9

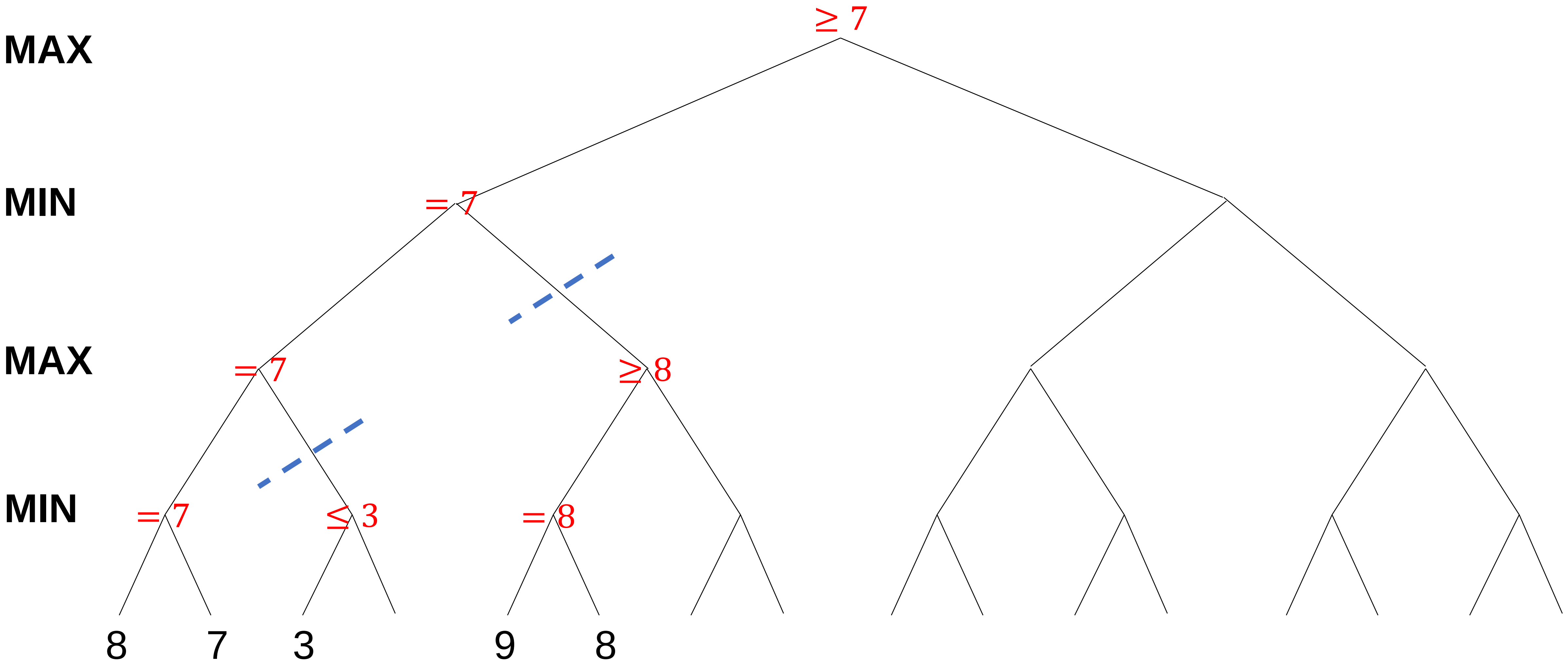

MAX
MIN
MAX
MIN
≤ 7
= 7
= 7
≤ 3
≤ 9
8
7
3
9
8



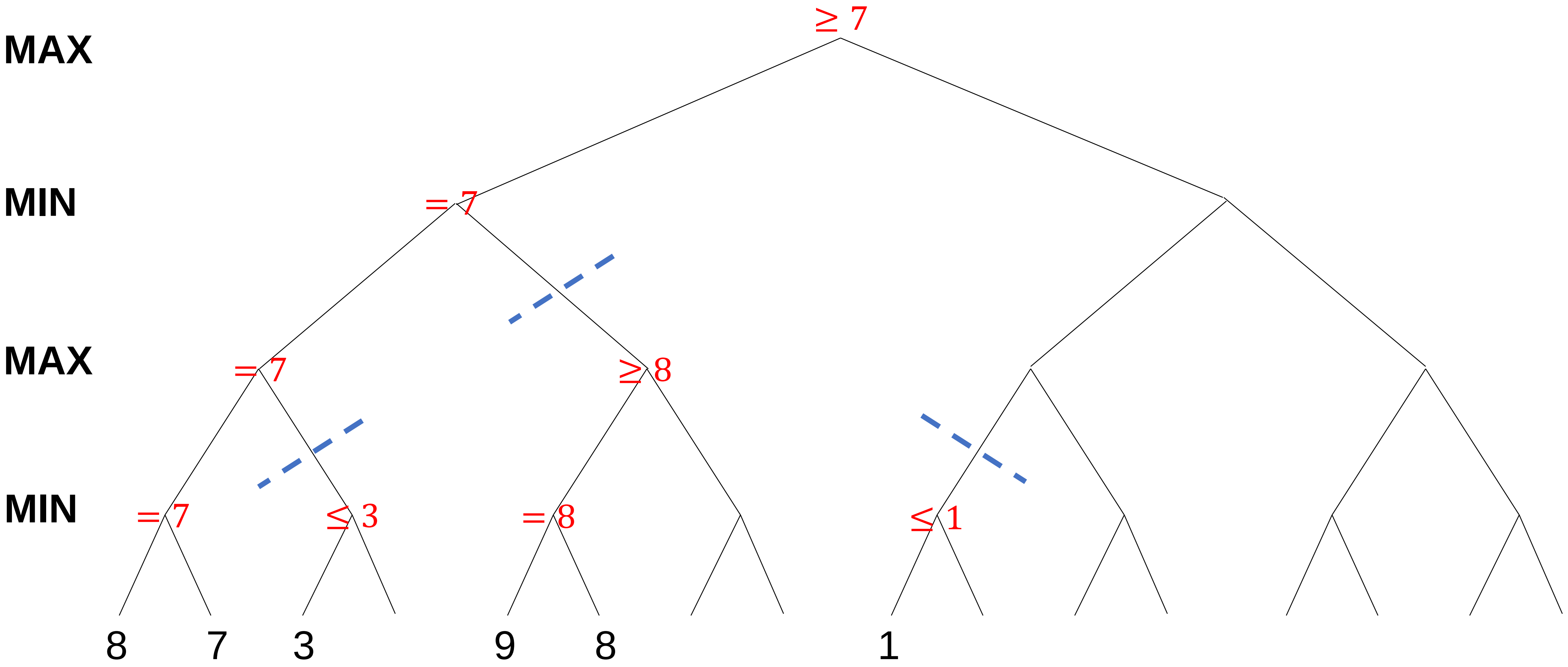
MAX
MIN
MAX
MIN
≤ 7
= 7
≥ 8
= 7
≤ 3
= 8
8
7
3
9
8



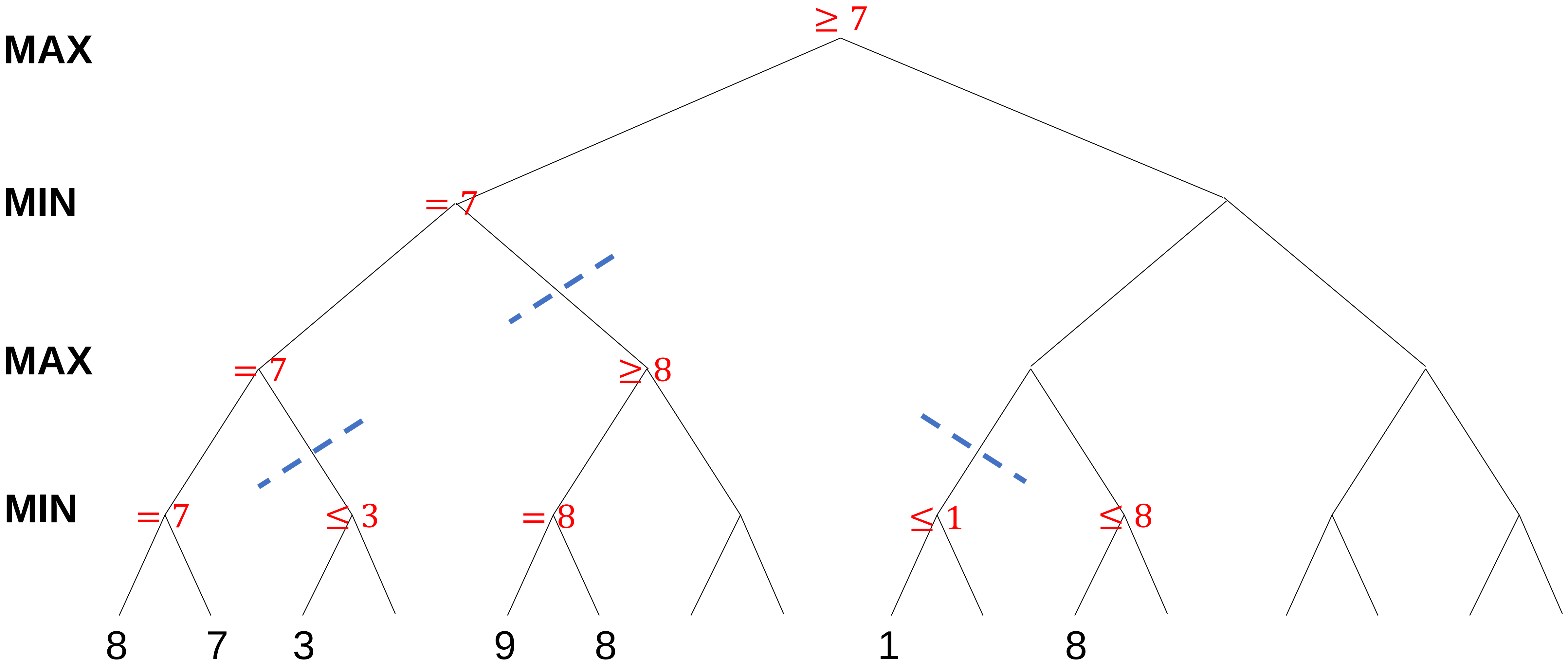
MAX
MIN
MAX
MIN
≤ 7
= 7
≥ 8
= 7
≤ 3
= 8
8
7
3
9
8



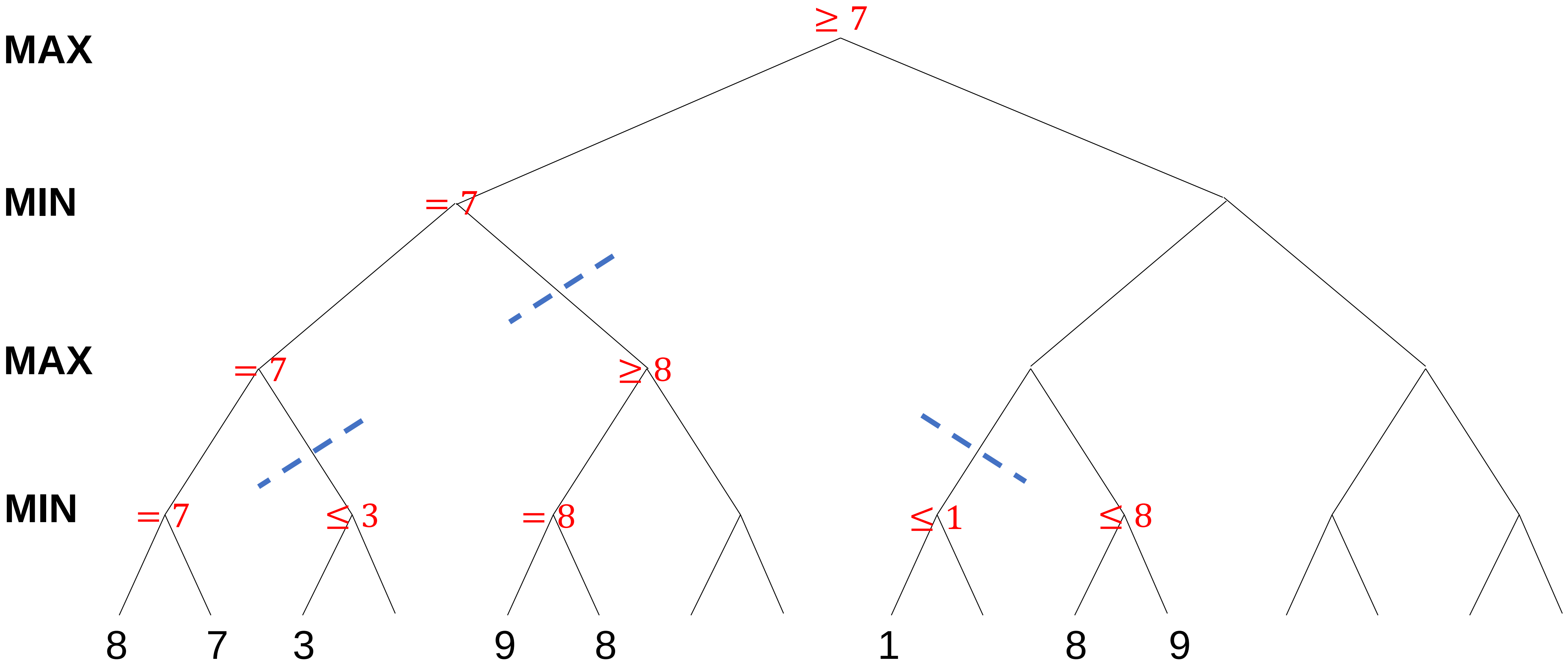
MAX
MIN
MAX
MIN
≥ 7
= 7
= 7
≥ 8
= 7
≤ 3
= 8
8
7
3
9
8

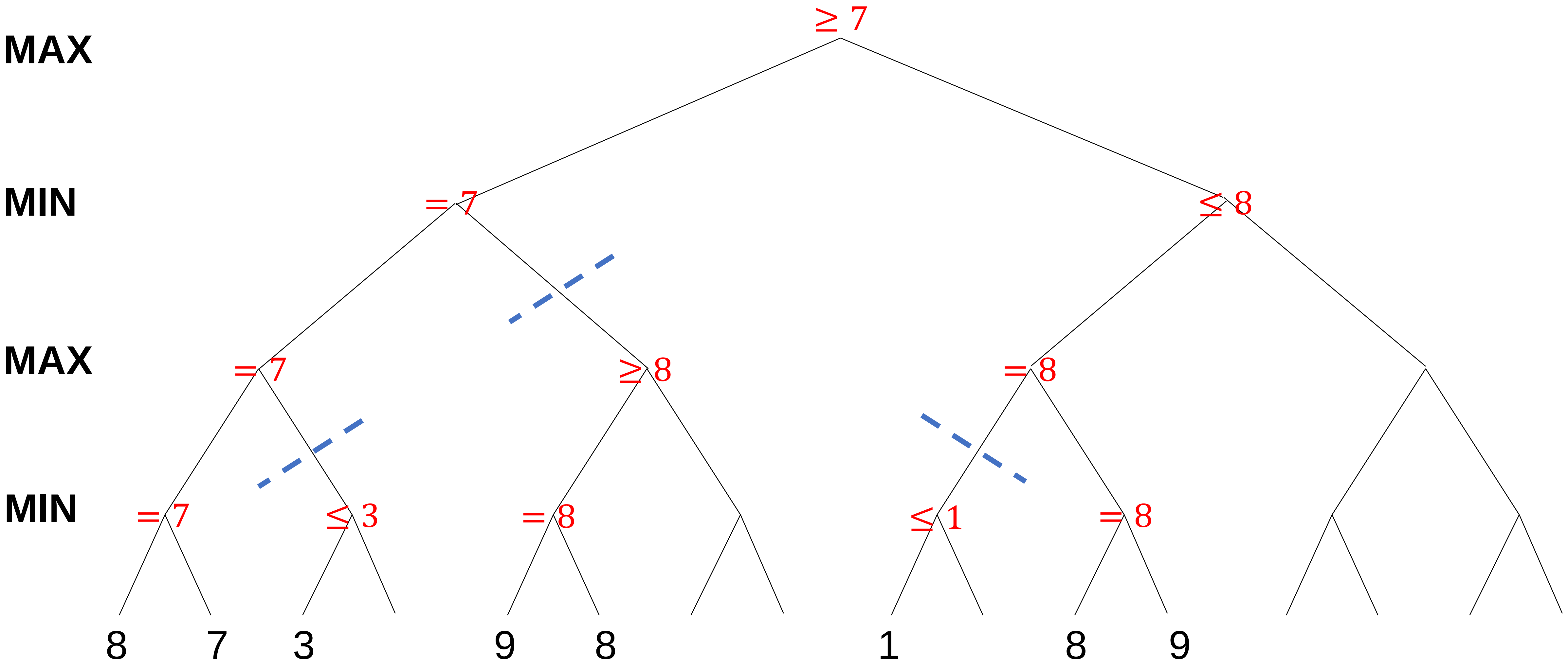


MAX
MIN
MAX
MIN
≥ 7
= 7
= 7
≥ 8
= 7
≤ 3
= 8
≤ 1
8
7
3
9
8
1

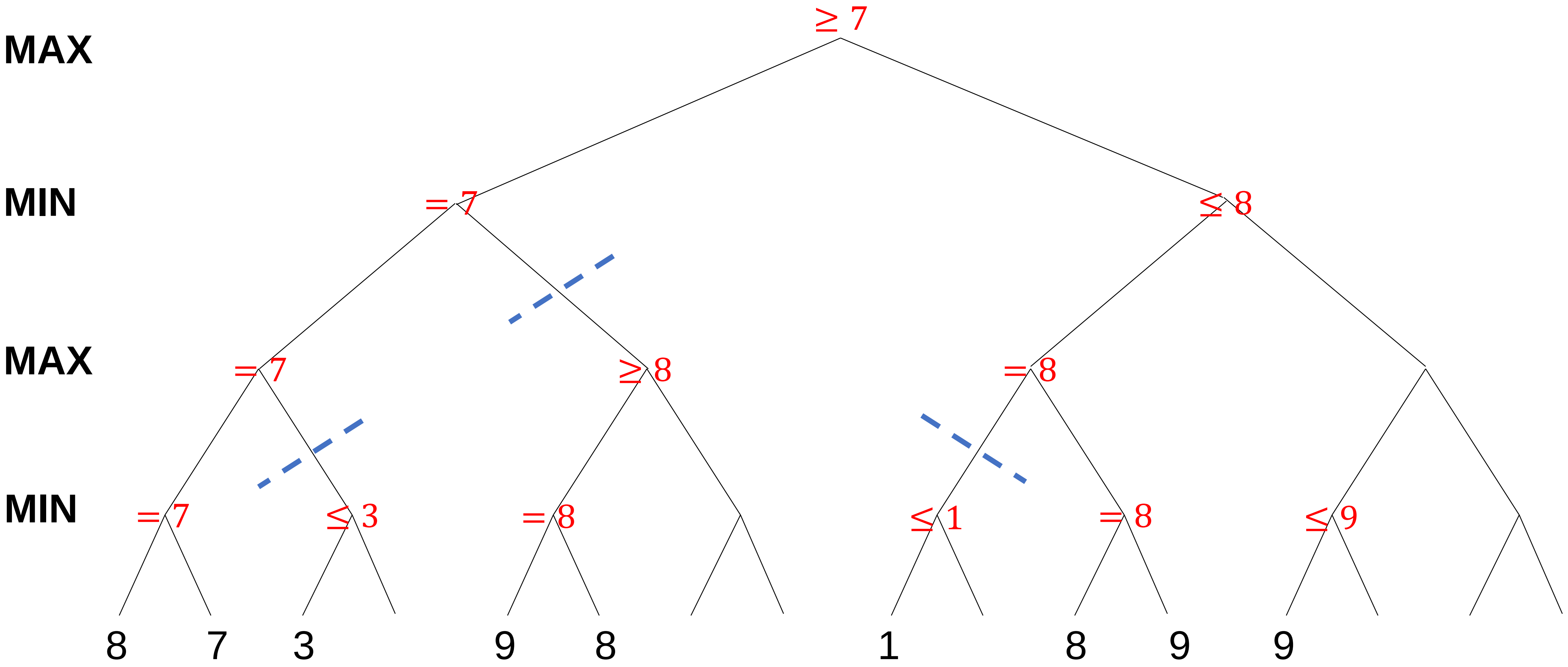


MAX
MIN
MAX
MIN
≥ 7
= 7
= 7
≥ 8
= 7
≤ 3
= 8
≤ 1
≤ 8
8
7
3
9
8
1
8

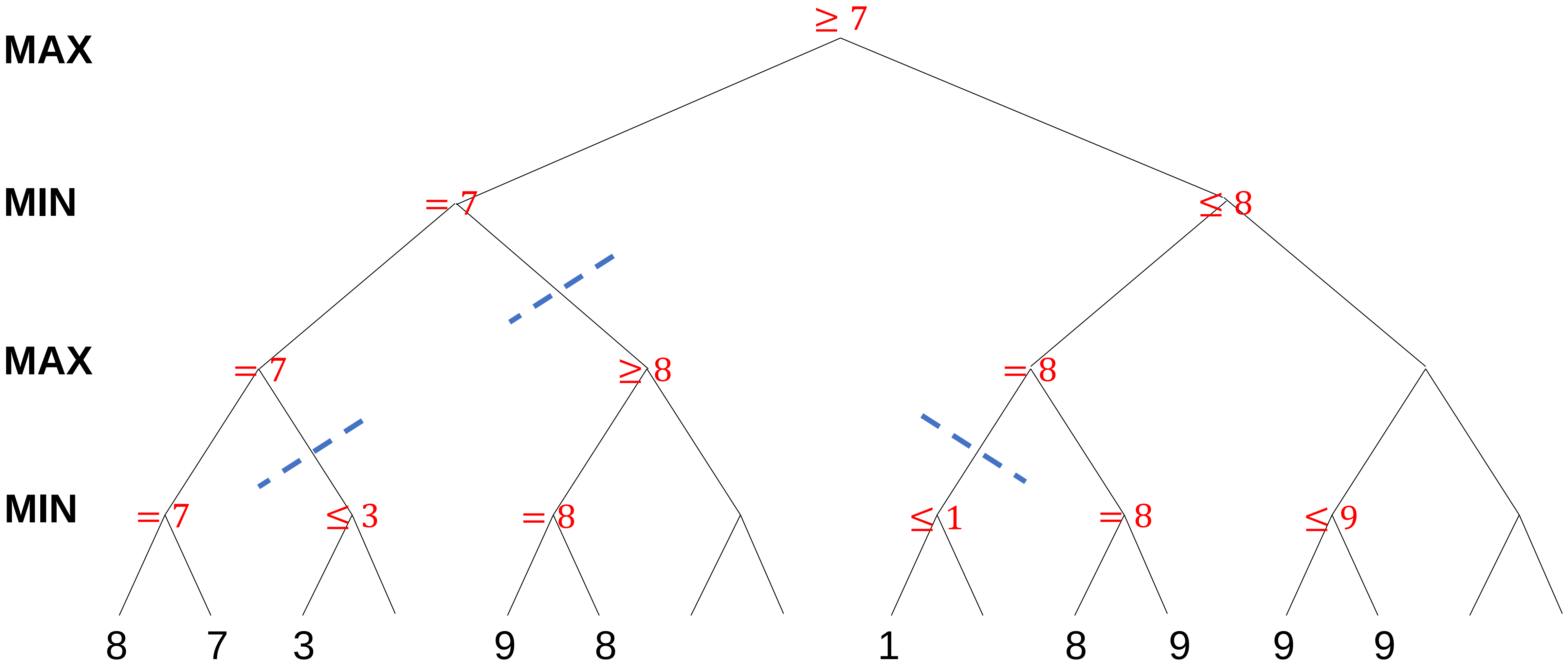

MAX
MIN
MAX
MIN
≥ 7
= 7
= 7
≥ 8
= 7
≤ 3
= 8
≤ 1
≤ 8
8
7
3
9
8
1
8
9

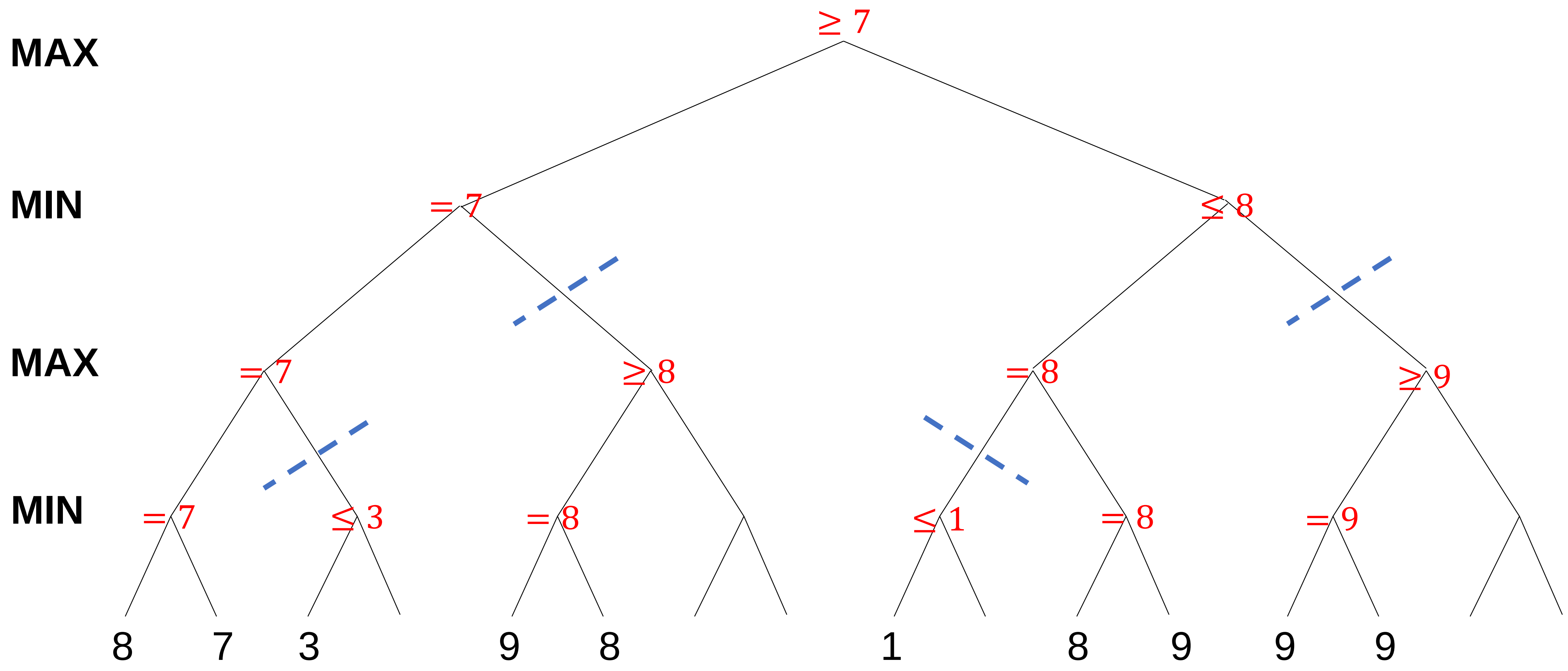


MAX
MIN
MAX
MIN
≥ 7
= 7
≤ 8
= 7
≥ 8
= 8
= 7
≤ 3
= 8
≤ 1
= 8
8
7
3
9
8
1
8
9

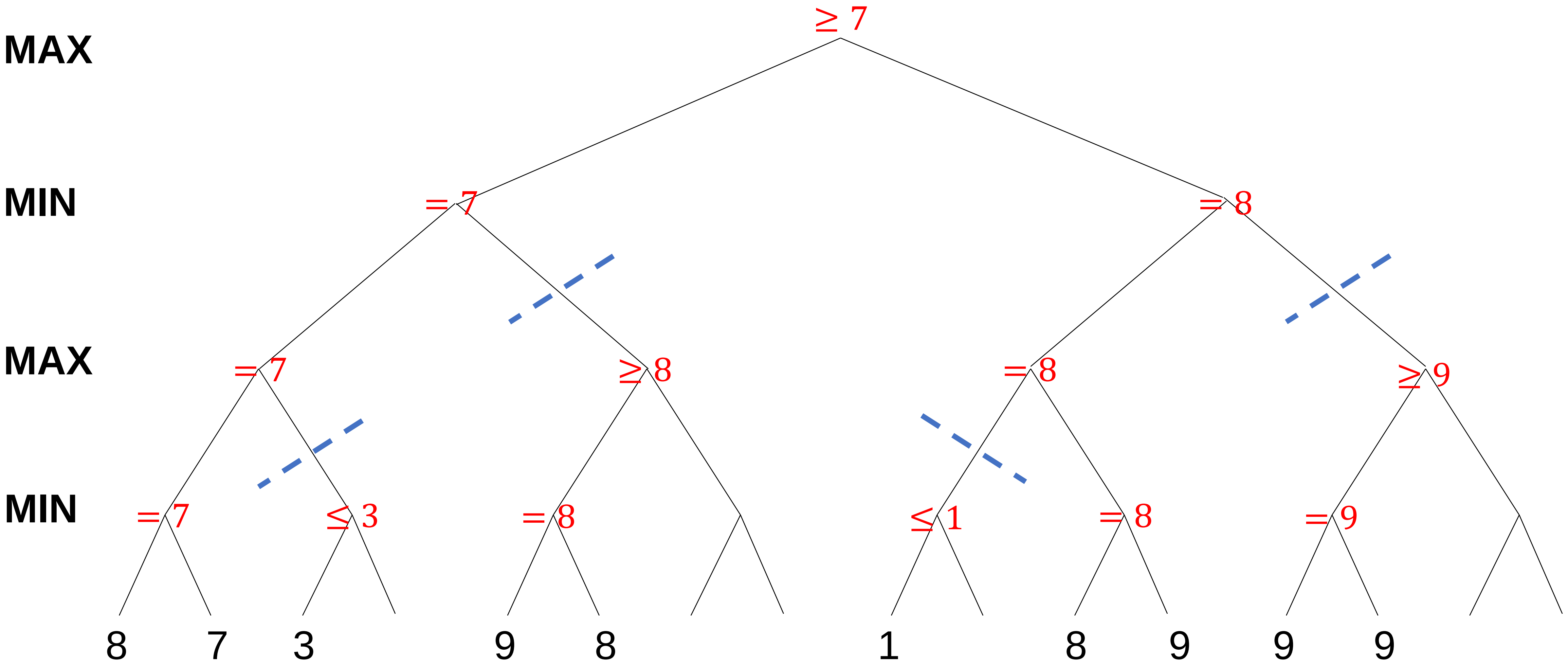


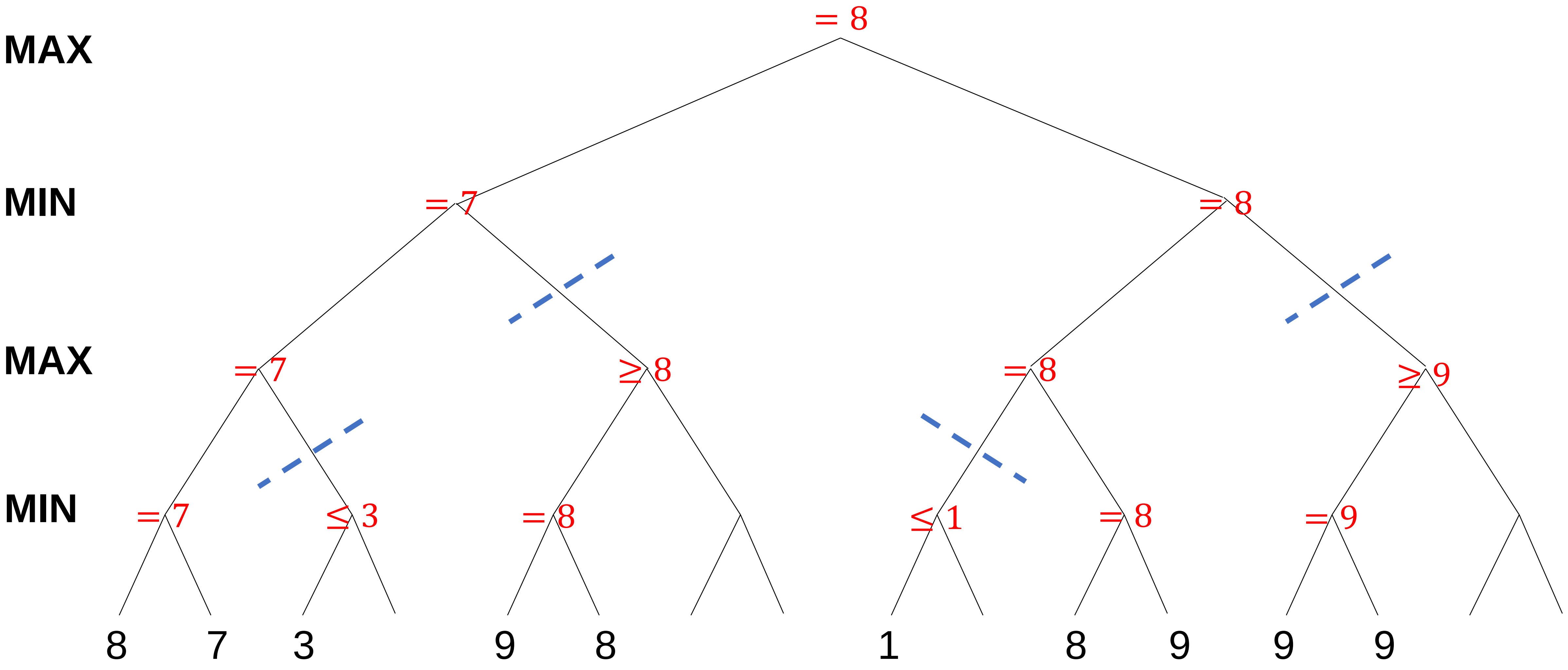
MAX
MIN
MAX
MIN
≥ 7
= 7
≤ 8
= 7
≥ 8
= 8
= 7
≤ 3
= 8
≤ 1
= 8
≤ 9
8
7
3
9
8
1
8
9
9


MAX
MIN
MAX
MIN
≥ 7
= 7
≤ 8
= 7
≥ 8
= 8
= 7
≤ 3
= 8
≤ 1
= 8
≤ 9
8
7
3
9
8
1
8
9
9
9



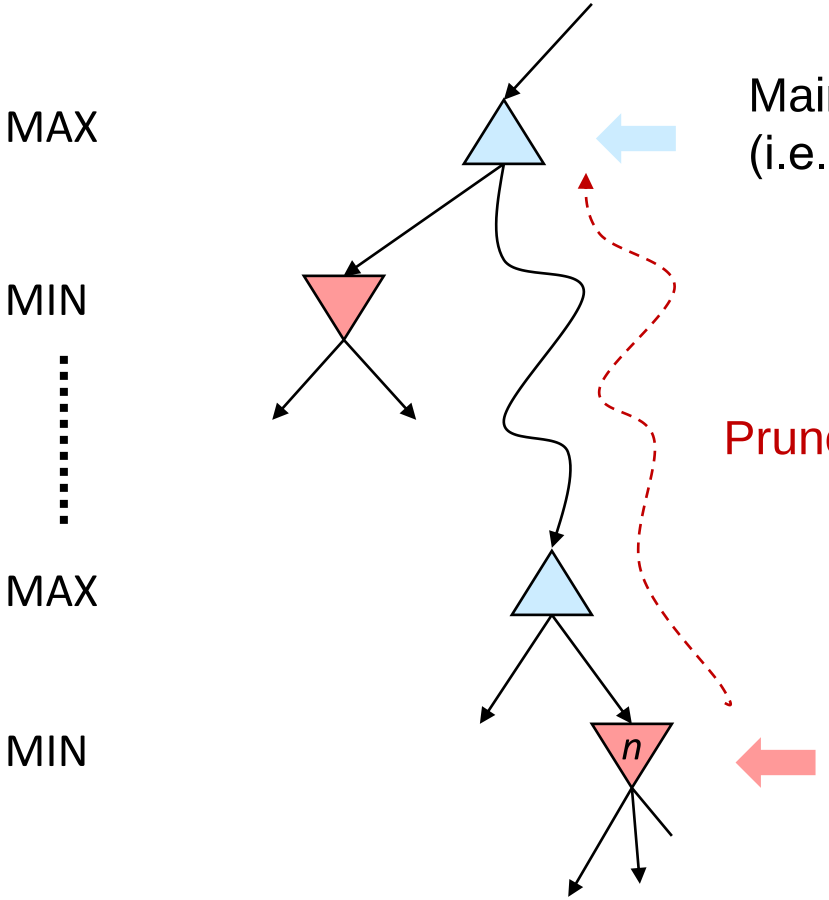
MAX
MIN
MAX
MIN
≥ 7
= 7
≤ 8
= 7
≥ 8
= 8
≥ 9
= 7
≤ 3
= 8
≤ 1
= 8
= 9
8
7
3
9
8
1
8
9
9
9


MAX
MIN
MAX
MIN
≥ 7
= 7
= 8
= 7
≥ 8
= 8
≥ 9
= 7
≤ 3
= 8
≤ 1
= 8
= 9
8
7
3
9
8
1
8
9
9
9



MAX
MIN
MAX
MIN
= 8
= 7
= 8
= 7
≥ 8
= 8
≥ 9
= 7
≤ 3
= 8
≤ 1
= 8
= 9
8
7
3
9
8
1
8
9
9
9

Alpha-Beta Pruning



Maintain a value **lower bound**
(i.e., “≥”) that increases over time

Prune if there is inconsistency with ancestors

Maintain a value **upper bound**
(i.e., “≤”) that decreases over time

Alpha-Beta Pruning

α: MAX's best option on path to root
β: MIN's best option on path to root

```
def max-value(state, α, β):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

Alpha-Beta Pruning

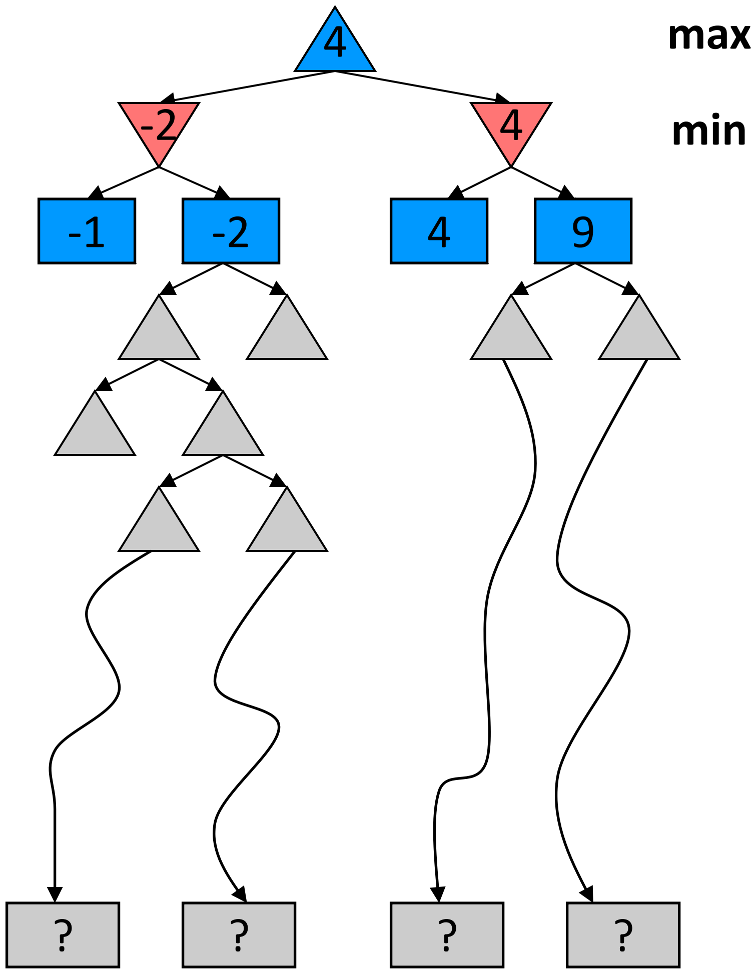
- The pruning has **no effect** on the minimax value computed for the root.

- Child ordering affects the efficiency
  - If a MAX node finds a larger children value (or a MIN node finds a smaller children value) quicker, then more time can be saved.

- With perfect ordering, the time complexity drops to $O(b^{m/2})$
  - Doubles solvable depth
  - Full search of, e.g., chess, is still hopeless

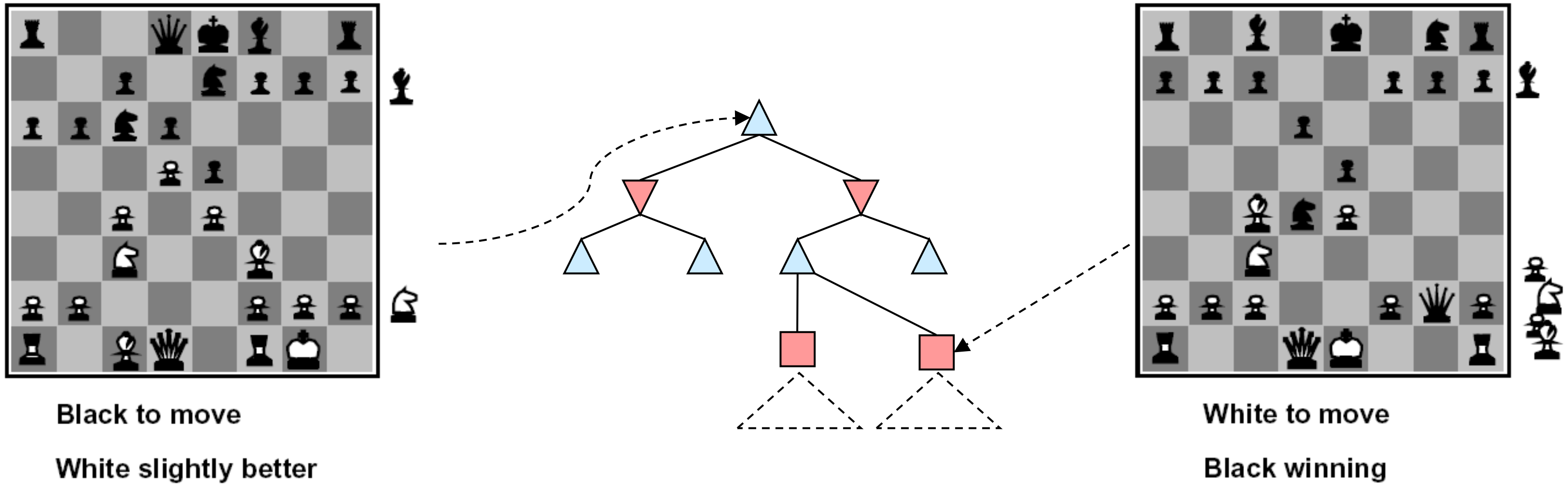
Resource Limits

- In realistic games, cannot search to leaves
- **Solution:** depth-limited search
  - Search only to a limited depth
  - At the last layer of the search, call the evaluation function (heuristic function)
- **Example**
  - Suppose we have 100 seconds, can explore 10K nodes / sec
  - So can check 1M nodes per move
  - α-β reaches about depth 8 – decent chess program
- Use iterative deepening for an anytime algorithm



Evaluation Functions

- Evaluation functions score non-terminal nodes in depth-limited search



- e.g., weighted linear sum of features:

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

where $f_1(s)$ = (num white queens – num black queens), etc.

Next Lecture

- Expectimax
- Monte-Carlo Tree Search