innovate achieve lead

BITS Pilani

Pilani Campus

Contents: Game Playing

(Minimax, alpha-beta pruning)

Brief about Constraint Satisfaction Problem

Board Games

- Two Person
  - Have exactly two players.
- Zero Sum
  - In zero sum games, the payoff is zero. One player's gain is the other player's loss. One wins; the other loses.
- Complete Information
  - Both players have access to all the information i.e., can see the board, and thus know the options the other play has.
- Alternate moves
  - Players take turns to make their moves.

What kind of games

Mainly games of strategy with the following characteristics:

1. Sequence of moves to play
2. Rules that specify possible moves
3. Rules that specify a payment for each move
4. Objective is to maximize your payment

Games vs Search Problem

- Unpredictable opponent → specifying a move for every possible opponent reply

- Time limits → unlikely to find goal, must approximate

Games as Adversarial Search

- States:
  - board configurations
- Initial state:
  - the board position and which player will move
- Successor function:
  - returns list of (move, state) pairs, each indicating a legal move and the resulting state
- Terminal test:
  - determines when the game is over
- Utility function:
  - gives a numeric value in terminal states (e.g., -1, 0, +1 for loss, tie, win)

Game Tree (2-player, Deterministic, Turns)

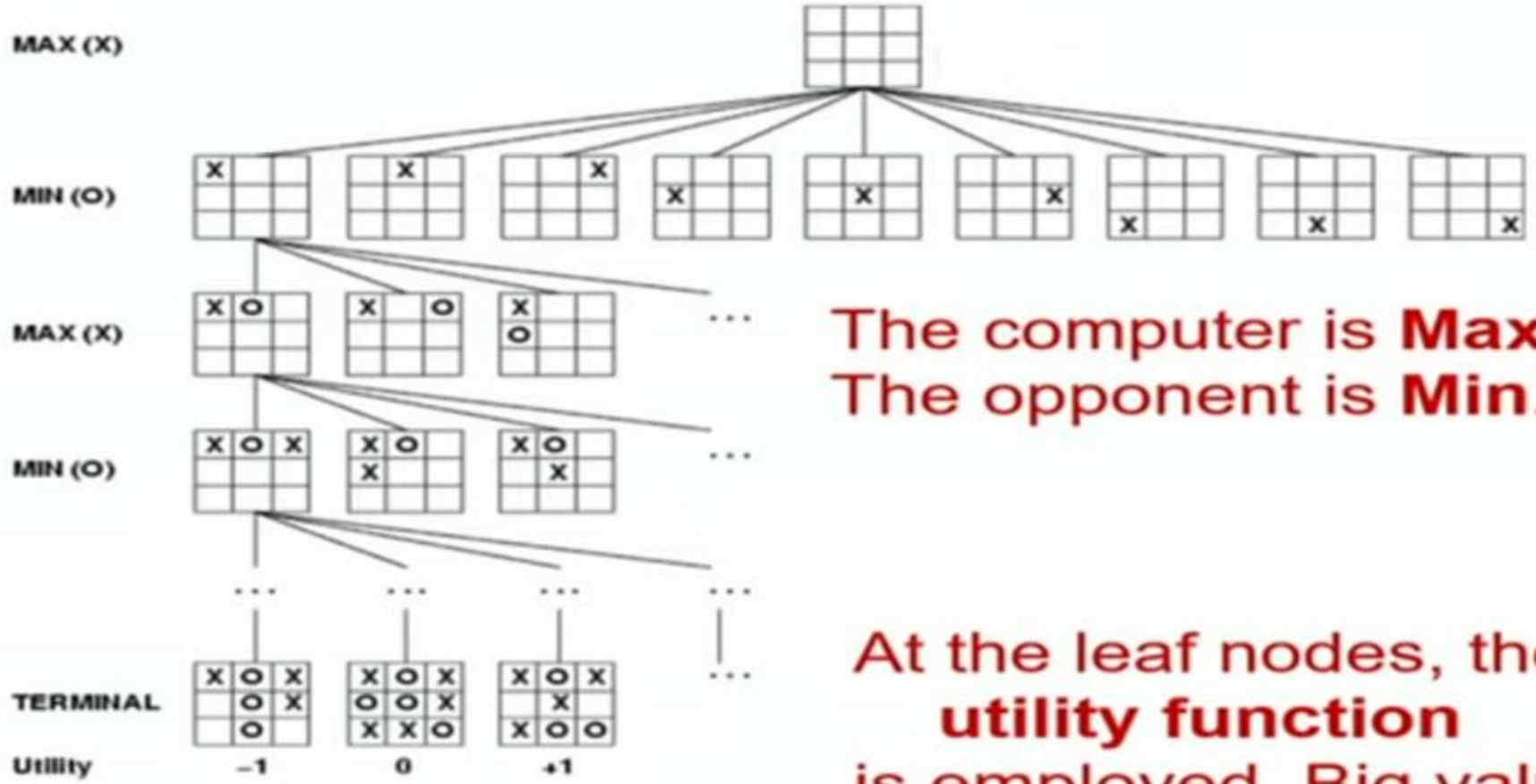
computer's turn

opponent's turn

computer's turn

opponent's turn

leaf nodes are evaluated



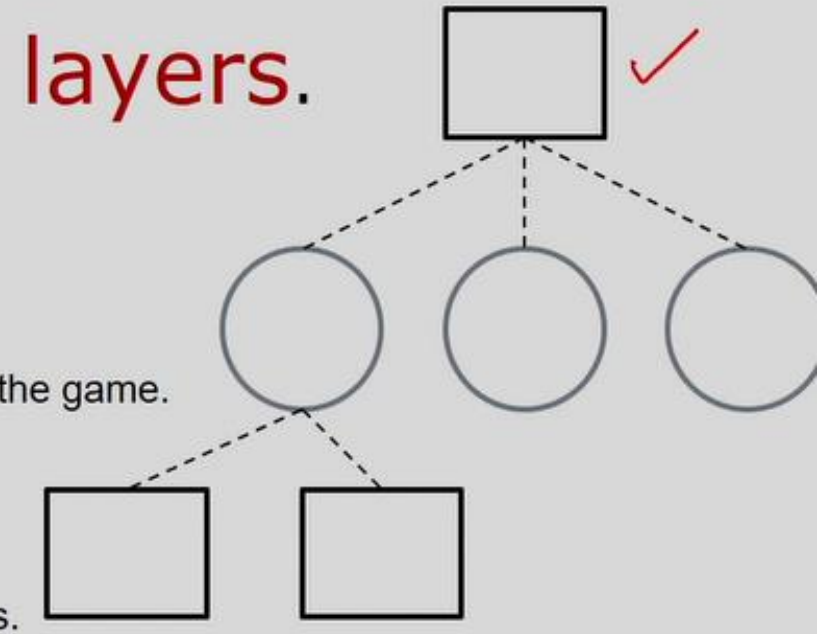
The computer is **Max**.
The opponent is **Min**.

At the leaf nodes, the **utility function** is employed. Big value means good, small is bad.

Game Trees

- A game is represented by a game tree.
  - Game tree is a layered tree in which at each alternating level, one or the other player makes the choice.
  - Layers - MAX layers and the MIN layers.

A game starts at the root with MAX playing first.



Leaves of a game tree are labelled with outcome of the game.

The game ends at the leaf nodes.

Minimax Procedure

- For complex games such as chess or checkers, search to termination is out of question.

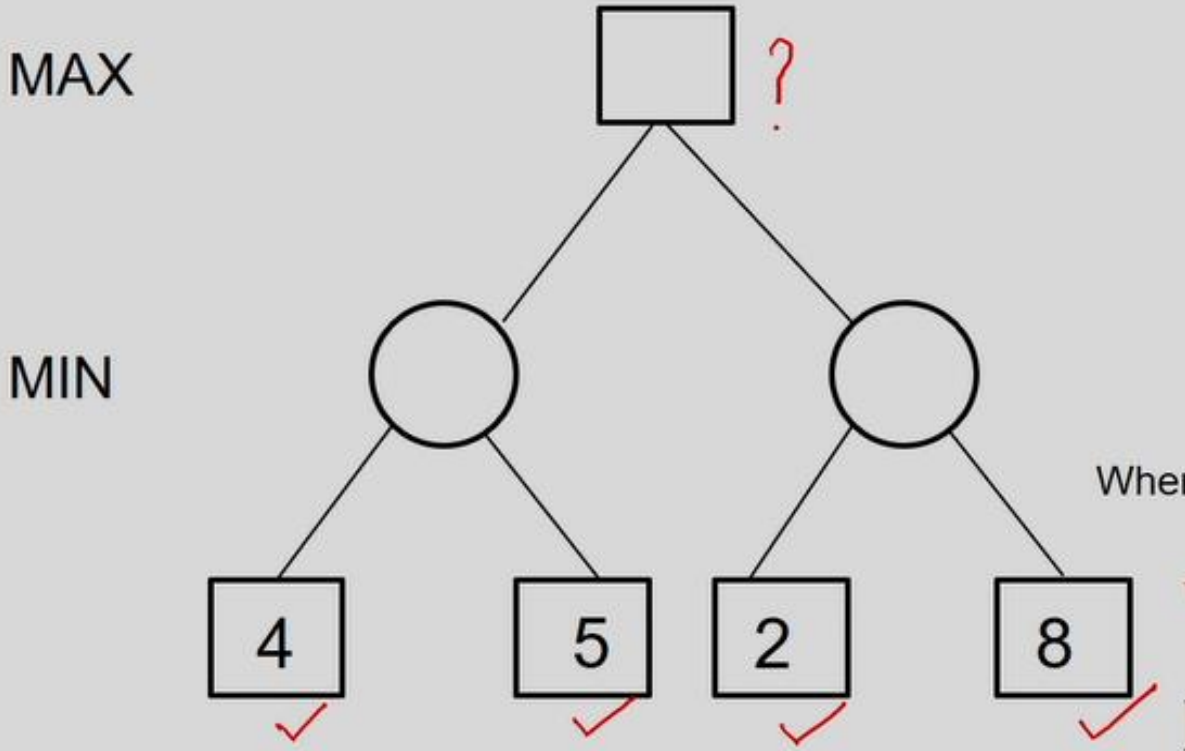
  Complete game tree for chess has approximately 10^{40} nodes.

- Good First Move?

  Even for a game as simple as Tic-tac-toe there are over 3,50,000 nodes in the complete game tree.

  - A good first move can be extracted by a procedure called the Minimax.
  - This estimate can be made by applying a static evaluation function to the leaf node.
  - Back-up values level by level.
    - MAX parent of MIN nodes is assigned backed-up value equal to maximum of the evaluations of the nodes.
    - MIN parent of MAX nodes is assigned the minimum.

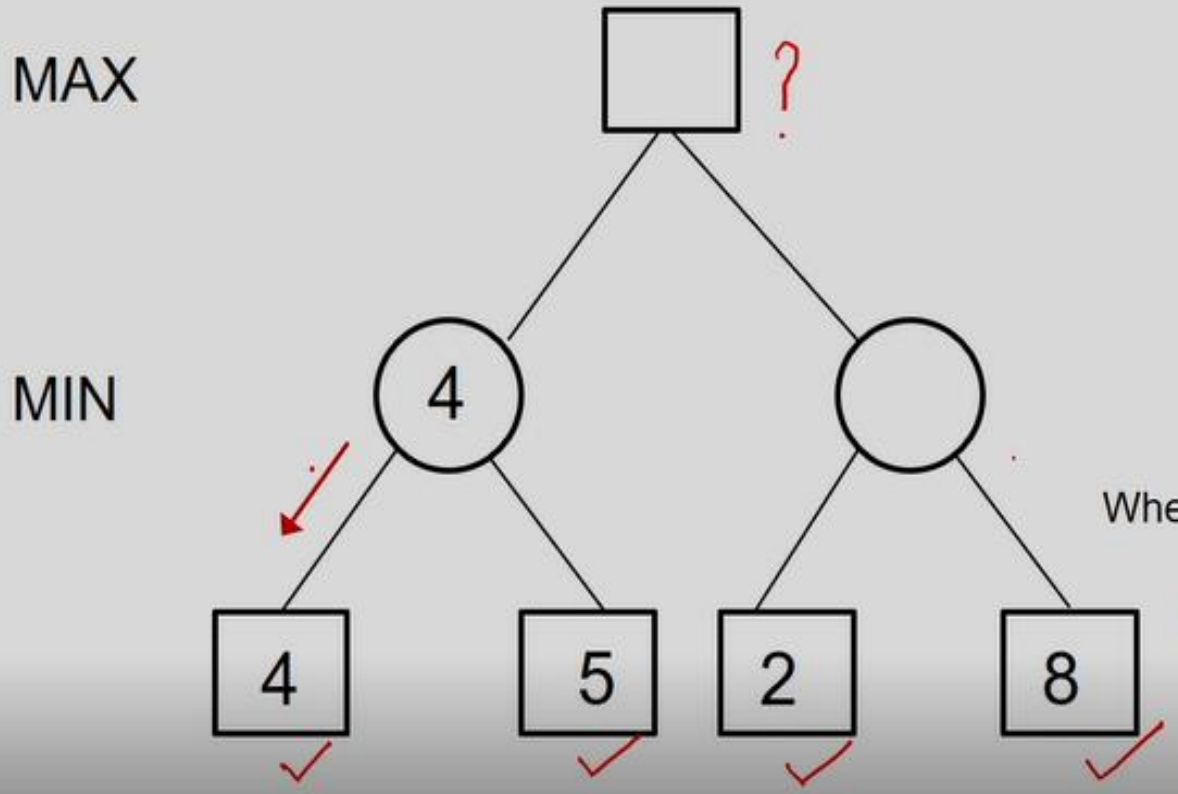
Minimax Procedure



Where is the play going to go?

Value of the board from the perspective of the MAX player.

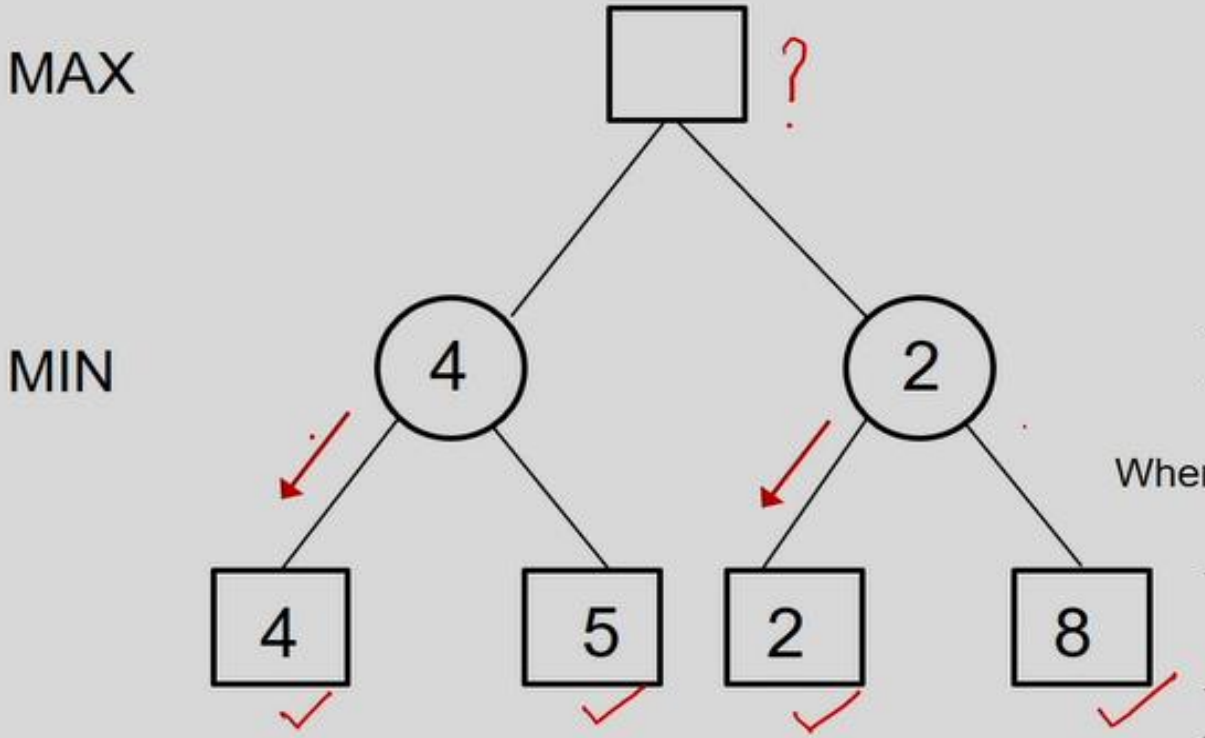
Minimax Procedure



Where is the play going to go?

Value of the board from the perspective of the MAX player.

Minimax Procedure



Back-up values one level.

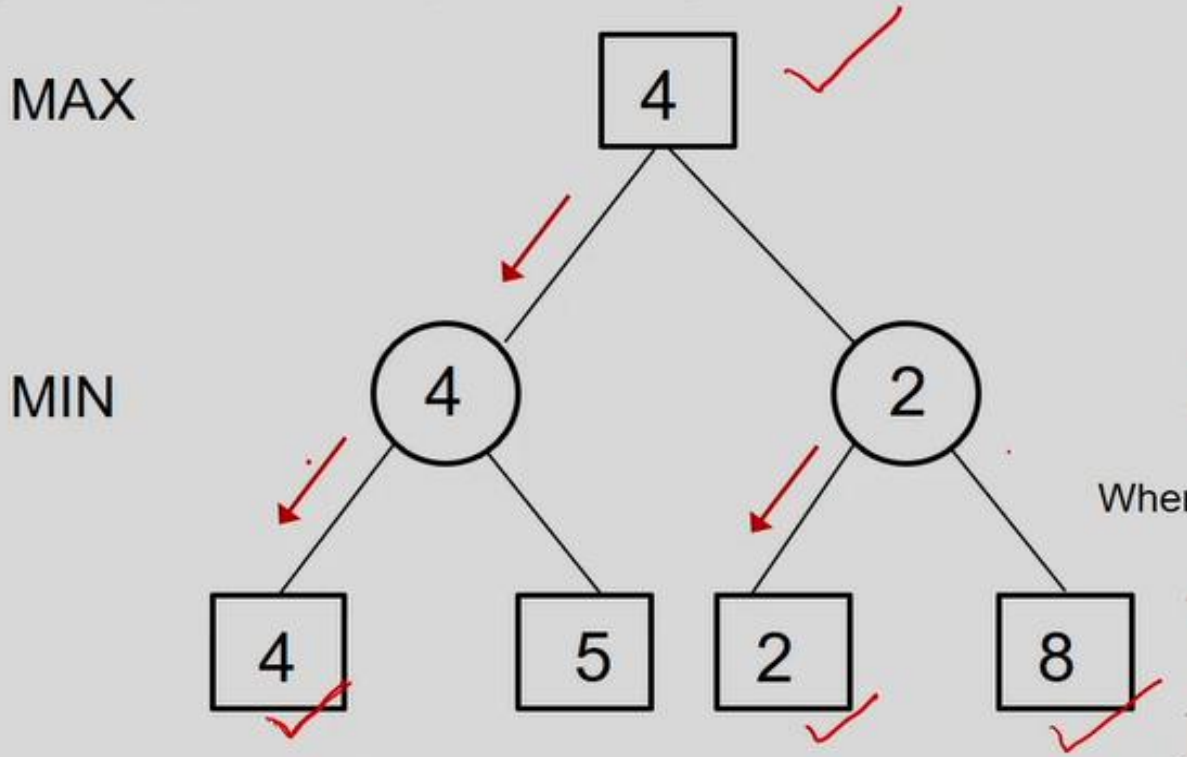
Where is the play going to go?

Value of the board from the perspective of the MAX player.

Minimax Procedure

Static evaluation functions measures the worth of a leaf node.

Measurement is based on features thought to influence the worth.
E.g. In checkers – relative piece advantage; control of center etc.



Adversarial game – Competing with each other.

Far shorter than 8, the MAX player wanted.
More than 2 that the MIN player wanted.

Minimax Algorithm

1. Go to the bottom of the tree.
2. Compute static values.
3. Back them up level-by-level.
4. Decide where to go.

Back-up values one level.

Where is the play going to go?

Value of the board from the perspective of the MAX player.

Do not expect to go where YOU wanted!

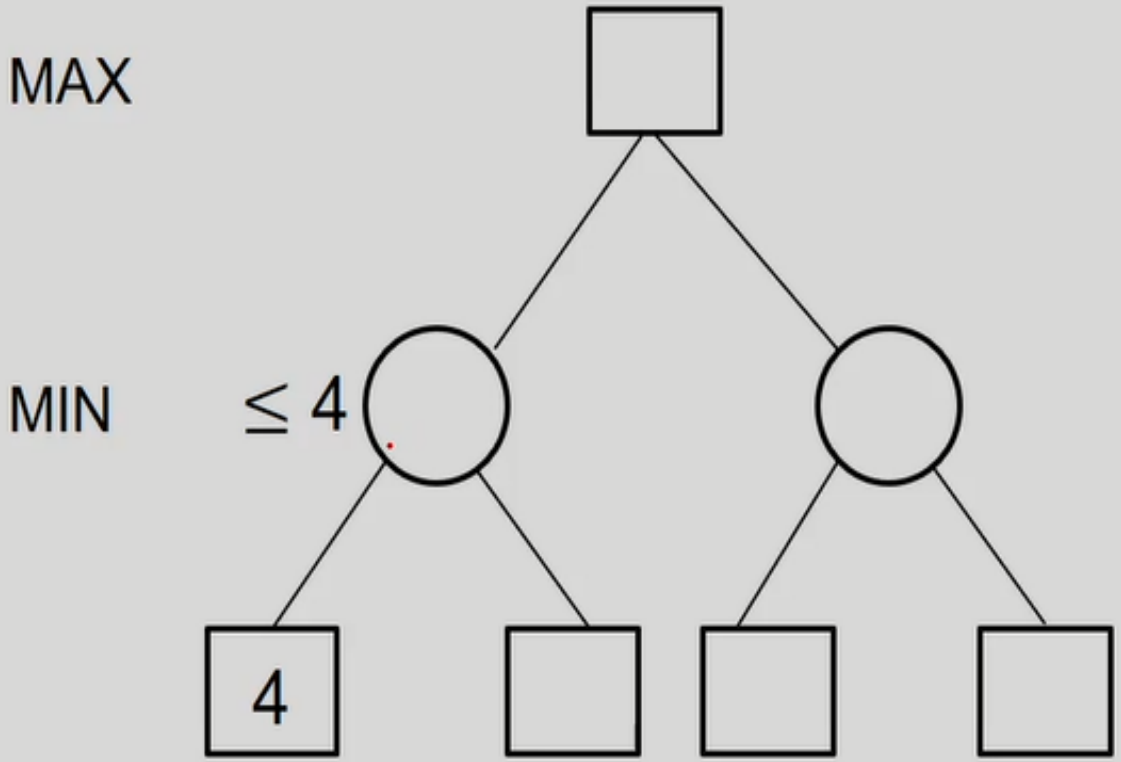
Minimax Algorithm

```
function MINIMAX-DECISION(state) returns an action
    v ← 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 ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

Properties of Minimax

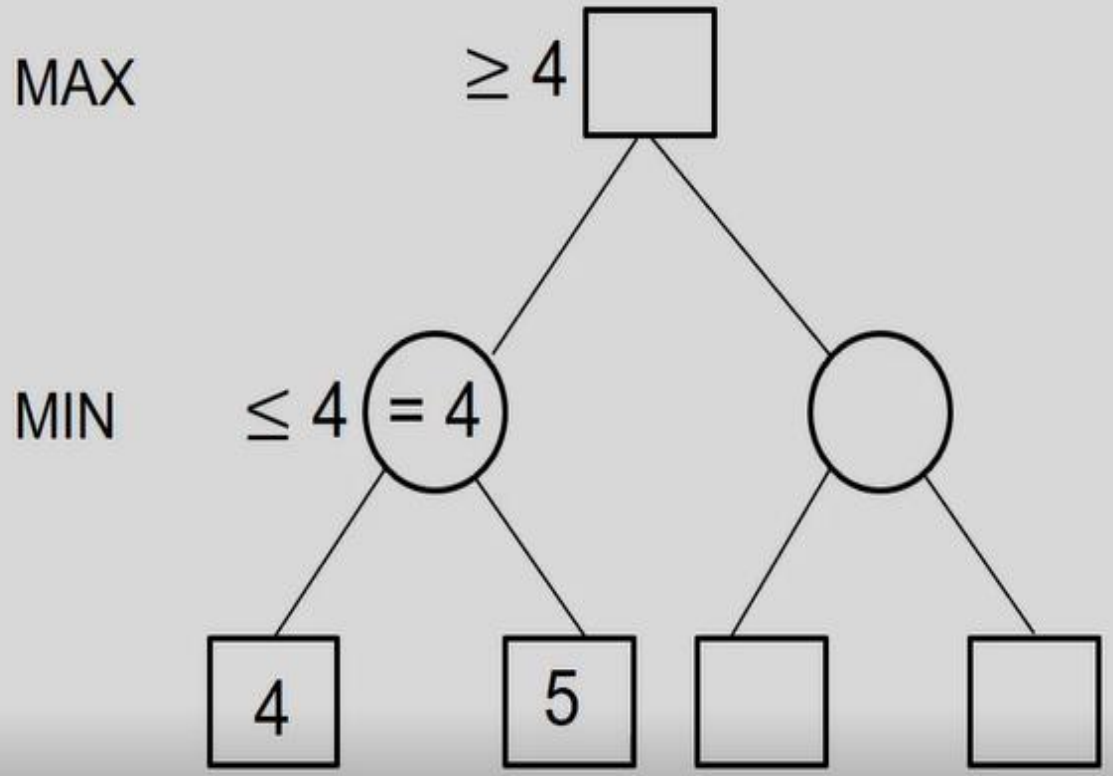
- Complete?
  - Yes (if tree is finite)
- Optimal?
  - Yes (against an optimal opponent)
  - No (does not exploit opponent weakness against suboptimal opponent)
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - O(bm) (depth-first exploration)

Alpha-beta Pruning



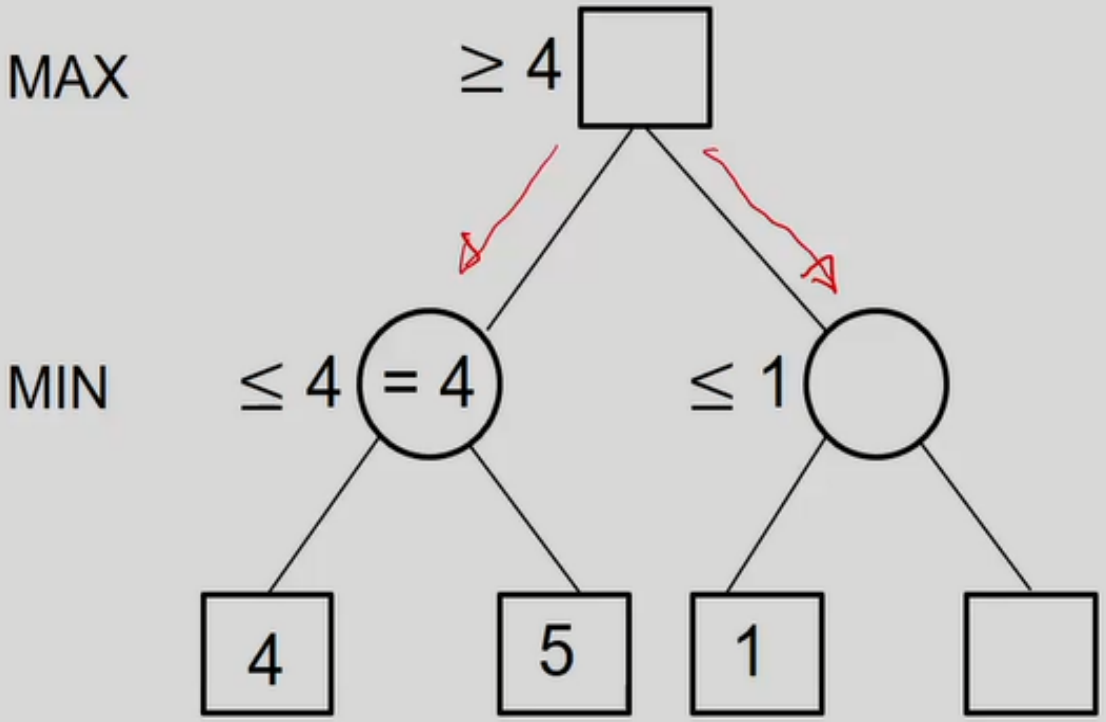
Compute static values one at a time.

Alpha-beta Pruning



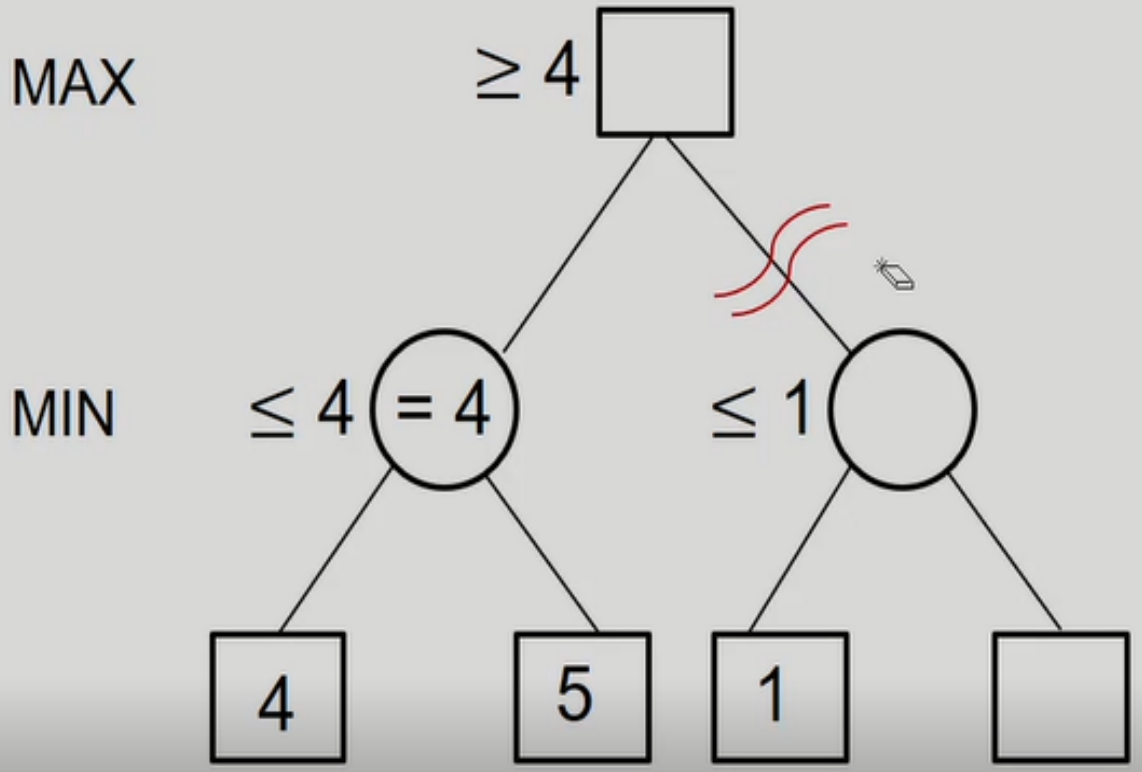
Compute static values one at a time.

Alpha-beta Pruning



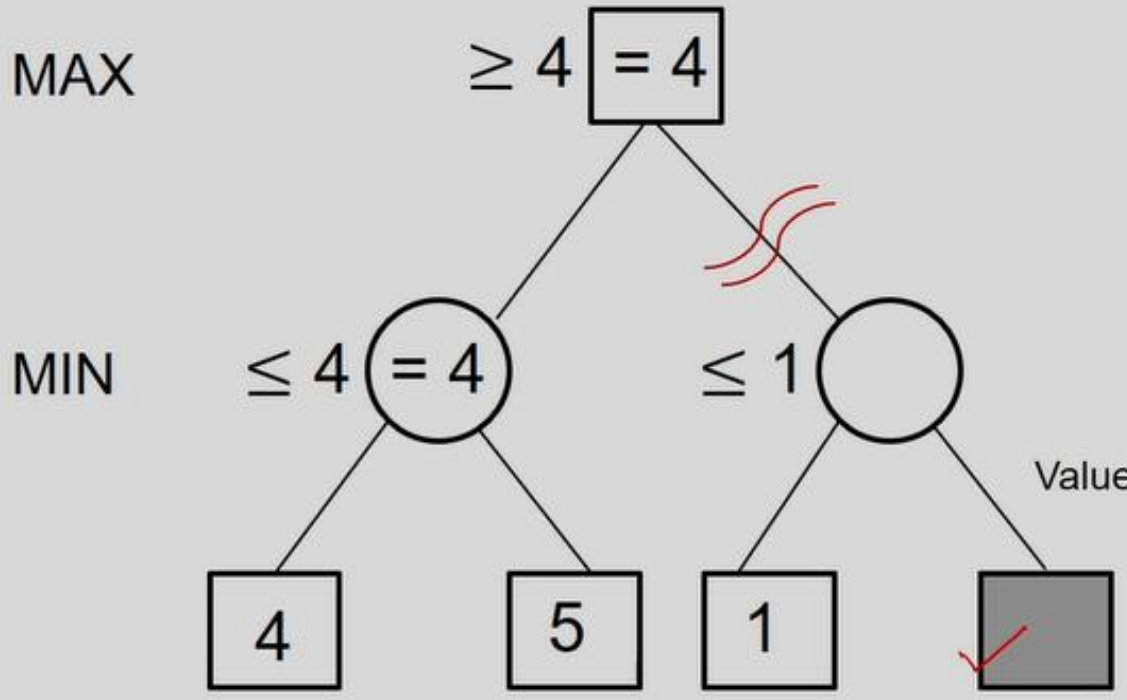
Compute static values one at a time.

Alpha-beta Pruning



Compute static values one at a time.

Alpha-beta Pruning



Alpha-beta Algorithm

1. Not a separate algorithm!
2. Layering on top of Minimax.

Essence of Alpha-Beta Pruning; cuts out sections of the search space

Values in the cut-out branch can't affect the value of the root node.

Compute static values one at a time.

We don't need to compute the value at this node.

Alpha-beta Pruning

- Alpha-beta pruning is not actually a new algorithm, rather an optimization technique for minimax algorithm.
- It reduces the computation time by a huge factor. This allows us to search much faster and even go into deeper levels in the game tree.
- It cuts off branches in the game tree which need not be searched because there already exists a better move available.
- It is called Alpha-Beta pruning because it passes two extra parameters in the minimax function

Alpha-beta Pruning

Traverse the search tree in depth-first order

alpha(n) The best value that MAX currently can guarantee; maximum value found so far.

beta(n) The best value that MIN currently can guarantee; minimum value found so far

Beta cutoff

Cutoff- Do not generate or examine any more of n's children.

Given a MAX node n

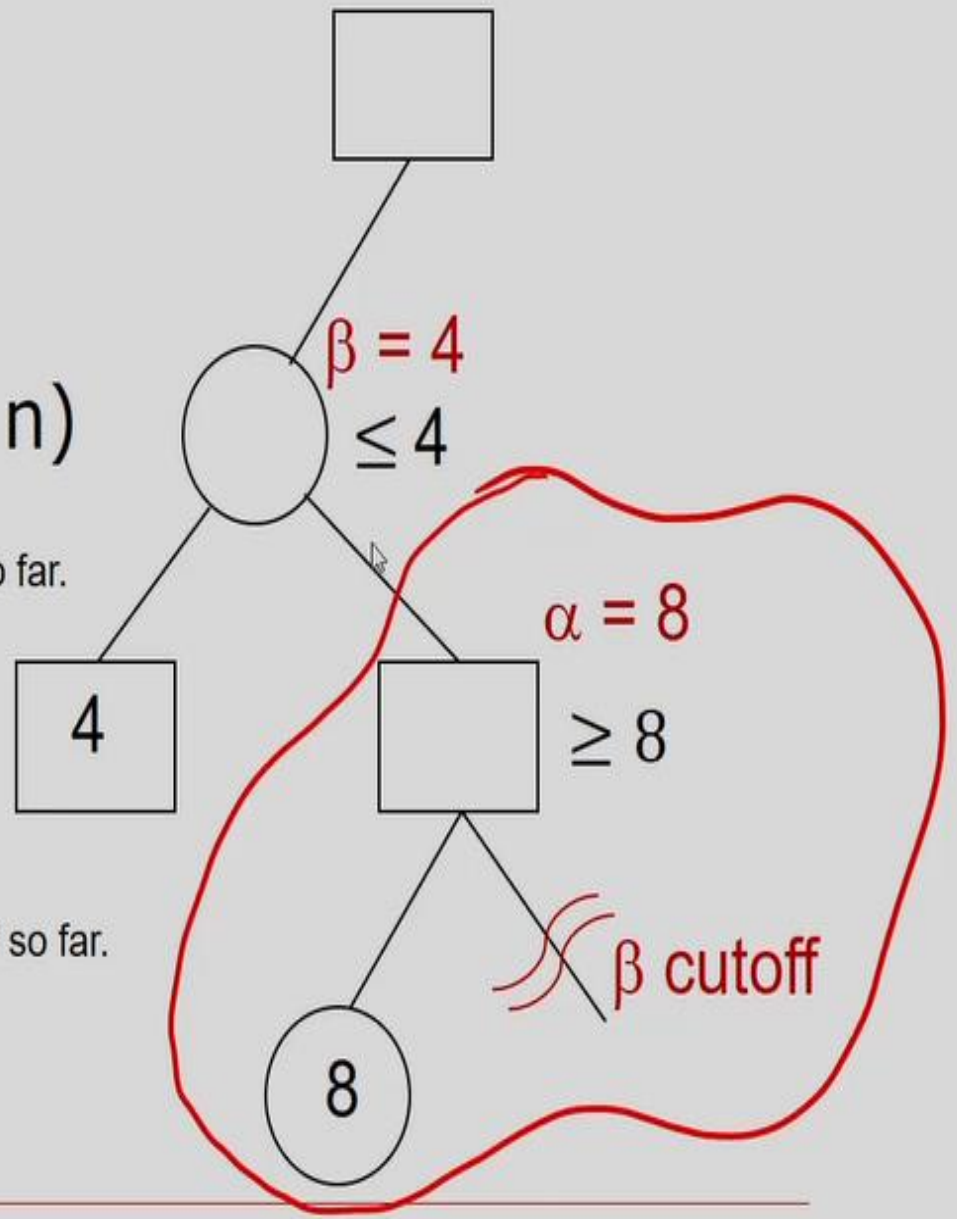
Cutoff ~~search~~ below n

if alpha(n) ≥ beta(i)

(for some MIN node ancestor i of n)

beta - best value that MIN currently can guarantee; minimum value found so far.

alpha - best value that MAX currently can guarantee; maximum value found so far.



Alpha cutoff

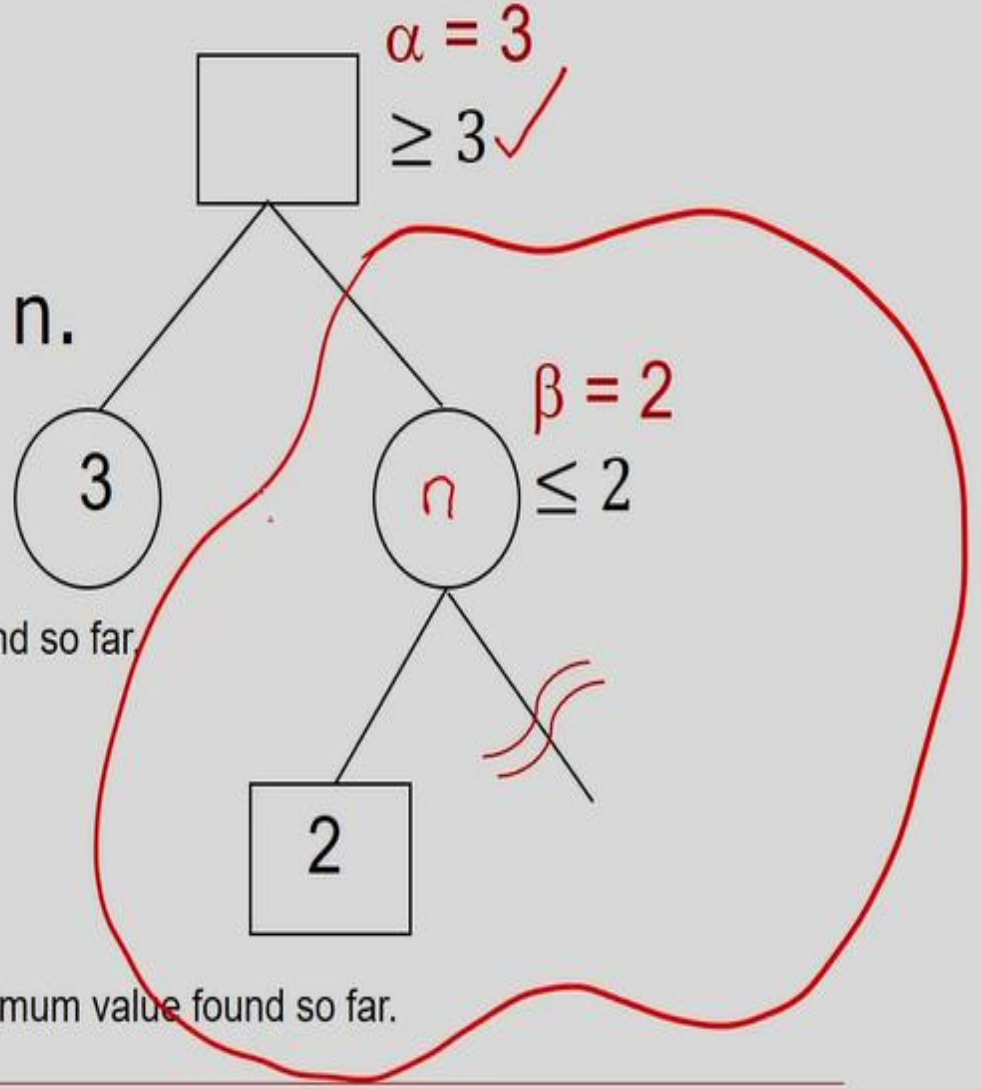
Cutoff- Do not generate or examine any more of n's children.

Given a MIN node n

Cutoff search below n

if beta(n) ≤ alpha(i)

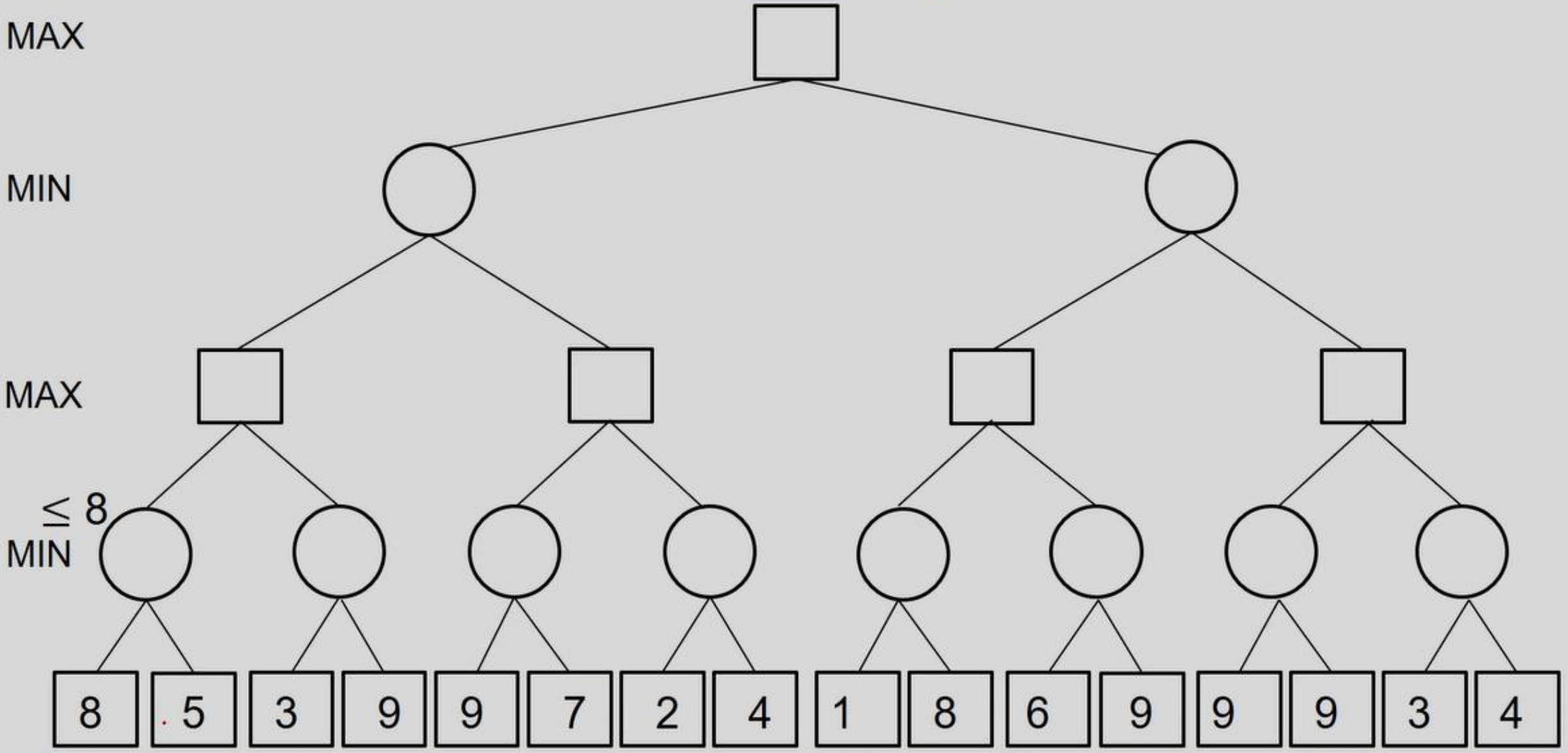
for some MAX node ancestor i of n.



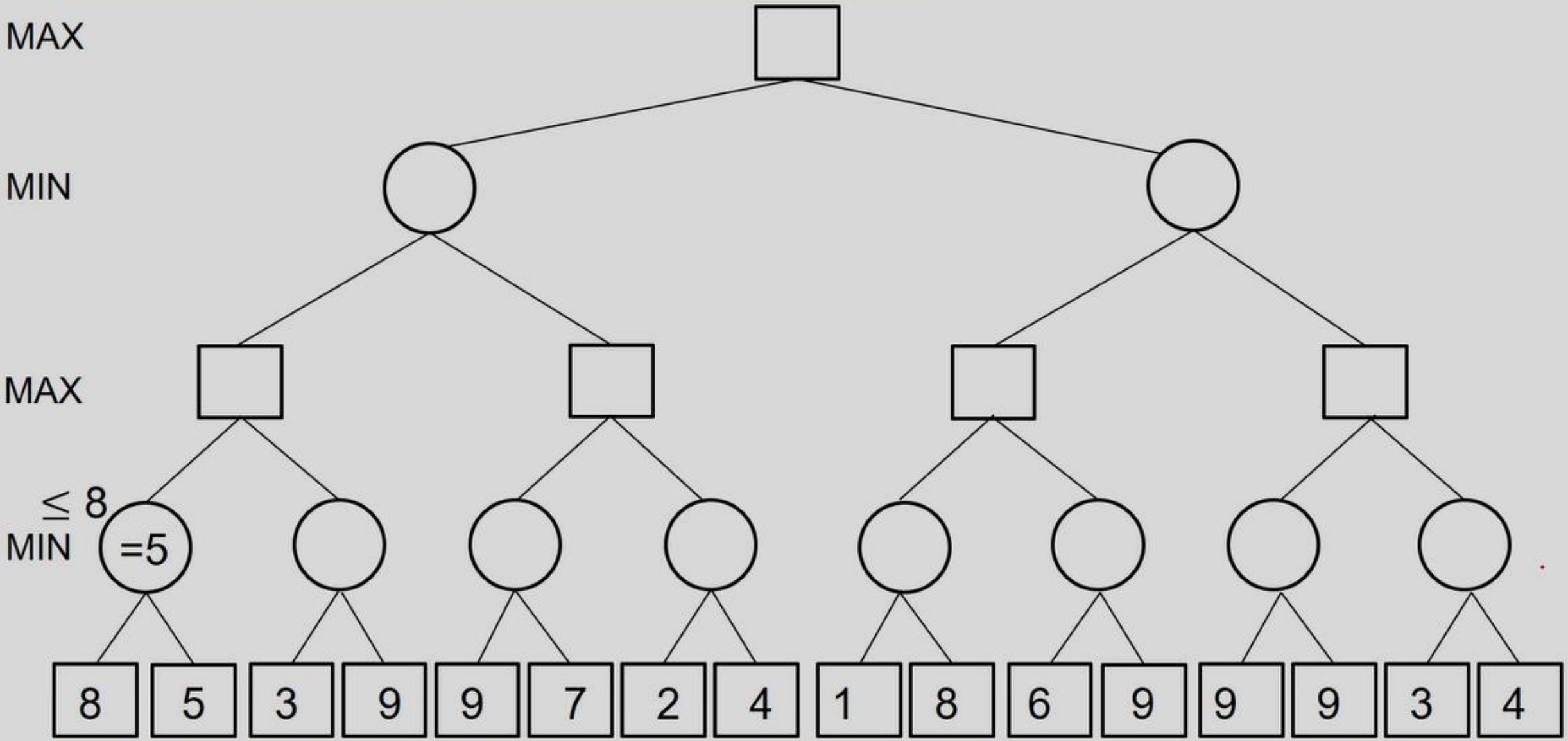
beta - best value that MIN currently can guarantee; minimum value found so far.

alpha - best value that MAX currently can guarantee; maximum value found so far.

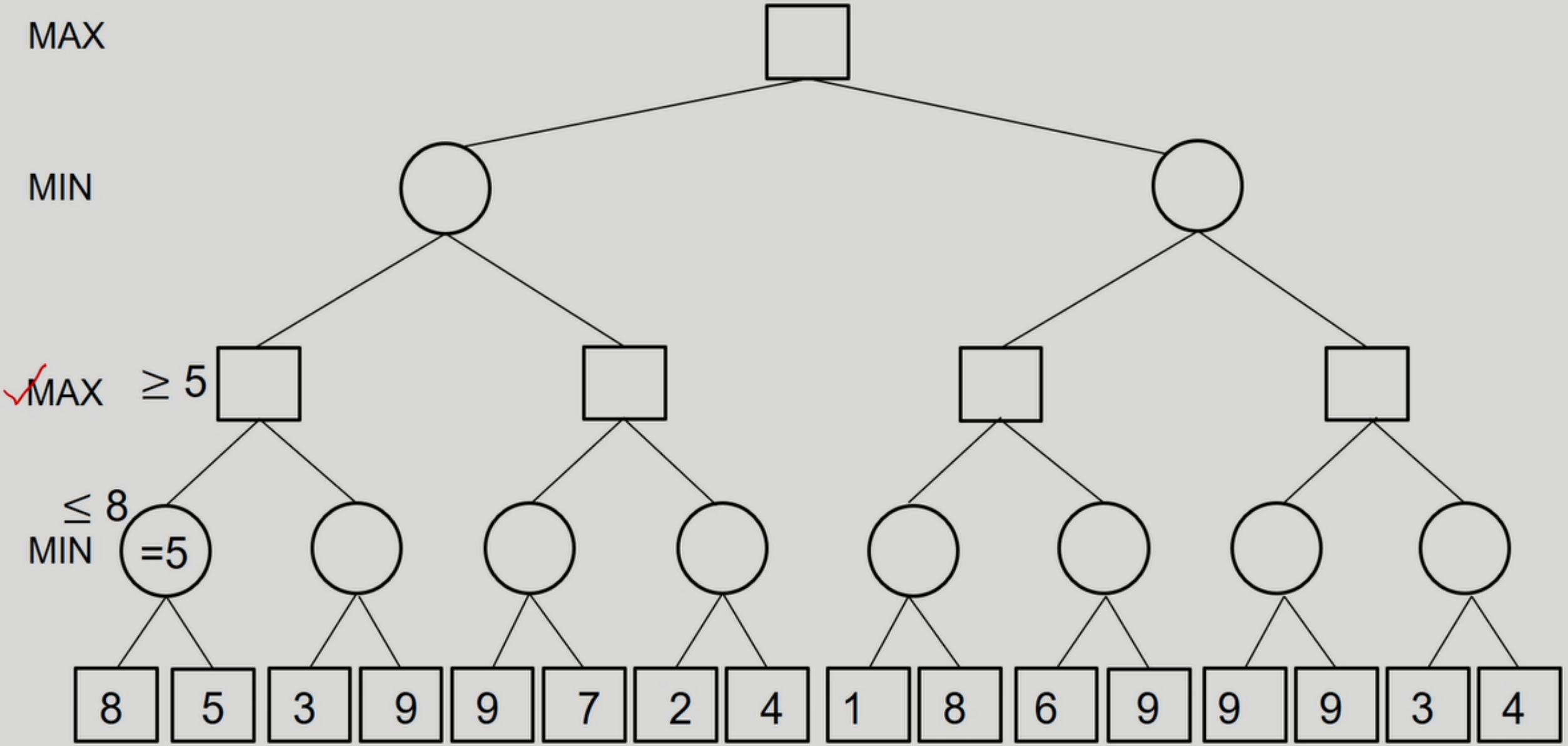
Alpha-beta Pruning



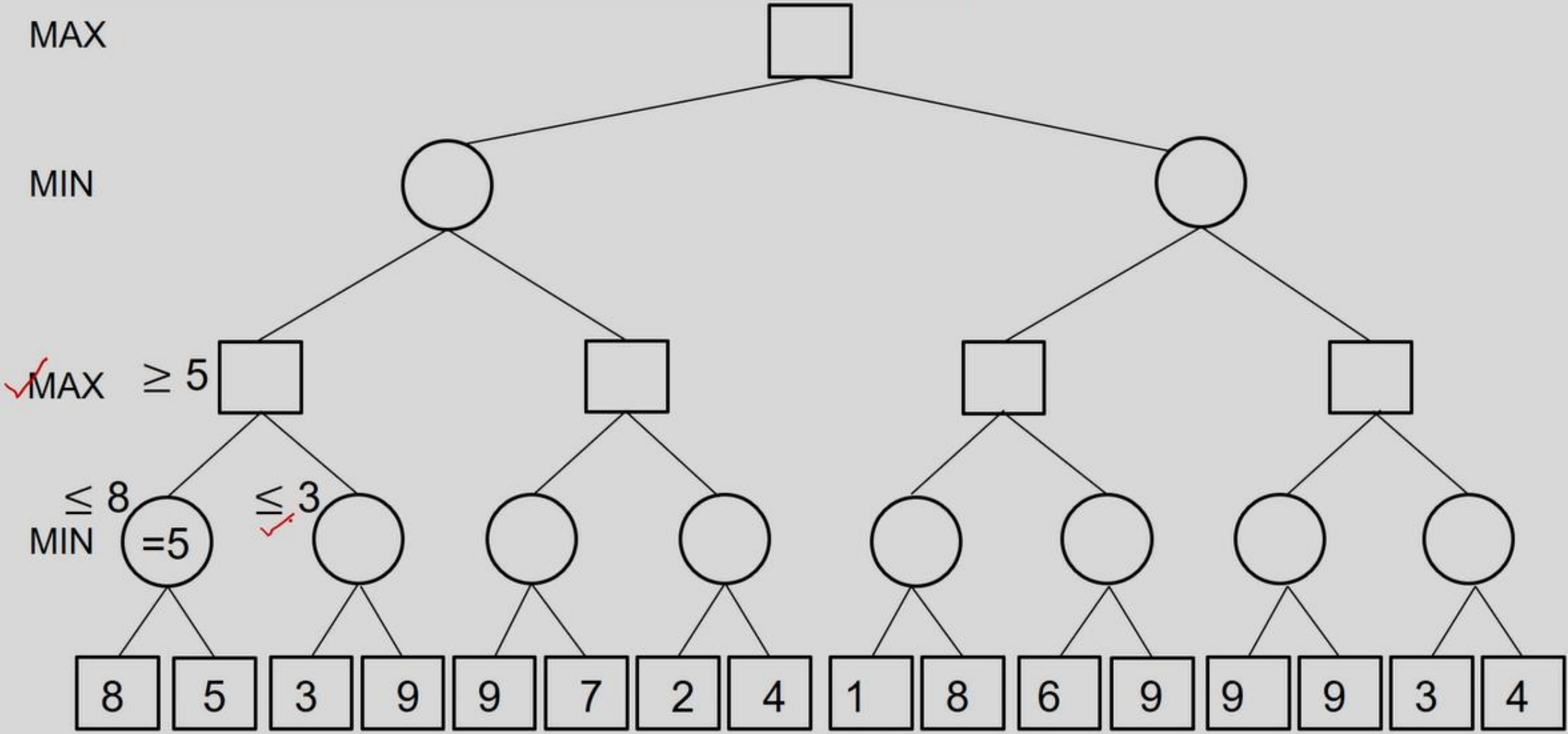
Alpha-beta Pruning



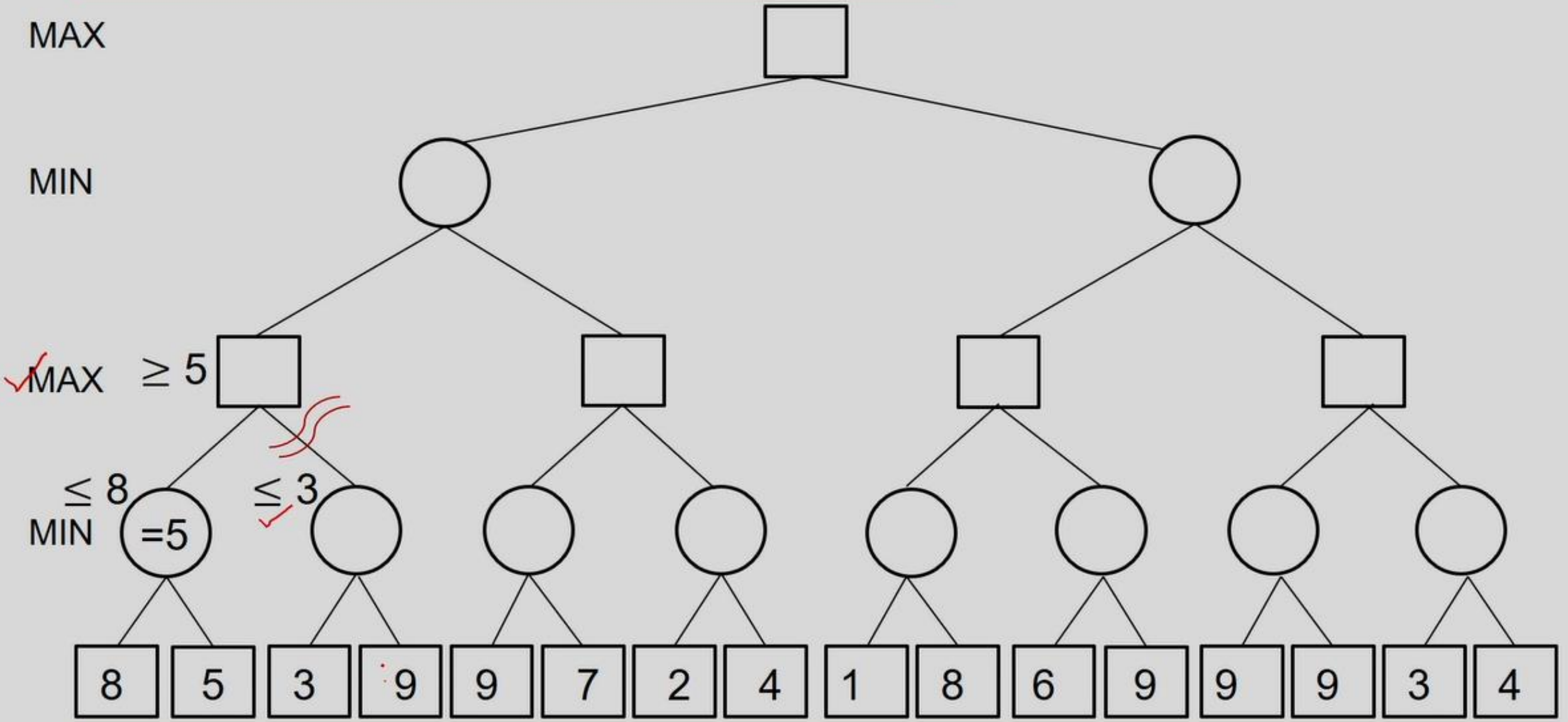
Alpha-beta Pruning



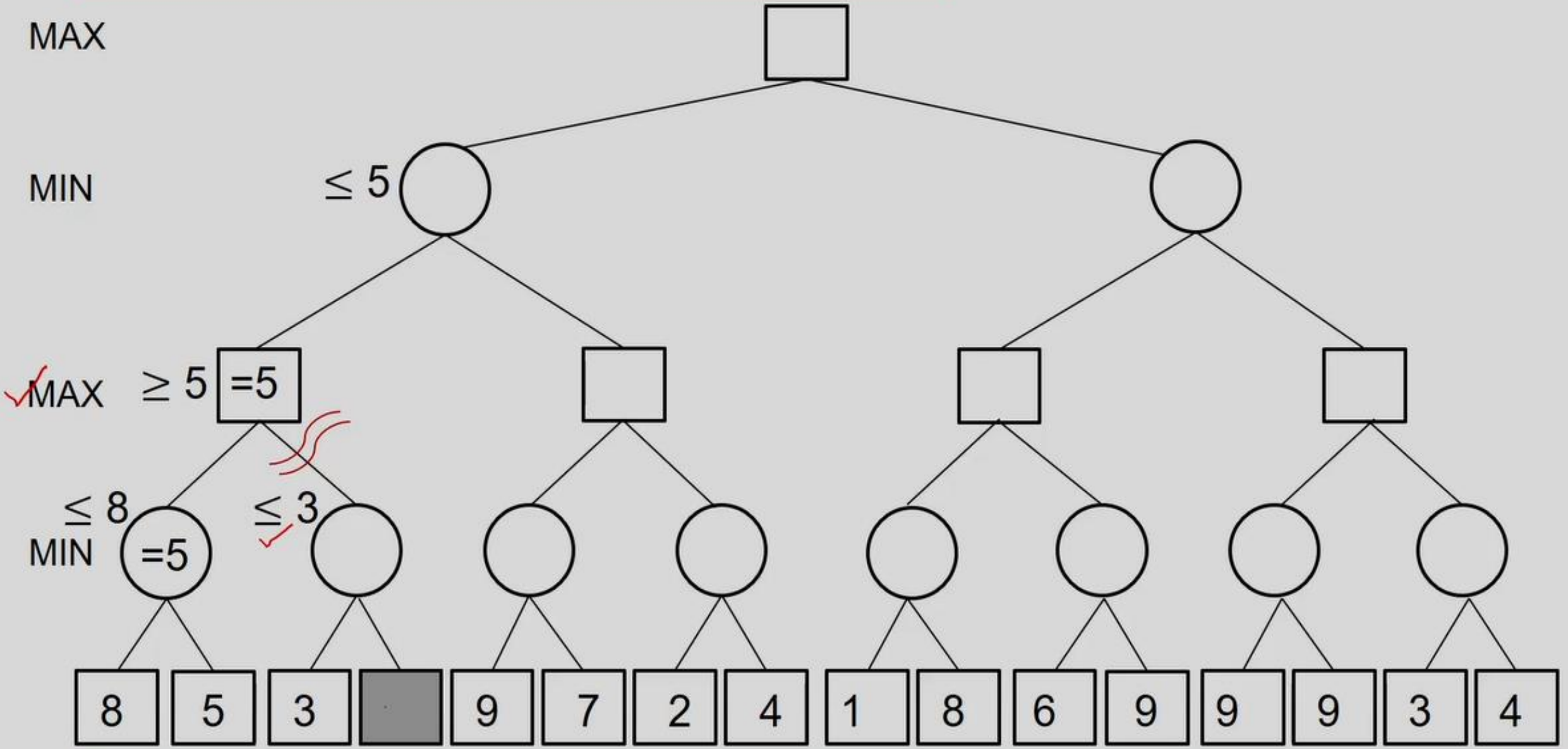
Alpha-beta Pruning



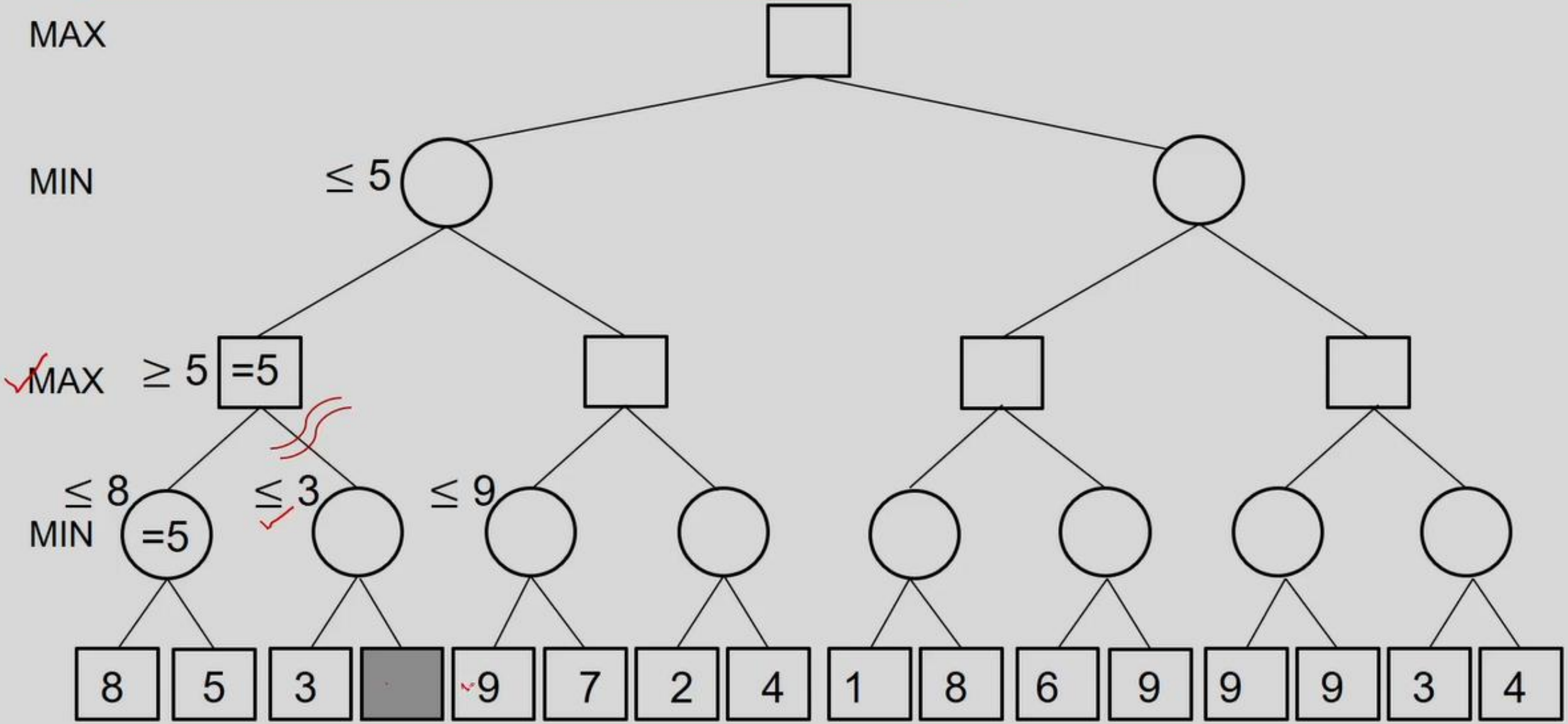
Alpha-beta Pruning



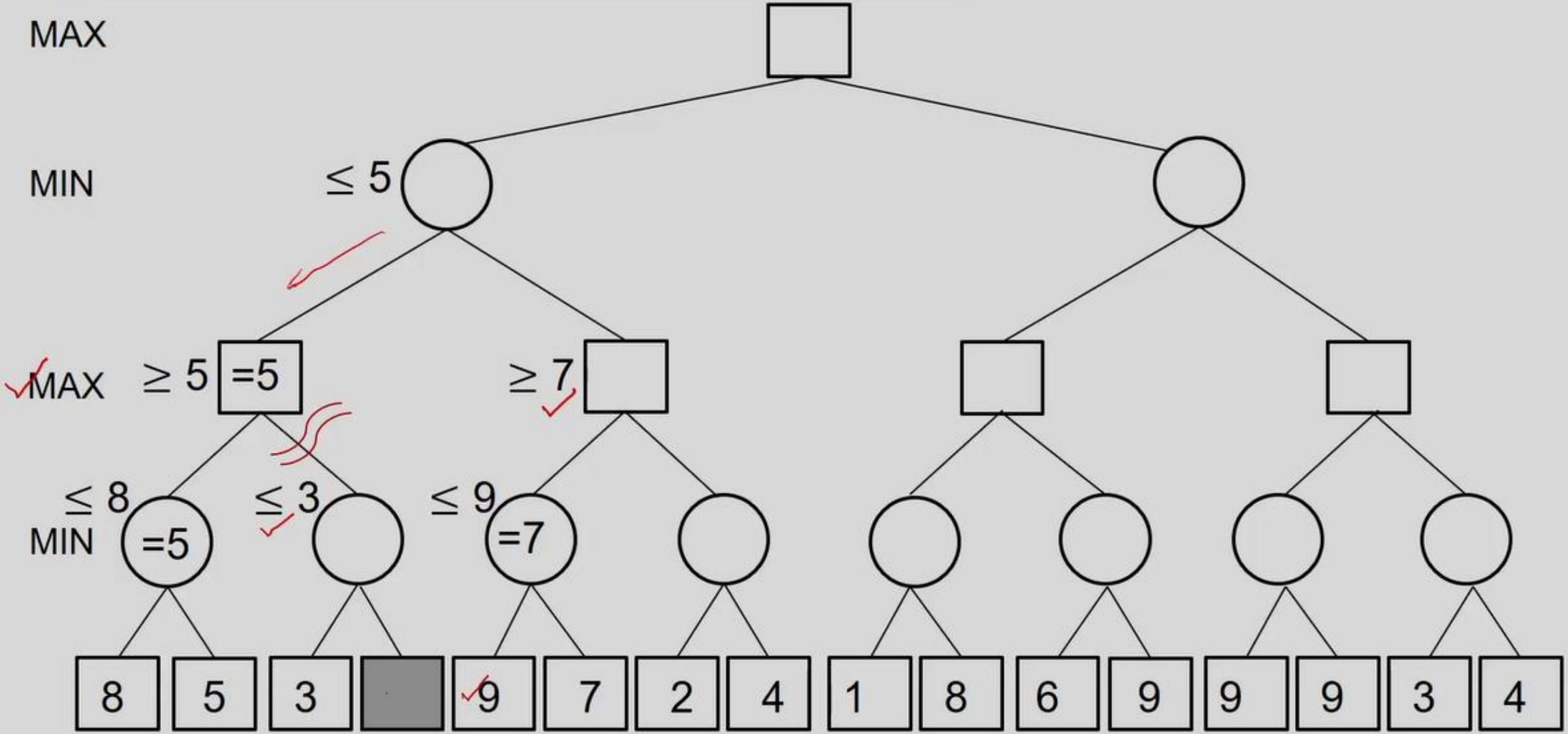
Alpha-beta Pruning



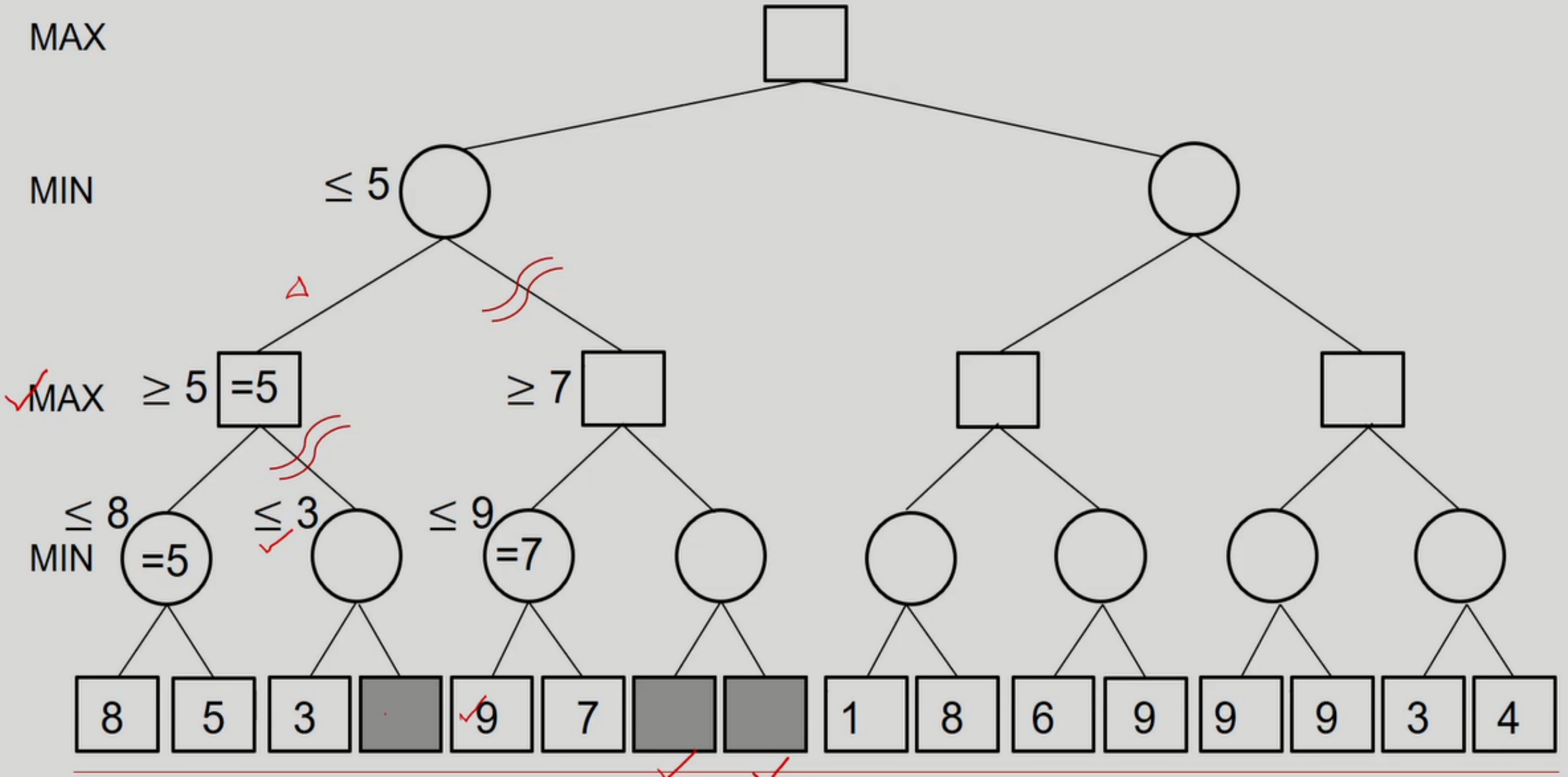
Alpha-beta Pruning



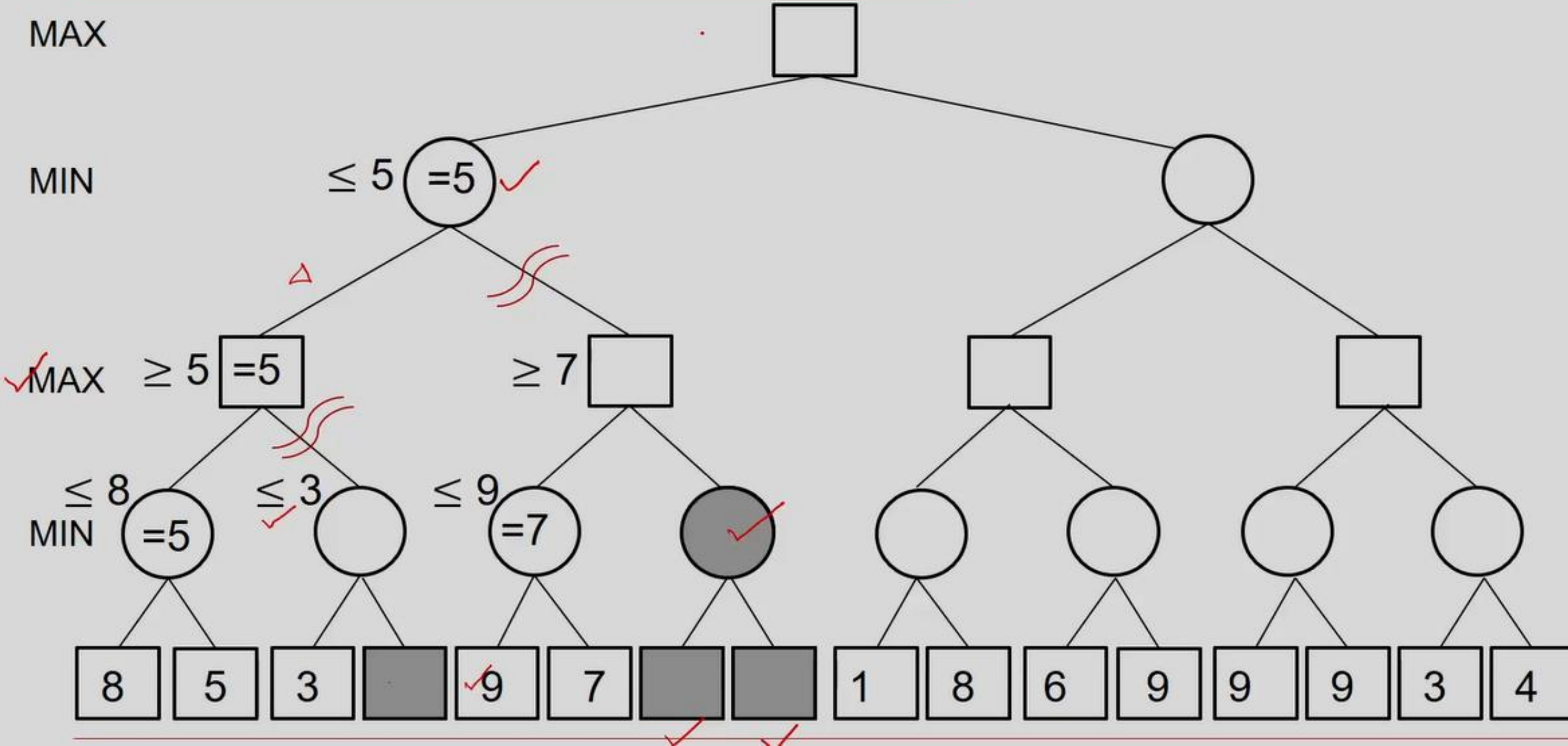
Alpha-beta Pruning



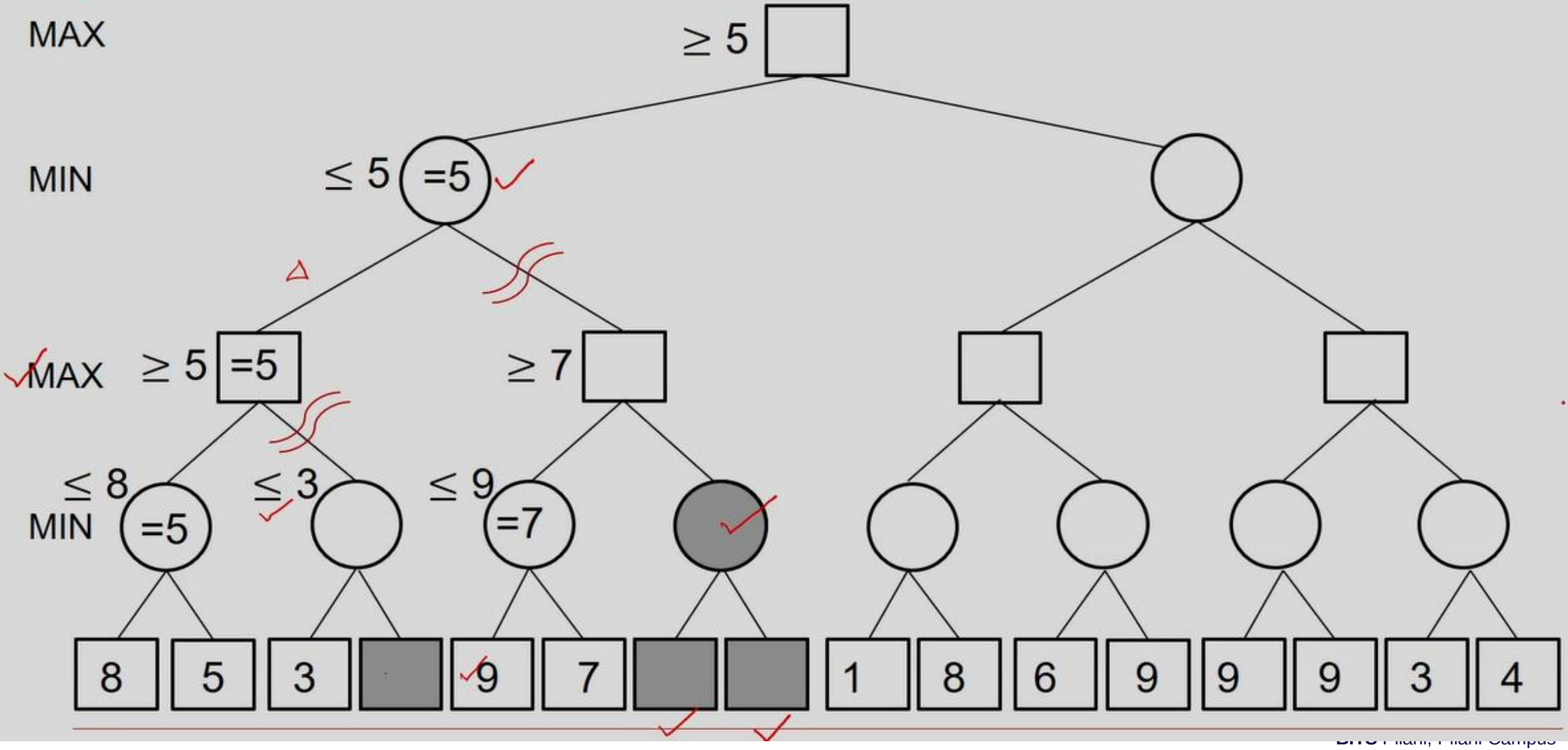
Alpha-beta Pruning



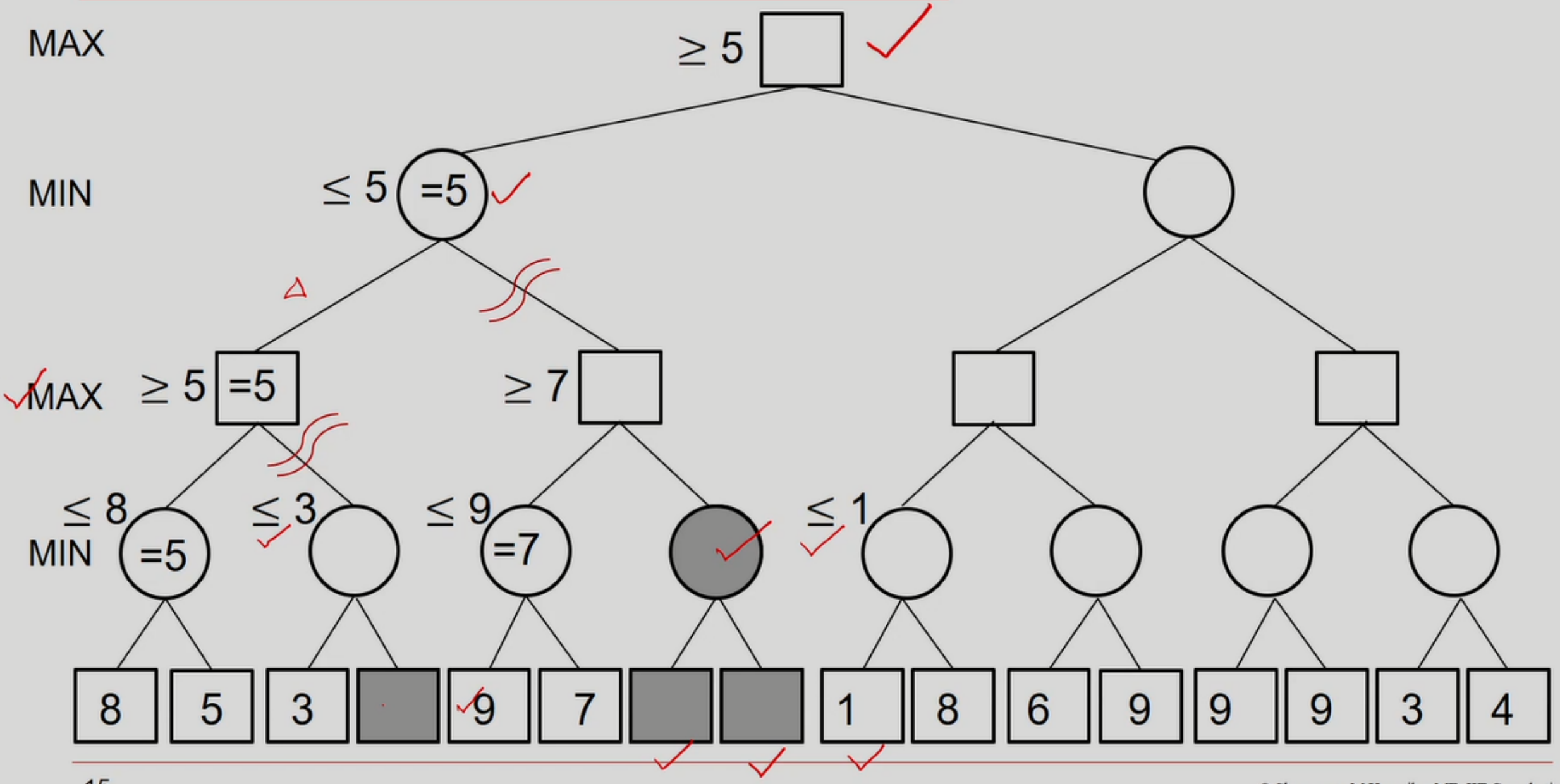
Alpha-beta Pruning



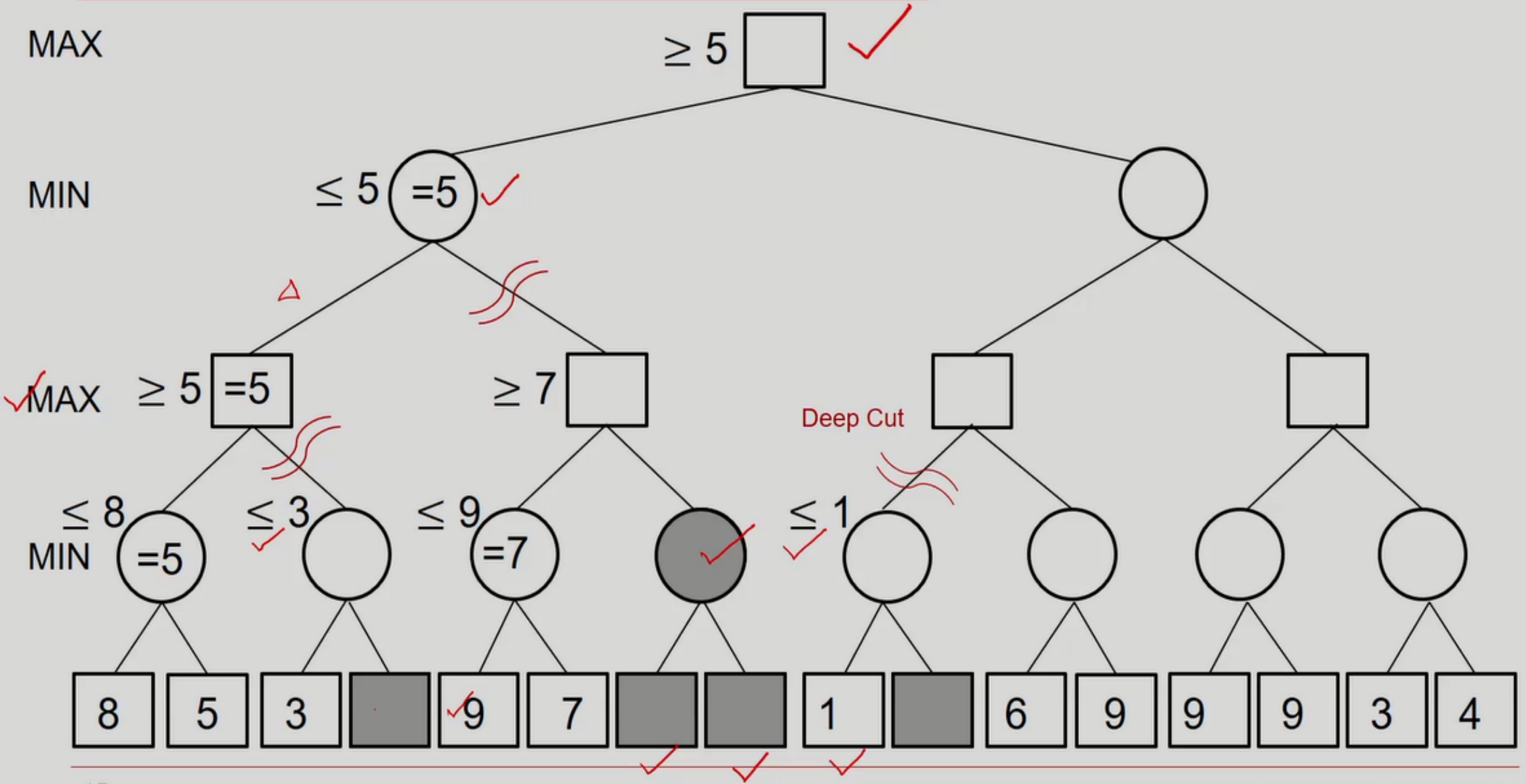
Alpha-beta Pruning



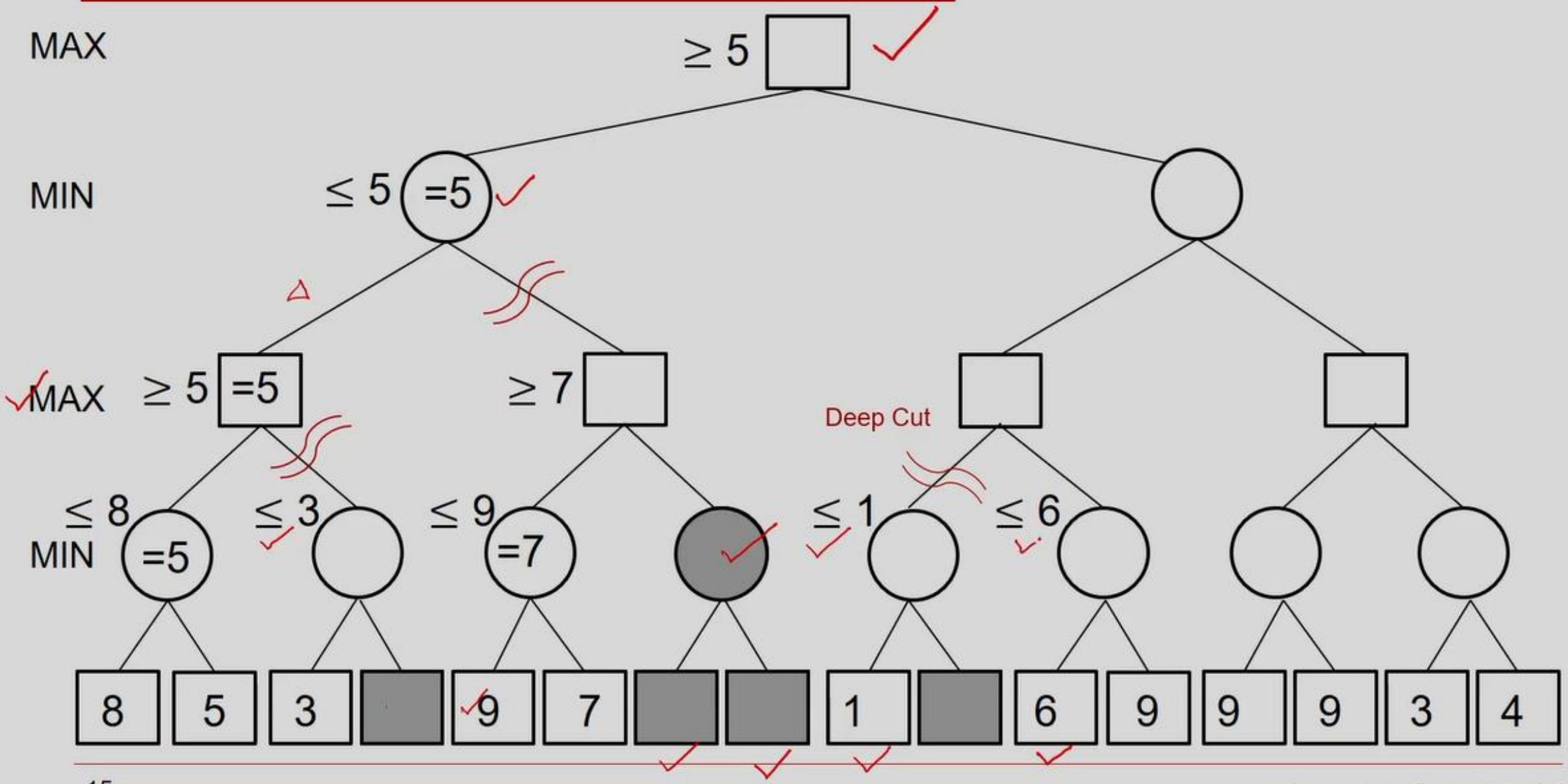
Alpha-beta Pruning



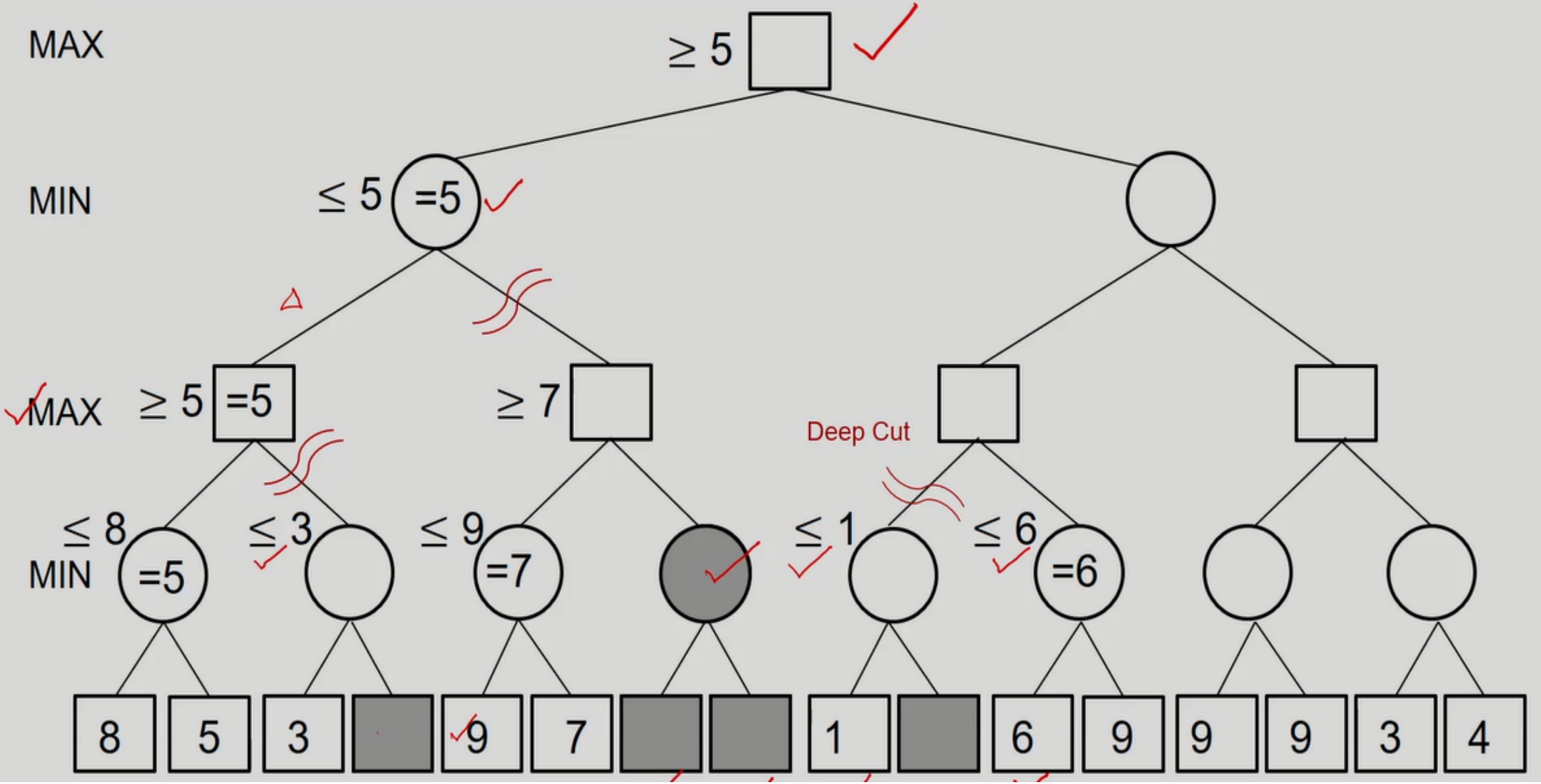
Alpha-beta Pruning



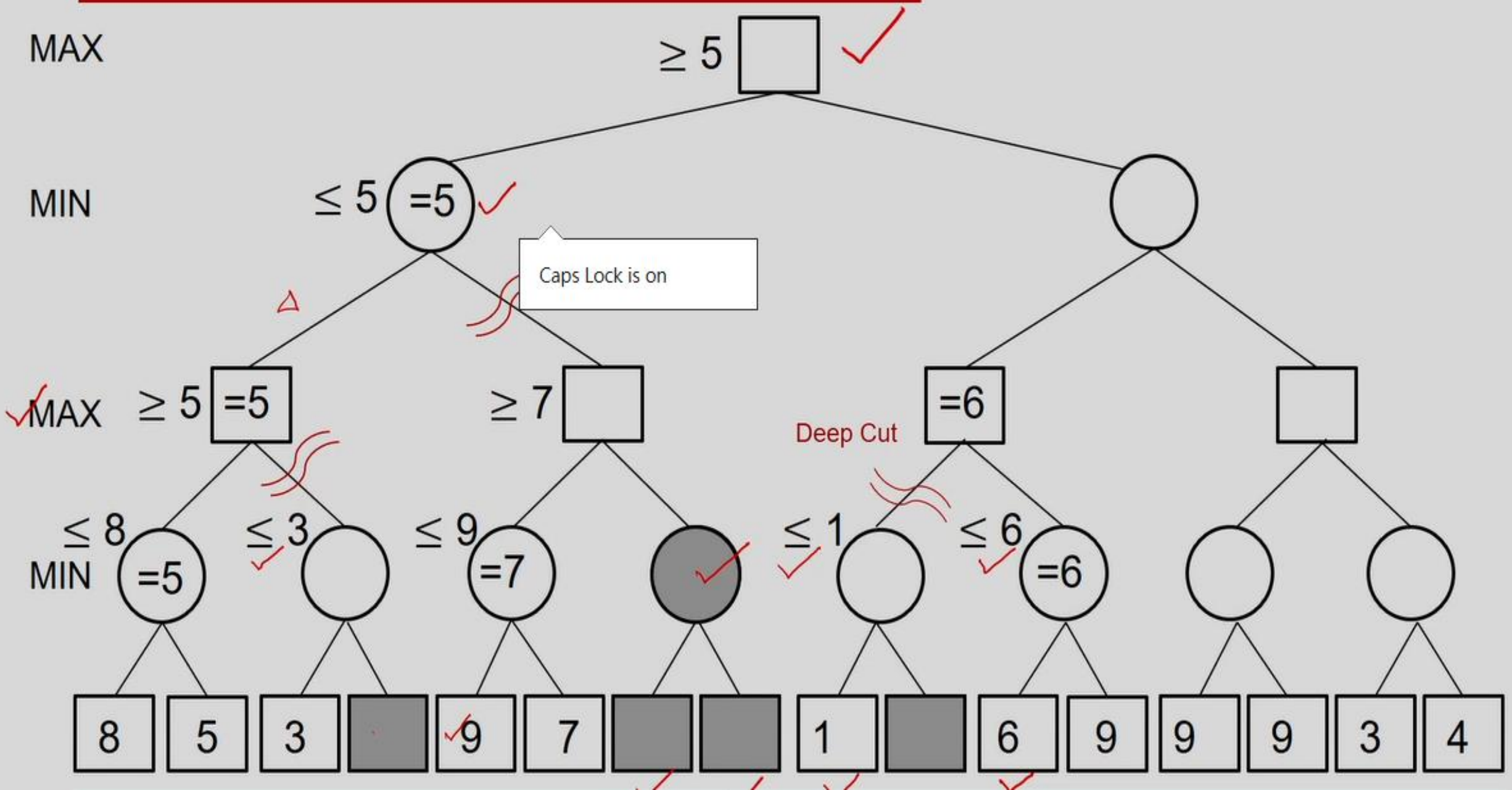
Alpha-beta Pruning



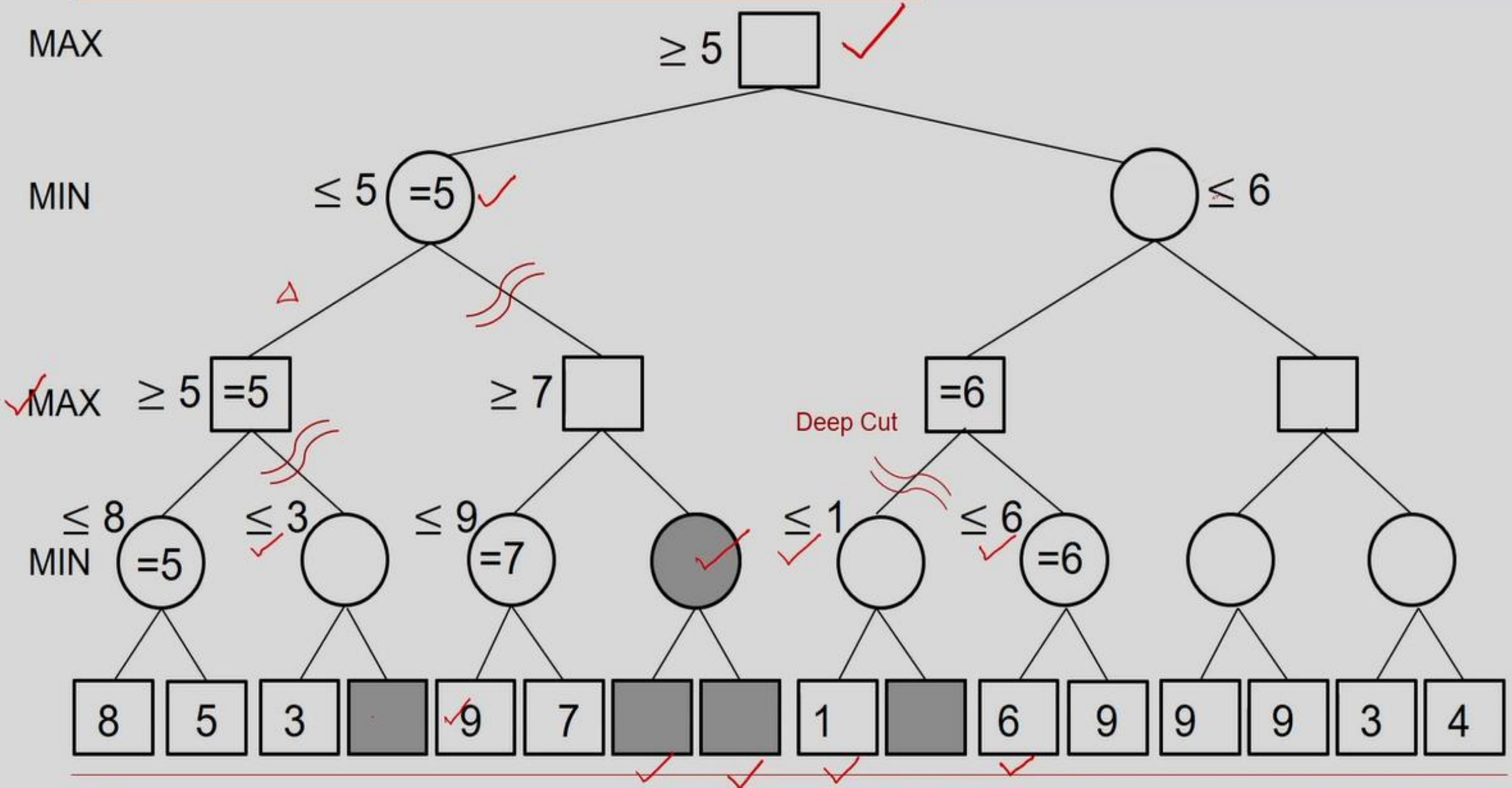
Alpha-beta Pruning



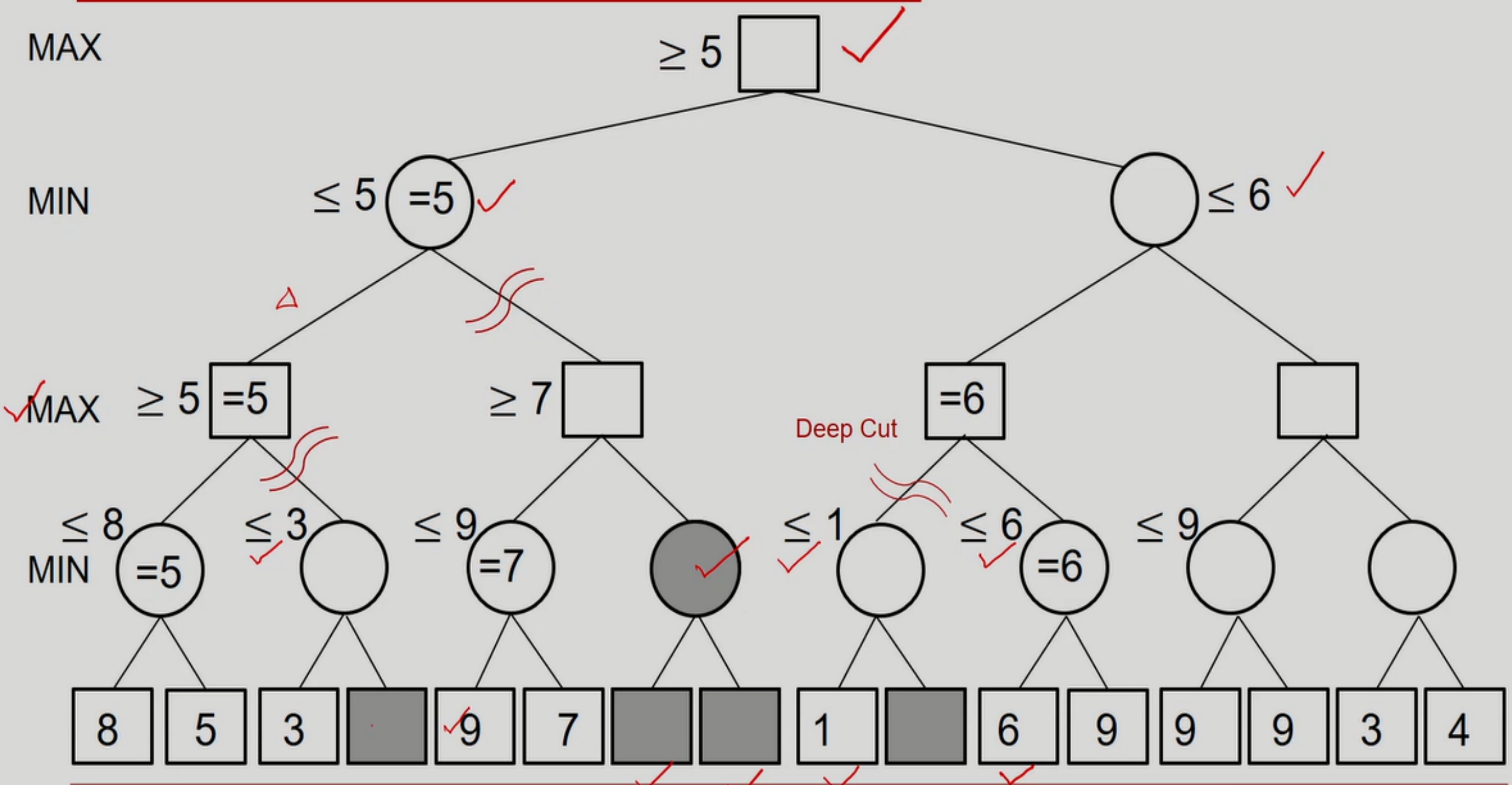
Alpha-beta Pruning



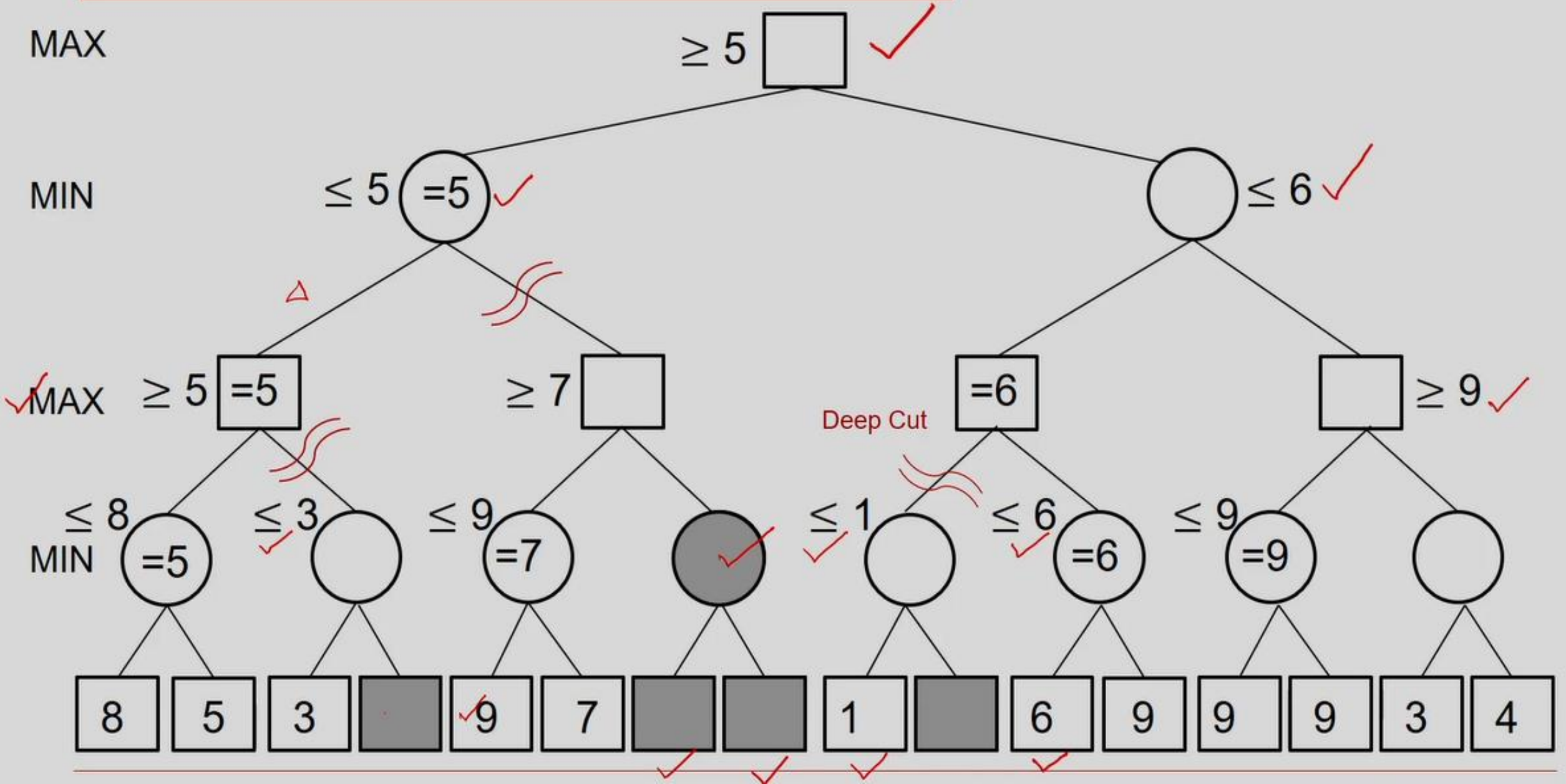
Alpha-beta Pruning



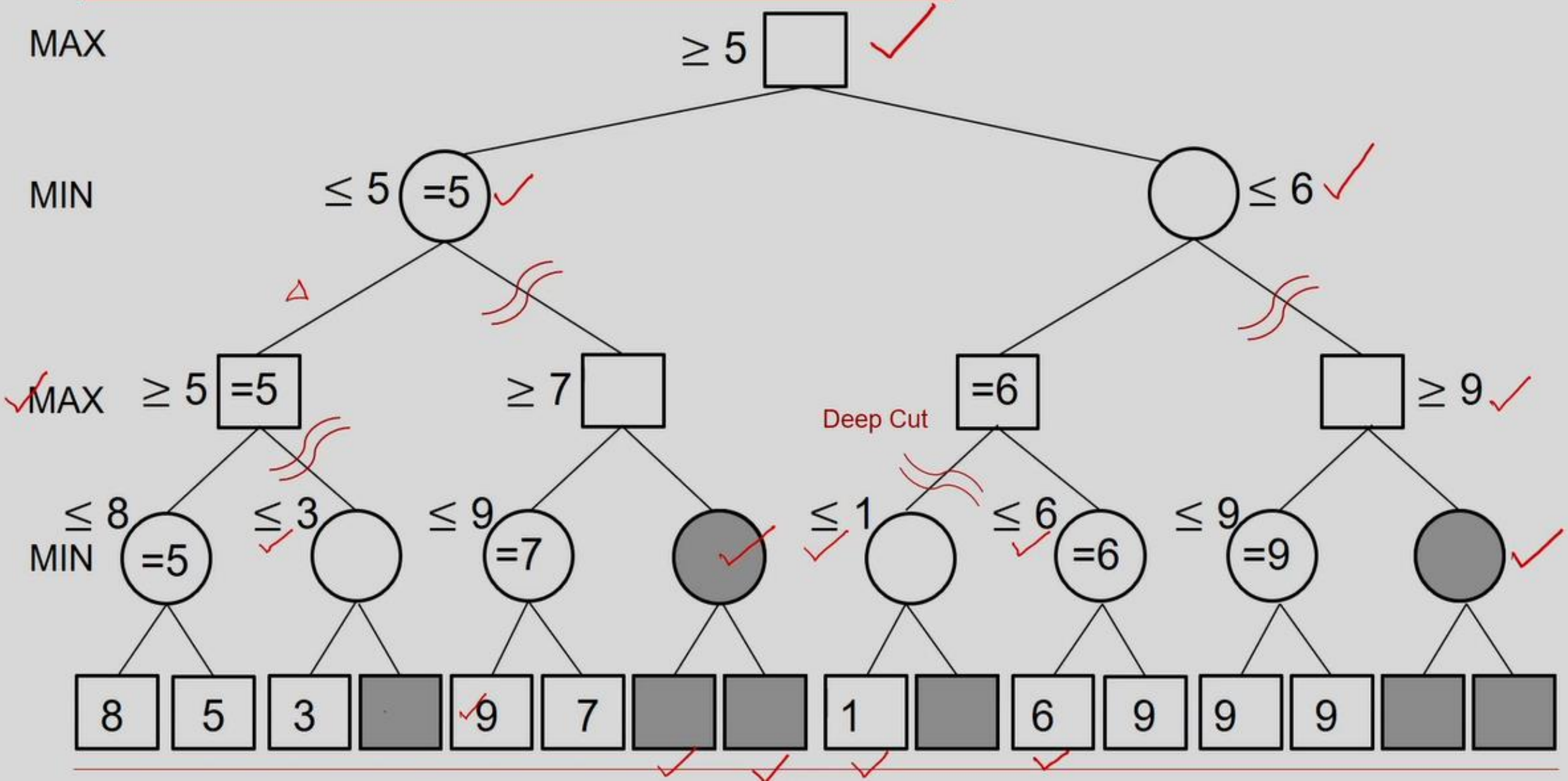
Alpha-beta Pruning



Alpha-beta Pruning



Alpha-beta Pruning

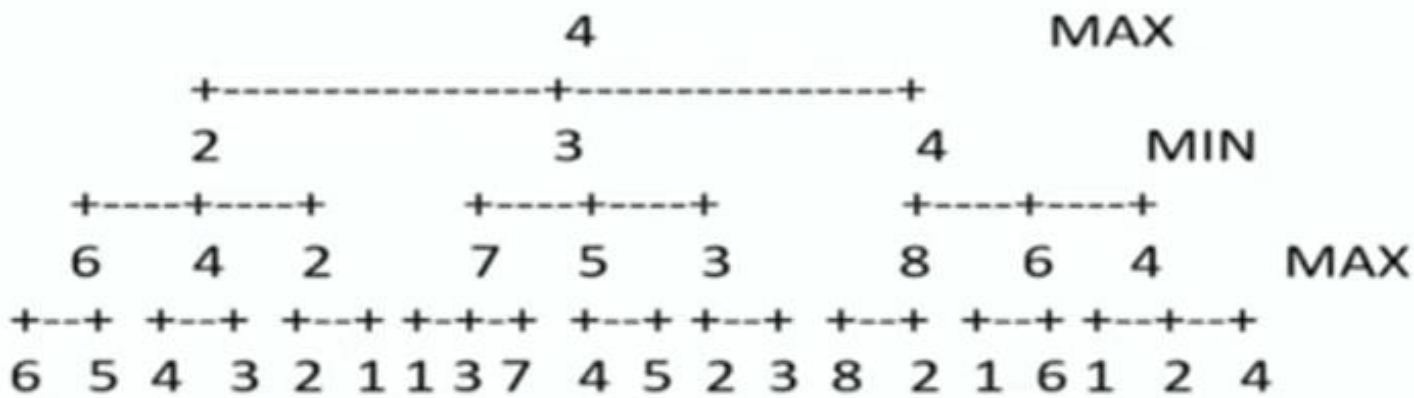


Algorithm for alpha beta pruning

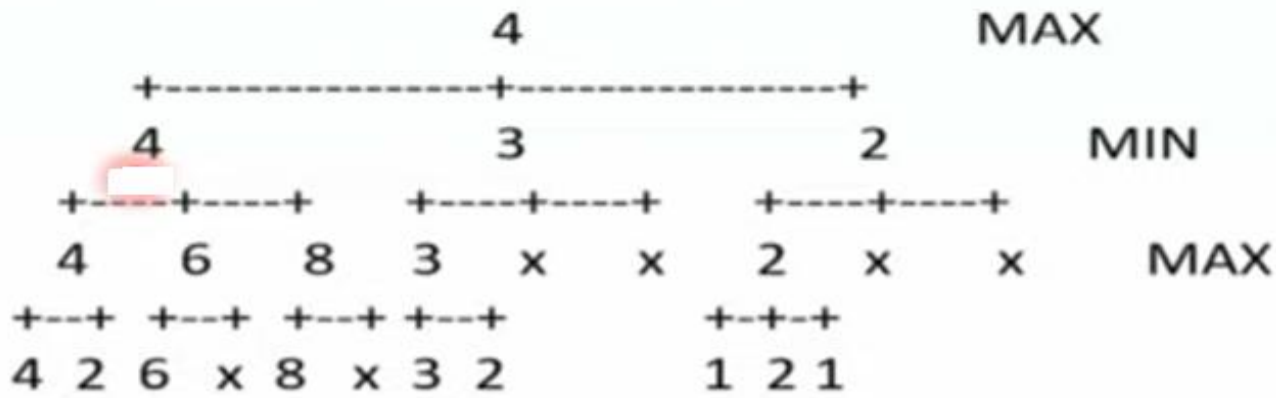
```
evaluate (node, alpha, beta)
        if node is a leaf
         return the heuristic value of node

    if node is a minimizing node
                    for each child of node
                    beta = min (beta, evaluate (child, alpha, beta))
                   if beta <= alpha
                    return beta
     return beta

      if node is a maximizing node
                     for each child of node
                   alpha = max (alpha, evaluate (child, alpha, beta))
                    if beta <= alpha
                    return alpha
     return alpha
```

Bad and Good Cases for Alpha-Beta Pruning

- Bad: Worst moves encountered first



- Good: Good moves ordered first



- If we can order moves, we can get more benefit

Let T(m) be time complexity of search for depth m

Normally:

$T(m) = b.T(m-1) + c \rightarrow T(m) = O(b^m)$

With ideal α-β pruning:

$T(m) = T(m-1) + (b-1)T(m-2) + c \rightarrow T(m) = O(b^{m/2})$

Properties of α-β

- Pruning does not affect final result. This means that it gets the exact same result as does full minimax.
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$

Constraint satisfaction problems (CSPs)

Standard search problem:

state is a "black box"—any old data structure
that supports goal test, eval, successor

CSP:

state is defined by variables X_i with values from domain D_i

goal test is a set of constraints specifying
allowable combinations of values for subsets of variables

Simple example of a **formal representation language**

Allows useful **general-purpose** algorithms with more power
than standard search algorithms

Example: Map-Coloring

Variables WA, NT, Q, NSW, V, SA, T

Domains $D_i = \{red, green, blue\}$

Constraints: adjacent regions must have different colors

$WA \neq NT$ (if the language allows this), or

$(WA, NT) \in \{(red, green), (red, blue), (green, red), (green, blue), \ldots\}$

Example: Map-Coloring contd.

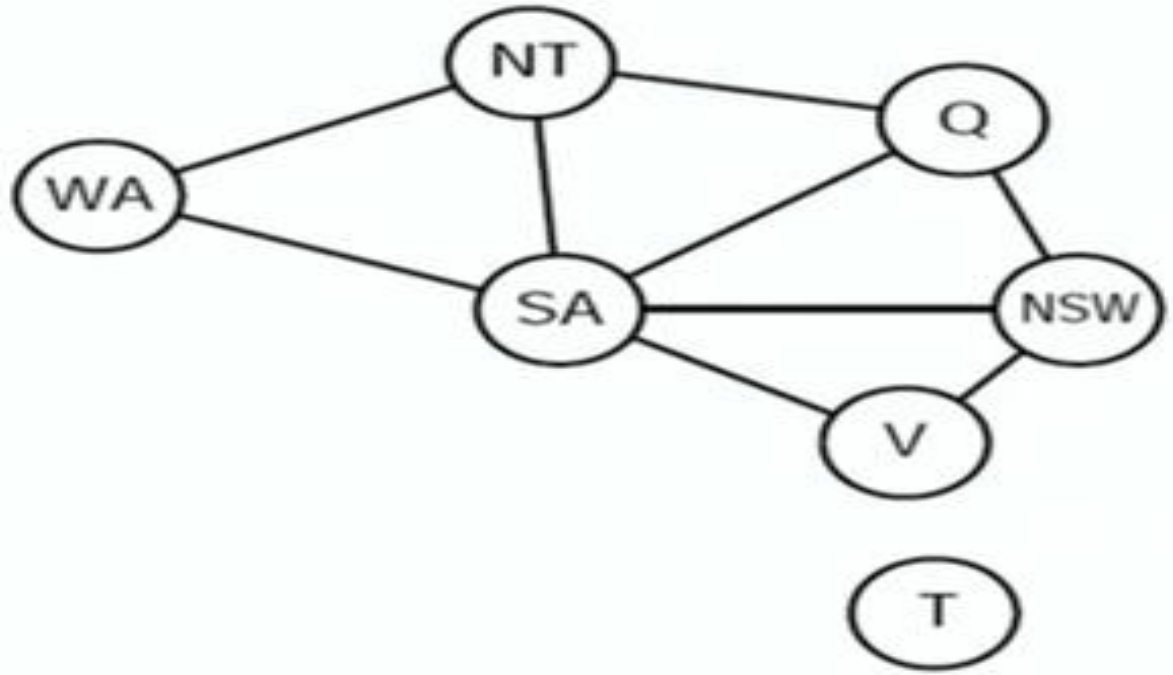
Solutions are assignments satisfying all constraints, e.g.,

$\{ \quad \mathbf{WA} = red, NT = green, Q = red, NSW = green, V = red, SA = blue$

Constraint graph

Binary CSP: each constraint relates at most two variables

Constraint graph: nodes are variables, arcs show constraints



Tasmania is an independent subproblem!

Varieties of CSPs

Discrete variables

- finite domains; size $d \Rightarrow O(d^n)$ complete assignments
  - ◊ e.g., Boolean CSPs, incl. Boolean satisfiability (NP-complete)
- infinite domains (integers, strings, etc.)
  - ◊ e.g., job scheduling, variables are start/end days for each job
  - ◊ need a constraint language, e.g., $StartJob_1 + 5 \leq StartJob_3$
  - ◊ linear constraints solvable, nonlinear undecidable

Continuous variables

- ◊ e.g., start/end times for Hubble Telescope observations
- ◊ linear constraints solvable in poly time by LP methods

Varieties of constraints

Unary constraints involve a single variable,

e.g., $SA \neq green$

Binary constraints involve pairs of variables,

e.g., $SA \neq WA$

Higher-order constraints involve 3 or more variables,

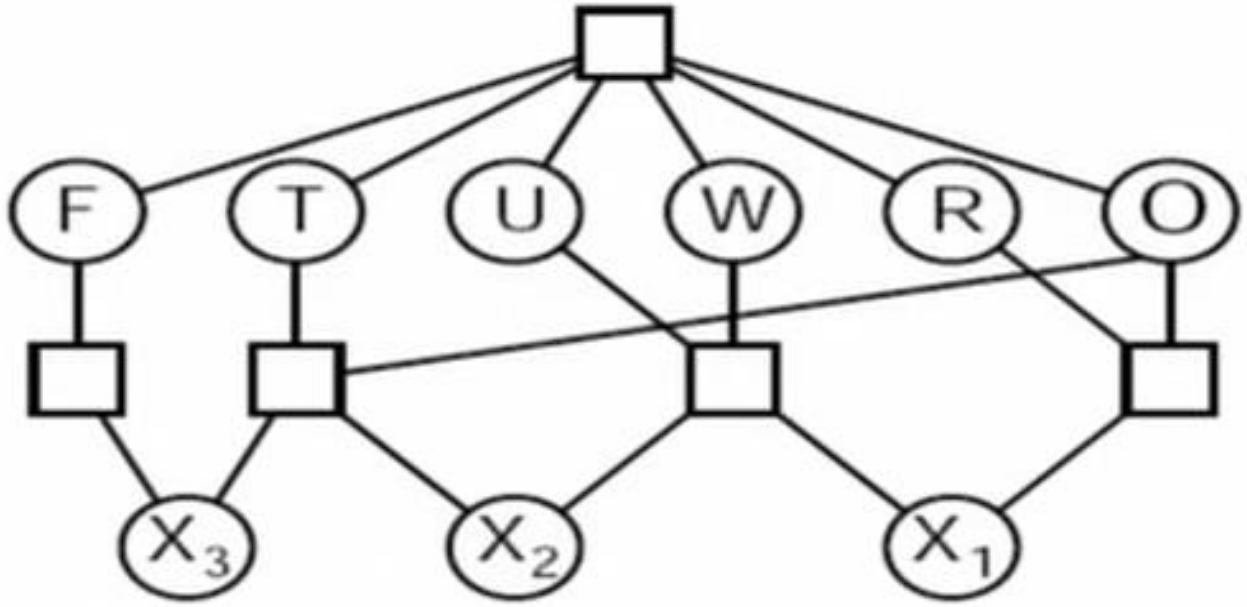
e.g., cryptarithmetic column constraints

Preferences (soft constraints), e.g., red is better than $green$
often representable by a cost for each variable assignment

$\rightarrow$ constrained optimization problems

Example: Cryptarithmetic

$$\begin{array}{r} \ T\ W\ O \\ +\ T\ W\ O \\ \hline F\ O\ U\ R \end{array}$$



Variables: $F\ T\ U\ W\ R\ O\ X_1\ X_2\ X_3$

Domains: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$

Constraints

alldiff(F, T, U, W, R, O)

$O + O = R + 10 \cdot X_1$, etc.