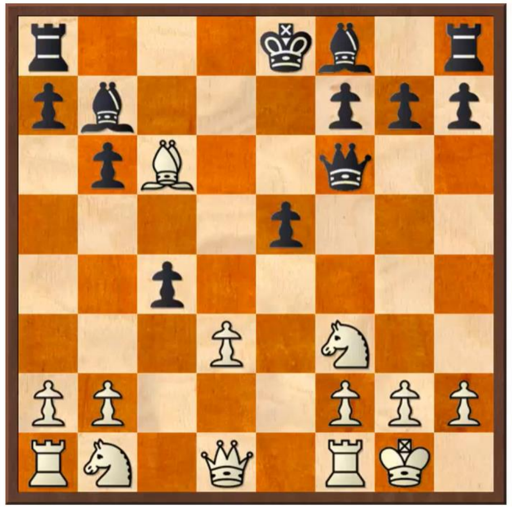
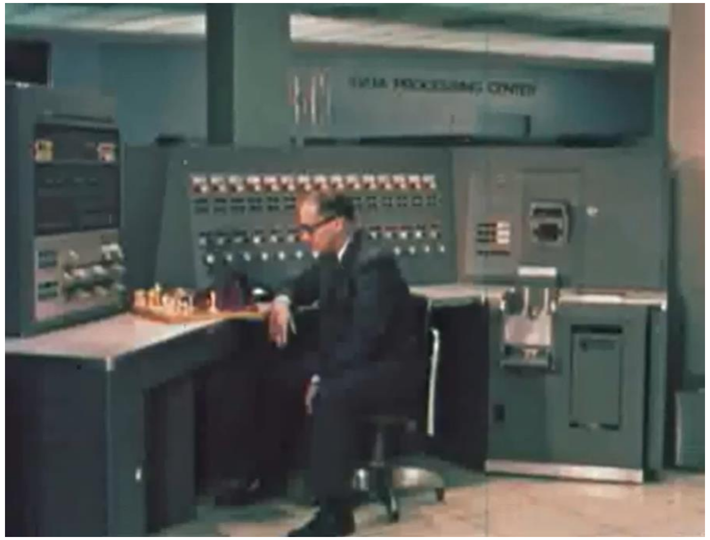
Search in Games

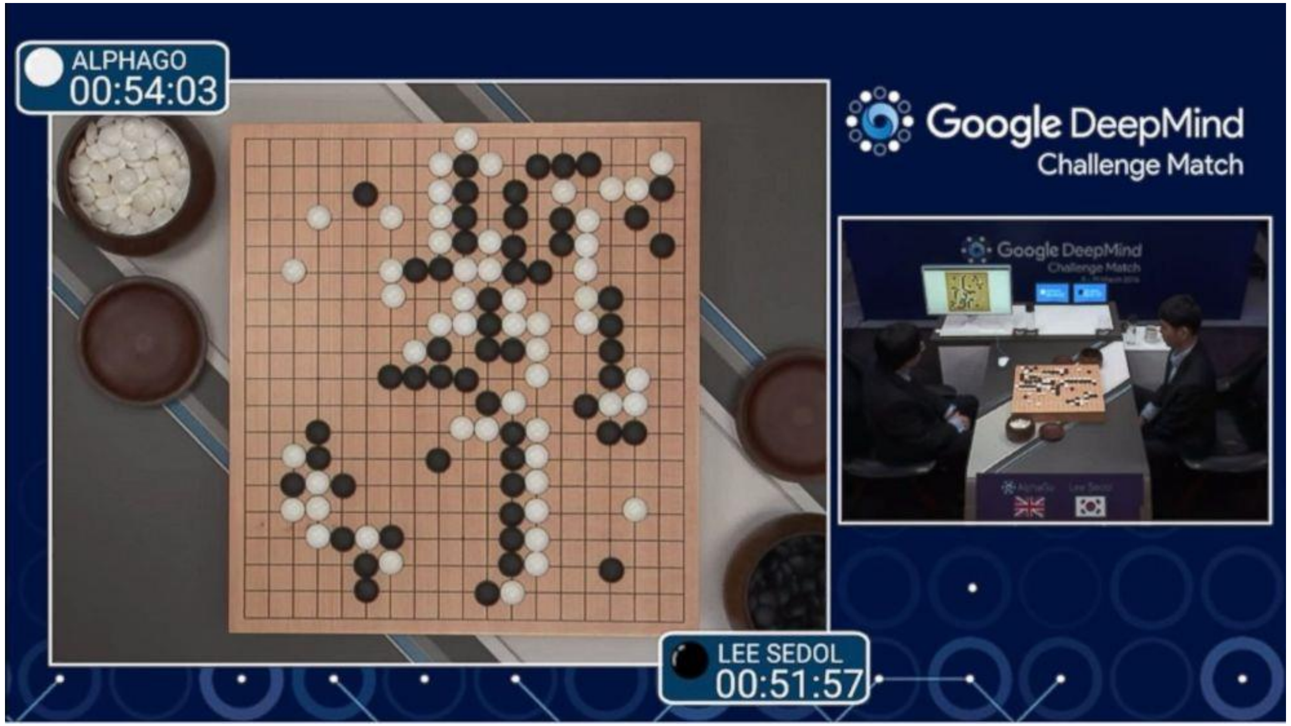
Chen-Yu Wei

Bernstein



Computer





Turn-Based Two-Player Game

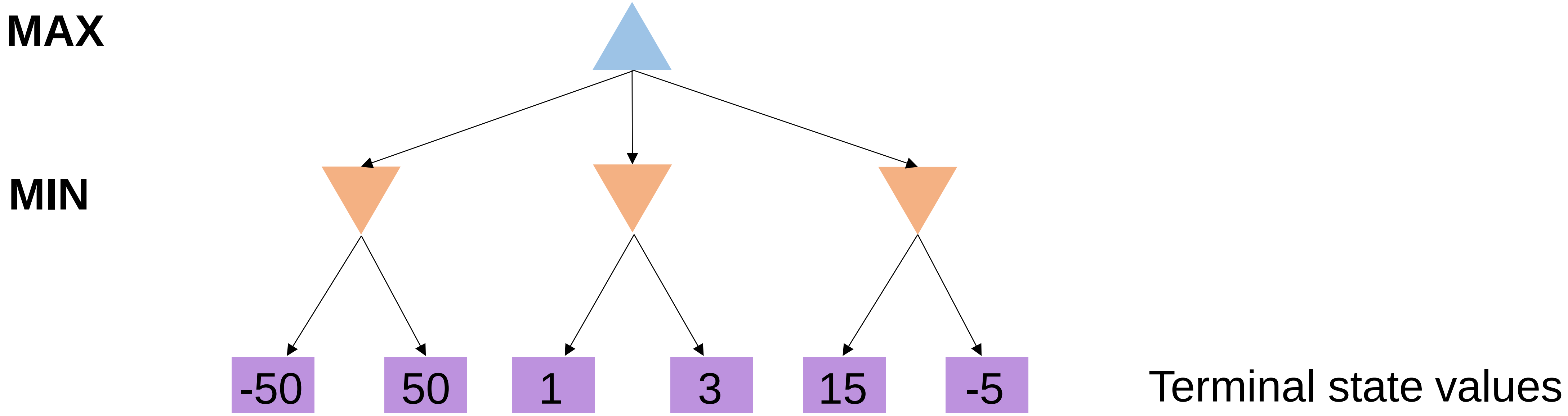
You choose one of the three bins. I choose a number from that bin. Your goal is to maximize the chosen number.

A	B	C
-50 50	1 3	15 -5

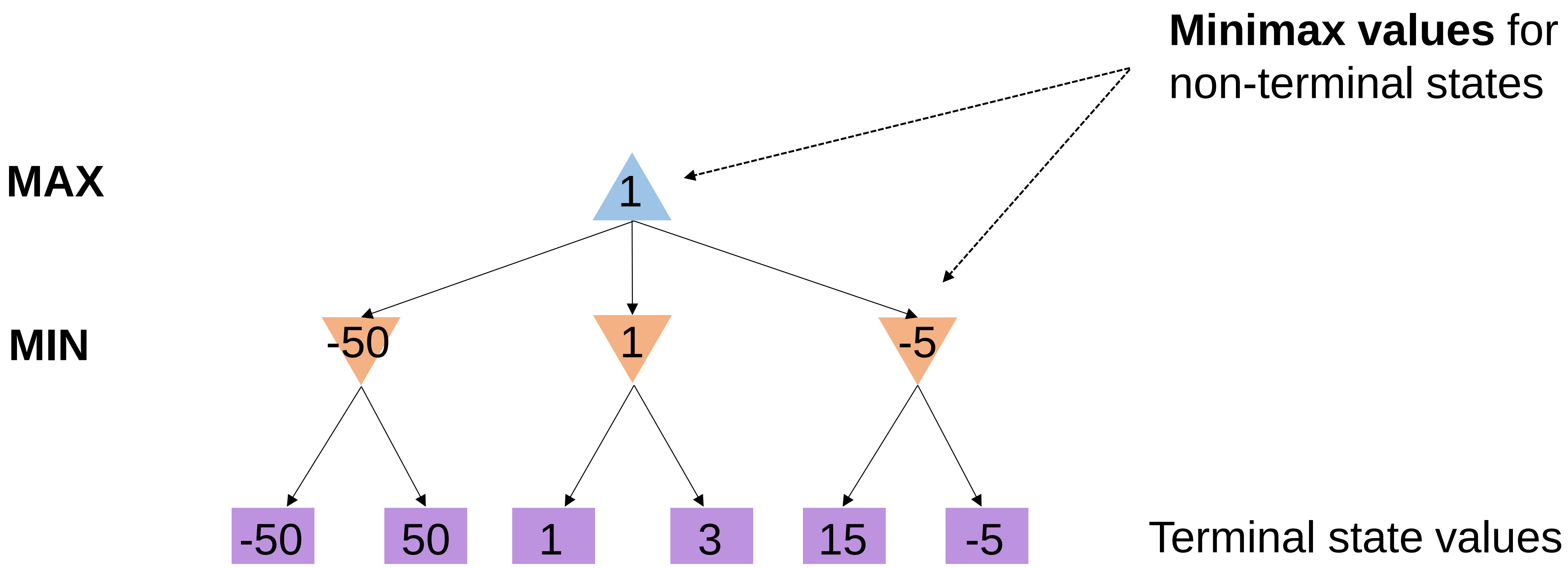
If I am

- adversarial
- random
- benign/cooperative

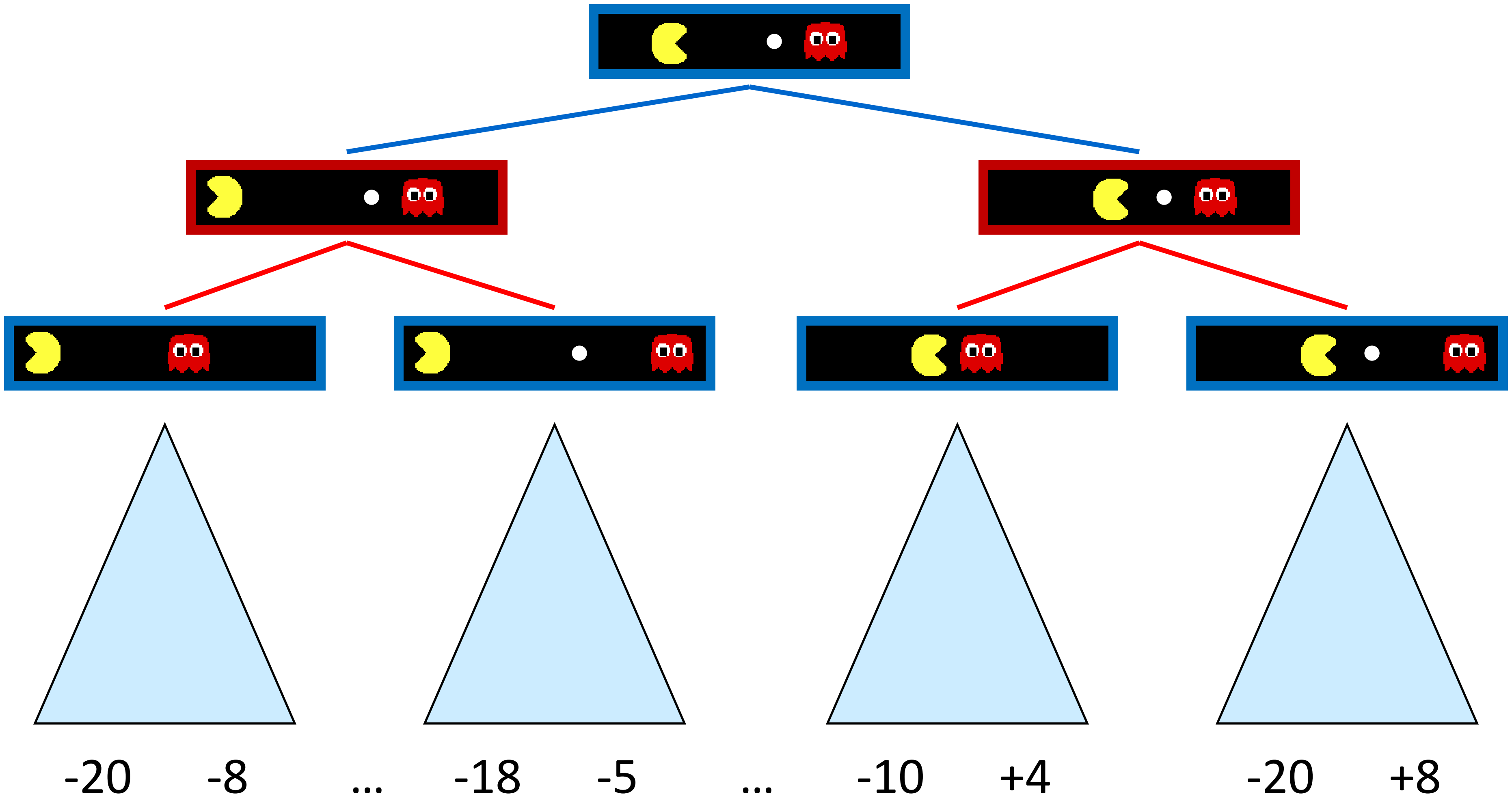
Turn-Based Two-Player Zero-Sum Games



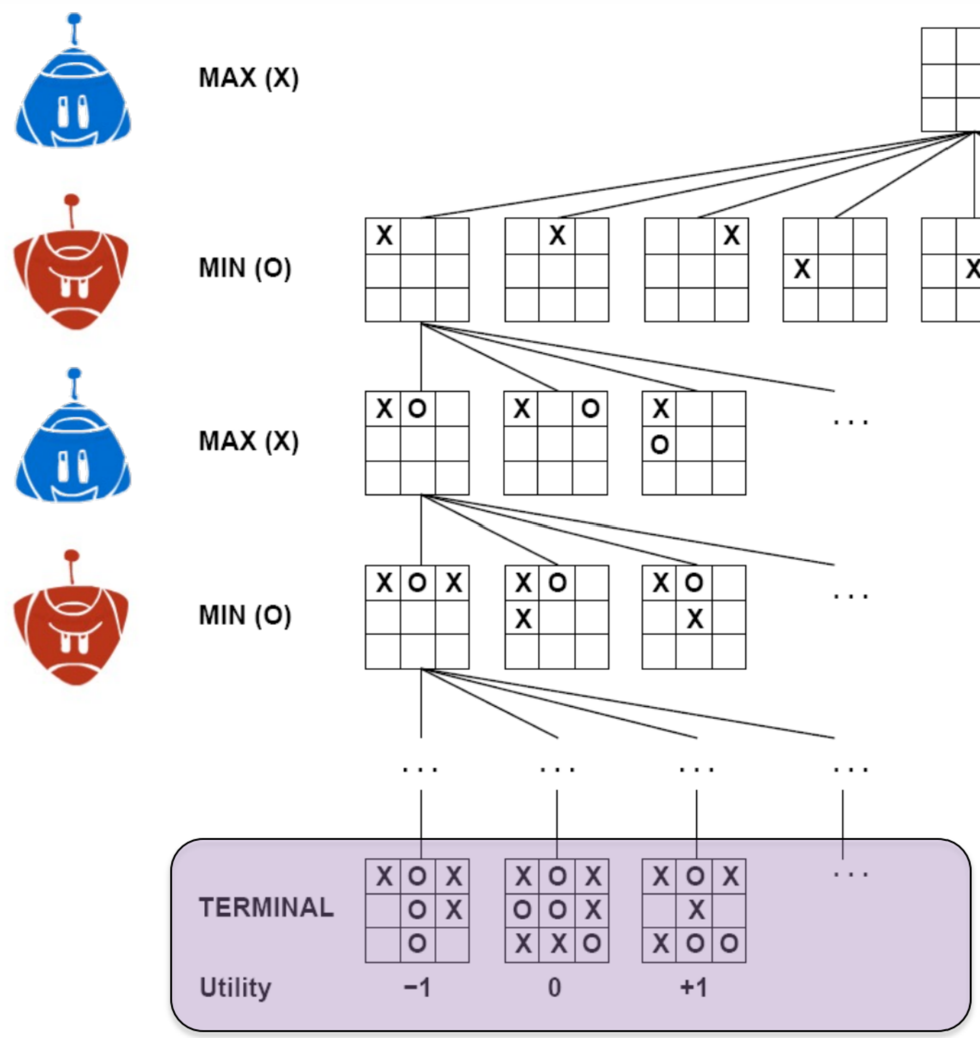
Turn-Based Two-Player Zero-Sum Games

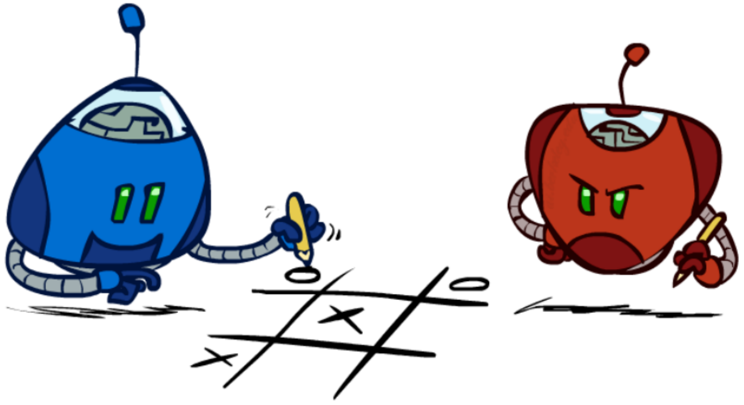


Example: PACMAN



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
```

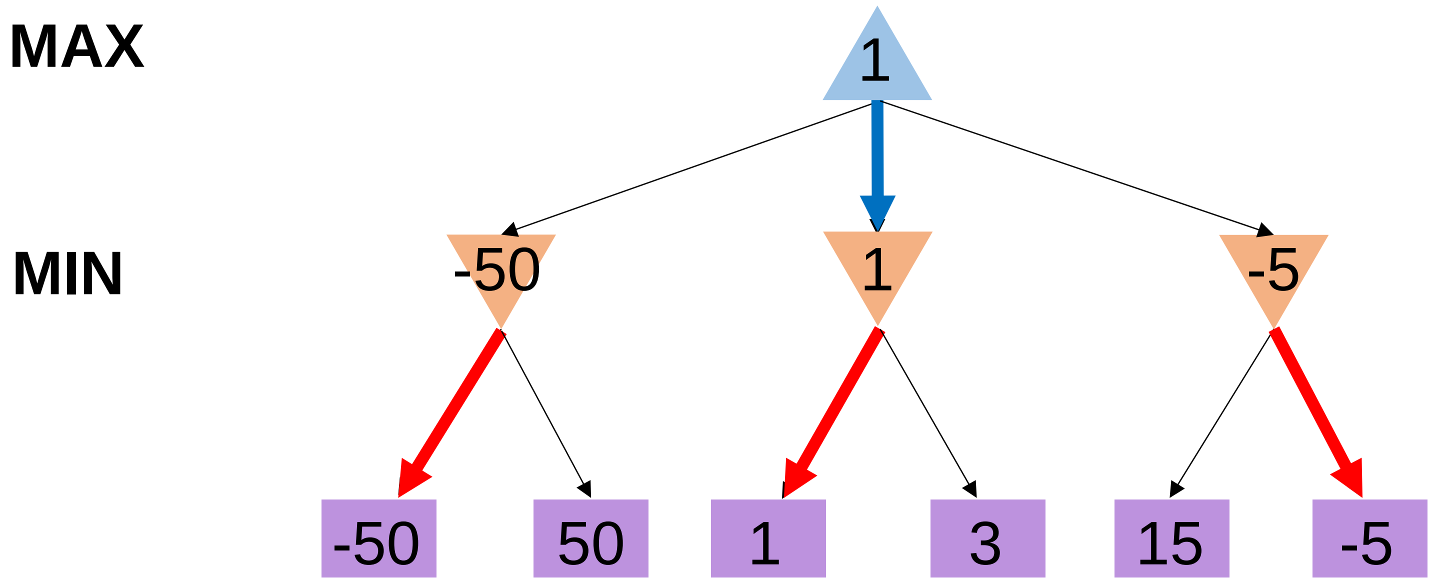
The Minimax Policy

“**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

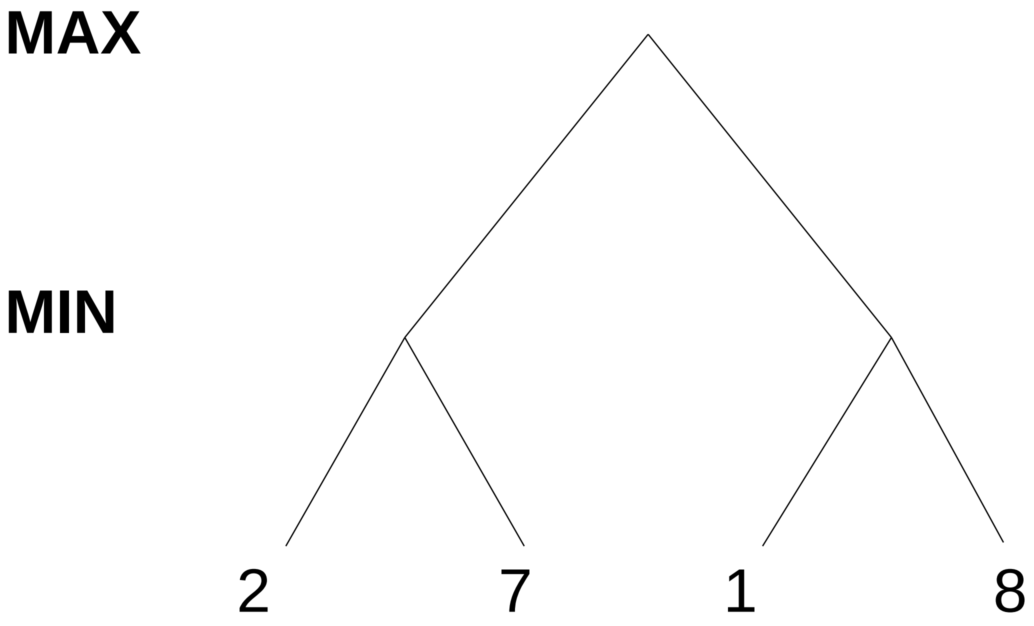


Time / Space Complexity

- 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

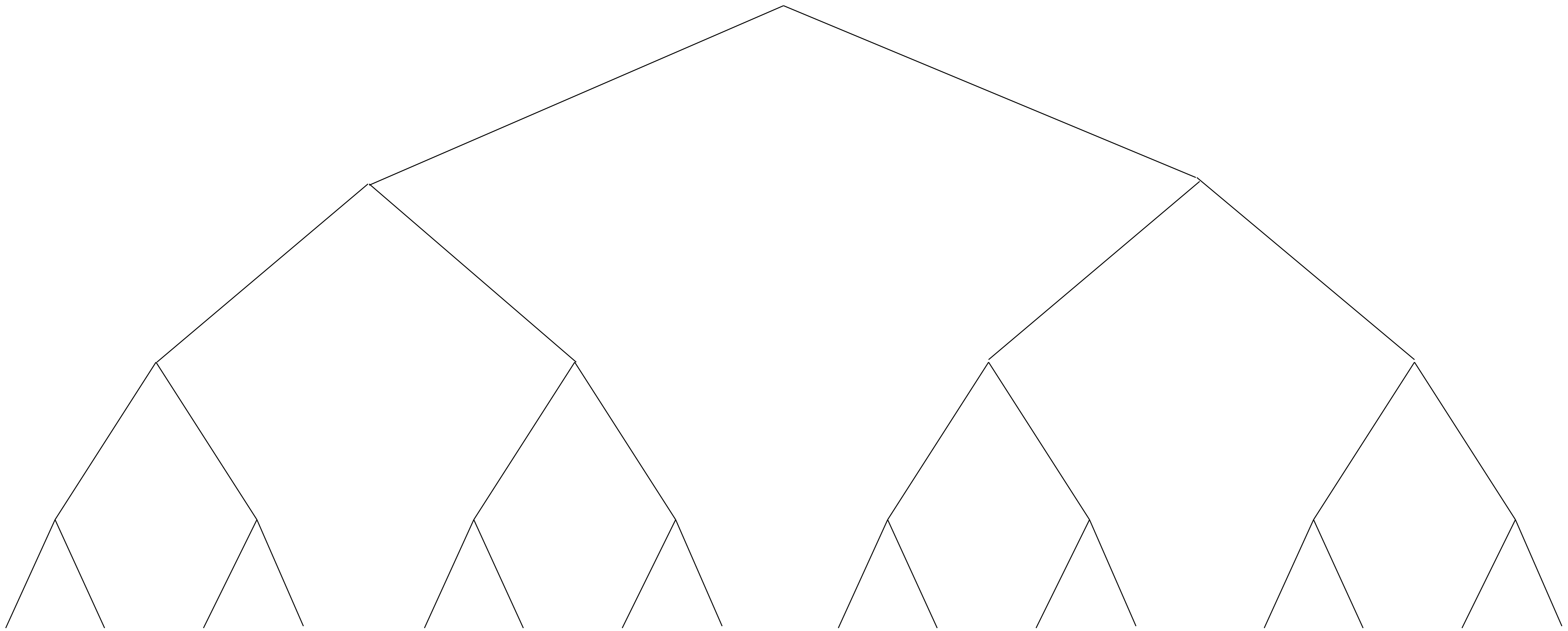

MAX
MIN
2
7
1
8

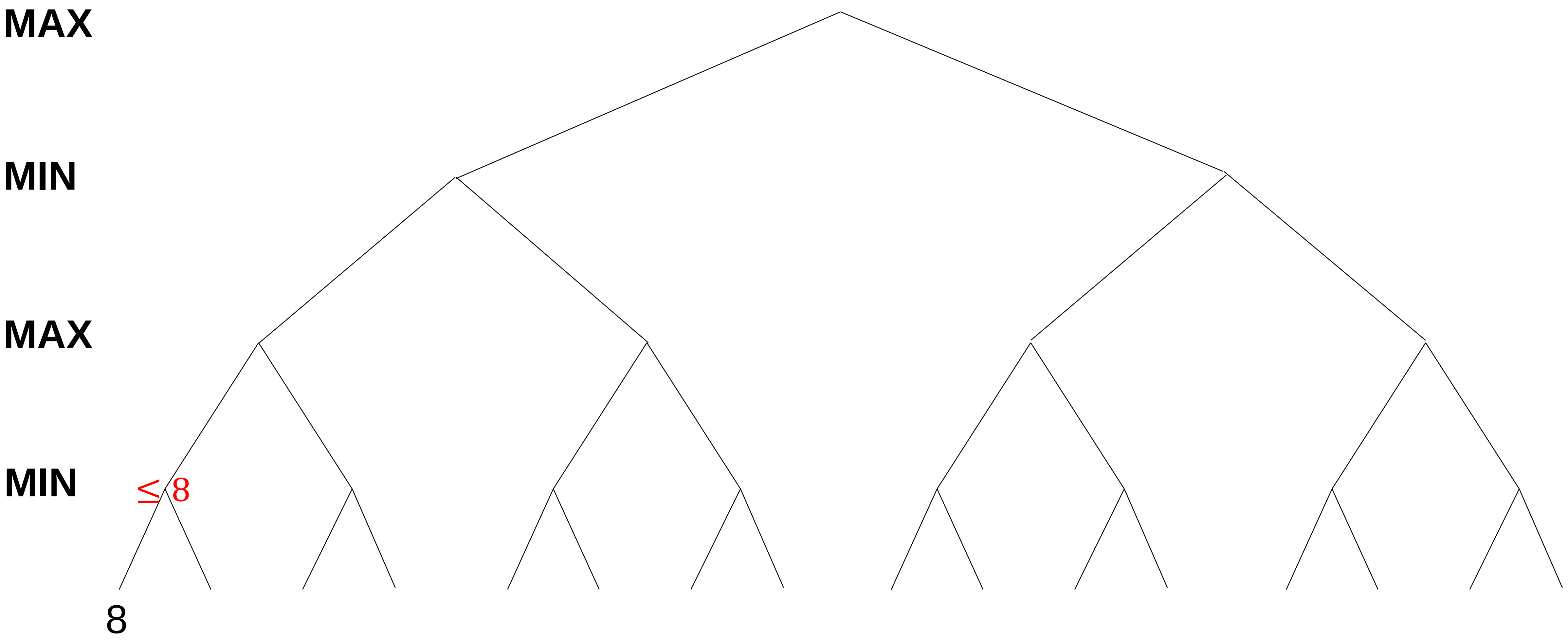
MAX

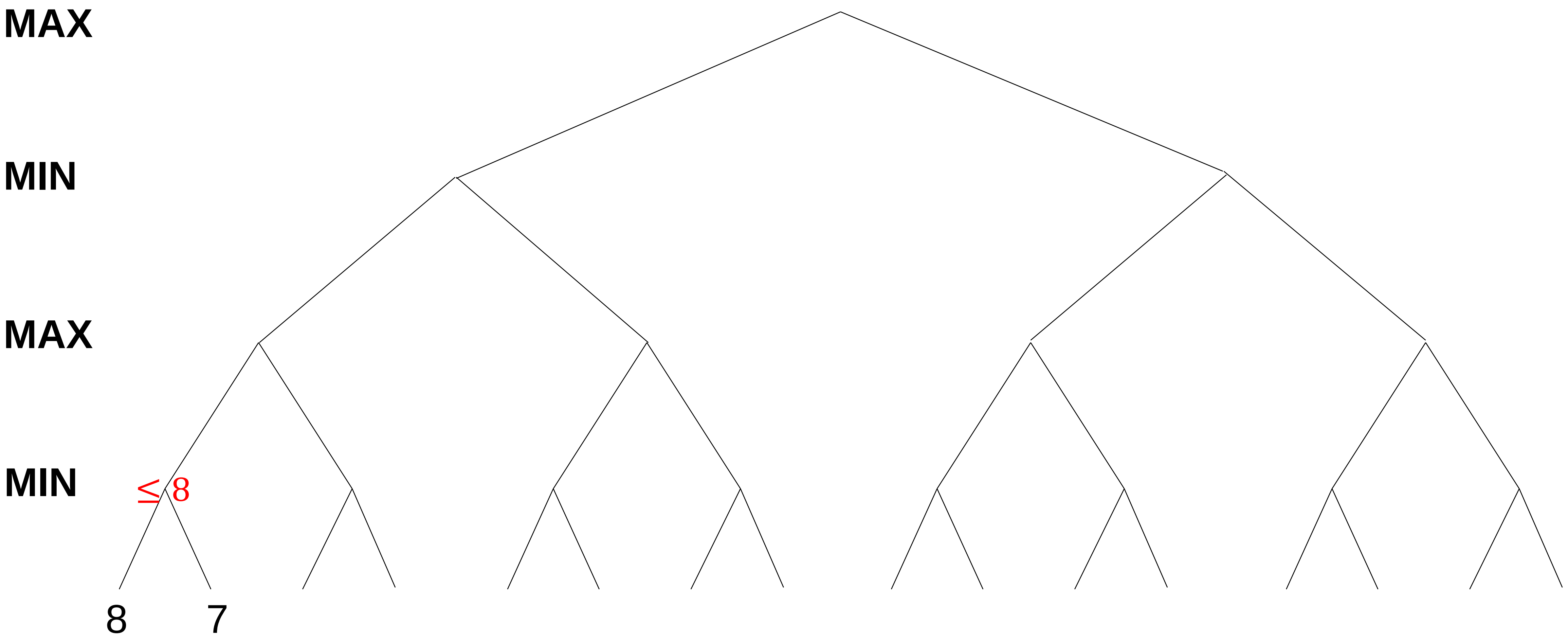
MIN

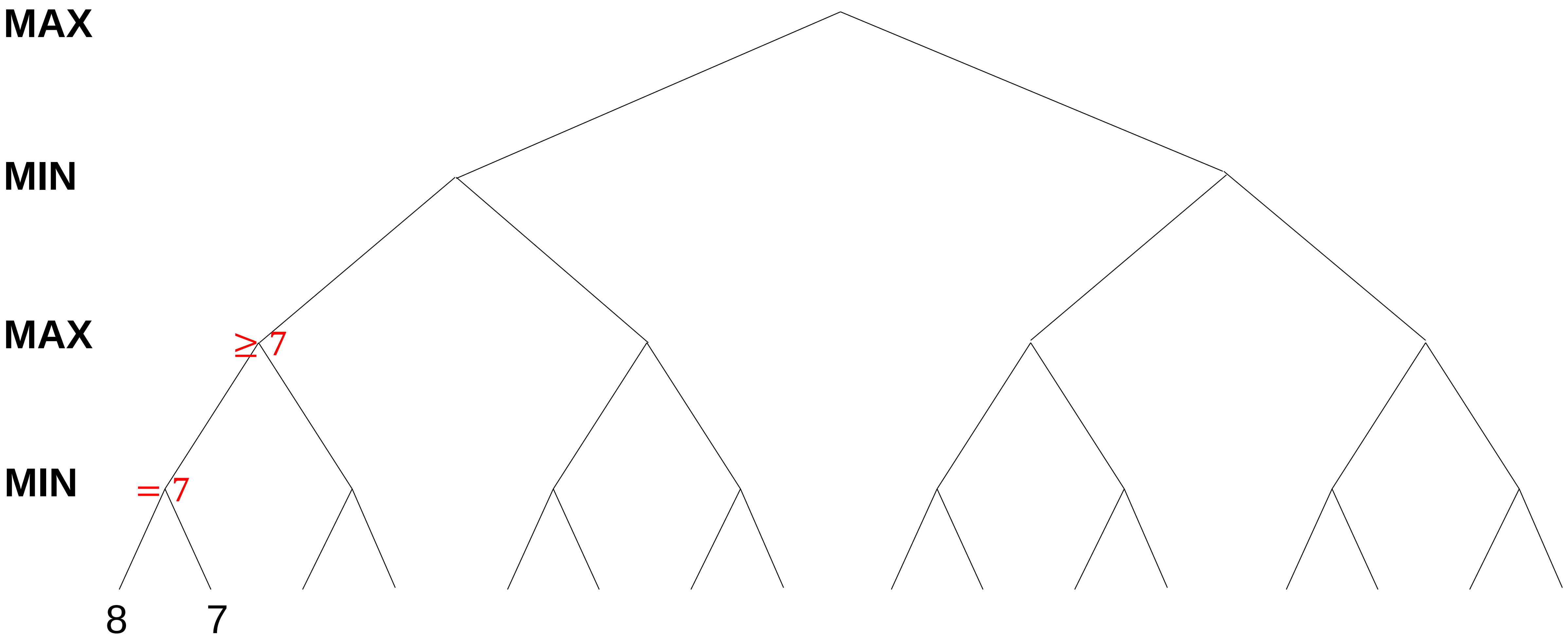
MAX

MIN

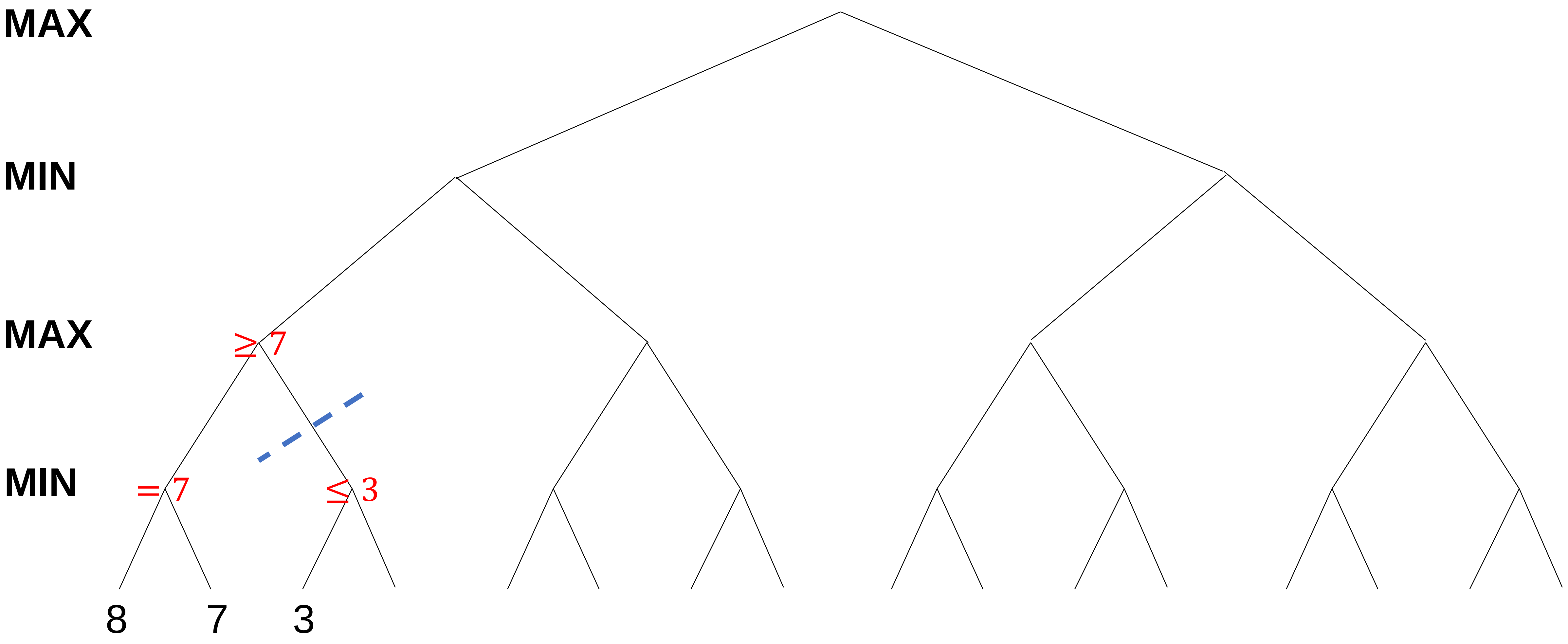


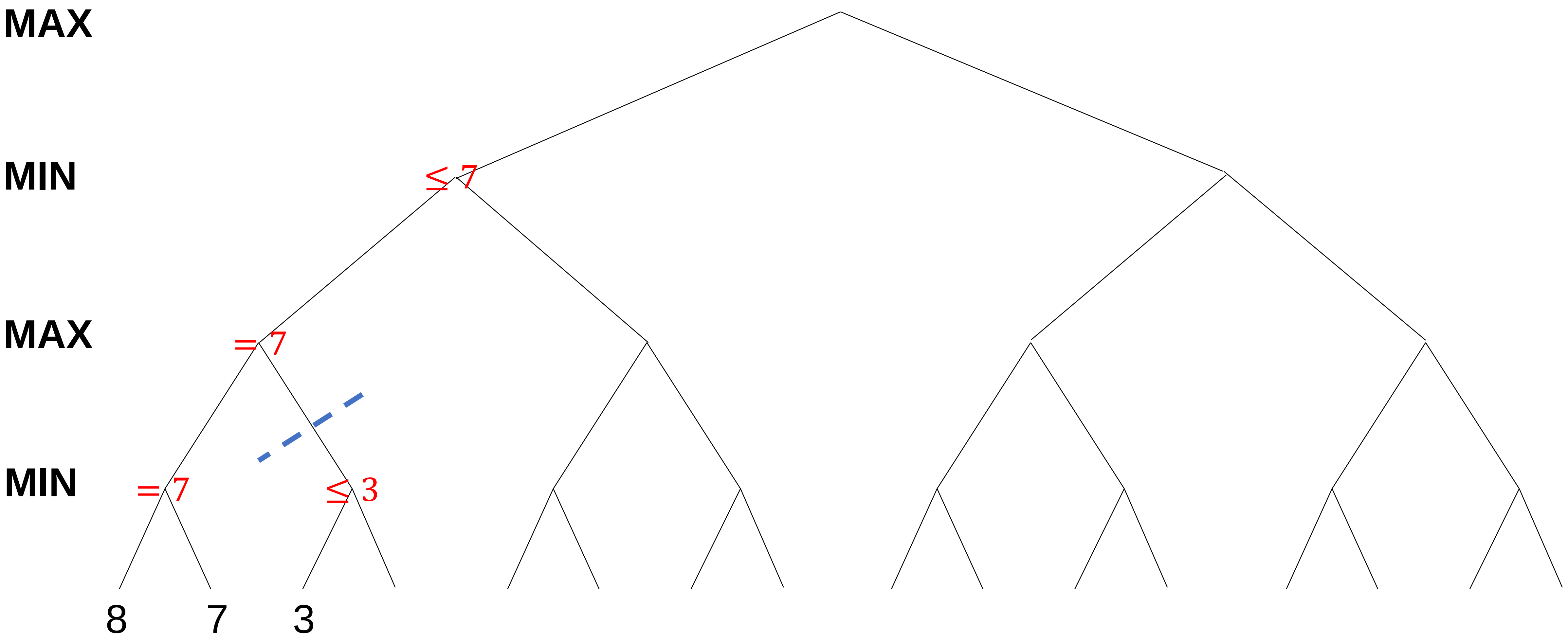

MAX
MIN
MAX
MIN
≤ 8
8

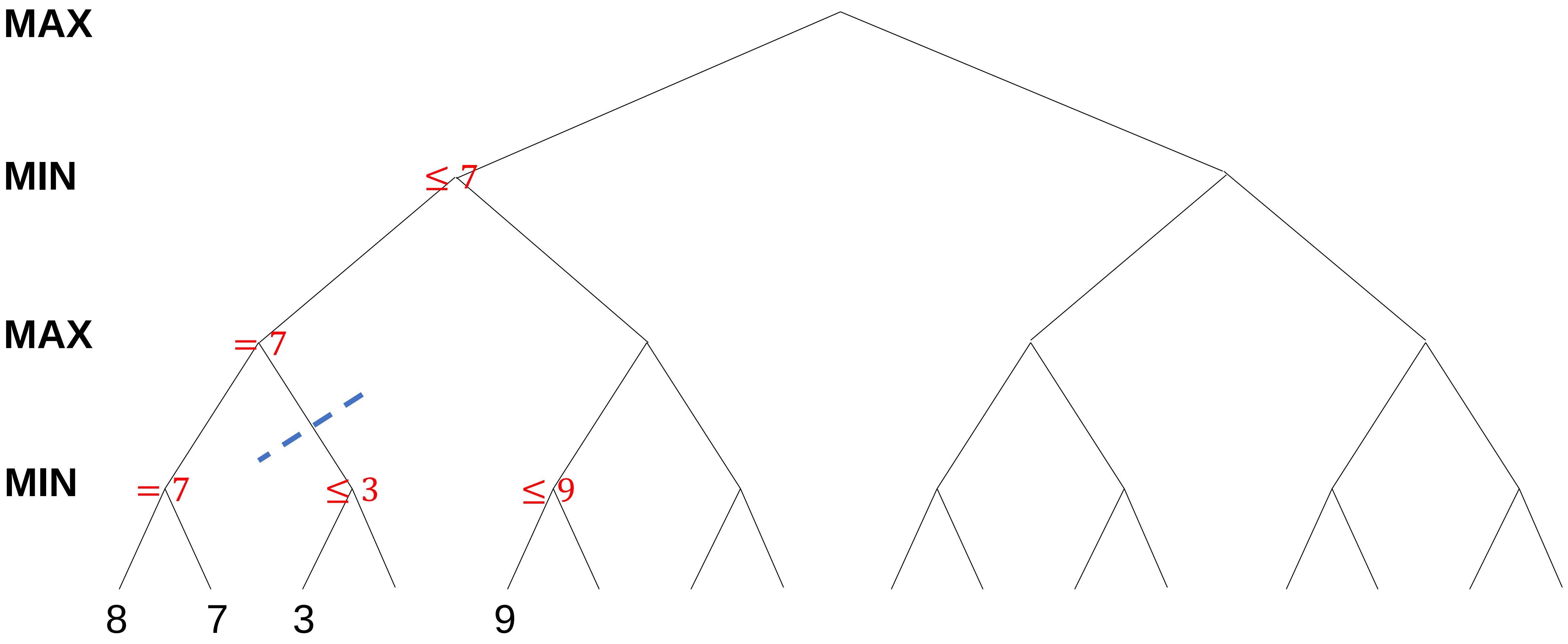

MAX
MIN
MAX
MIN
≤ 8
8
7

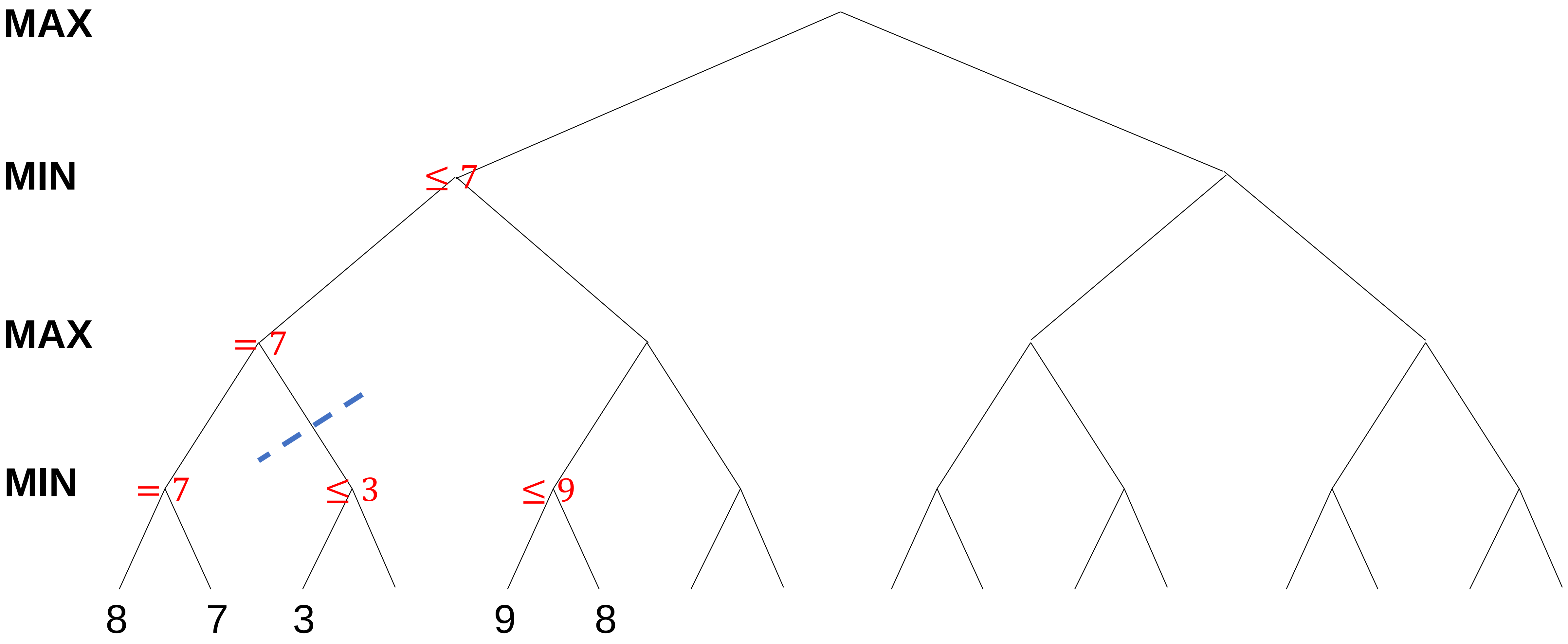


MAX
MIN
MAX
MIN
≥ 7
= 7
8
7

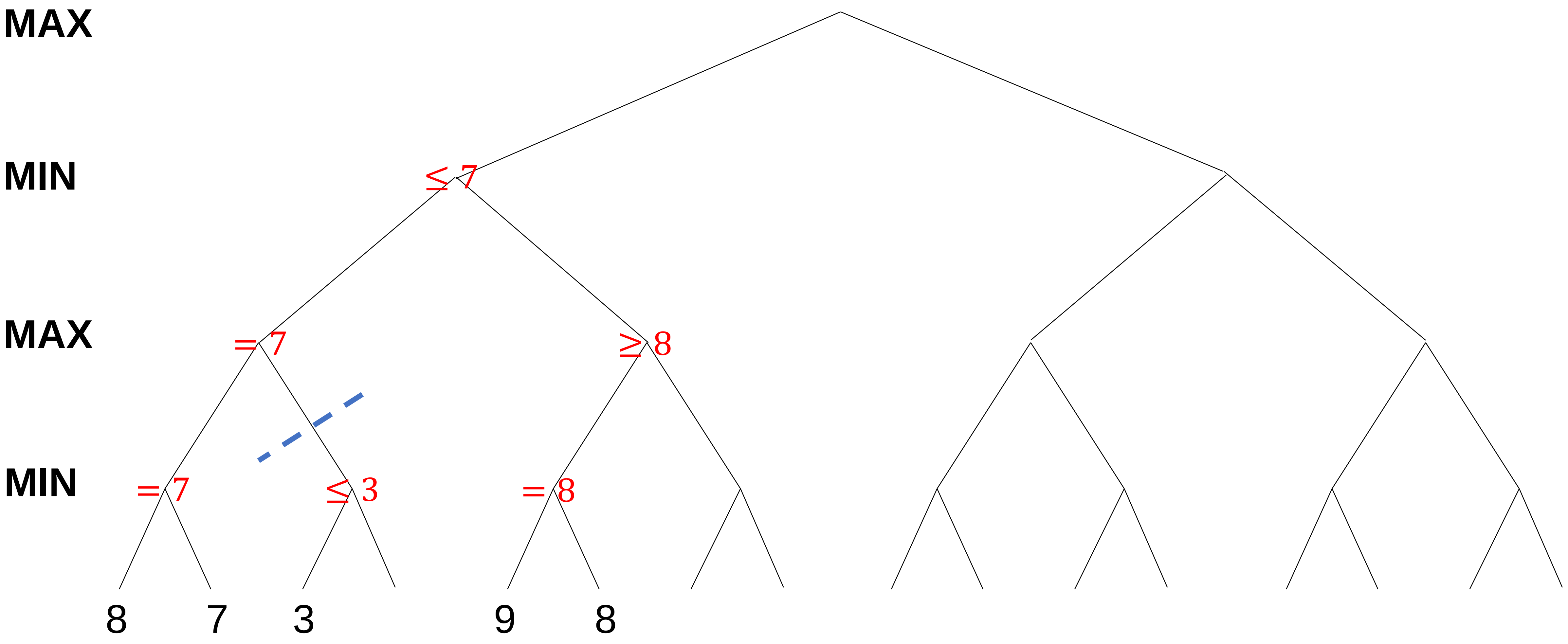

MAX
MIN
MAX
MIN
≥ 7
= 7
≤ 3
8
7
3


MAX
MIN
MAX
MIN
≤ 7
= 7
= 7
≤ 3
8
7
3

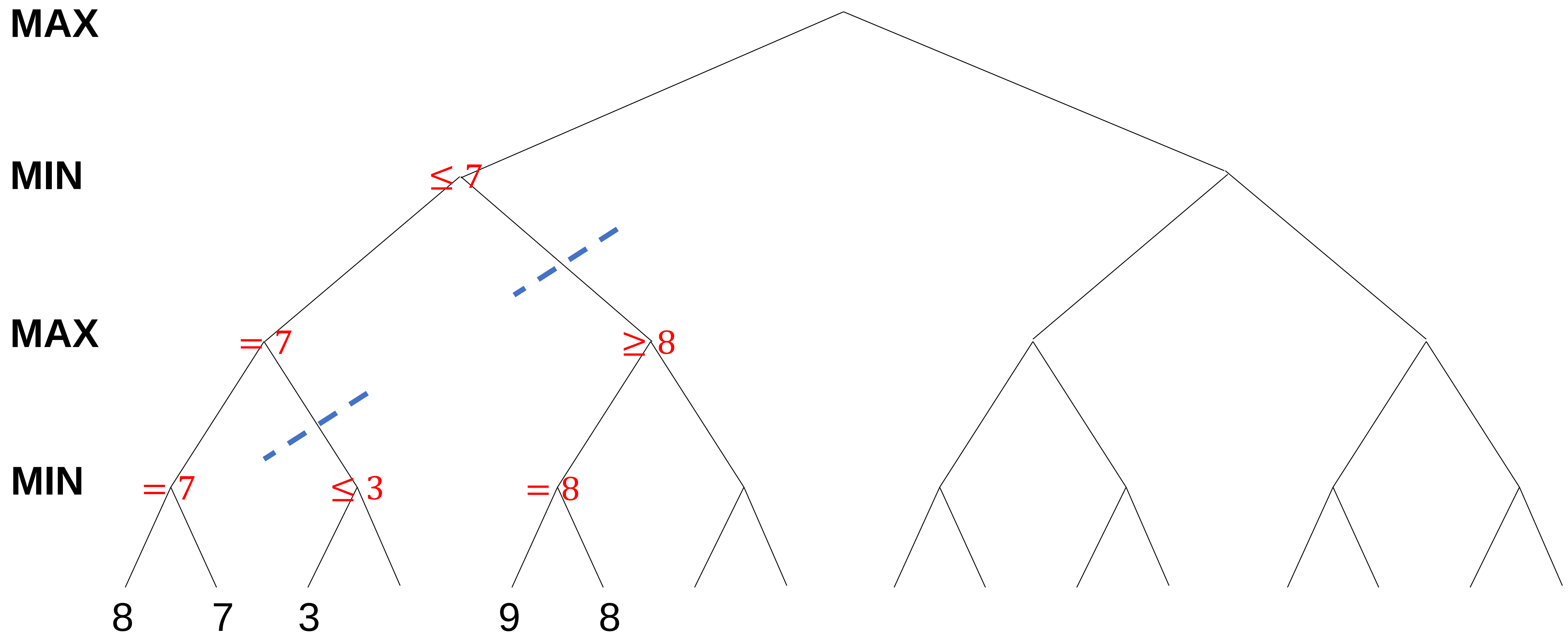

MAX
MIN
MAX
MIN
≤ 7
= 7
= 7
≤ 3
≤ 9
8
7
3
9



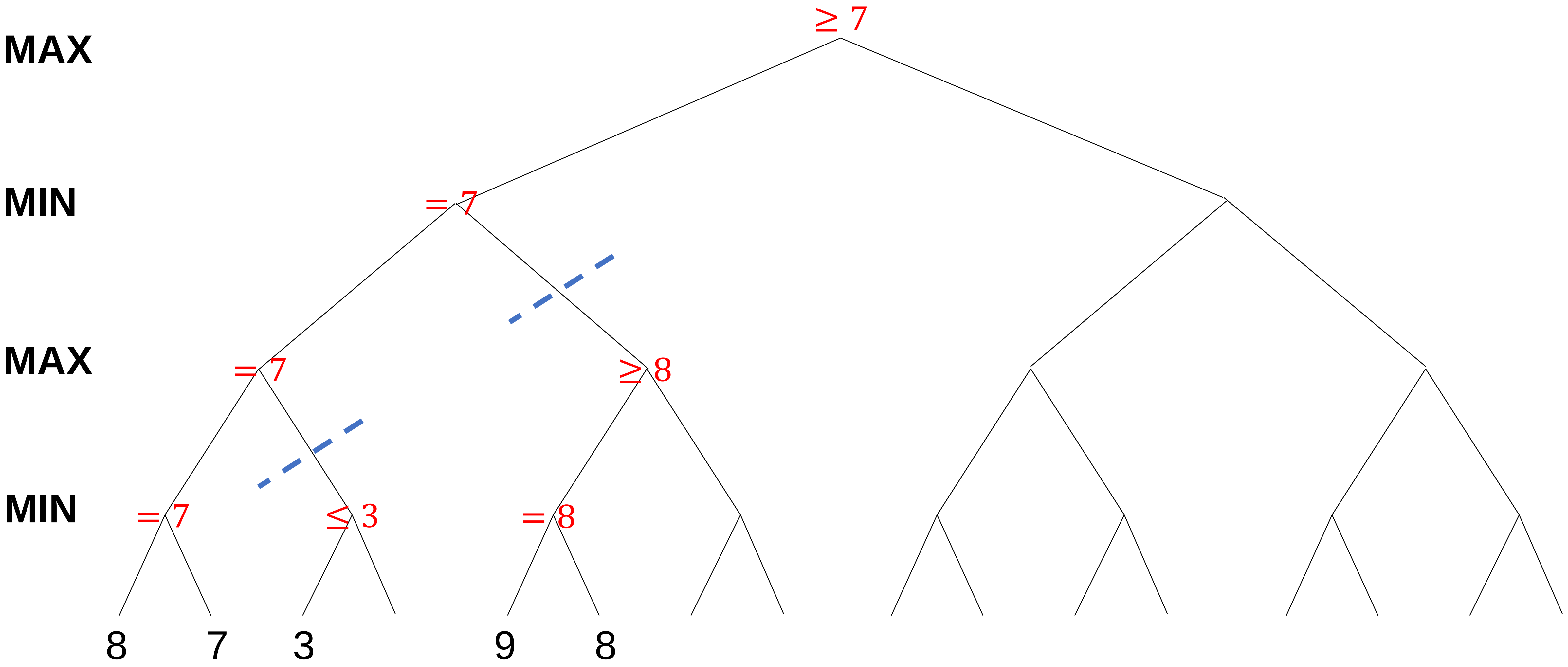
MAX
MIN
MAX
MIN
≤ 7
$= 7$
$= 7$
≤ 3
≤ 9
8
7
3
9
8

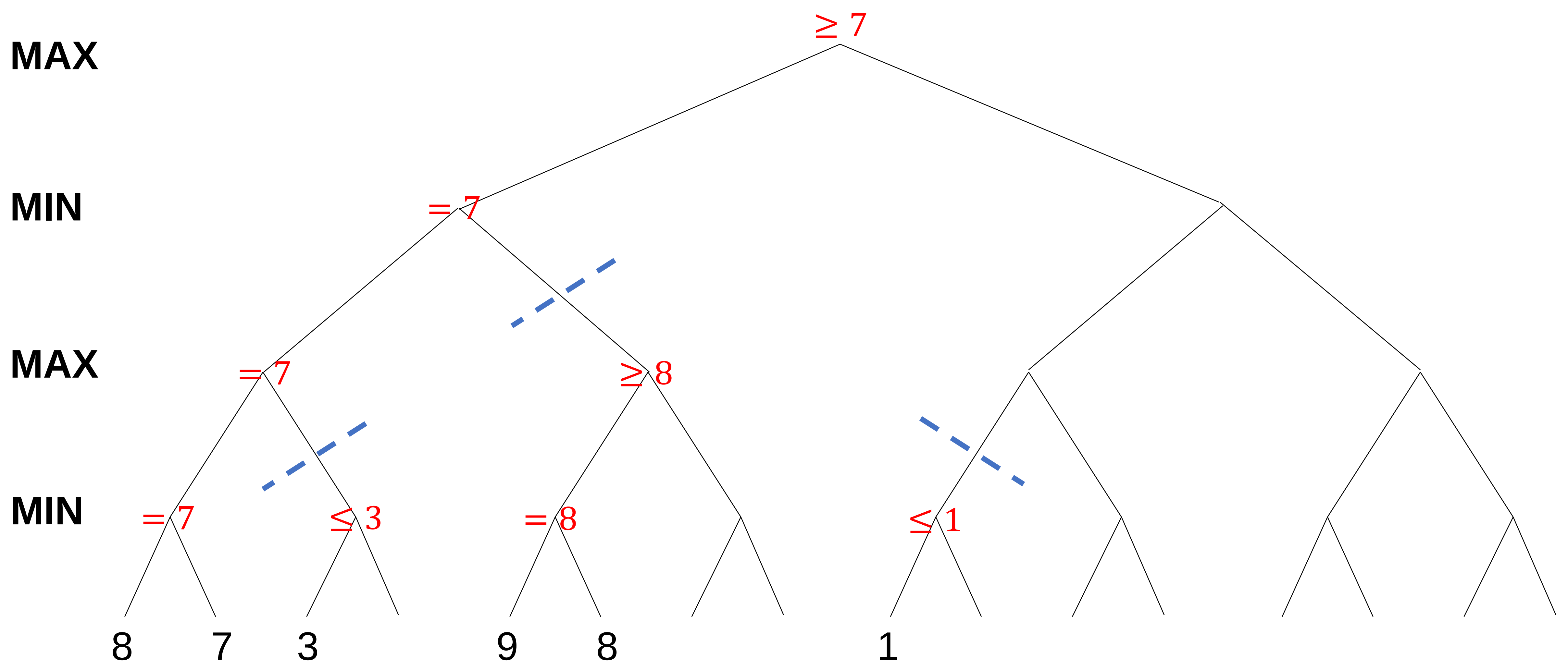


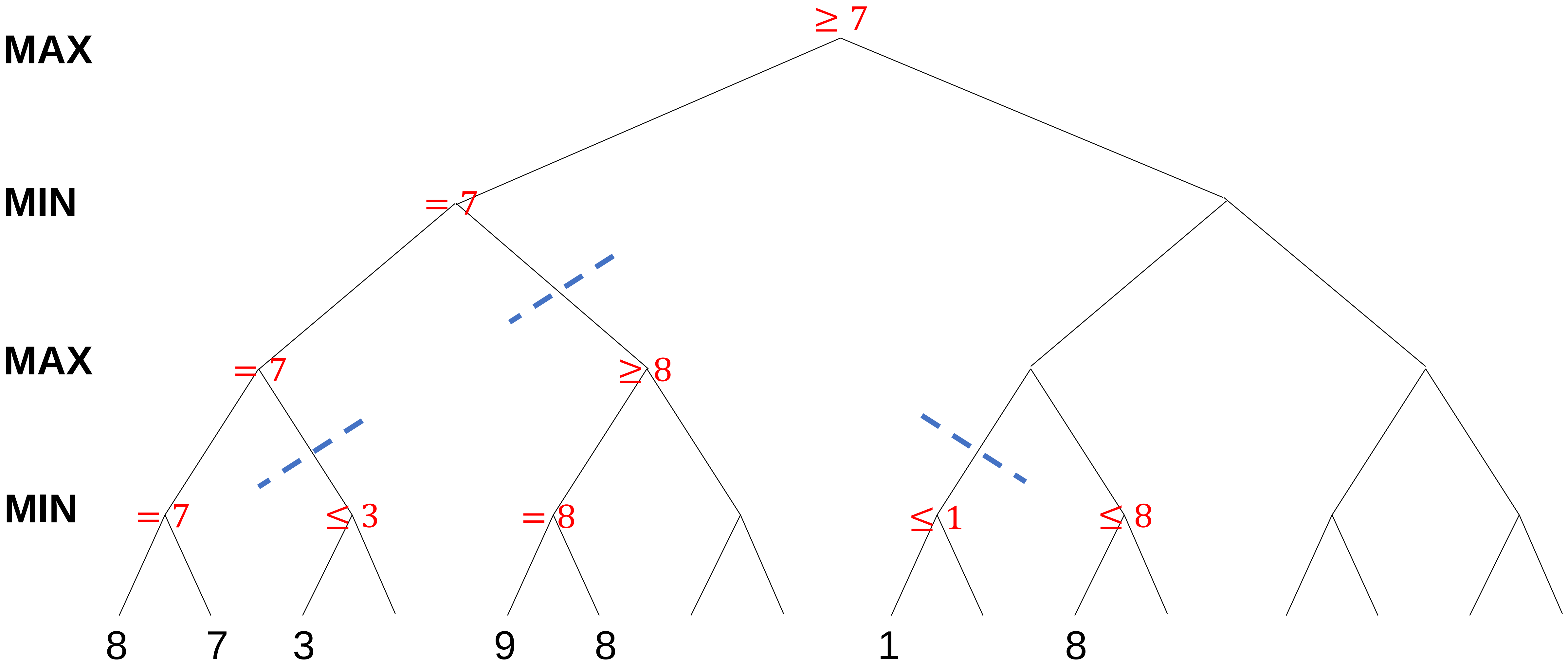
MAX
MIN
MAX
MIN
≤ 7
= 7
≥ 8
= 7
≤ 3
= 8
8
7
3
9
8

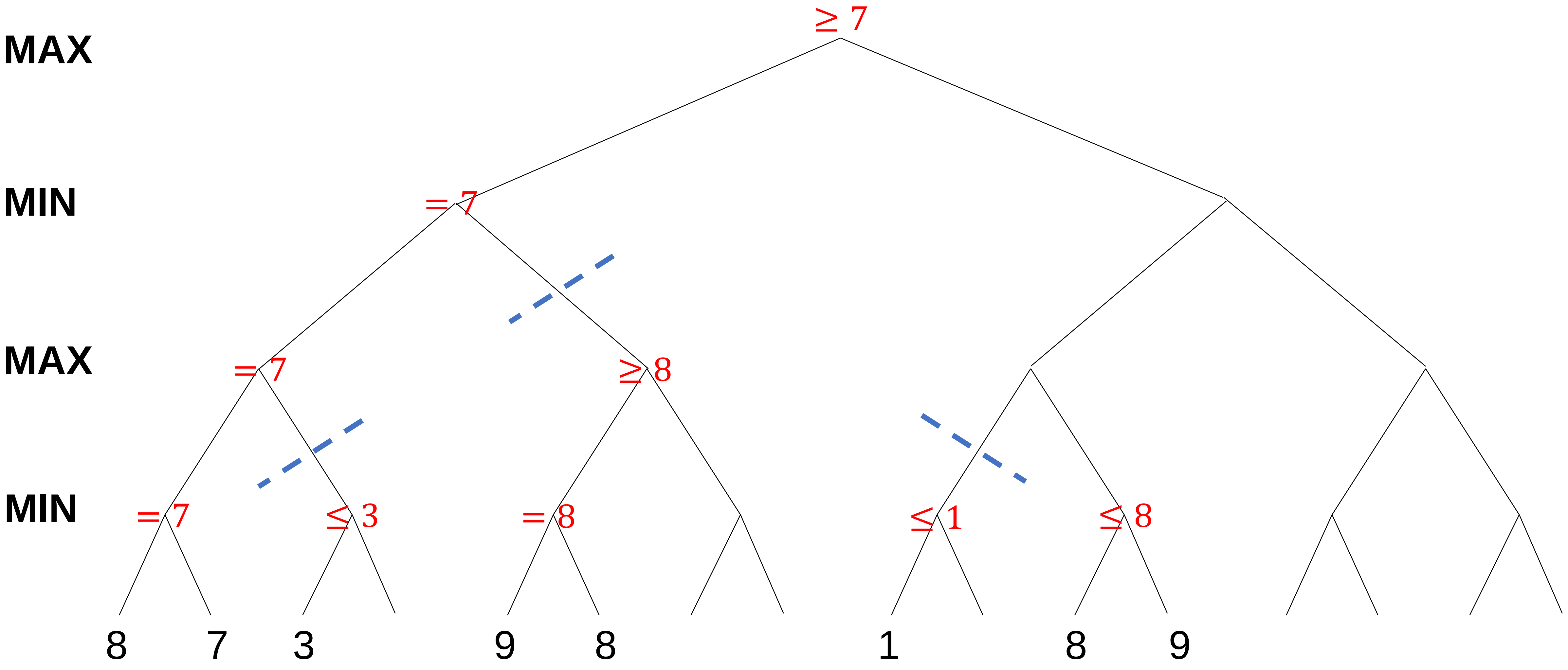


MAX
MIN
MAX
MIN
≤ 7
= 7
≥ 8
= 7
≤ 3
= 8
8
7
3
9
8

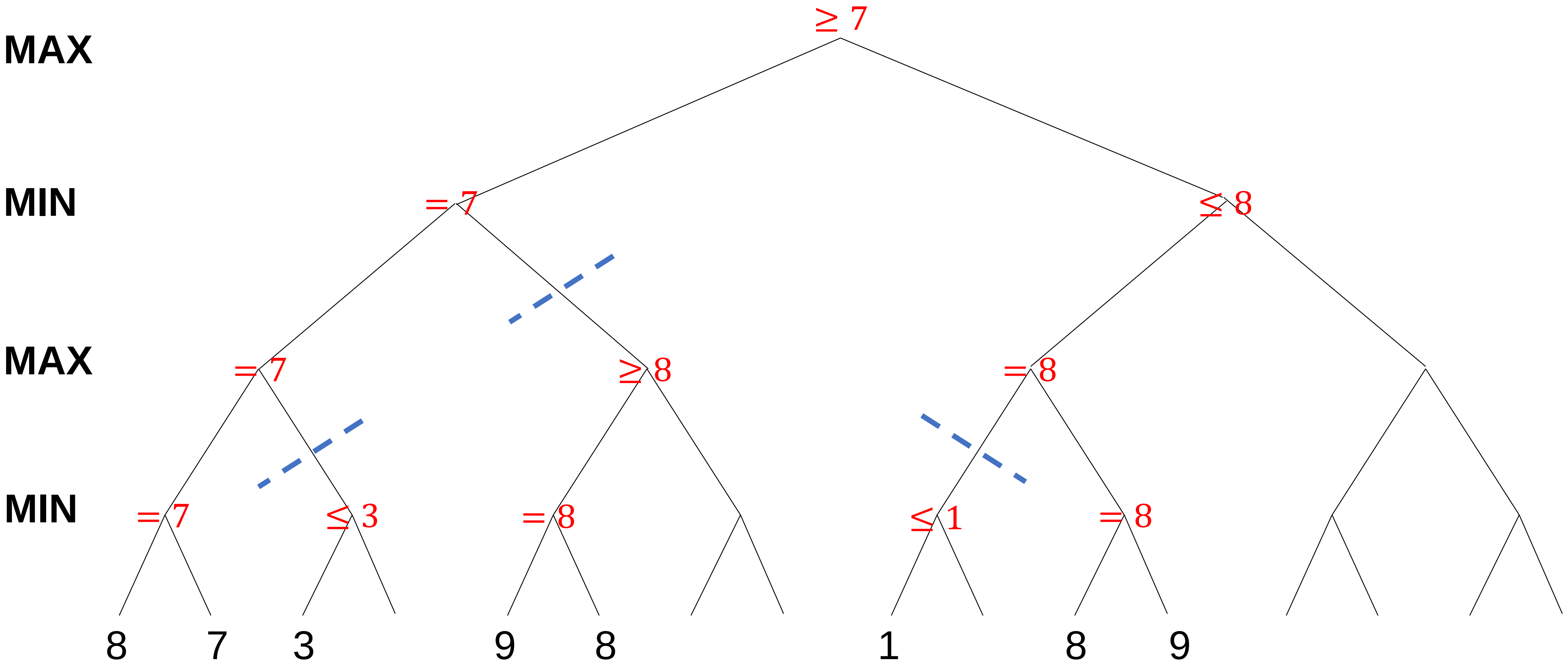

MAX
MIN
MAX
MIN
≥ 7
= 7
= 7
≥ 8
= 7
≤ 3
= 8
8
7
3
9
8


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

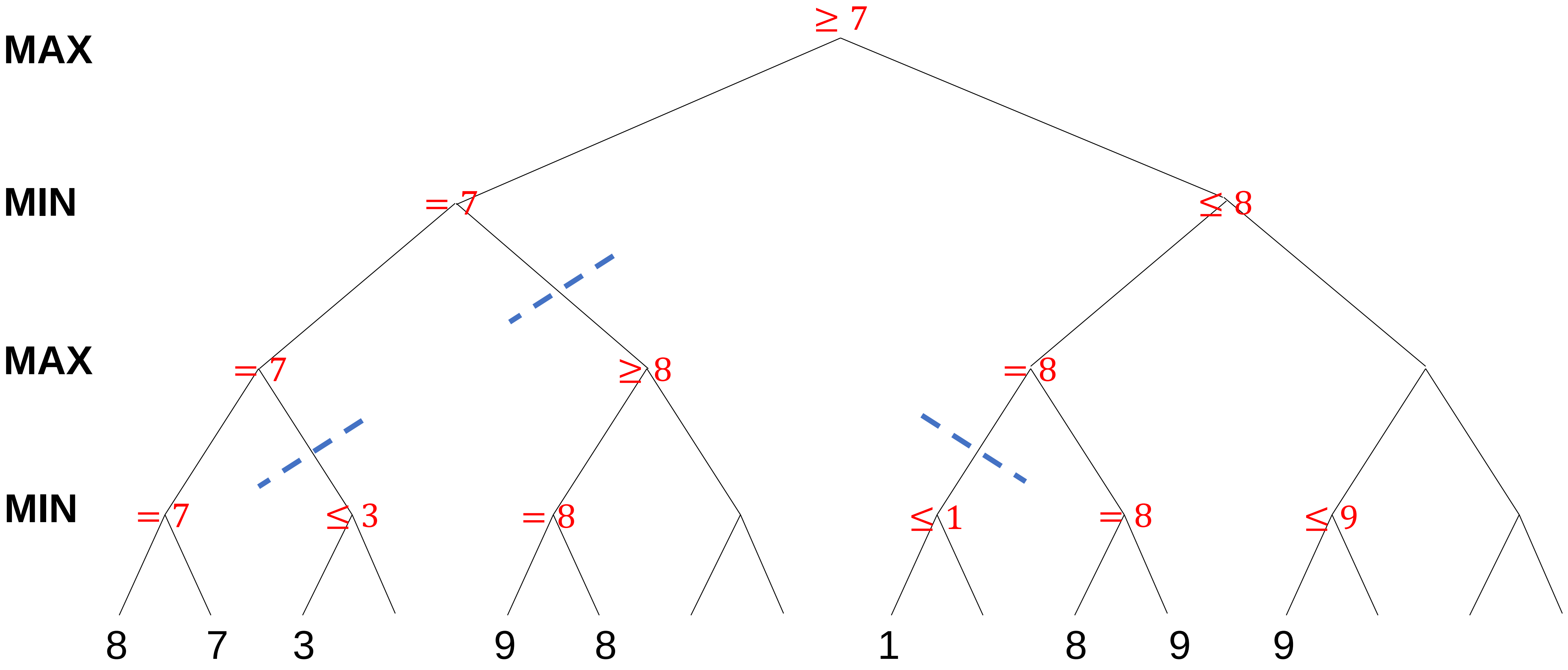

MAX
MIN
MAX
MIN
≥ 7
= 7
= 7
≥ 8
= 7
≤ 3
= 8
≤ 1
≤ 8
8
7
3
9
8
1
8



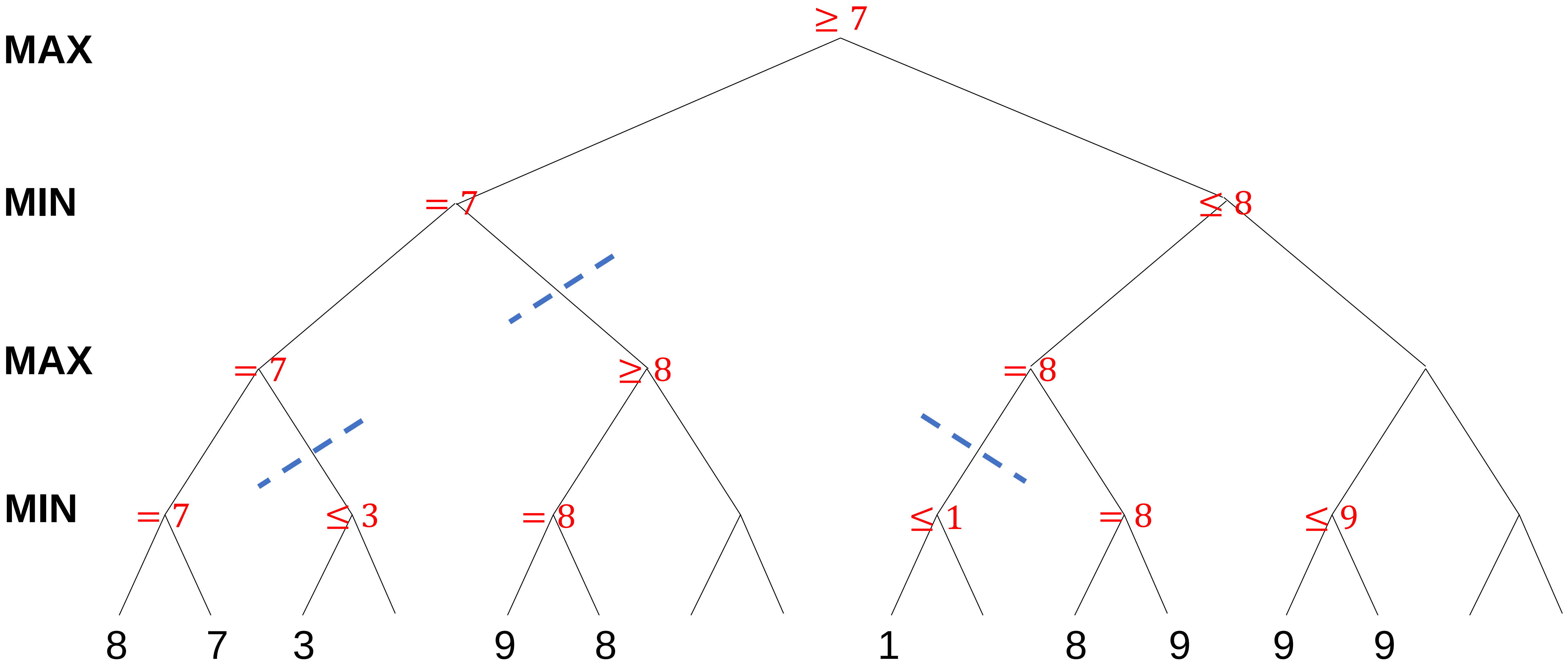
MAX
MIN
MAX
MIN
≥ 7
= 7
= 7
≥ 8
= 7
≤ 3
= 8
≤ 1
≤ 8
8
7
3
9
8
1
8
9



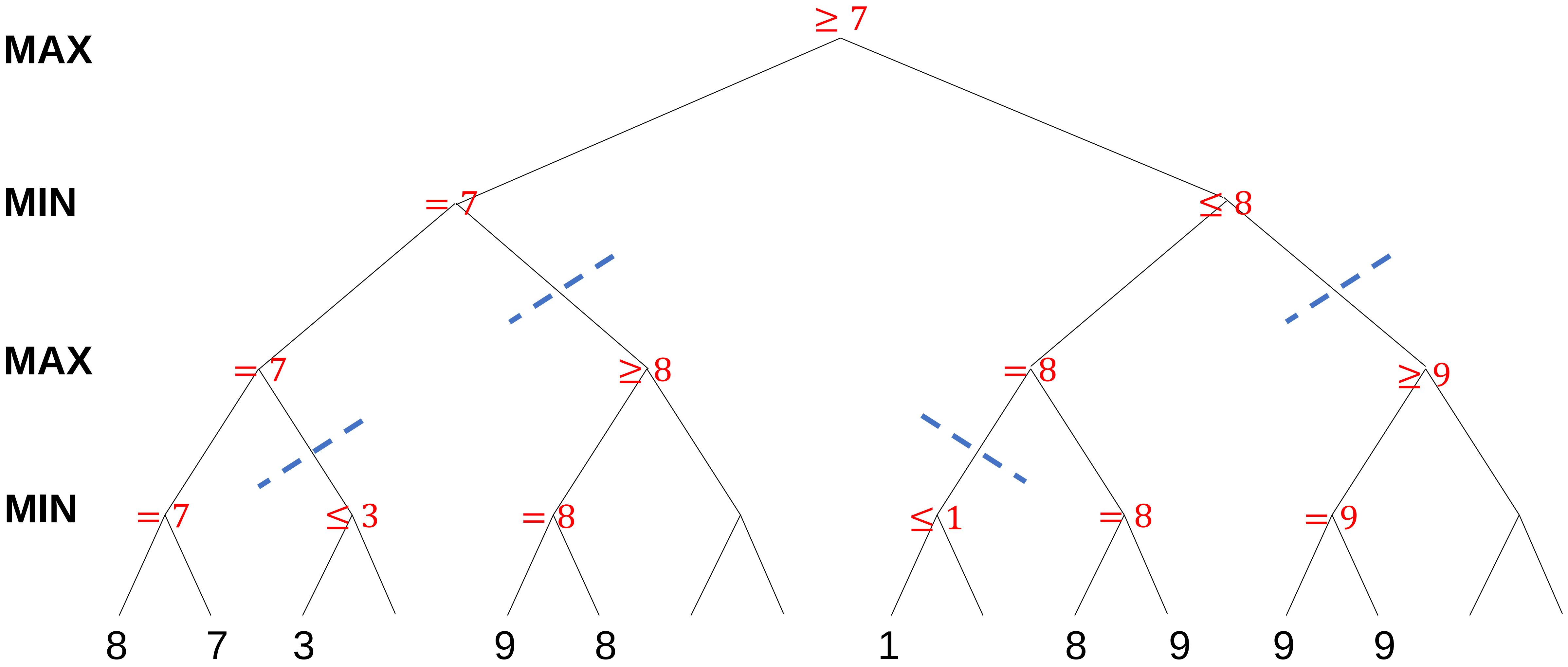
MAX
MIN
MAX
MIN
≥ 7
= 7
≤ 8
= 7
≥ 8
= 8
= 7
≤ 3
= 8
≤ 1
= 8
8
7
3
9
8
1
8
9

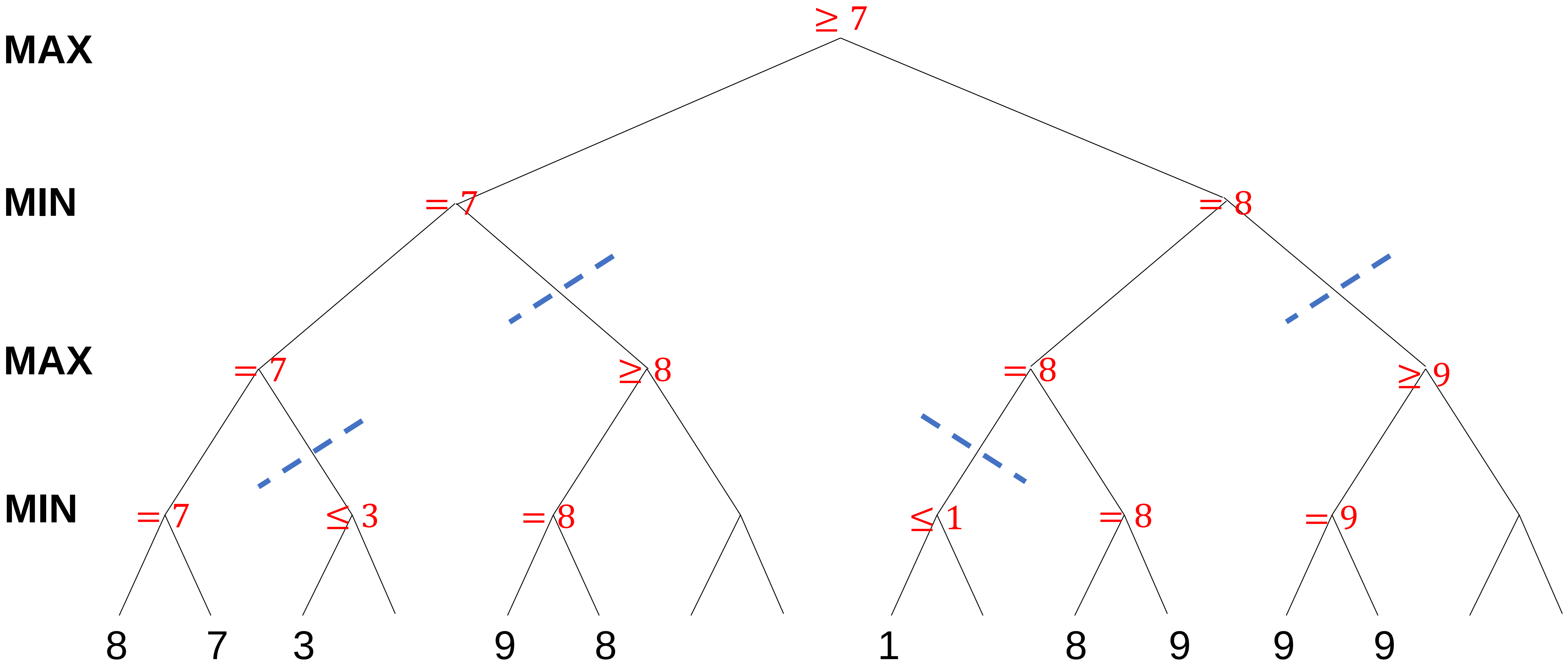


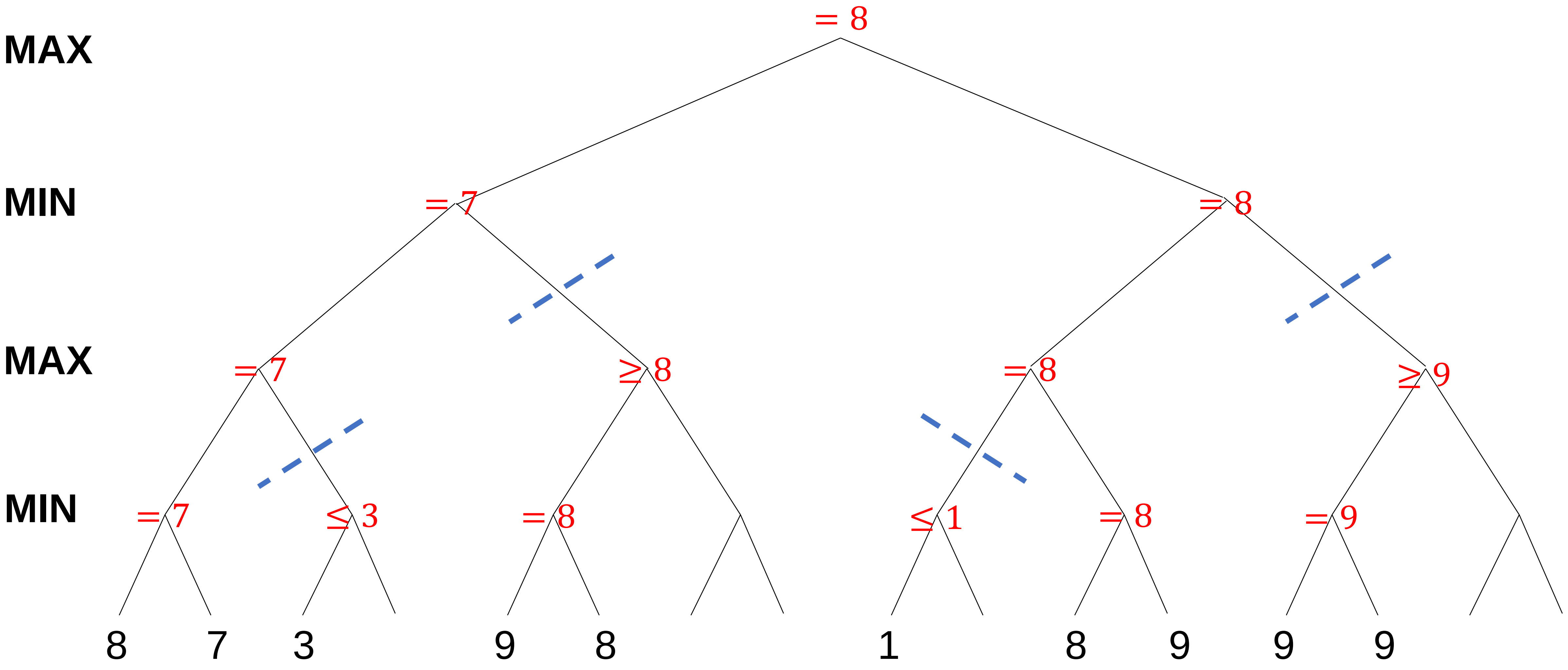
MAX
MIN
MAX
MIN
≥ 7
= 7
≤ 8
= 7
≥ 8
= 8
= 7
≤ 3
= 8
≤ 1
= 8
≤ 9
8
7
3
9
8
1
8
9
9



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

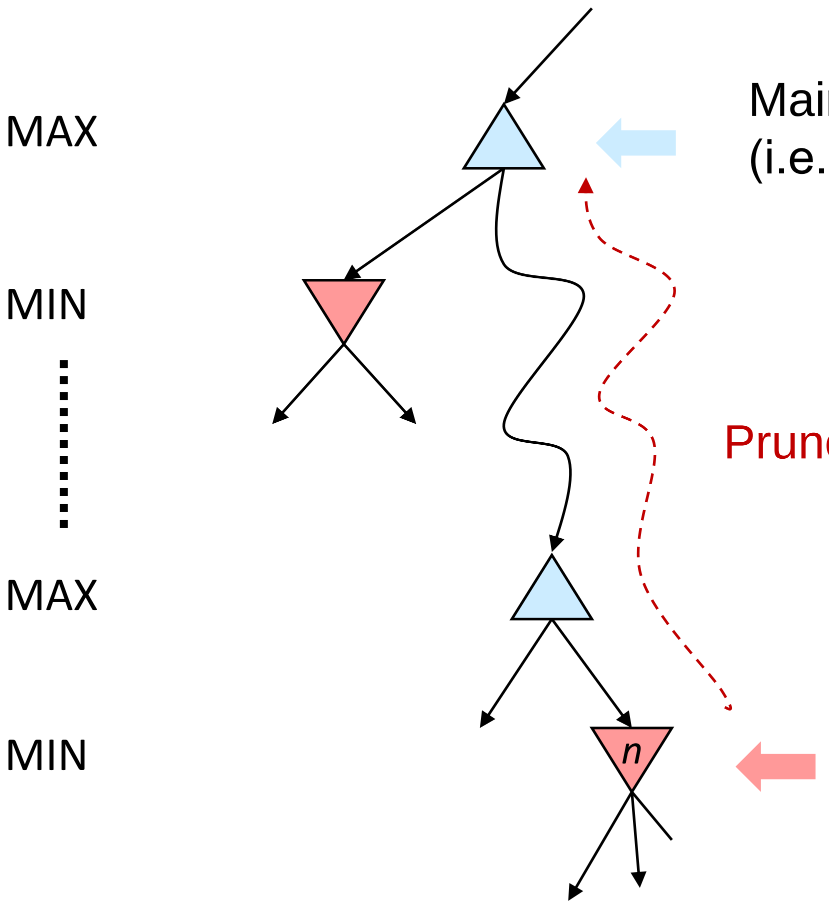

MAX
MIN
MAX
MIN
≥ 7
= 7
≤ 8
= 7
≥ 8
= 8
≥ 9
= 7
≤ 3
= 8
≤ 1
= 8
= 9
8
7
3
9
8
1
8
9
9
9


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

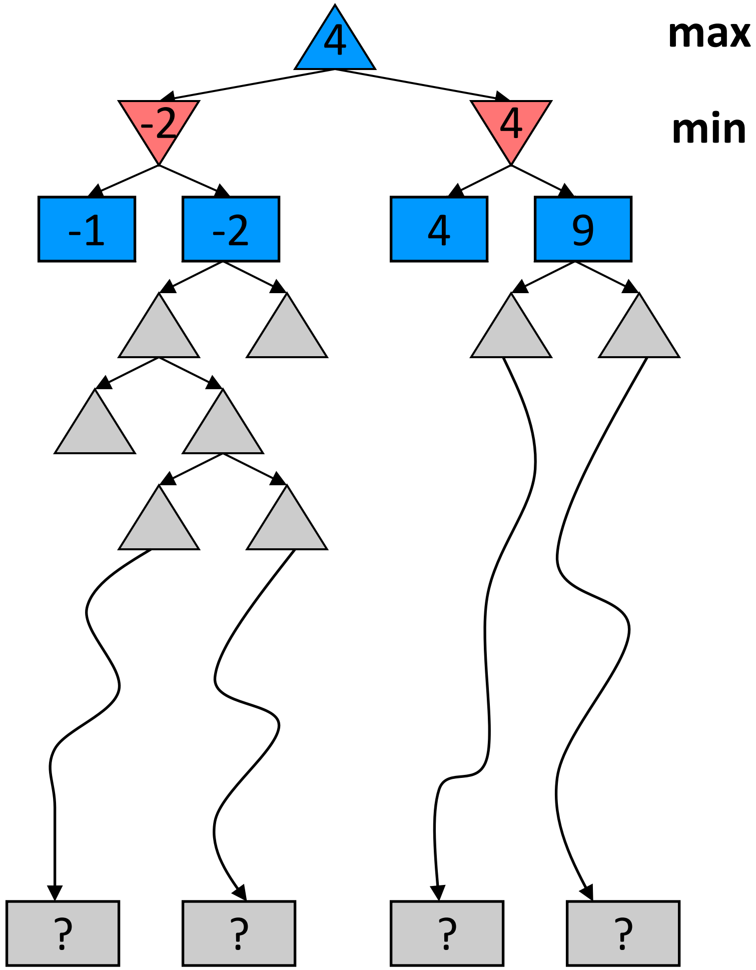
- The pruning has **no effect** on the minimax value computed for the root.

- 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

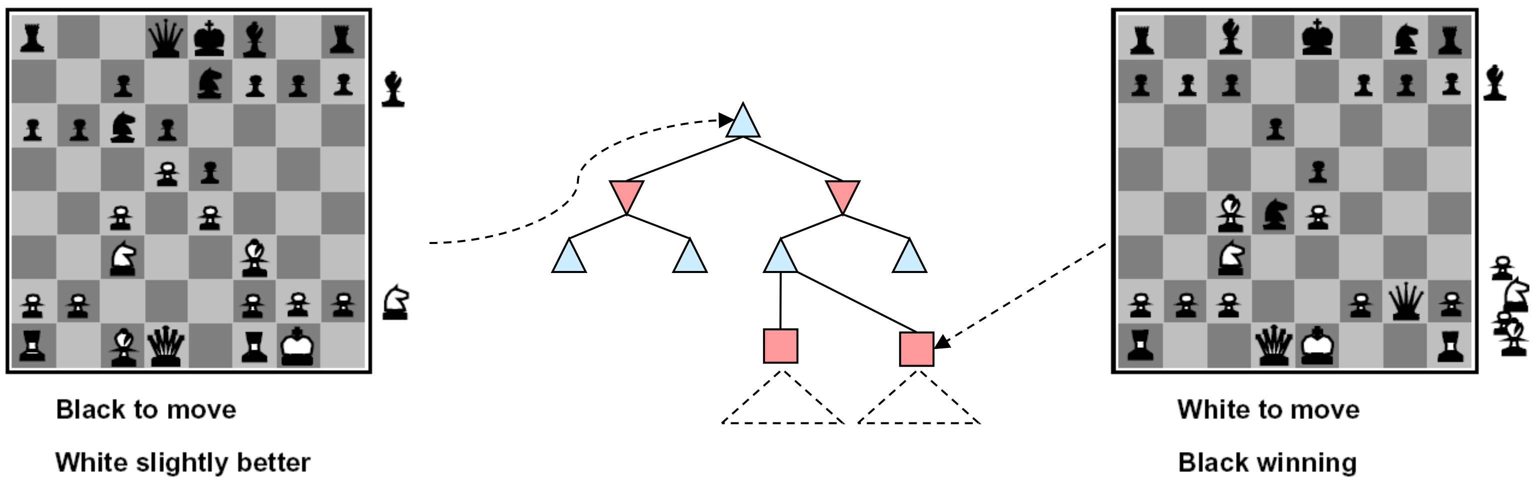
Resource Limits

- 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

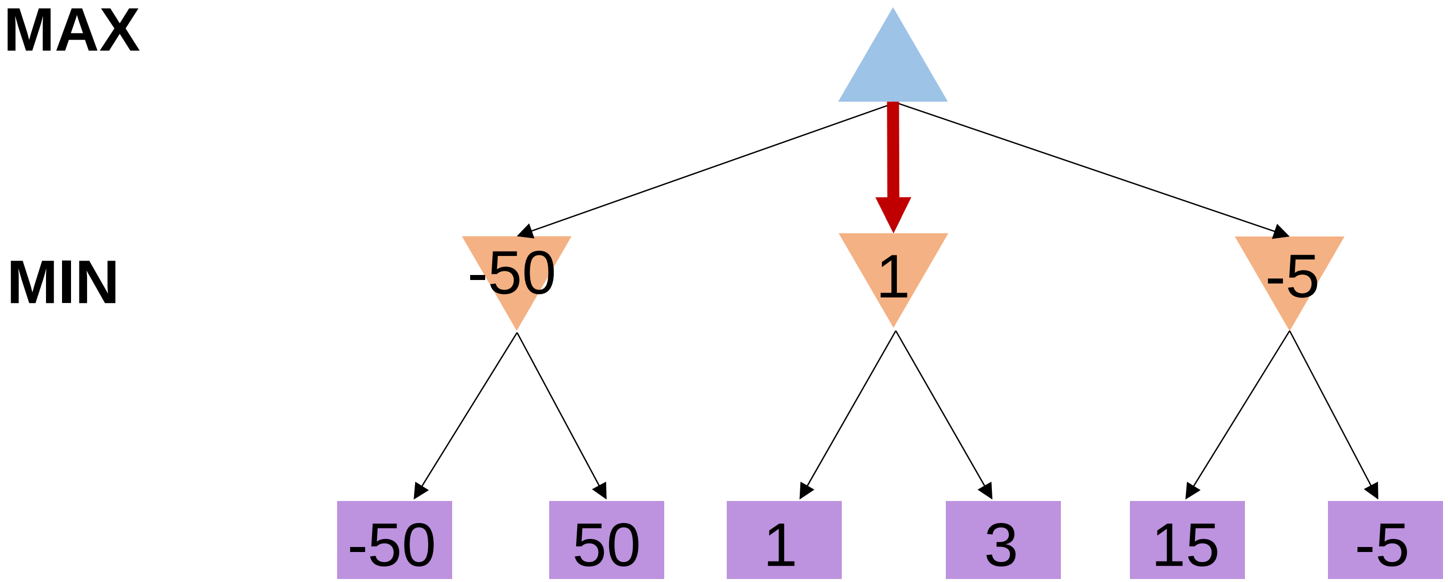


- E.g., weighted linear sum of features: $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$
  where $f_1(s)$ = (num white queens – num black queens), etc.

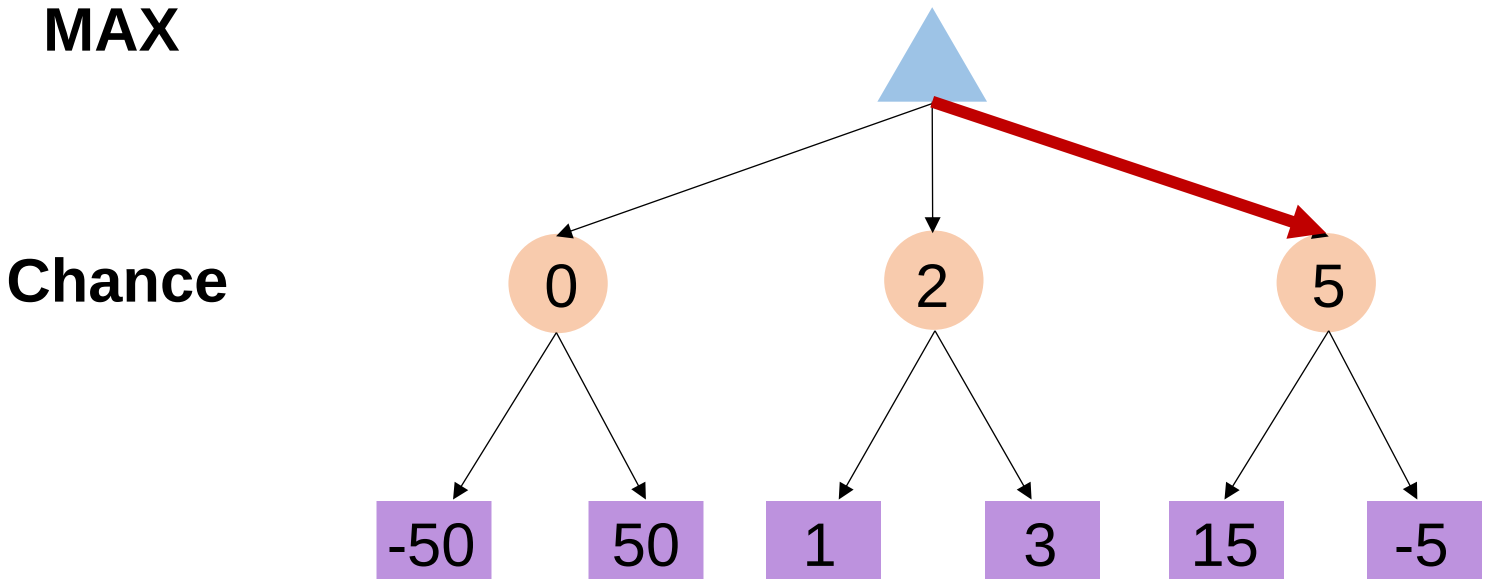
- Evaluation function can provide guidance to expand most promising nodes first (allowing alpha-beta pruning to prune more)

Expectimax

Two-Player Turn-Based Game



Two-Player Turn-Based Game



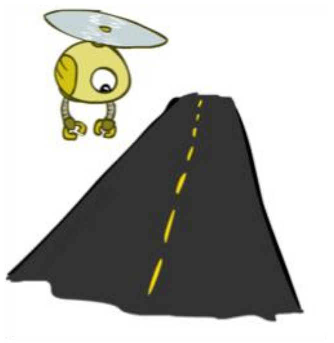
Expectimax Search

- When do we have randomness?
  - Explicit randomness: rolling dice
  - Unpredictable opponents: the ghosts respond randomly
  - Actions can fail: when moving a robot, wheels might slip

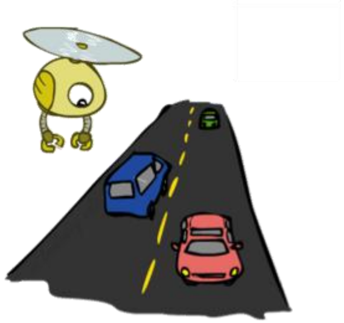
- Values now reflect average-case (**expectimax**) outcomes, not worst-case (**minimax**) outcomes.

Reminder: Probabilities

- Example: Traffic on freeway
  - Random variable: T = whether there's traffic
  - Outcomes: T in {none, light, heavy}
  - Distribution: P(T=none) = 0.25, P(T=light) = 0.50, P(T=heavy) = 0.25



0.25



0.50

0.25

Time:	20 min		30 min		60 min		
	x	+	x	+	x	⇒	35 min
Probability:	0.25		0.50		0.25		

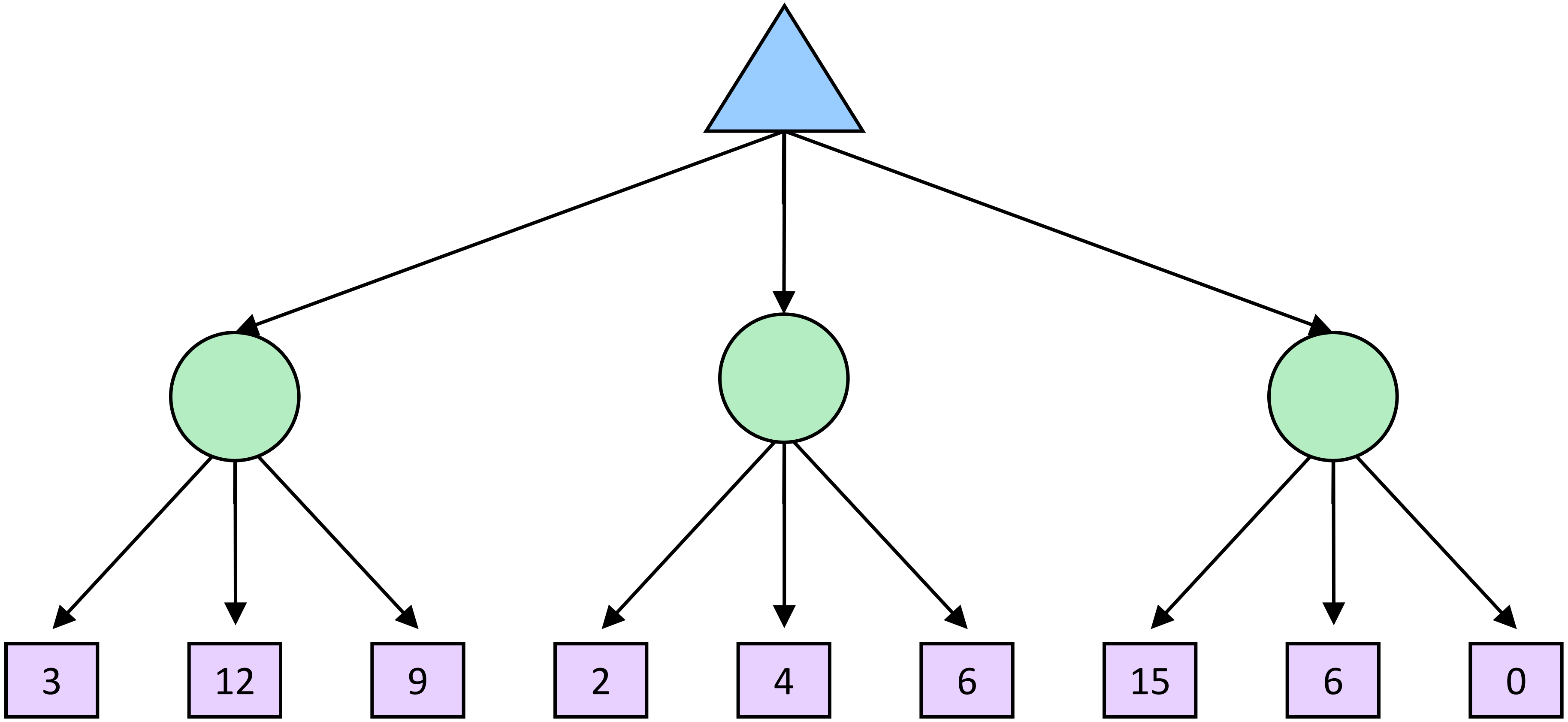
Expectimax

```
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 EXP: return exp-value(state)
```

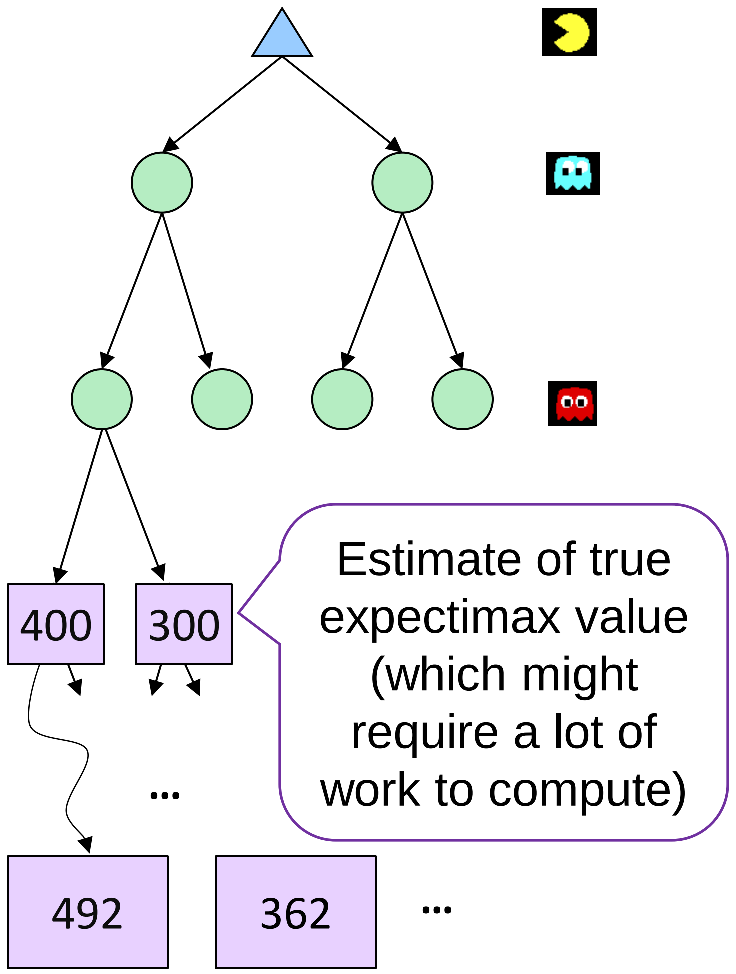
```
def max-value(state):
    initialize v = -∞
    for each successor of state:
        v = max(v, value(successor))
    return v
```

```
def exp-value(state):
    initialize v = 0
    for each successor of state:
        p = probability(successor)
        v += p * value(successor)
    return v
```

Expectimax



Depth-Limited Expectimax



What Probabilities to Use?

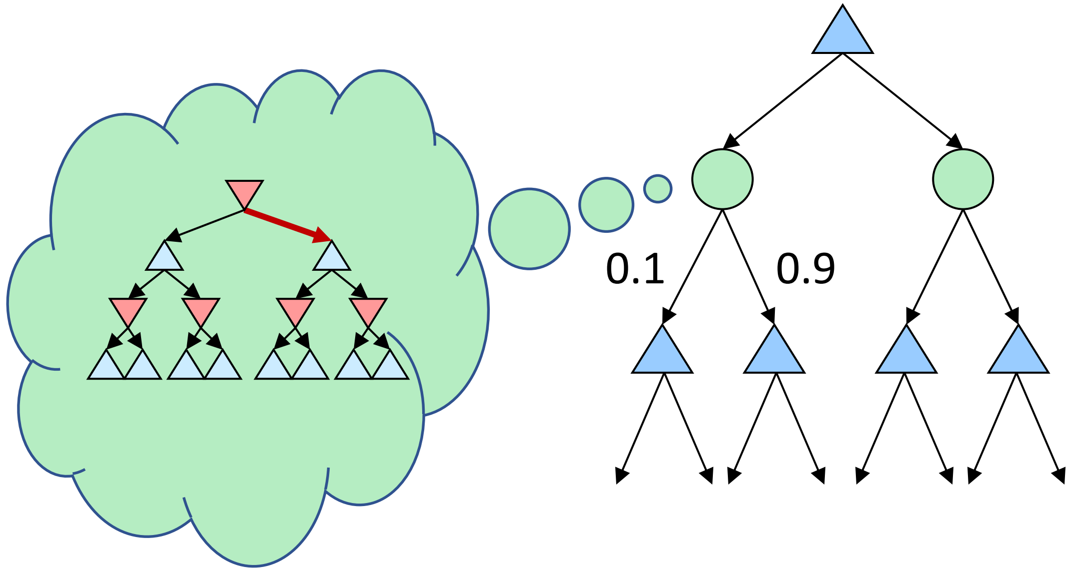
- In expectimax search, we have a **probabilistic model** of how the opponent (or environment) will behave.
  - Model could be a simple uniform distribution (roll a die)
  - Model could be sophisticated and require a great deal of computation
  - We have a chance node for any outcome out of our control: opponent or environment

- For now, assume each chance node magically comes along with probabilities that specify the distribution over its outcomes

- You'll get more ideas about how to produce such probabilistic models later in the semester when we talk about "learning from data"

Quiz: Informed Probabilities

- Suppose you know that your opponent is running a depth-2 minimax, using the result 80% of the time, and moving randomly otherwise
- Question: What tree search should you use?



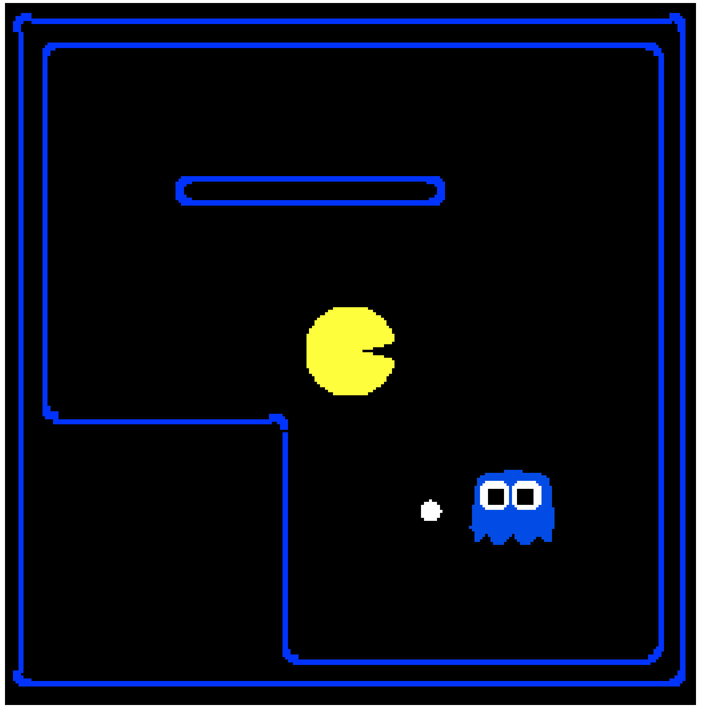
- **Answer: Expectimax!**
  - To figure out EACH chance node's probabilities, you have to run a simulation of your opponent
  - This kind of thing gets very slow very quickly
  - Minimax search, on the other hand, has the nice property that it all collapses into one game tree

The Dangers of Optimism and Pessimism

Dangerous Optimism
Assuming chance when the world is adversarial

Dangerous Pessimism
Assuming the worst case when it's not likely

Assumptions vs. Reality

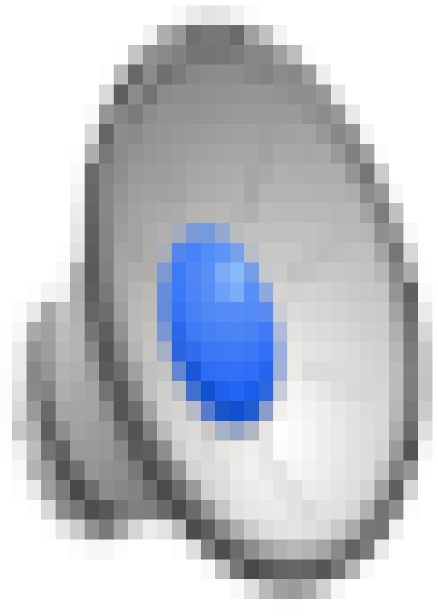


	Adversarial Ghost	Random Ghost
Minimax Pacman	Won 5/5 Avg. Score: 483	Won 5/5 Avg. Score: 493
Expectimax Pacman	Won 1/5 Avg. Score: -303	Won 5/5 Avg. Score: 503

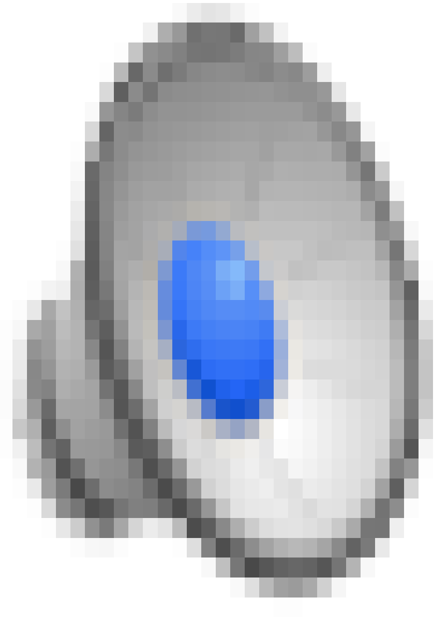
Results from playing 5 games

Pacman used depth 4 search with an eval function that avoids trouble
Ghost used depth 2 search with an eval function that seeks Pacman

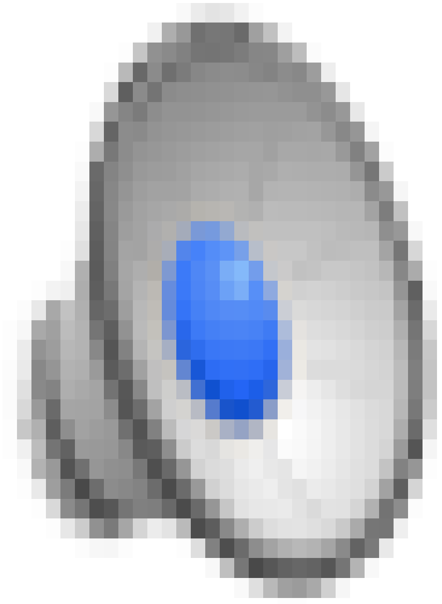
Random Ghost – Expectimax Pacman



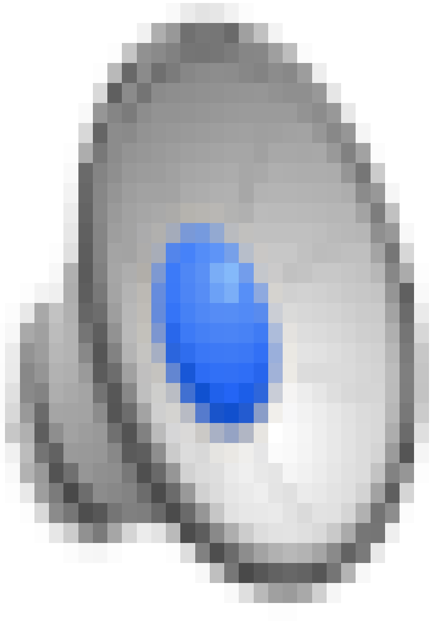
Adversarial Ghost – Minimax Pacman



Adversarial Ghost – Expectimax Pacman



Random Ghost – Minimax Pacman



Monte-Carlo Tree Search

Issues of the Search Methods We Introduced So Far

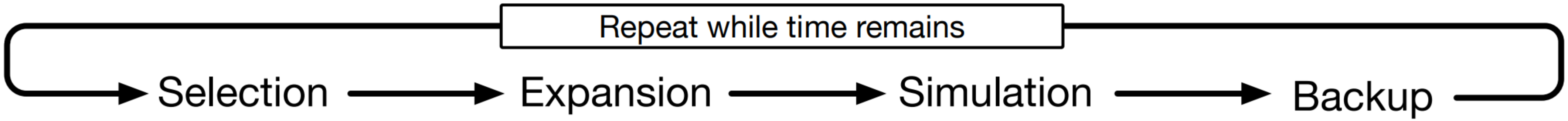
- When branching factor (i.e., number of possible actions) is large, the search cannot go deep
  - In Go, the branching factor could be > 300
  - α-β search would be limited to 4 or 5 layers
- Sometimes it's difficult to define a good evaluation function

Monte-Carlo Tree Search (MCTS)

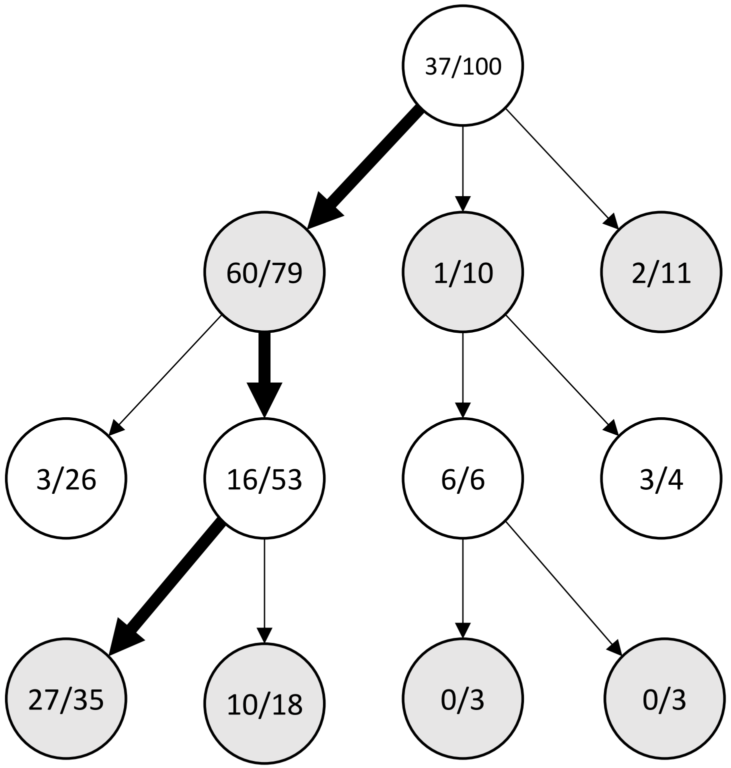
- Selective search
  - Do **not** try to explore all possible actions
  - Only explore parts of the tree that has more potential to improve for the root

- Evaluation by rollouts
  - Play multiple games **to termination** from a state (using some rollout policy), and evaluate through win rate

Monte-Carlo Tree Search (MCTS)



Monte-Carlo Tree Search (MCTS)



Selection

- Starting from the root node, execute **tree policy** until reaching a leaf node
- One effective tree policy is given by UCB1, which chooses an action based on

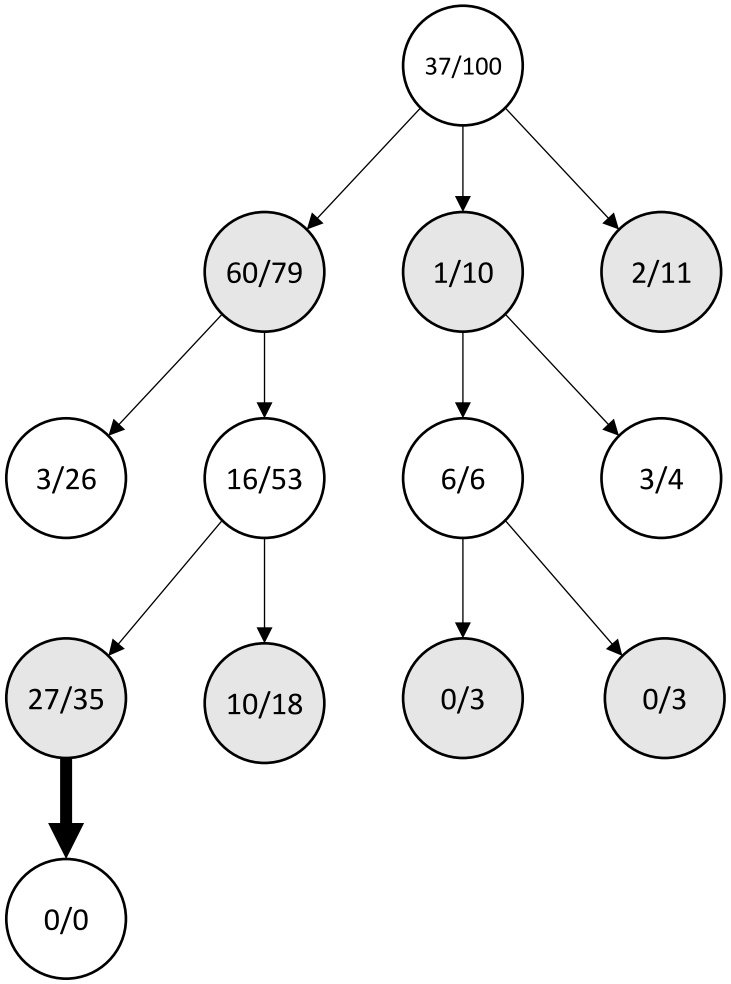
$$\frac{W(n)}{N(n)} + C \times \sqrt{\frac{\log N(\text{parent}(n))}{N(n)}}$$

$W(n)$: total #wins of all playouts that went through node n
$N(n)$: total #playouts that went through node n

Monte-Carlo Tree Search (MCTS)

Expand

- On some iterations, grow the search tree from selected leaf nodes by adding one or more child nodes

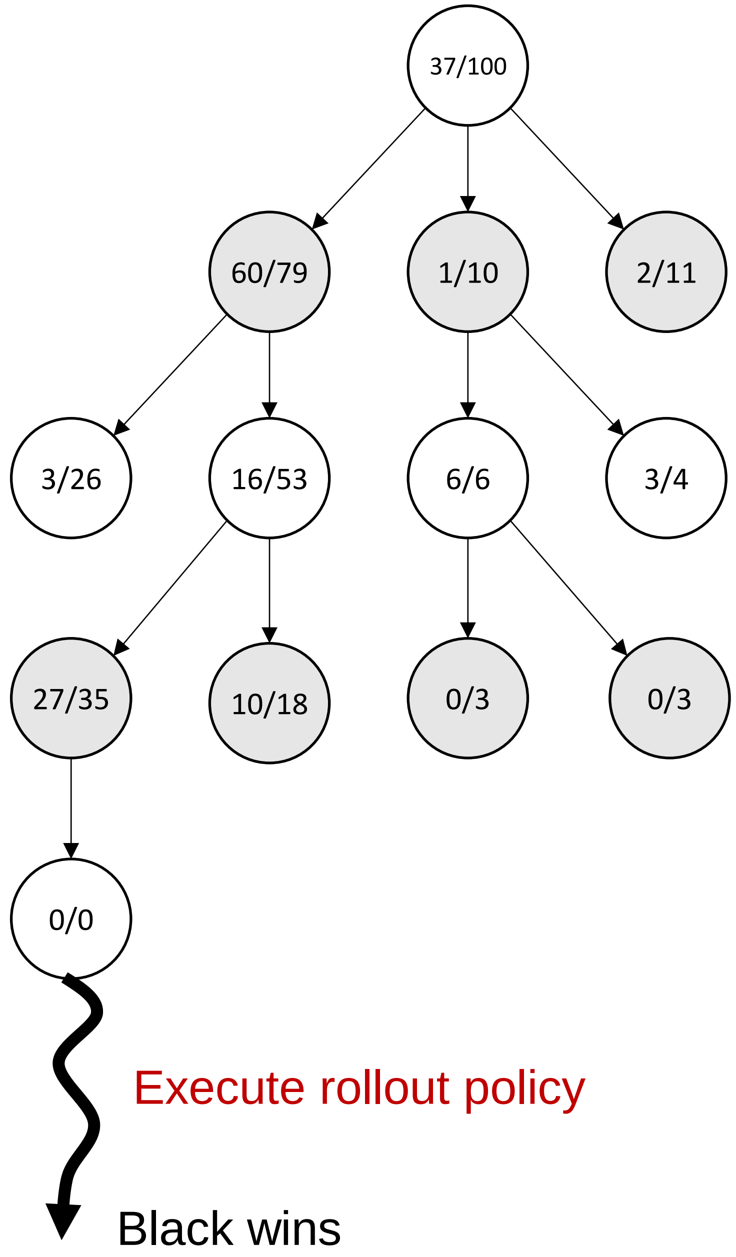


Monte-Carlo Tree Search (MCTS)

Simulation

- From the selected or expanded node (if any), execute the **rollout policy** to the end of the game

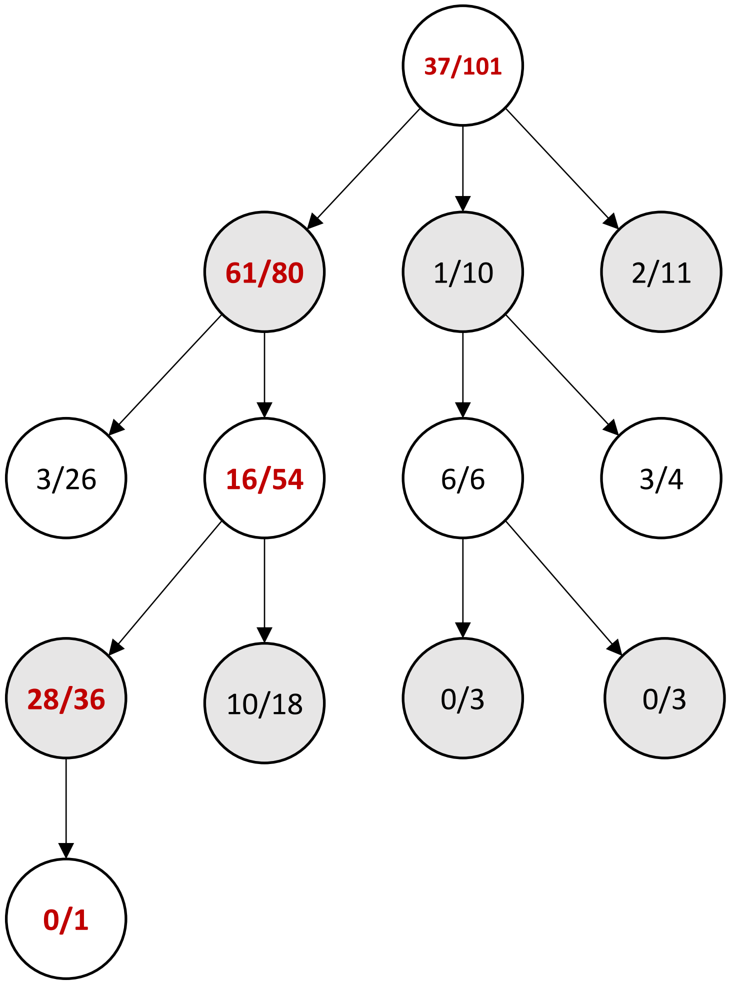
- Rollout policy
  - Could be heuristics, such as “consider capture moves” in chess
  - Could be learned through neural networks by self-play



Monte-Carlo Tree Search (MCTS)

Backup

- Update the #wins and #playouts on nodes along the tree policy



Monte-Carlo Tree Search (MCTS)

Finally,

- Choose the action from the root node that has the largest visit count.
  - Why not the action with the highest win rate?
- After the opponent’s move, start the same procedure from the new state (can keep the statistics from the previous state)

Application of MCTS in AlphaGo and AlphaGo Zero

Check Section 16.6 of https://www.andrew.cmu.edu/course/10-703/textbook/BartoSutton.pdf