GAME PLAYING AS SEARCH

α-β pruning

TYPES OF GAMES

	deterministic	chance
perfect information	chess, checkers, go, othello	backgammon monopoly
imperfect information	battleship Kriegspiel	bridge, poker, scrabble nuclear war

TYPICAL ASSUMPTIONS

Two agents whose actions alternate

Utility values for each agent are the opposite of the other
- This creates the adversarial situation

Fully observable environments

In game theory terms:
- "Deterministic, turn-taking, zero-sum games of perfect information"

Generalizes to stochastic games, multiple players, non zero-sum, etc.

Compare to, e.g., "Prisoner's Dilemma"

"Deterministic, NON-turn-taking, NON-zero-sum game of imperfect information"

SEARCH VERSUS GAMES

Search – no adversary

- Solution is (heuristic) method for finding goal
- Heuristics and CSP techniques can find *optimal* solution
- Evaluation function: estimate of cost from start to goal through given node
- Examples: path planning, scheduling activities

Games – adversary

- Solution is strategy
  - strategy specifies move for every possible opponent reply.
- Time limits force an *approximate* solution
- Evaluation function: evaluate “goodness” of game position
- Examples: chess, checkers, Othello, backgammon

AN OPTIMAL PROCEDURE: THE MIN-MAX METHOD

Designed to find the optimal strategy for Max and find best move:

1. Generate the whole game tree, down to the leaves.
2. Apply utility (payoff) function to each leaf.
3. Back-up values from leaves through branch nodes:
   - a Max node computes the Max of its child values
   - a Min node computes the Min of its child values
4. At root: choose the move leading to the child of highest value.

GAME TREES

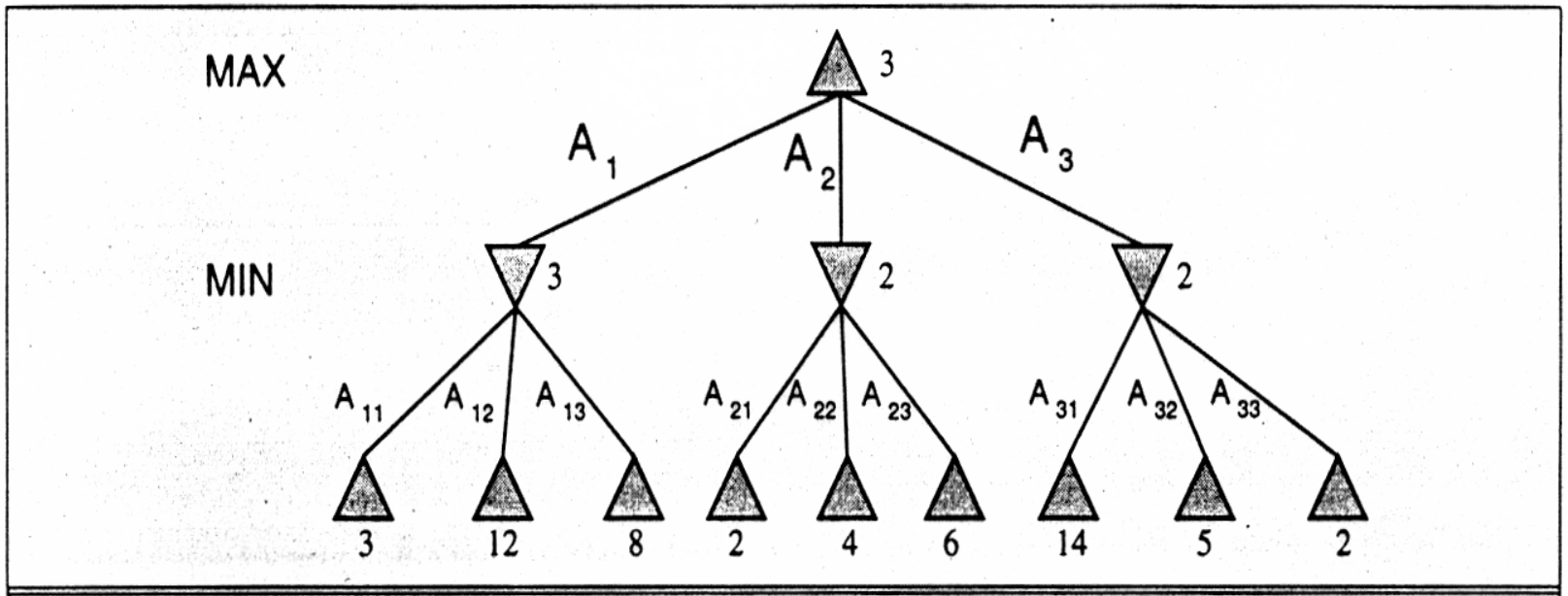


Figure 5.2 A two-ply game tree as generated by the minimax algorithm. The △ nodes are moves by MAX and the ▽ nodes are moves by MIN. The terminal nodes show the utility value for MAX computed by the utility function (i.e., by the rules of the game), whereas the utilities of the other nodes are computed by the minimax algorithm from the utilities of their successors. MAX's best move is A_1, and MIN's best reply is A_{11}.

PROPERTIES OF MINIMAX

Complete?

- Yes (if tree is finite).

Optimal?

- Yes (against an optimal opponent).
- Can it be beaten by an opponent playing sub−optimally?
  - No. (Why not?)

Time complexity?

- $O(b^m)$

Space complexity?

- O(bm) (depth−first search, generate all actions at once)
- O(m) (backtracking search, generate actions one at a time)

GAME TREE SIZE CHALLENGE

Tic–Tac–Toe

- b ≈ 5 legal actions per state on average, total of 9 plies in game.
  - "ply" = one action by one player, "move" = two plies.
- $5^9 = 1{,}953{,}125$
- $9! = 362{,}880$ (Computer goes first)
- $8! = 40{,}320$ (Computer goes second)

→ **exact solution quite reasonable**

Chess

- b ≈ 35 (approximate average branching factor)
- d ≈ 100 (depth of game tree for "typical" game)
- $b^d \approx 35^{100} \approx 10^{154}$ nodes!!

→ **exact solution completely infeasible**

It is usually impossible to develop the whole search tree.

OBSERVATION

- Some branches will never be played by rational players since they include sub-optimal decisions (for either player)

ALPHA-BETA PRUNING
EXPLOITING THE FACT OF AN ADVERSARY

If a position is provably bad:

- It is NO USE expending search time to find out exactly how bad

If the adversary can force a bad position:

- It is NO USE expending search time to find out the good positions that the adversary won't let you achieve anyway

Bad = not better than we already know we can achieve elsewhere.

Contrast normal search:

- ANY node might be a winner.
- ALL nodes must be considered.
- (A* avoids this through knowledge, i.e., heuristics)

ALPHA-BETA PRUNING

A way to reduce the number of nodes that need to be generated and evaluated

A branch and bound idea

Alpha Cutoff: when the value of a min position is less than or equal to the alpha−value of its parent, stop generating further successors

Beta Cutoff: when the value of a max position is greater than the beta−value of its parent, stop generating further successors.

ALPHA-BETA PRUNING

Main idea: Avoid processing subtrees that have no effect on the result

A way to reduce the number of nodes that need to be generated and evaluated

Two new parameters

- α: The best value for MAX seen so far
- β: The best value for MIN seen so far

α is used in MIN nodes, and is assigned in MAX nodes

β is used in MAX nodes, and is assigned in MIN nodes

IN OTHER WORDS

- Alpha: a ***lower bound*** on the value that a max node may ultimately be assigned

$$v > \alpha$$

- Beta: an ***upper bound*** on the value that a minimizing node may ultimately be assigned

$$v < \beta$$

- α–values at Max nodes never decrease.
- β –values at Min nodes never increase.

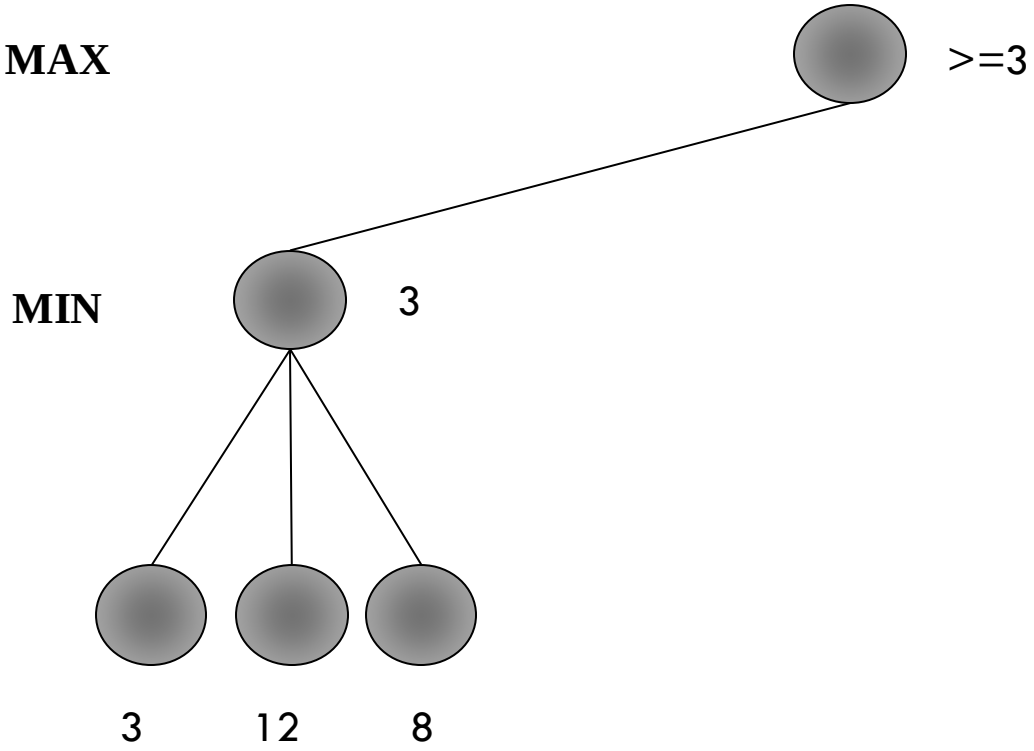
ALPHA BETA MINI-MAX

Keep track of α, β values and send on to children :

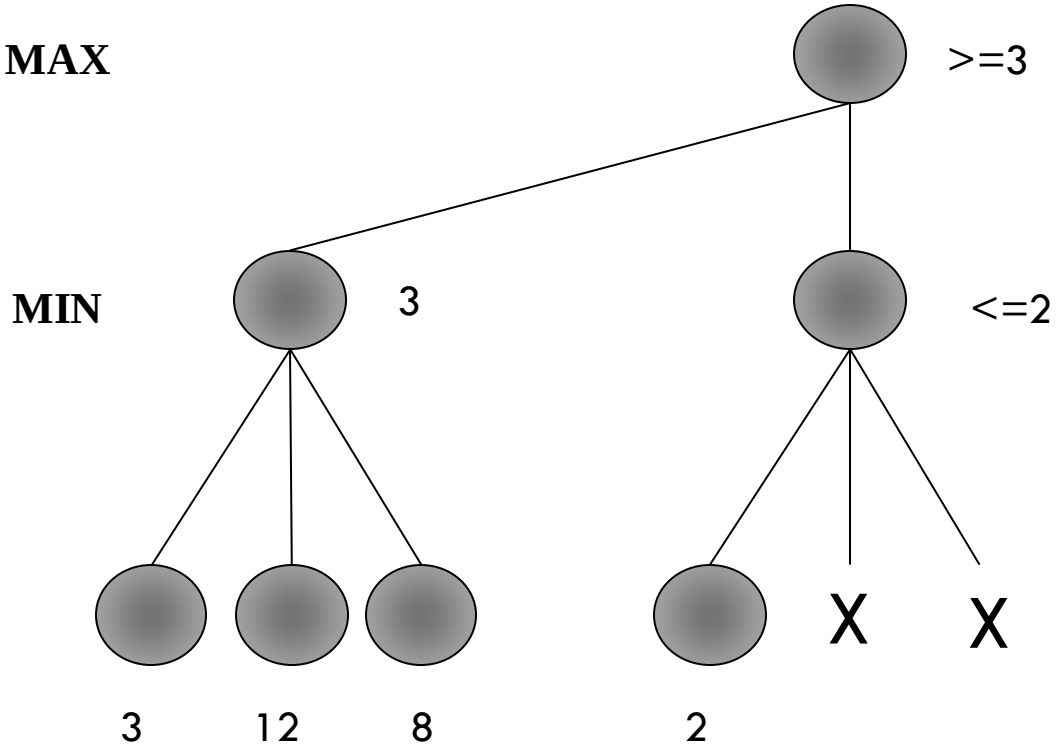
1) Search discontinued (loop breaks) below any Min node with $\beta <= \alpha$ of one of it's ancestors.
2) Set final value of node to be this β value.

3) Search discontinued (loop breaks) below any Max node with $\alpha >= \beta$ of one of it's ancestors.
4) Set final value of node to be this α value.

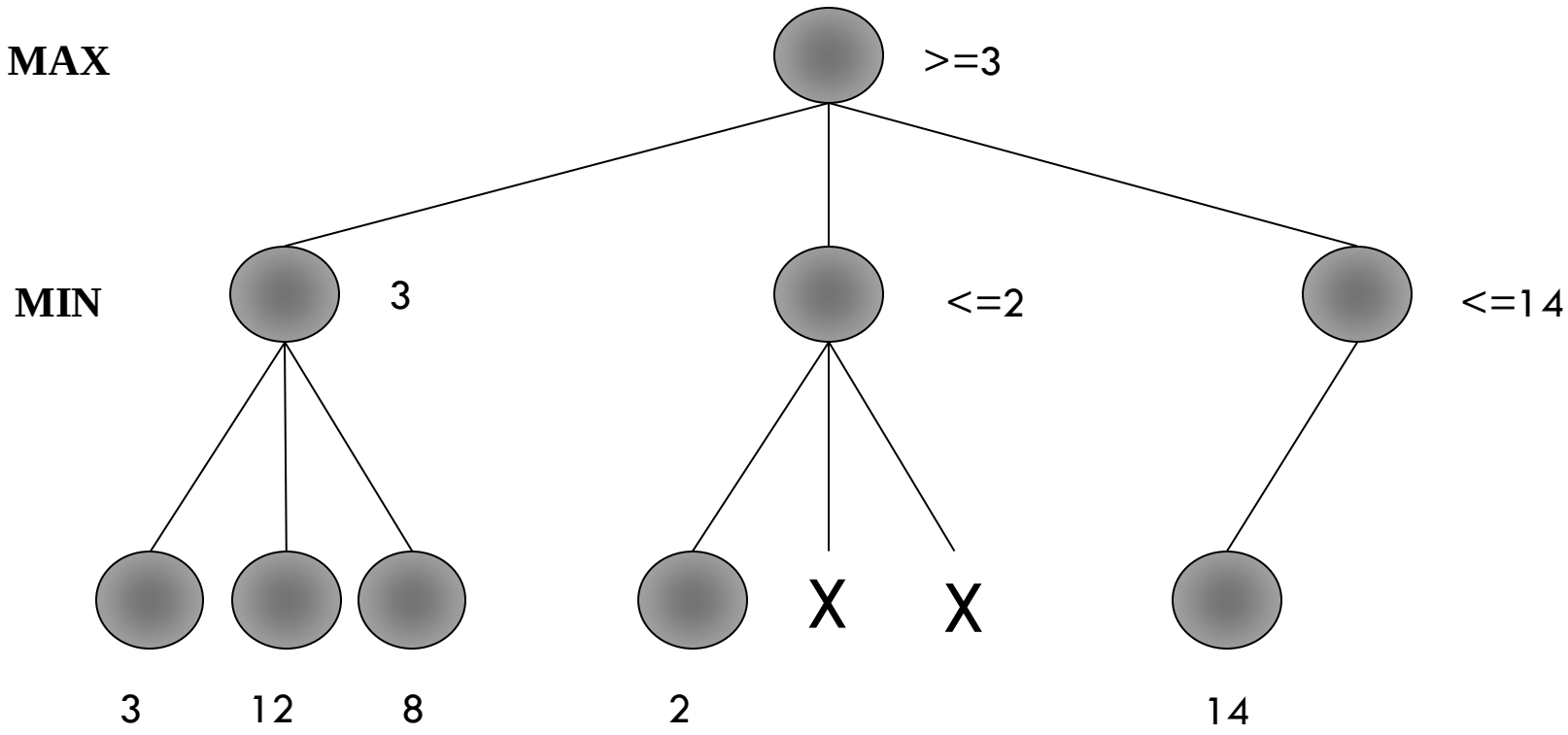
A-B PRUNING EXAMPLE



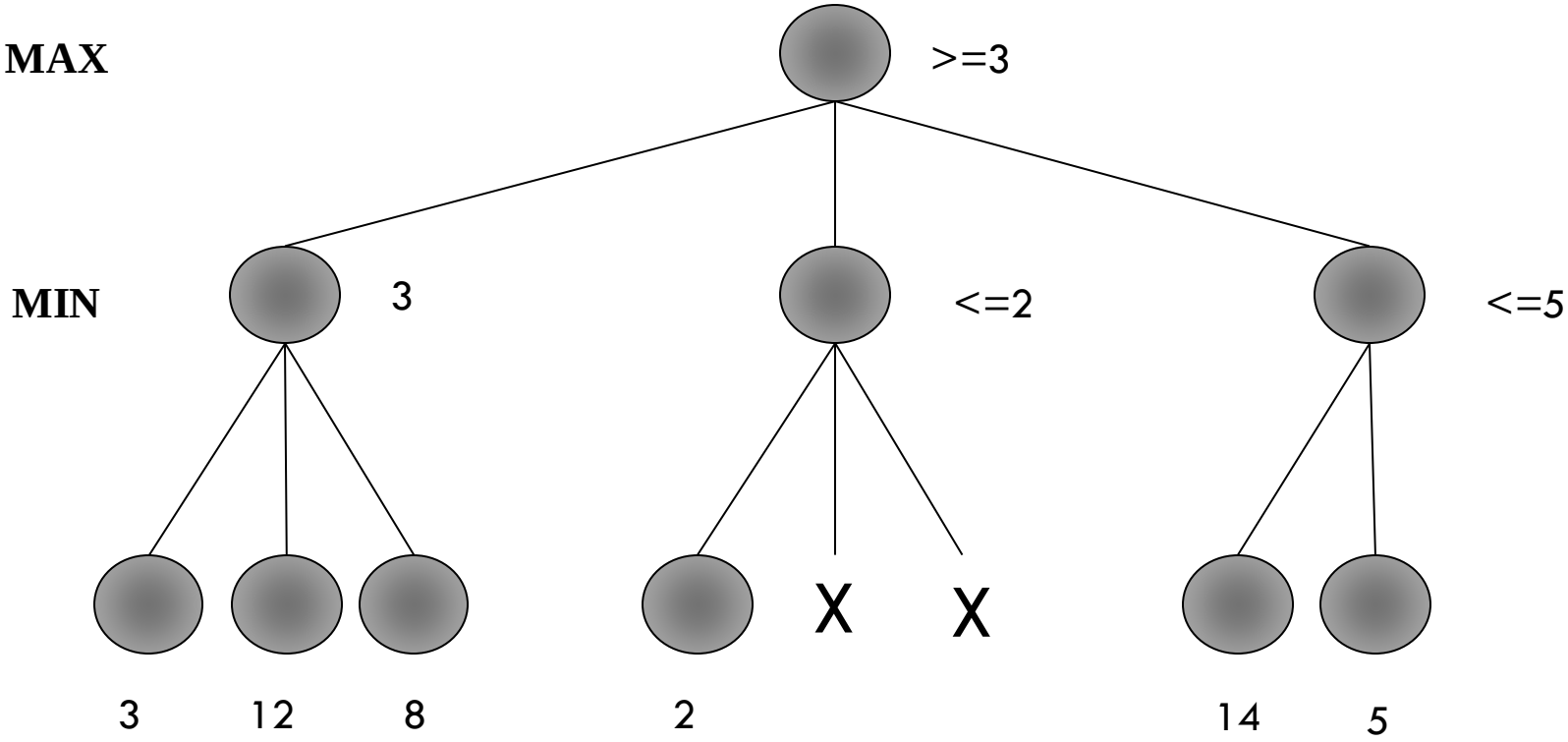
A-B PRUNING EXAMPLE



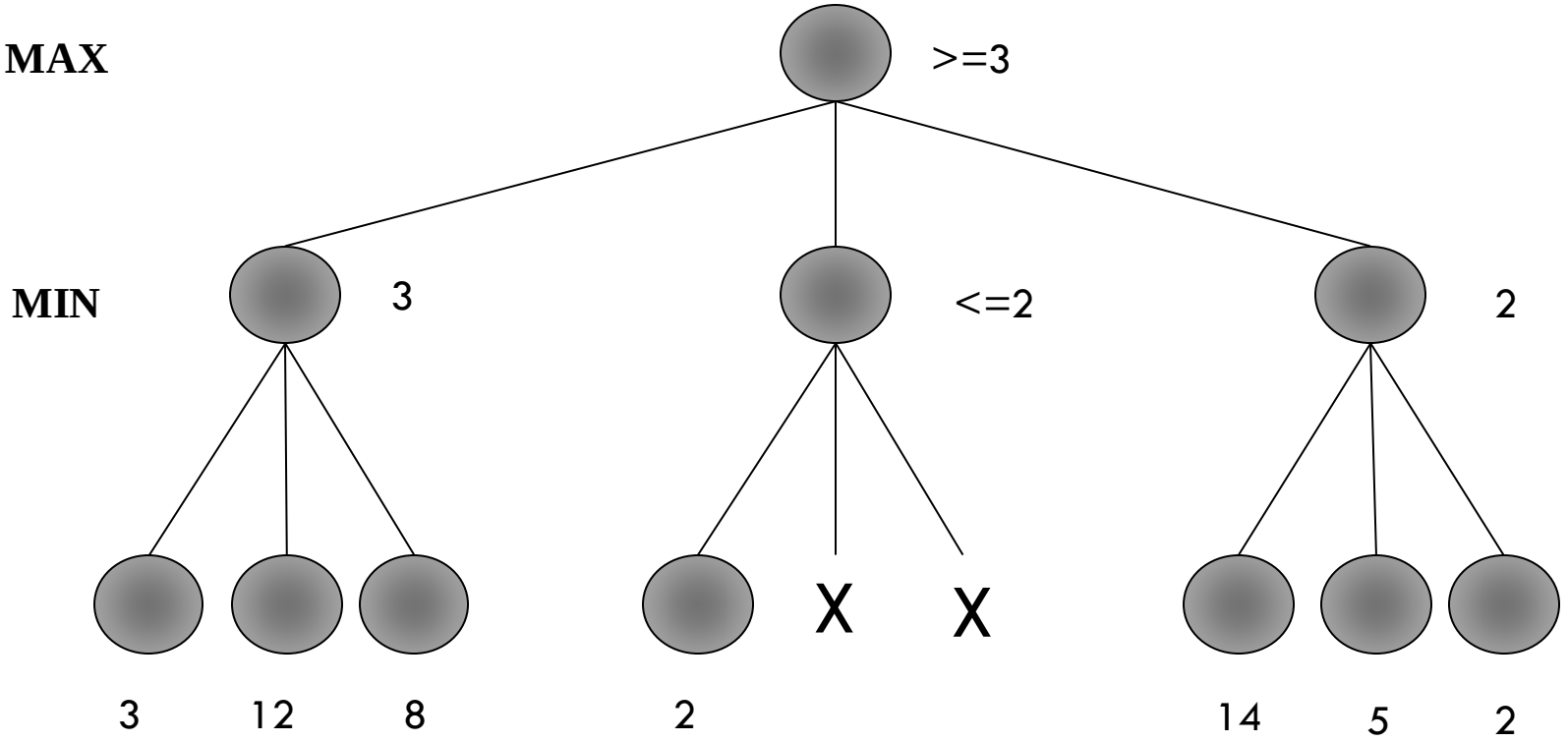
A-B PRUNING EXAMPLE

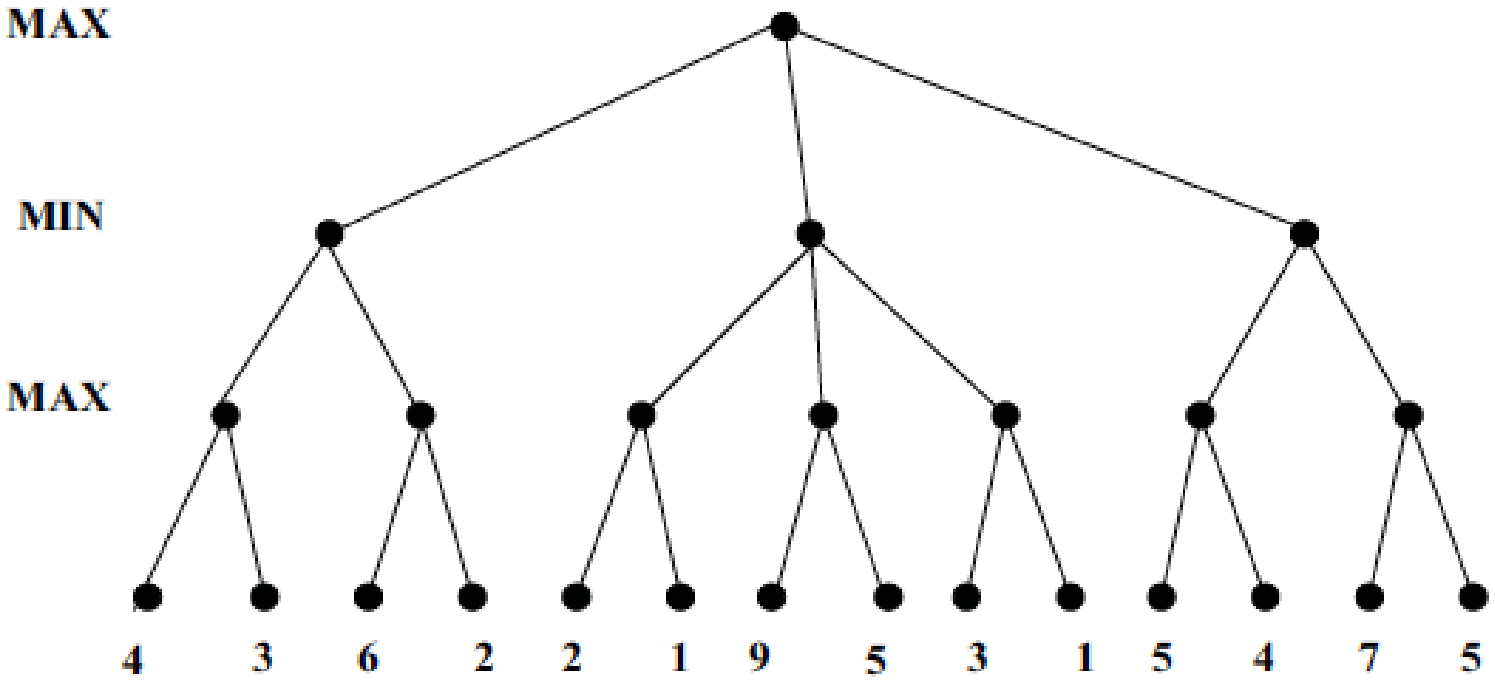


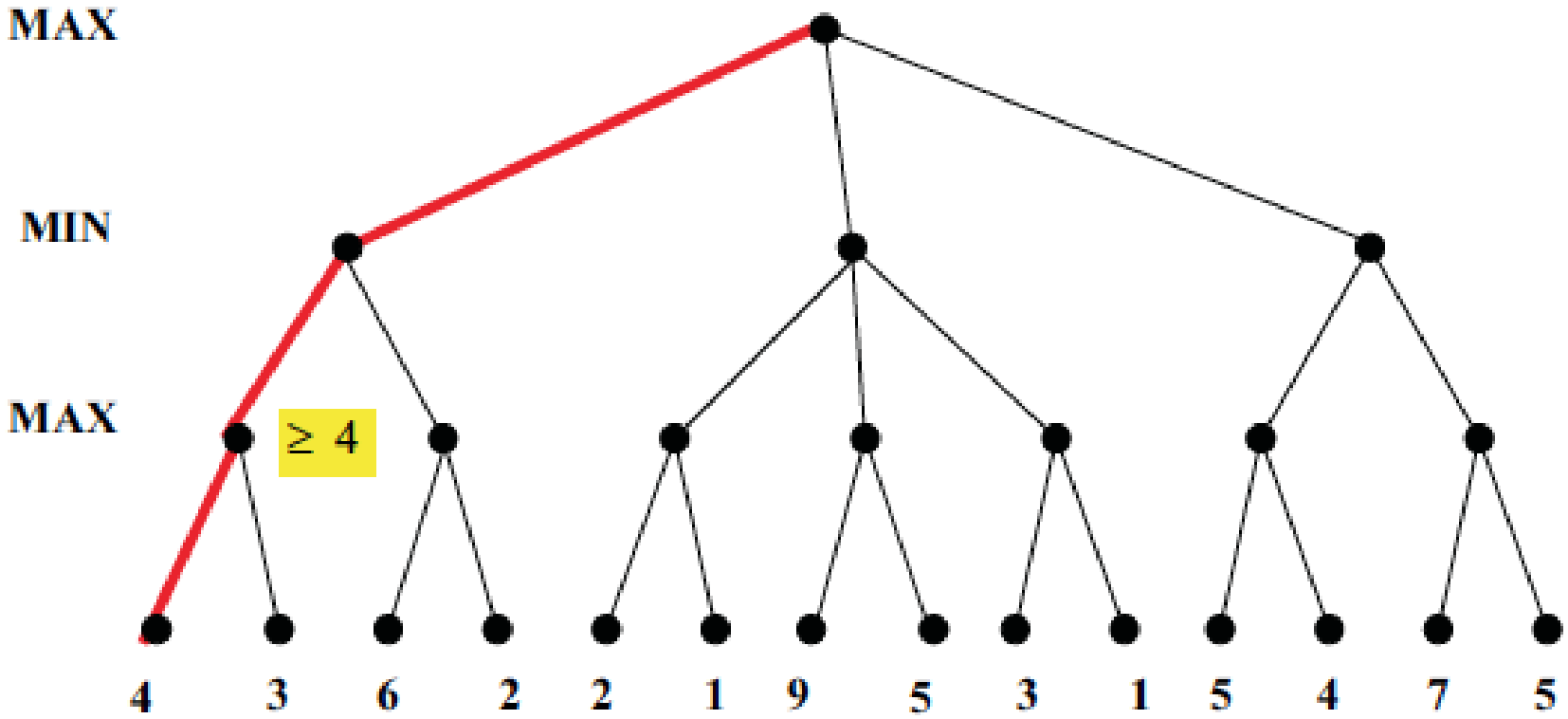
A-B PRUNING EXAMPLE

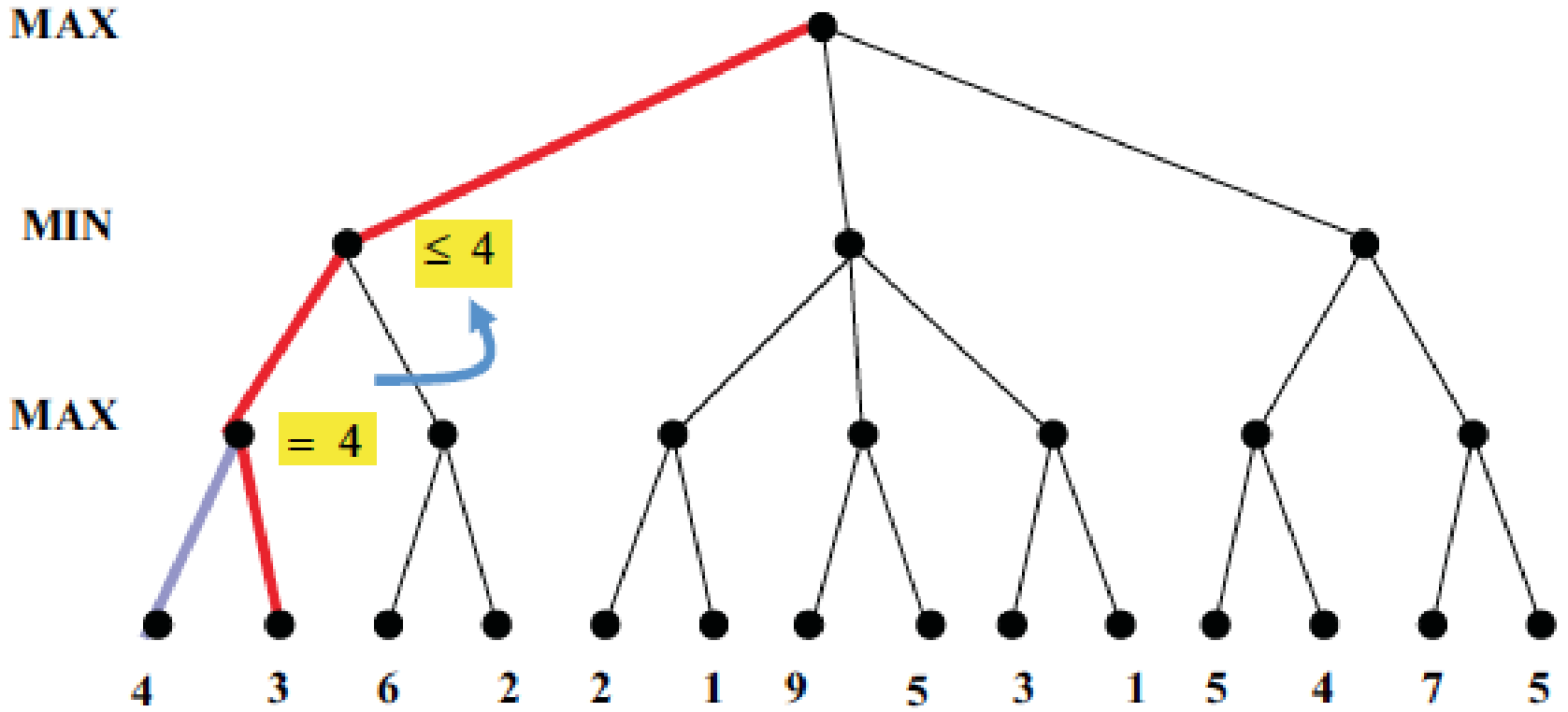


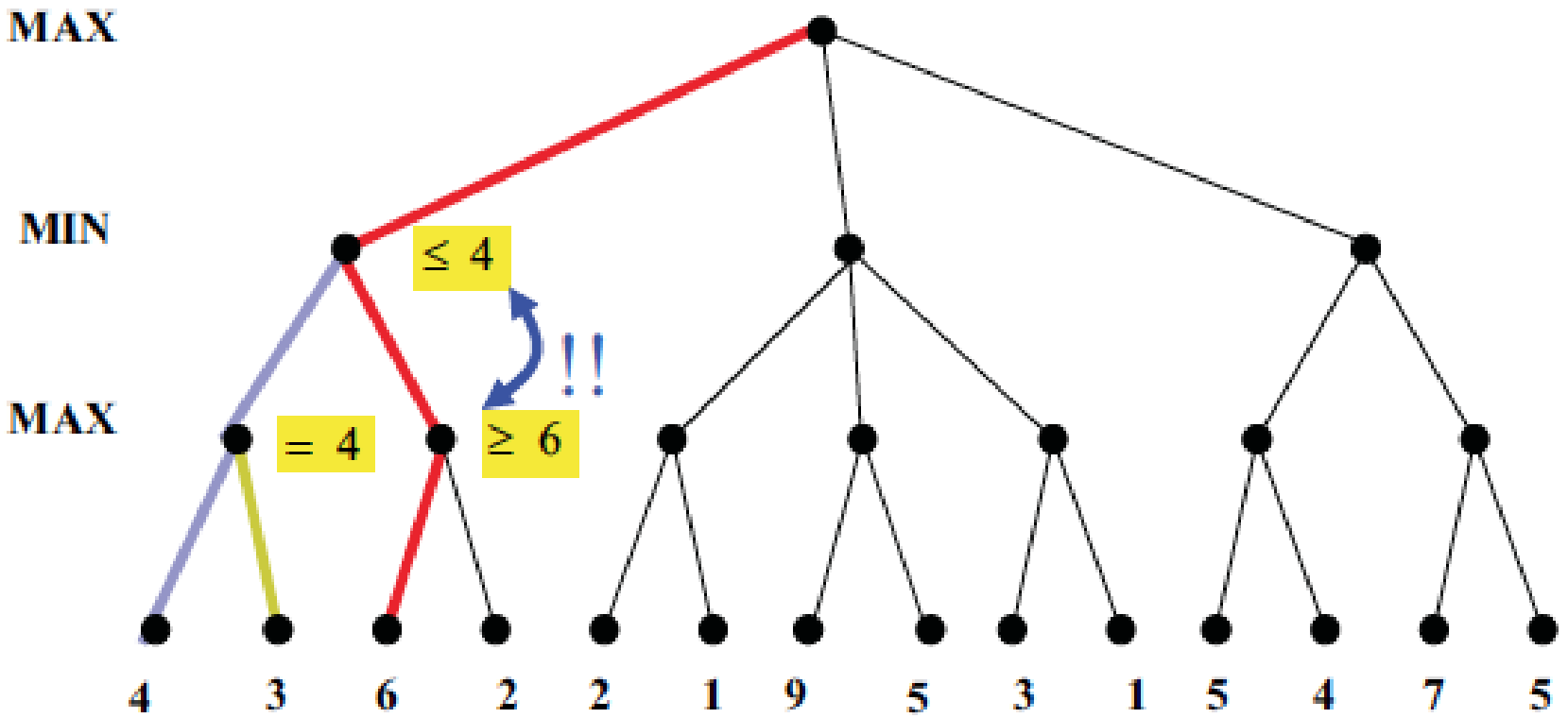
A-B PRUNING EXAMPLE

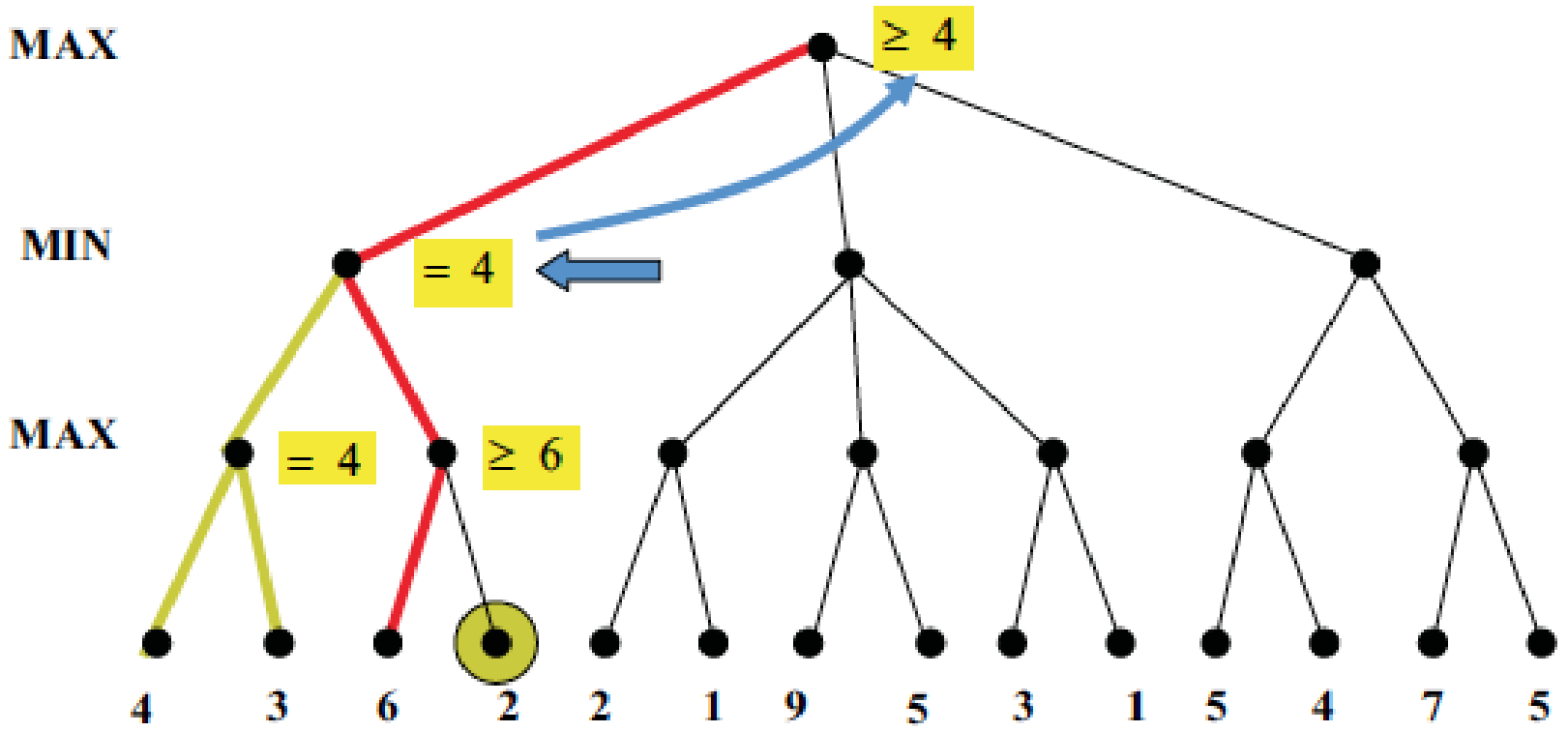


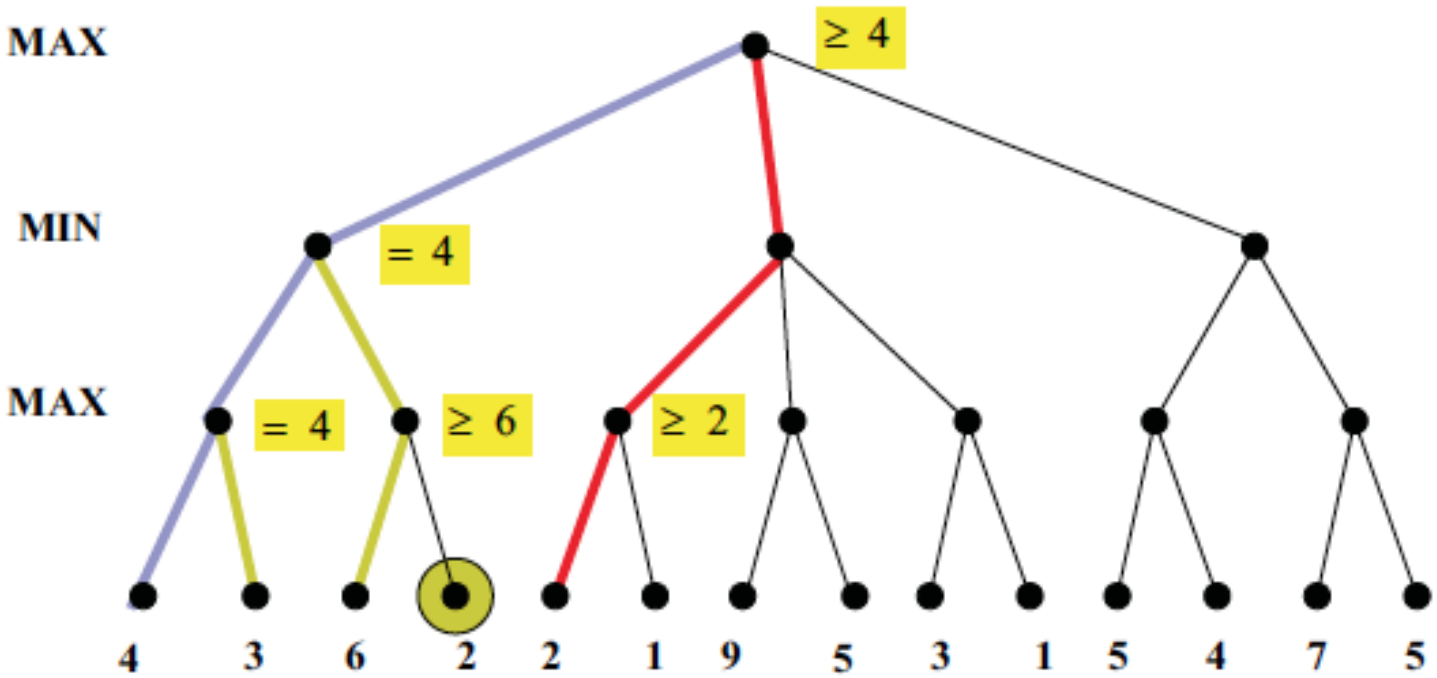

MAX
MIN
MAX
4
3
6
2
2
1
9
5
3
1
5
4
7
5

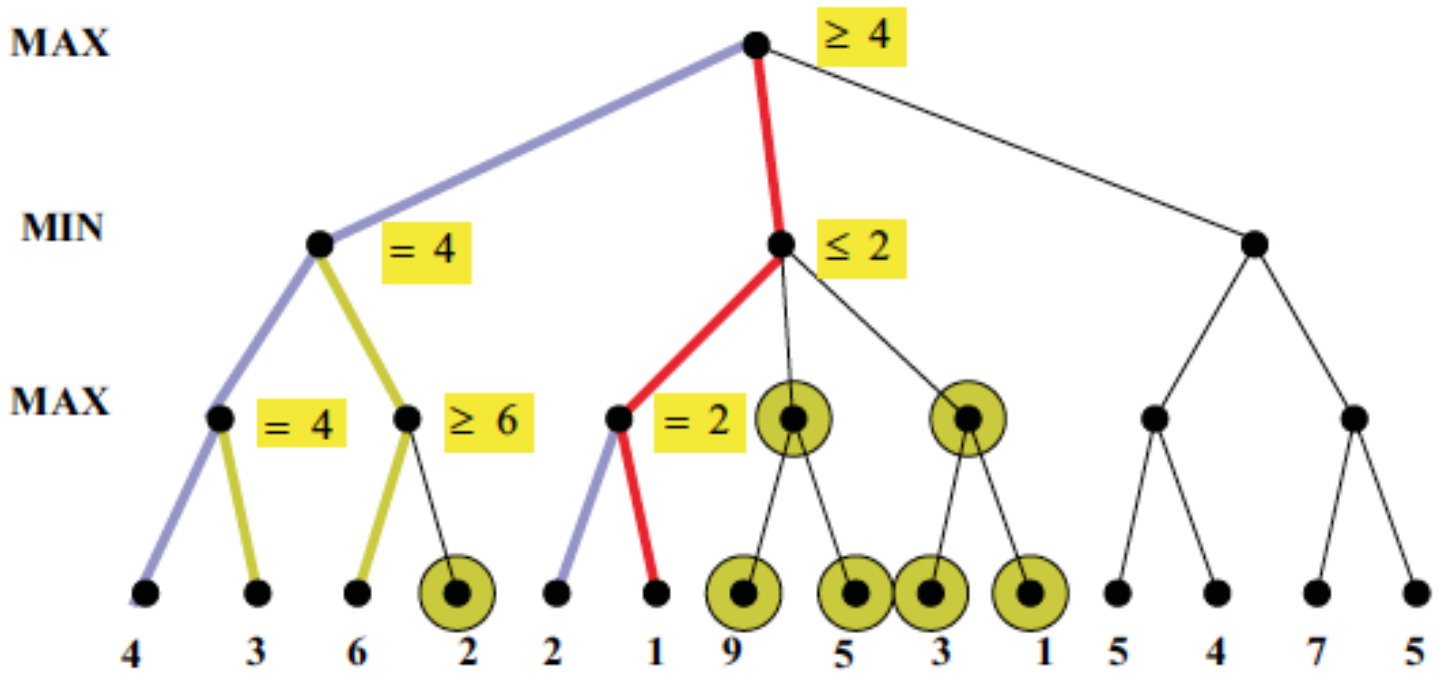

MAX
MIN
MAX
≥ 4
4 3 6 2 2 1 9 5 3 1 5 4 7 5

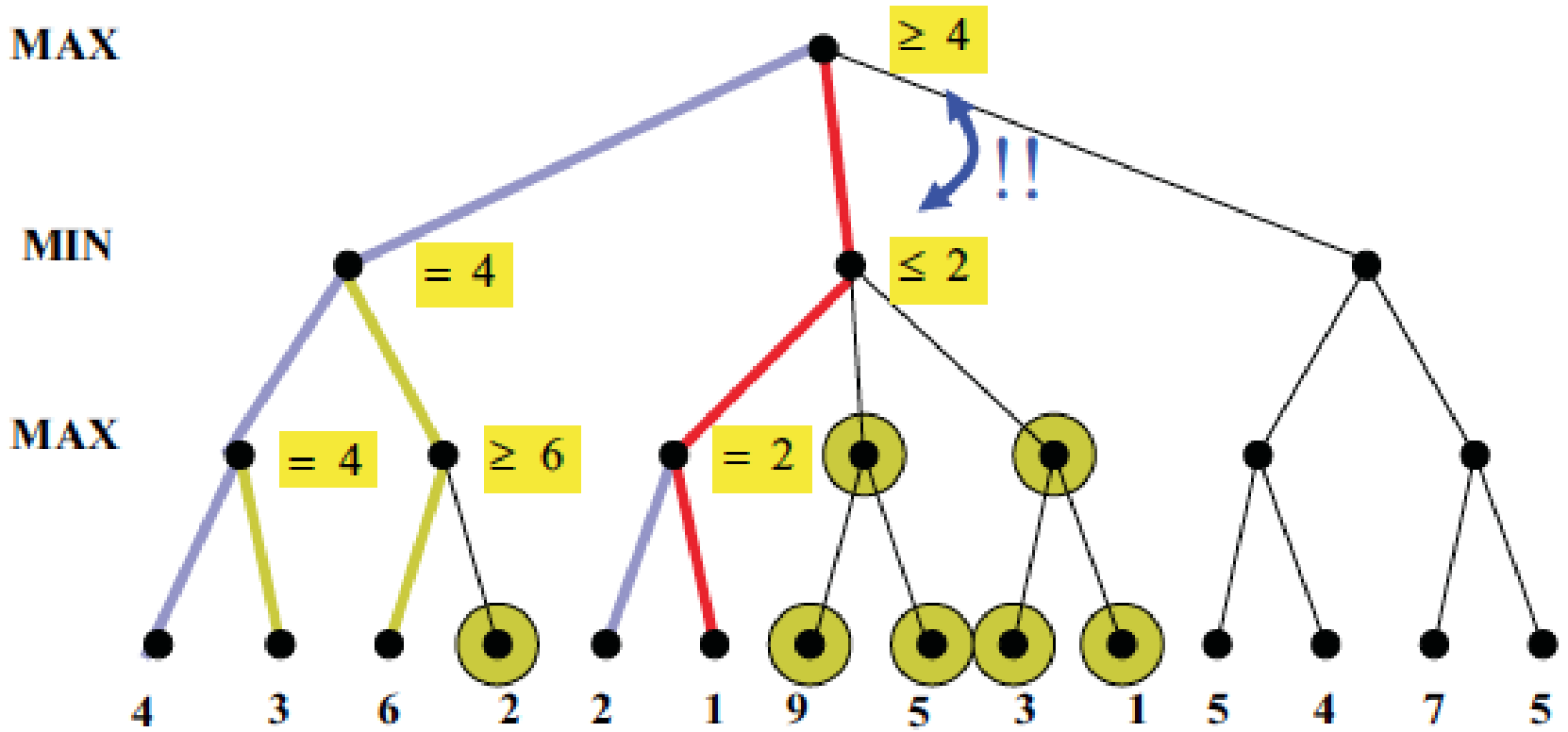

MAX
MIN
MAX
≤ 4
= 4
4 3 6 2 2 1 9 5 3 1 5 4 7 5

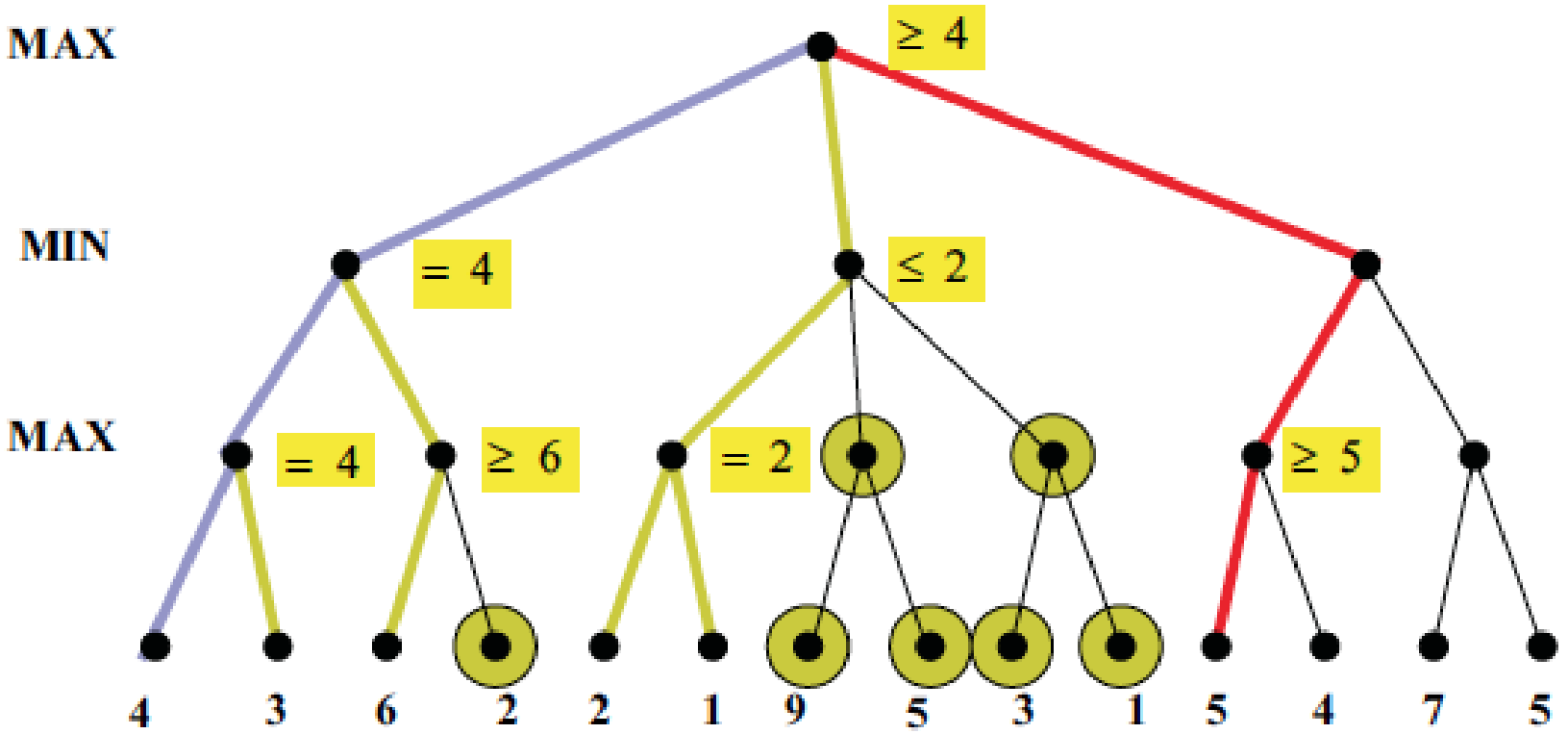

MAX
MIN
MAX
≤ 4
!!
= 4
≥ 6
4 3 6 2 2 1 9 5 3 1 5 4 7 5

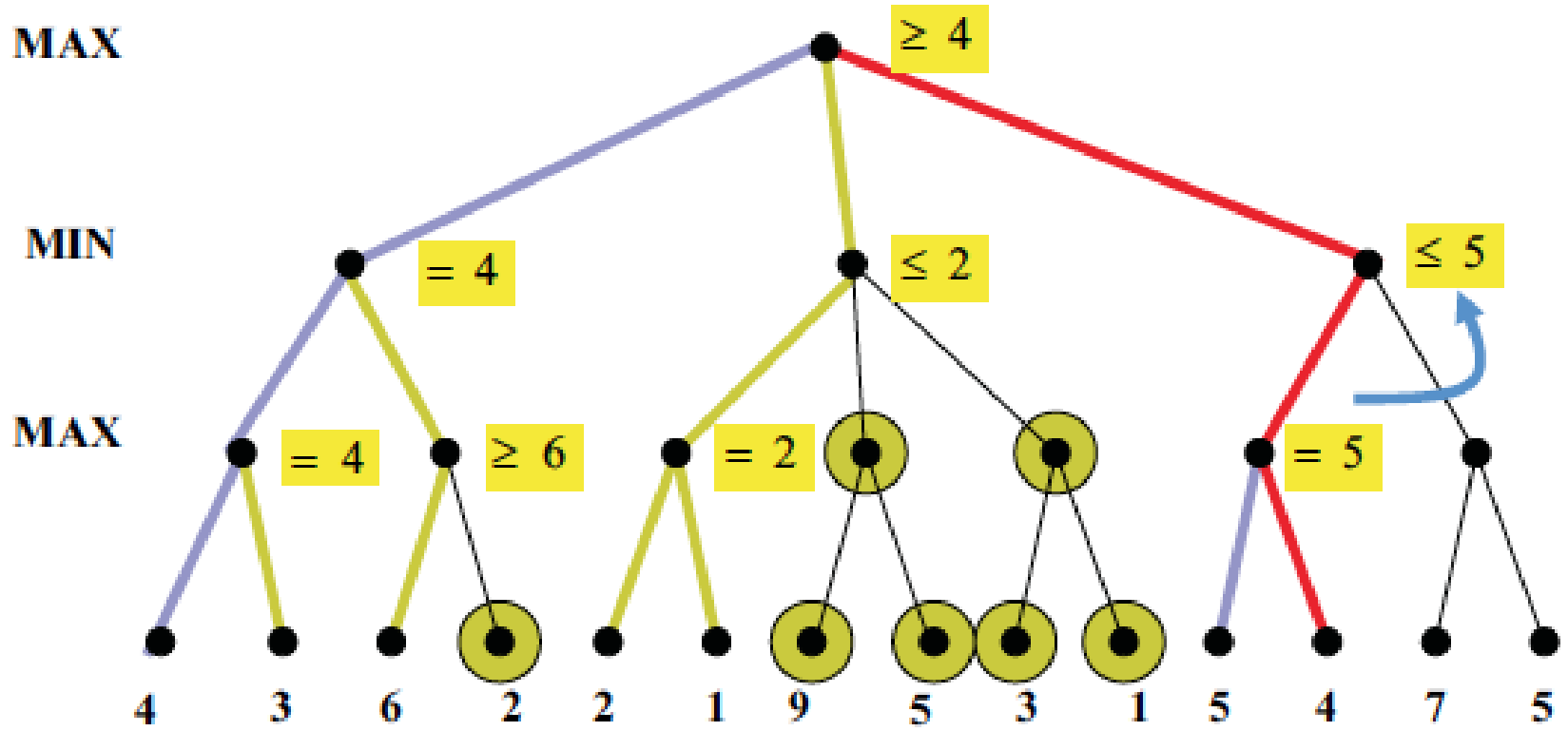

MAX
MIN
MAX
≥ 4
= 4
= 4
≥ 6
4 3 6 2 2 1 9 5 3 1 5 4 7 5

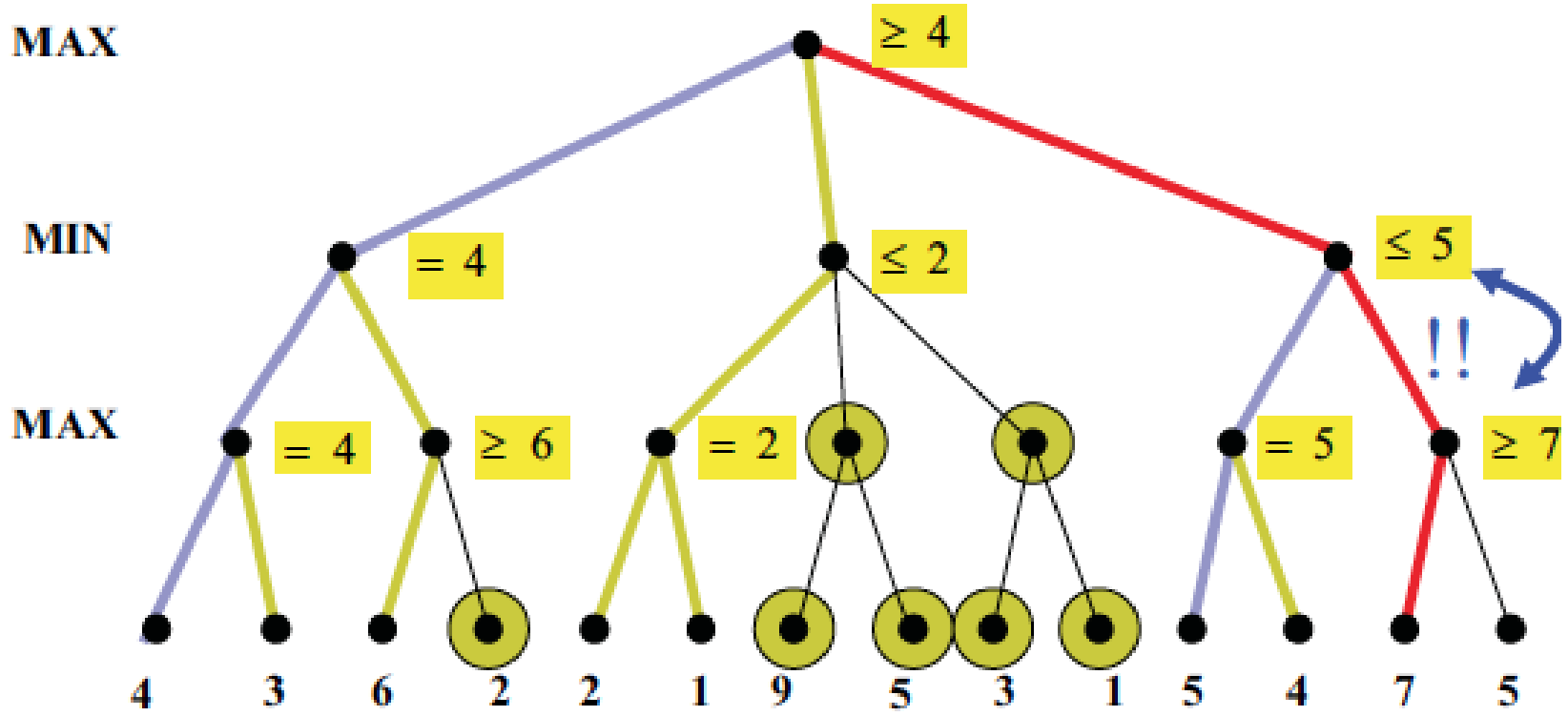

MAX
MIN
MAX
≥ 4
= 4
= 4
≥ 6
≥ 2
4 3 6 2 2 1 9 5 3 1 5 4 7 5

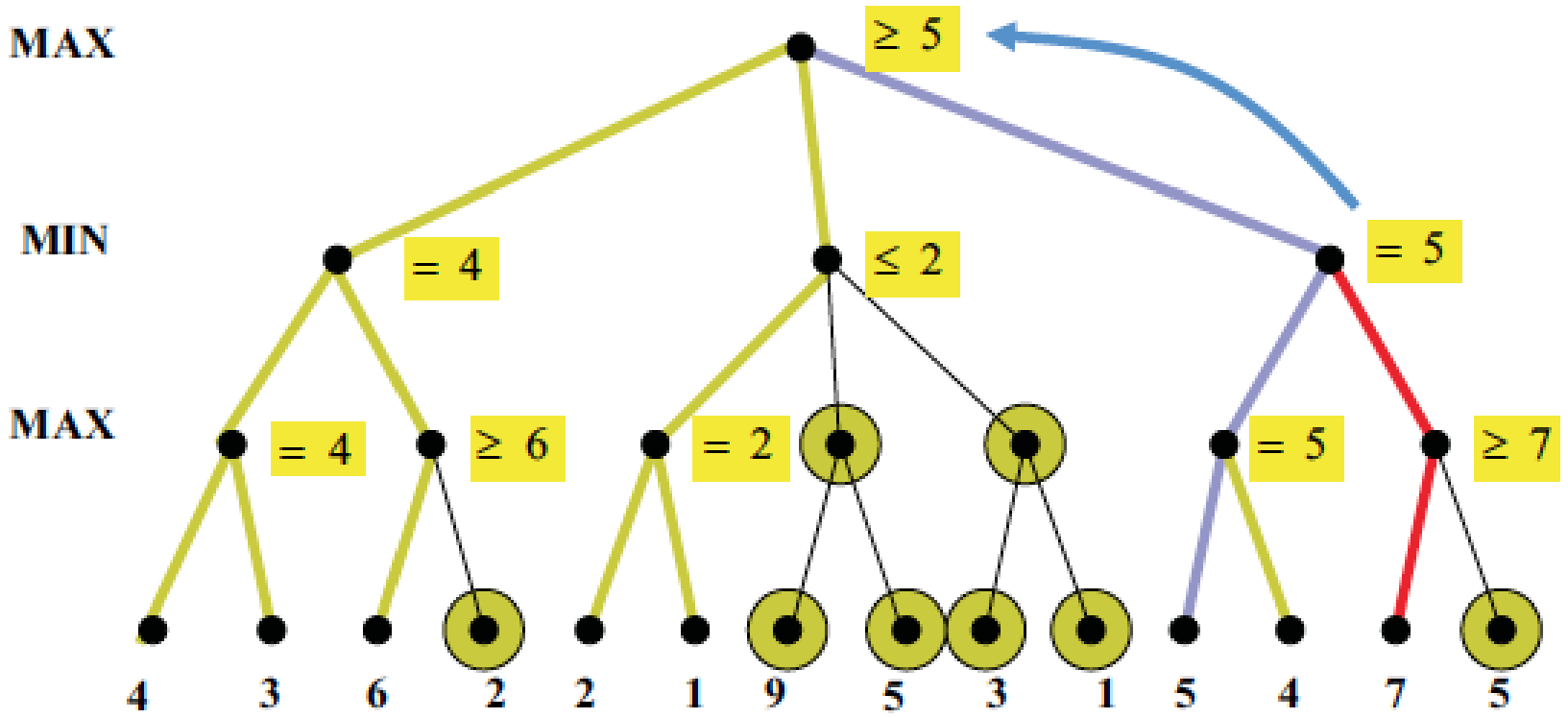

MAX
MIN
MAX
≥ 4
= 4
≤ 2
= 4
≥ 6
= 2
4 3 6 2 2 1 9 5 3 1 5 4 7 5

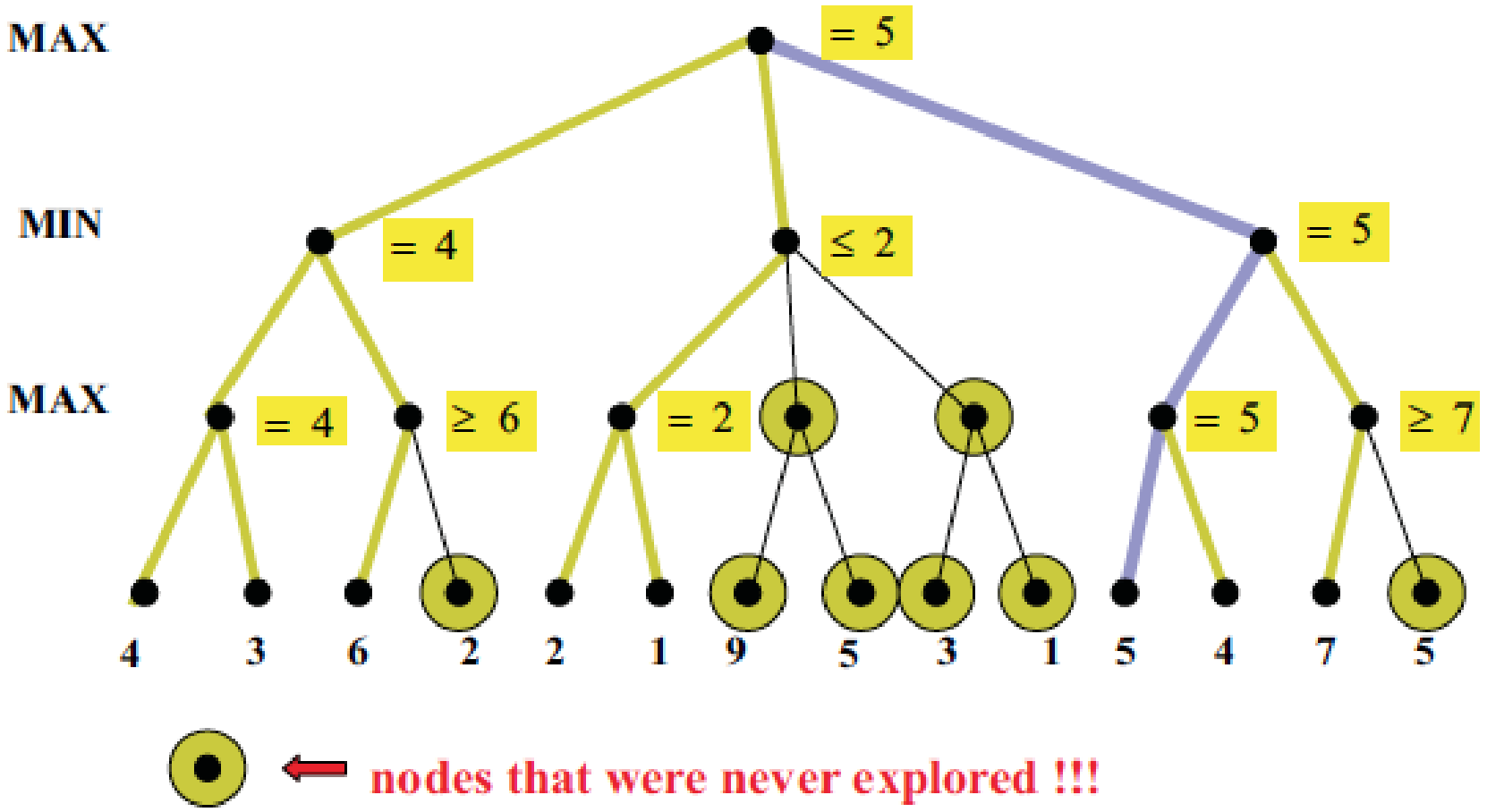

MAX
MIN
MAX
≥ 4
!!
= 4
≤ 2
= 4
≥ 6
= 2
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
≥ 4
= 4
≤ 2
= 4
≥ 6
= 2
≥ 5
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
≥ 4
= 4
≤ 2
≤ 5
= 4
≥ 6
= 2
= 5
4 3 6 2 2 1 9 5 3 1 5 4 7 5

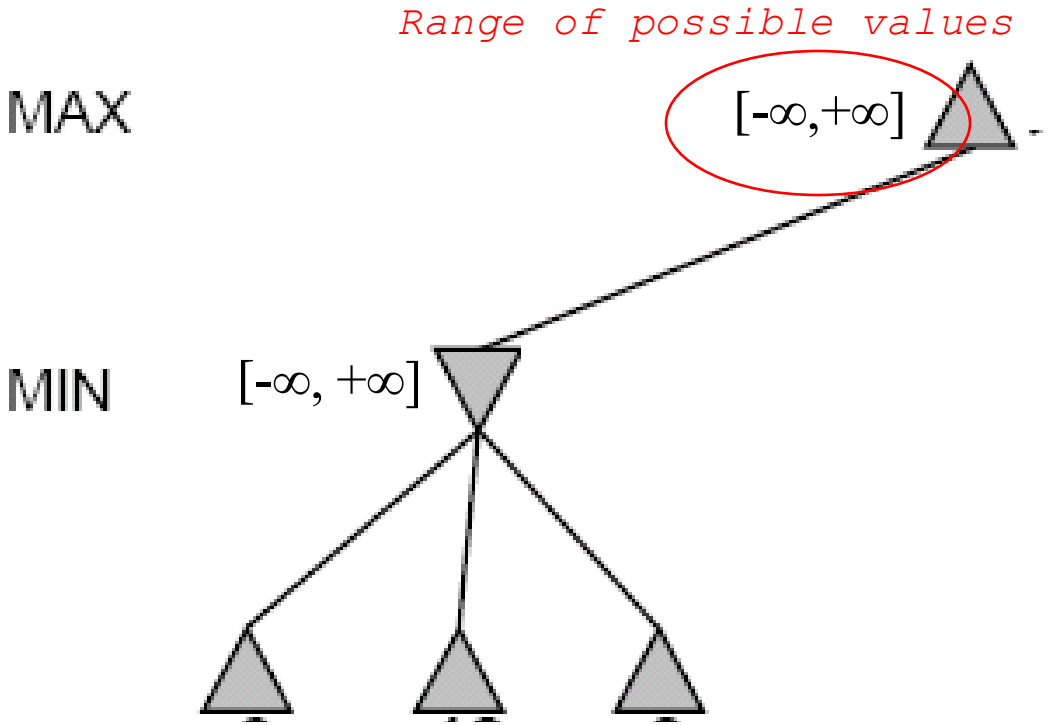

MAX
MIN
MAX
≥ 4
= 4
≤ 2
≤ 5
!!
= 4
≥ 6
= 2
= 5
≥ 7
4 3 6 2 2 1 9 5 3 1 5 4 7 5


MAX
MIN
MAX
≥ 5
= 4
≤ 2
= 5
= 4
≥ 6
= 2
= 5
≥ 7
4
3
6
2
2
1
9
5
3
1
5
4
7
5

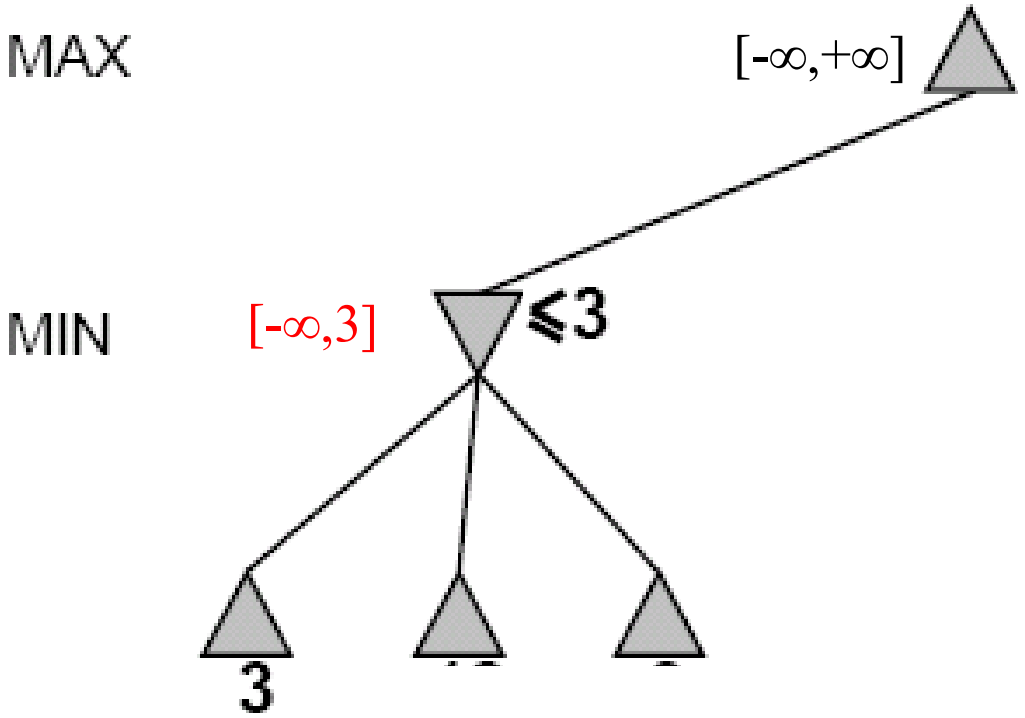

MAX
MIN
MAX
= 5
= 4
≤ 2
= 5
= 4
≥ 6
= 2
= 5
≥ 7
4 3 6 2 2 1 9 5 3 1 5 4 7 5
nodes that were never explored !!!

ANOTHER ALPHA-BETA EXAMPLE

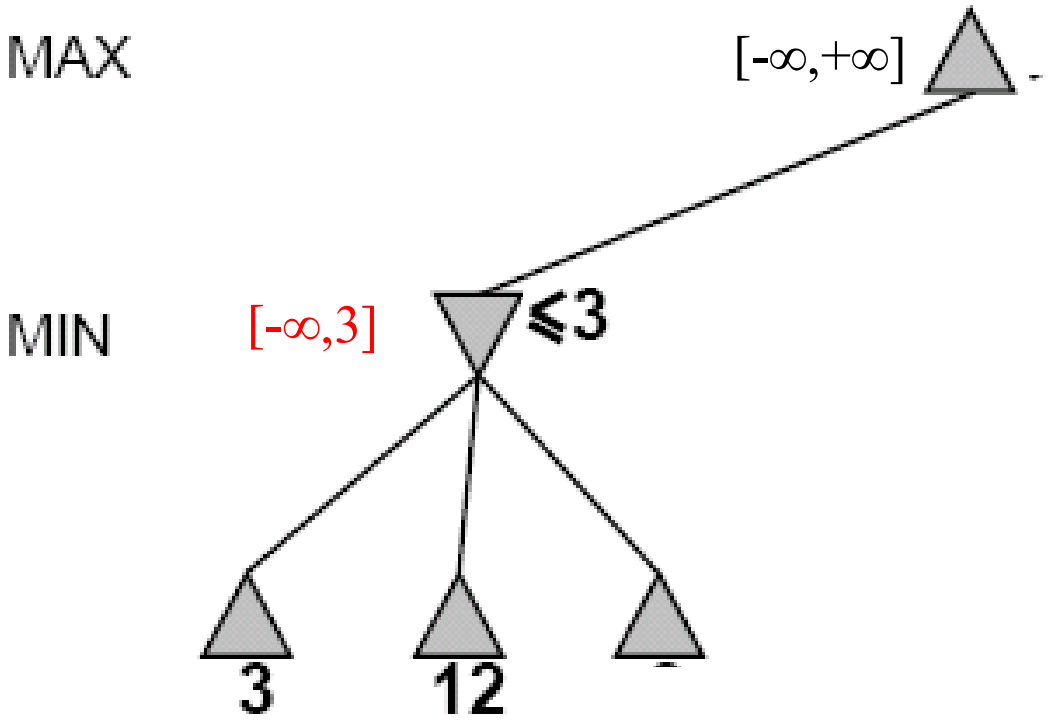
Do DF-search until first leaf



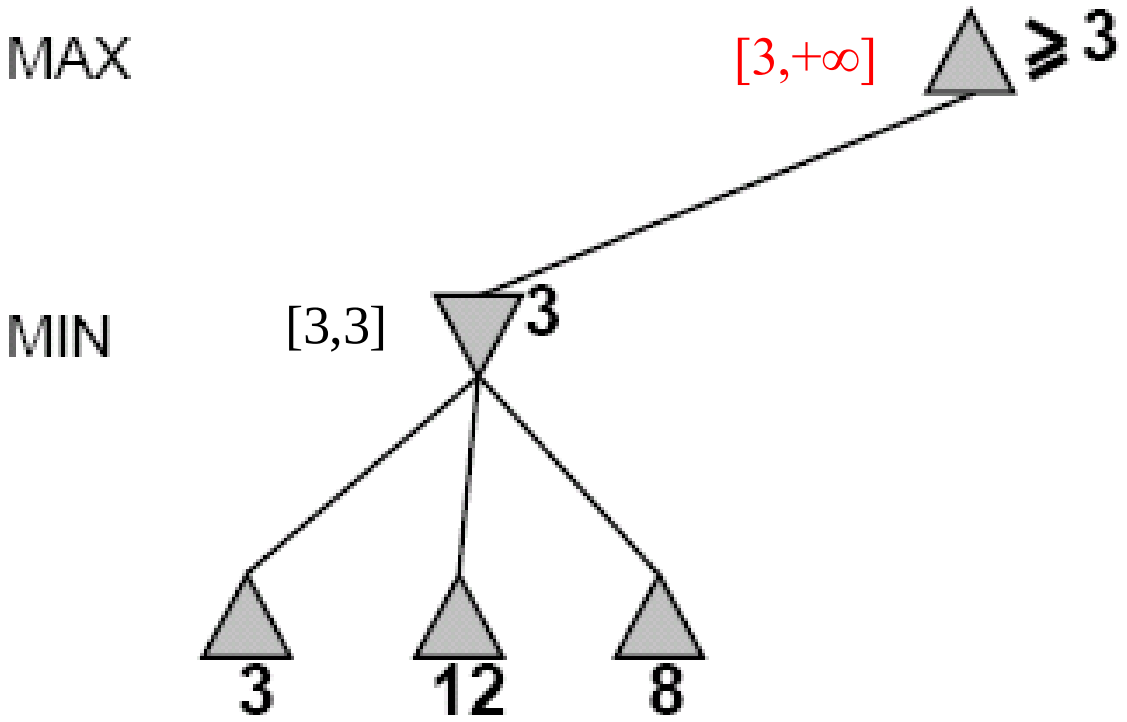
ALPHA-BETA EXAMPLE (CONTINUED)



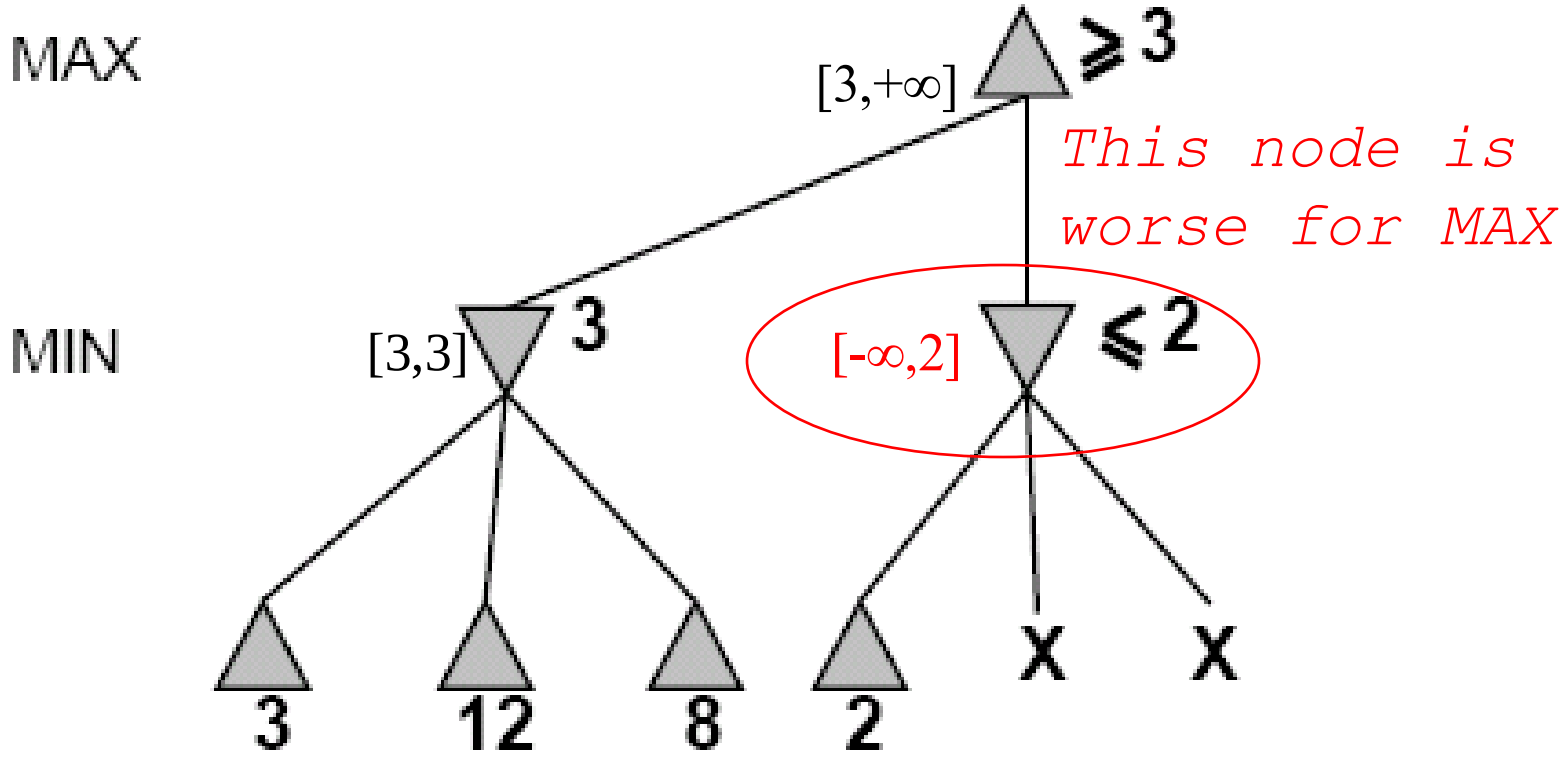
ALPHA-BETA EXAMPLE (CONTINUED)



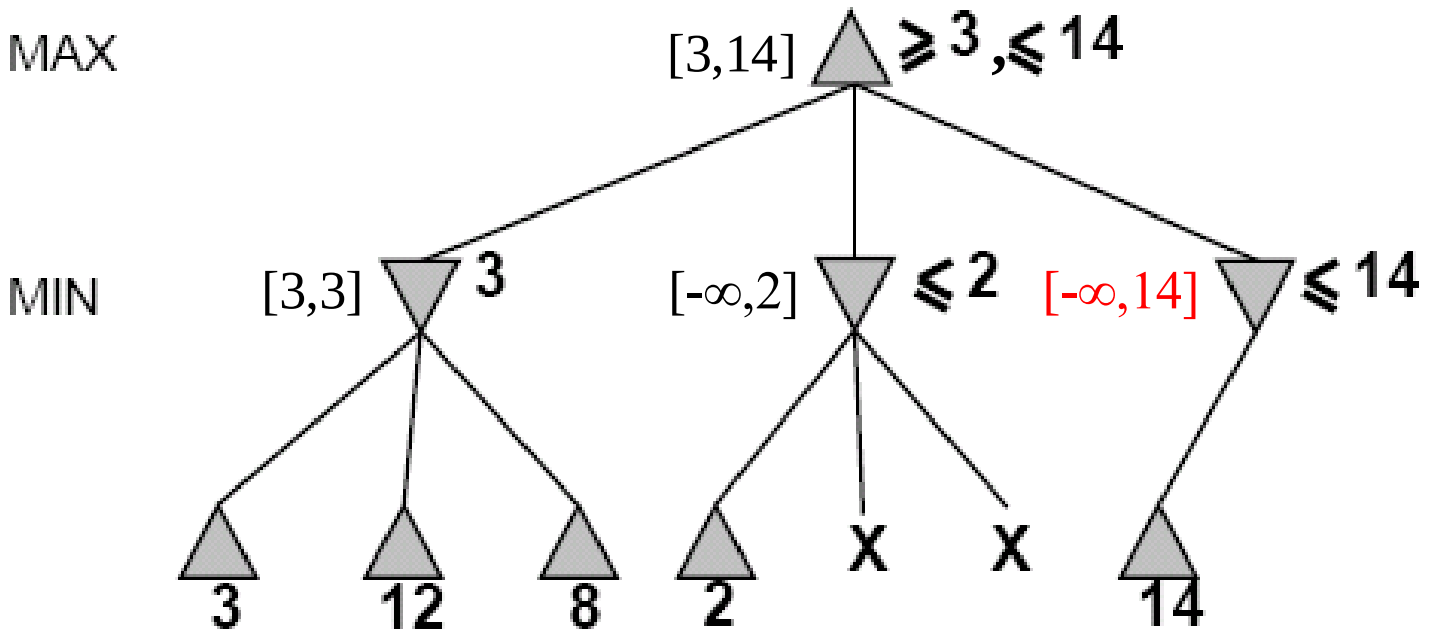
ALPHA-BETA EXAMPLE (CONTINUED)



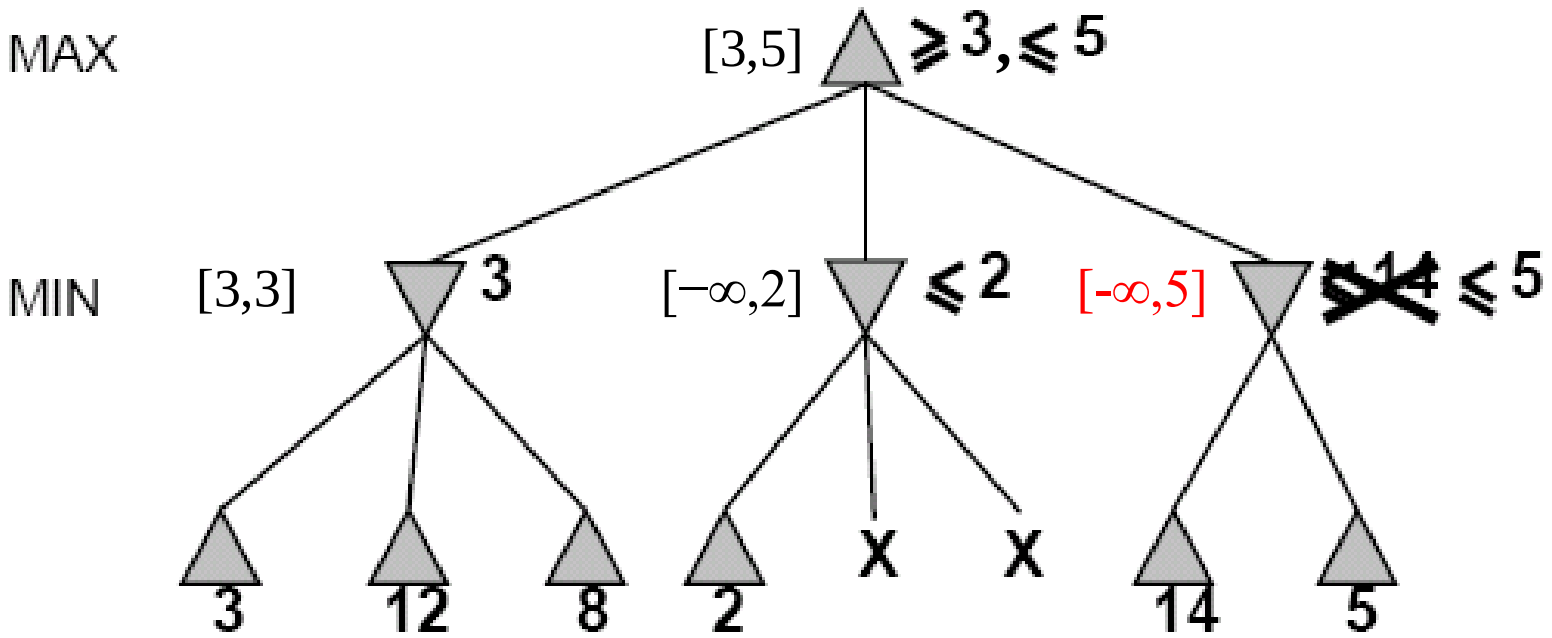
ALPHA-BETA EXAMPLE (CONTINUED)



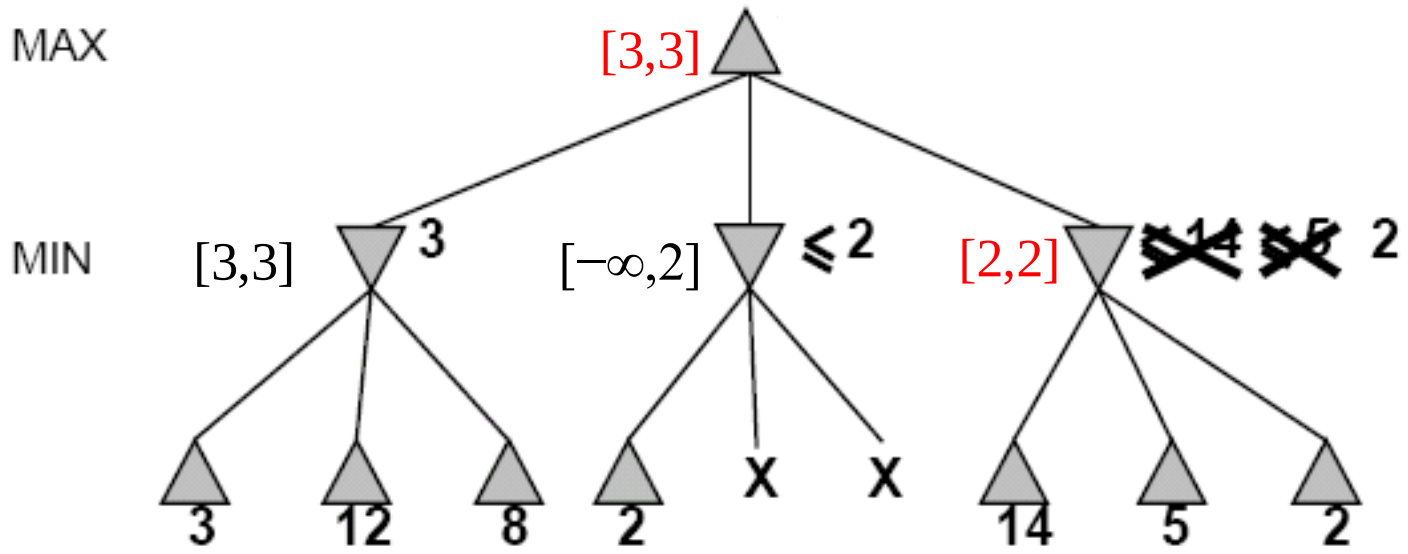
ALPHA-BETA EXAMPLE (CONTINUED)



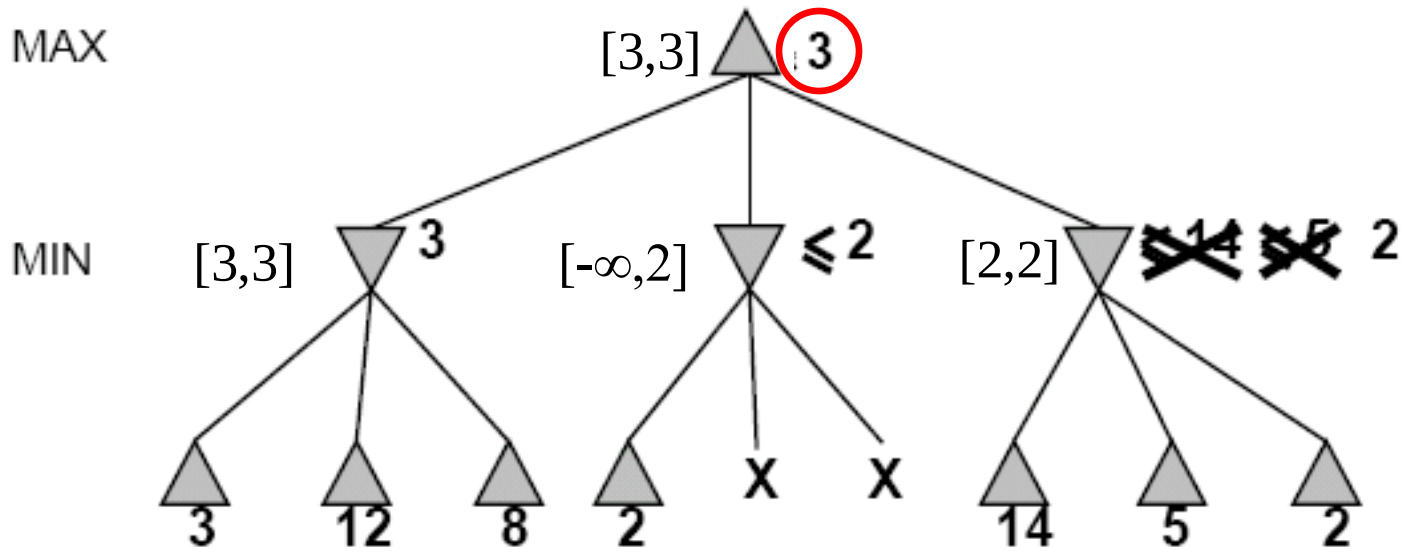
ALPHA-BETA EXAMPLE (CONTINUED)



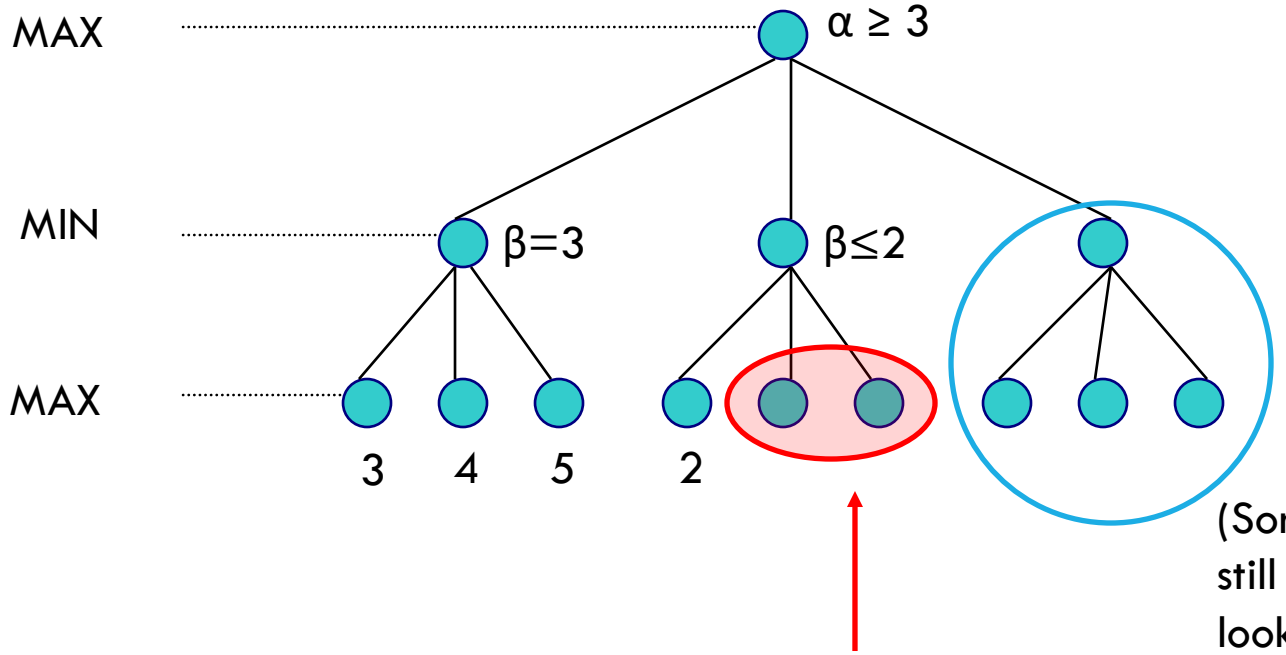
ALPHA-BETA EXAMPLE (CONTINUED)



ALPHA-BETA EXAMPLE (CONTINUED)



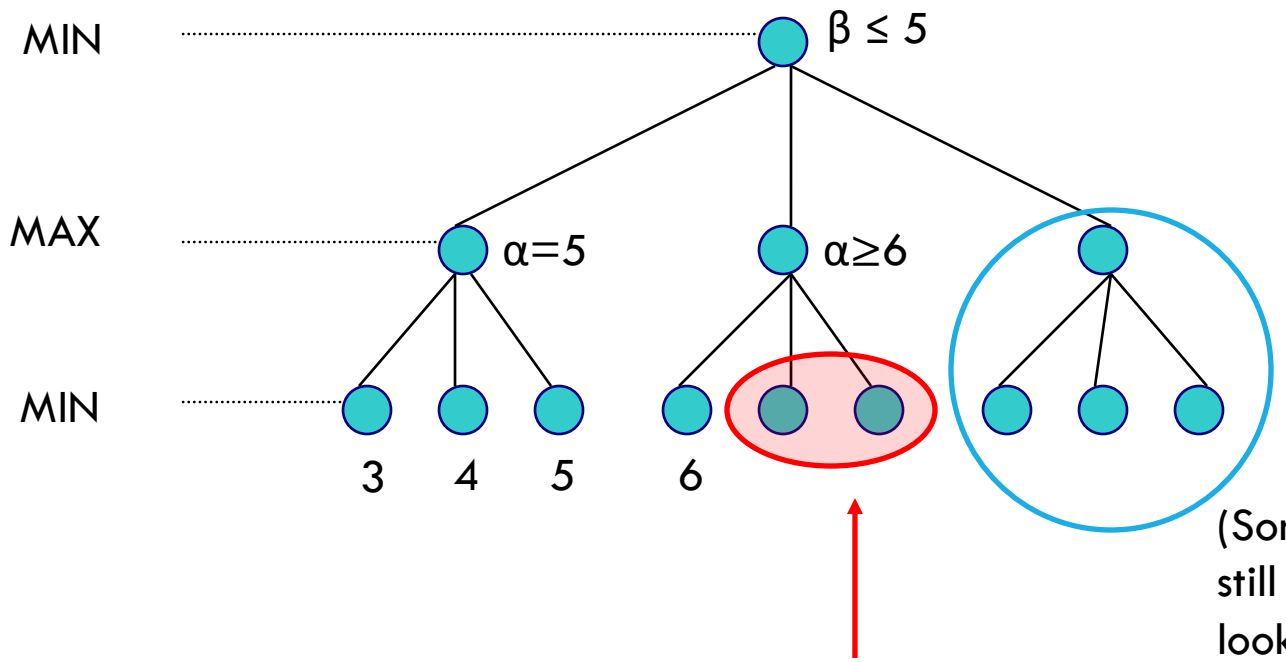
EXAMPLE WITH MAX



(Some of) these still need to be looked at

As soon as the node with value 2 is generated, we know that the beta value will be less than 3, we don't need to generate these nodes (and the subtree below them)

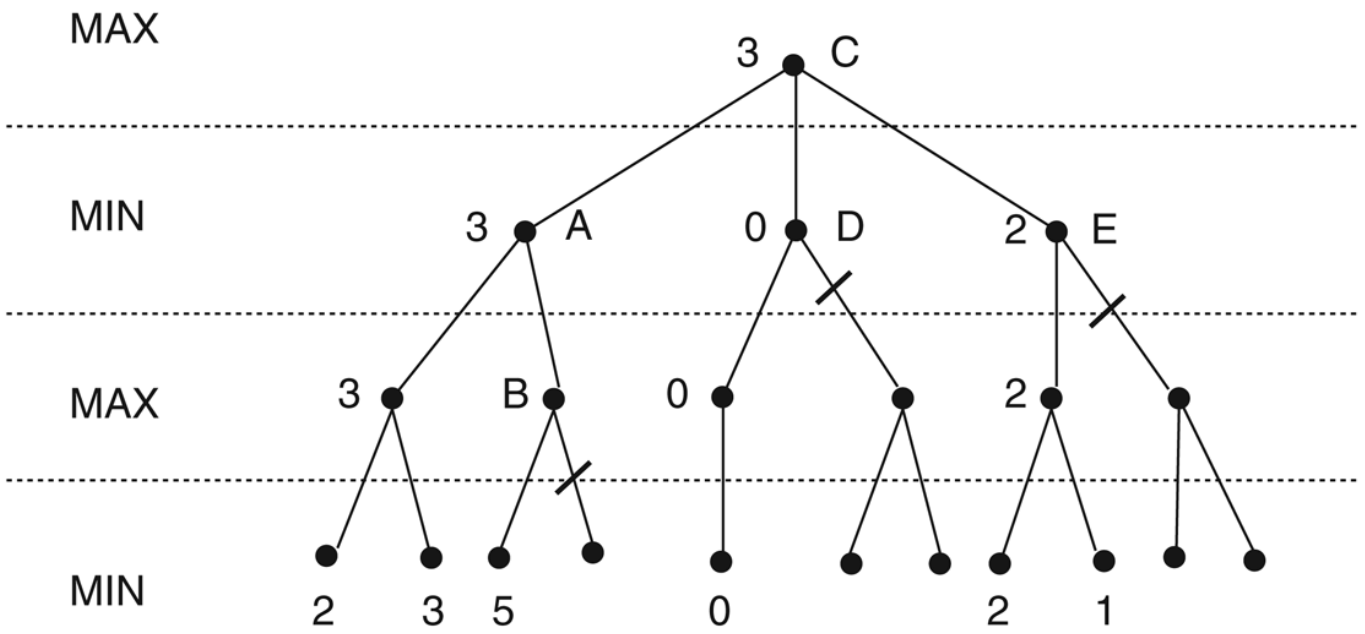
EXAMPLE WITH MIN



(Some of) these still need to be looked at

As soon as the node with value 6 is generated, we know that the alpha value will be larger than 6, we don't need to generate these nodes (and the subtree below them)

ALPHA-BETA PRUNING APPLIED TO THE STATE SPACE OF FIG. 4.15



A has $\beta = 3$ (A will be no larger than 3)
B is β pruned, since 5 > 3
C has $\alpha = 3$ (C will be no smaller than 3)
D is α pruned, since 0 < 3
E is α pruned, since 2 < 3
C is 3

GENERAL ALPHA-BETA PRUNING

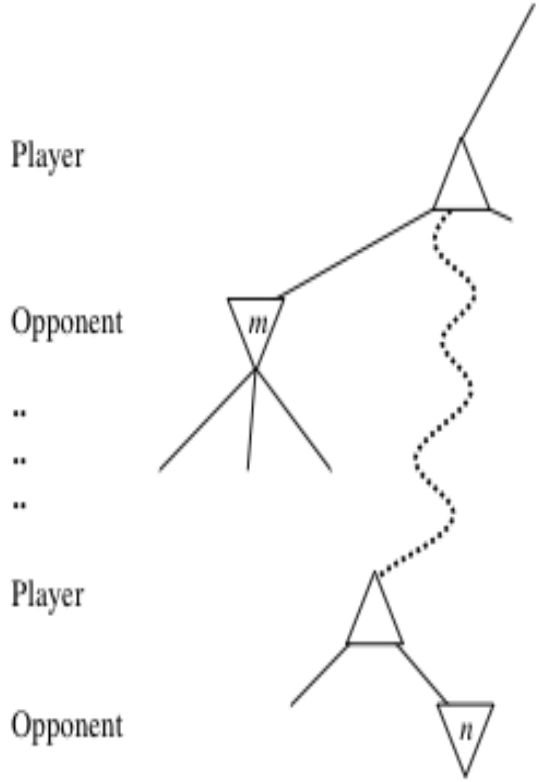
Consider a node *n* in the tree ---

If player has a better choice at:

- Parent node of n
- Or any choice point further up

Then *n* will never be reached in play.

Hence, when that much is known about *n*, it can be pruned.



ALPHA-BETA PRUNING

It is guaranteed to return exactly the same value as the Min−Max algorithm.

It is a pure optimization without any approximations or tradeoffs.

In a perfectly ordered tree, with the best moves on the left, alpha beta reduces the cost of the search from order $\mathbf{b^d}$ to order $\mathbf{b^{(d/2)}}$, that is, we can search twice as deep.

Worst case it won't prune any nodes.