Alpha Beta Pruning

- Alpha – Minimum value the max node can take
- Beta – Maximum value the min node can take

- Conditions : alpha >= beta

- Some games may require large amount of knowledge – Chess
- Improve the effectiveness of a search-based problem solving program
    - Improve the generate procedure so that only good moves are generated.
    - Improve the test procedure so that the best paths will be recognized and explored first.
    - Game playing requires both of these things.
    - Chess – 35 legal moves at each turn.
    - Plausible move generator.
    - Incorporating heuristic knowledge into both the generator and the tester, the performance of the overall system can be improved.
- Simulate forward thinking
- Every possible game move from start to last.

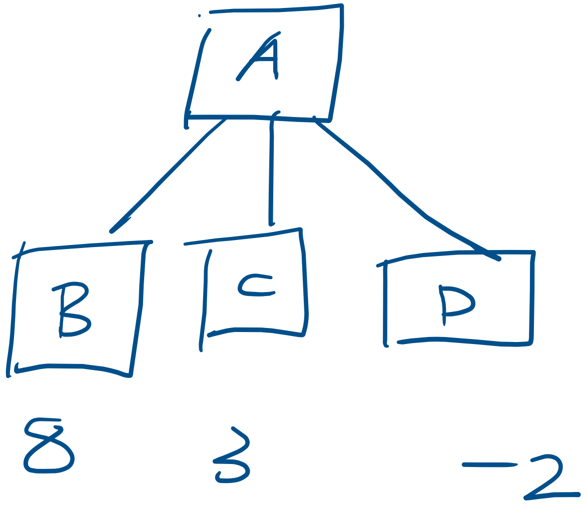
- Universe – Represented using game tree
- Level – next players turn
- No of possible moves decreases with each level.

- Ideal way for a search procedure – to find a solution is to generate moves through the problem space until a goal state is reached.
- Best move needs to be chosen – Static evaluation function
- Which move will contribute in win and which in losses is not easy.
- So two important thing – good plausible move generator and good static evaluation function.

Minimax Algorithm

- Search procedure
- Depth first – Depth limited
- Idea
    - Start from the current position
    - Use plausible move generator to generate next moves
    - Apply static evaluation function on those to choose the best.
    - After this we back this value up to the starting node.

- Goal of computer – leave human to lower score.
- Terminal nodes – cost values
- Min = +infinity
- Max = -infinity



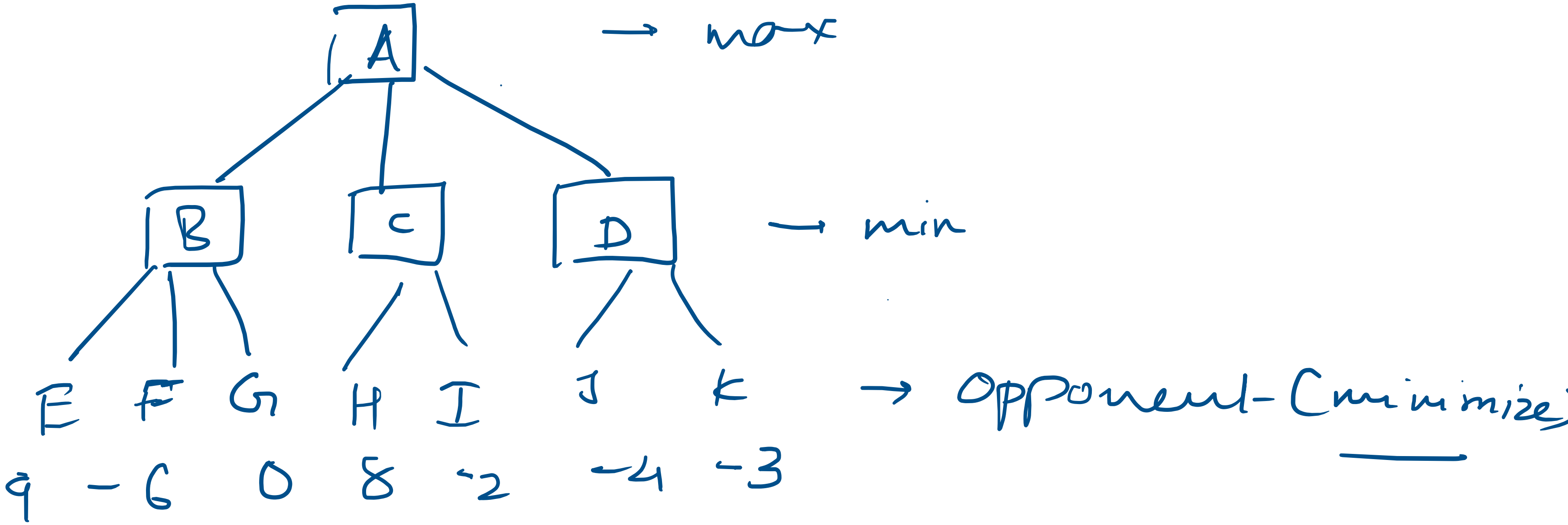
Static evaluation function value ranges from −10 to 10.

=> Goal → maximize

↓

so choose

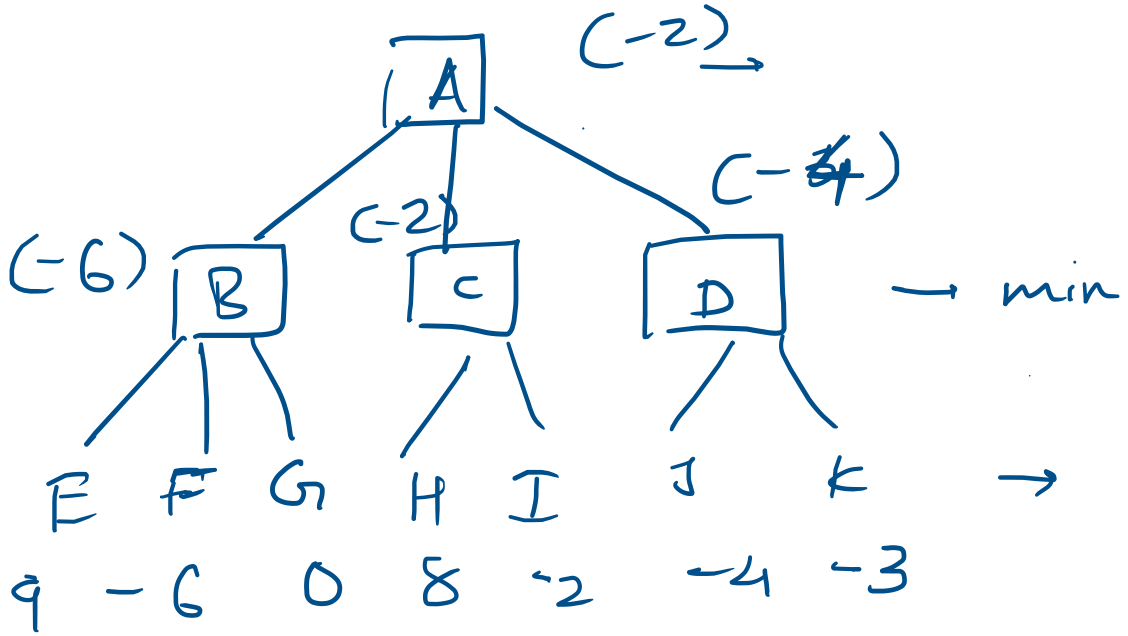
B



→ move the search to next ply.

Example: Chess → middle of a move.

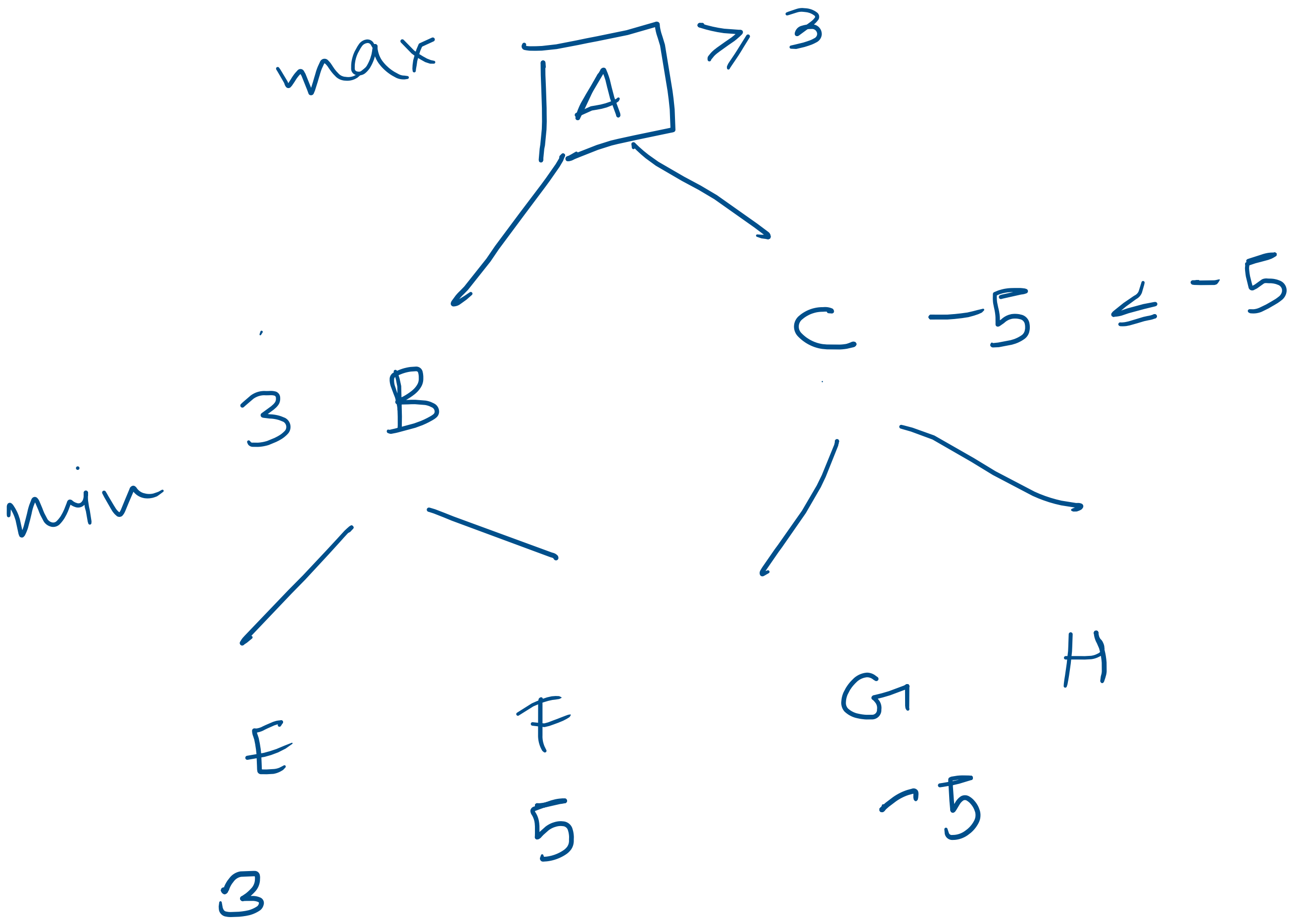
→ After one move the situation appears very good, but if we look one more ahead → we will see our pieces get captured.



→ make move B → then actual configuration gives value as (−6). so, C is a better move. Since there is nothing worse the opponent can do to produce value than (−2).

- The alternation of maximizing and minimizing at alternate ply when evaluations are being pushed back up corresponds to the opposing strategies of two players and gives this method the name minimax.
- Maximizing player is guaranteed atleast a value of -2 by choosing to move to C.

- Alpha – beta pruning
- Alpha – maximum score maximizing player can achieve
- Beta – minimum score minimizing player can achieve
- Alpha - -infinity
- Beta - + infinity
- Condition = alpha >=beta

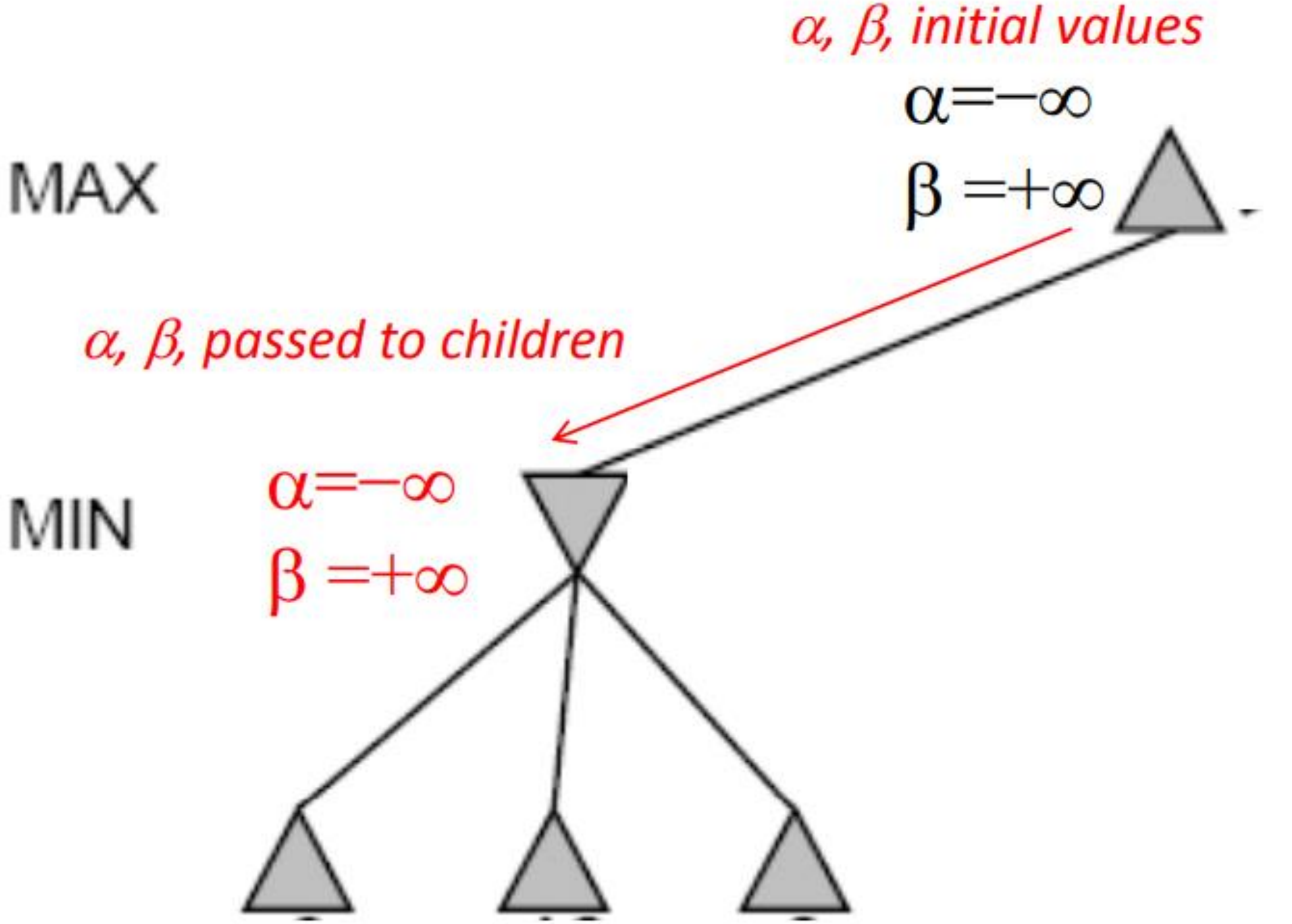


any move that produces less than 3 is worse move

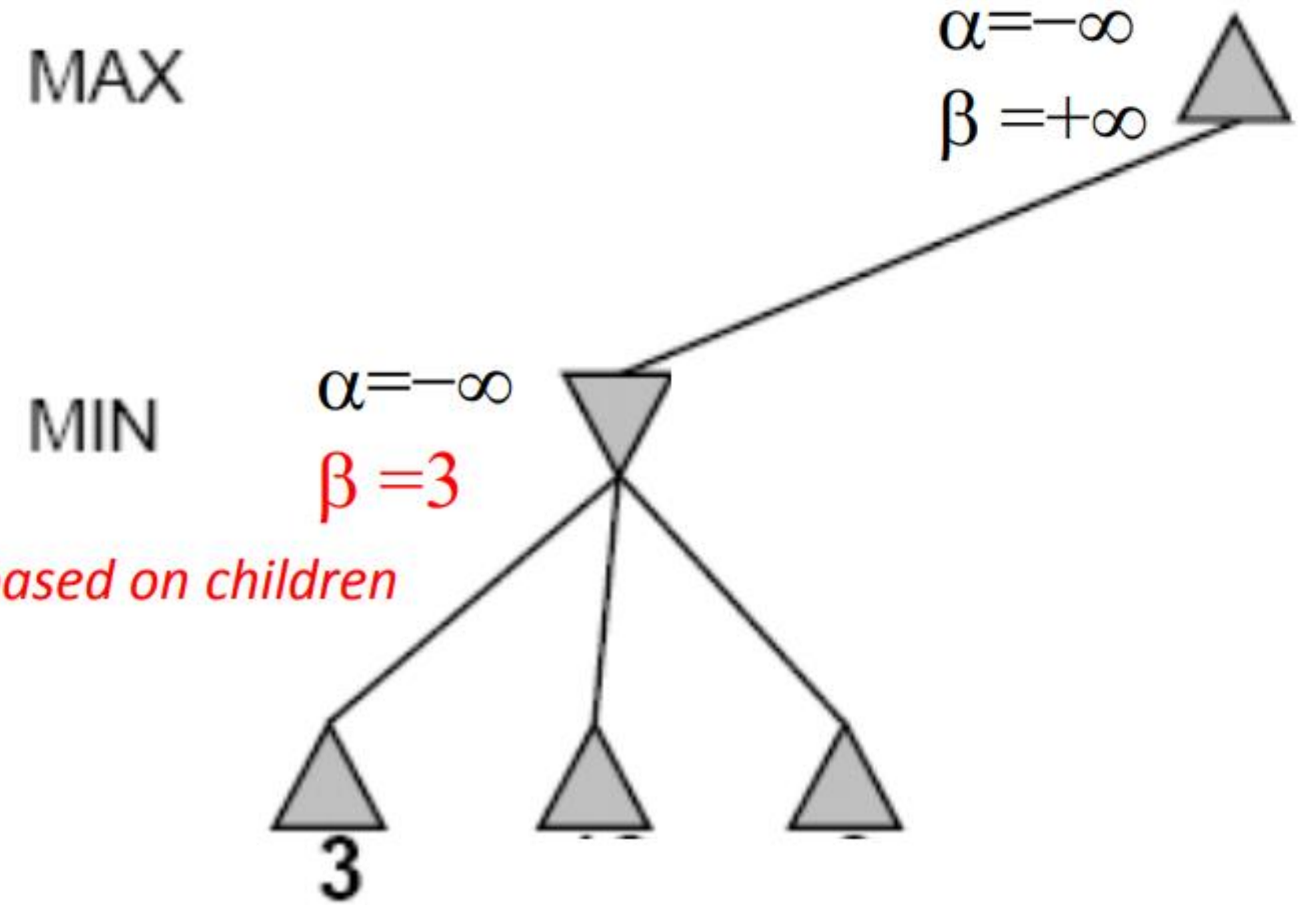
So, no need to explore C further.

- At maximizing levels, we can rule out a move early if it becomes clear that its value is less than the current threshold, while at the minimizing levels, search will be terminated if values are greater than the current threshold.

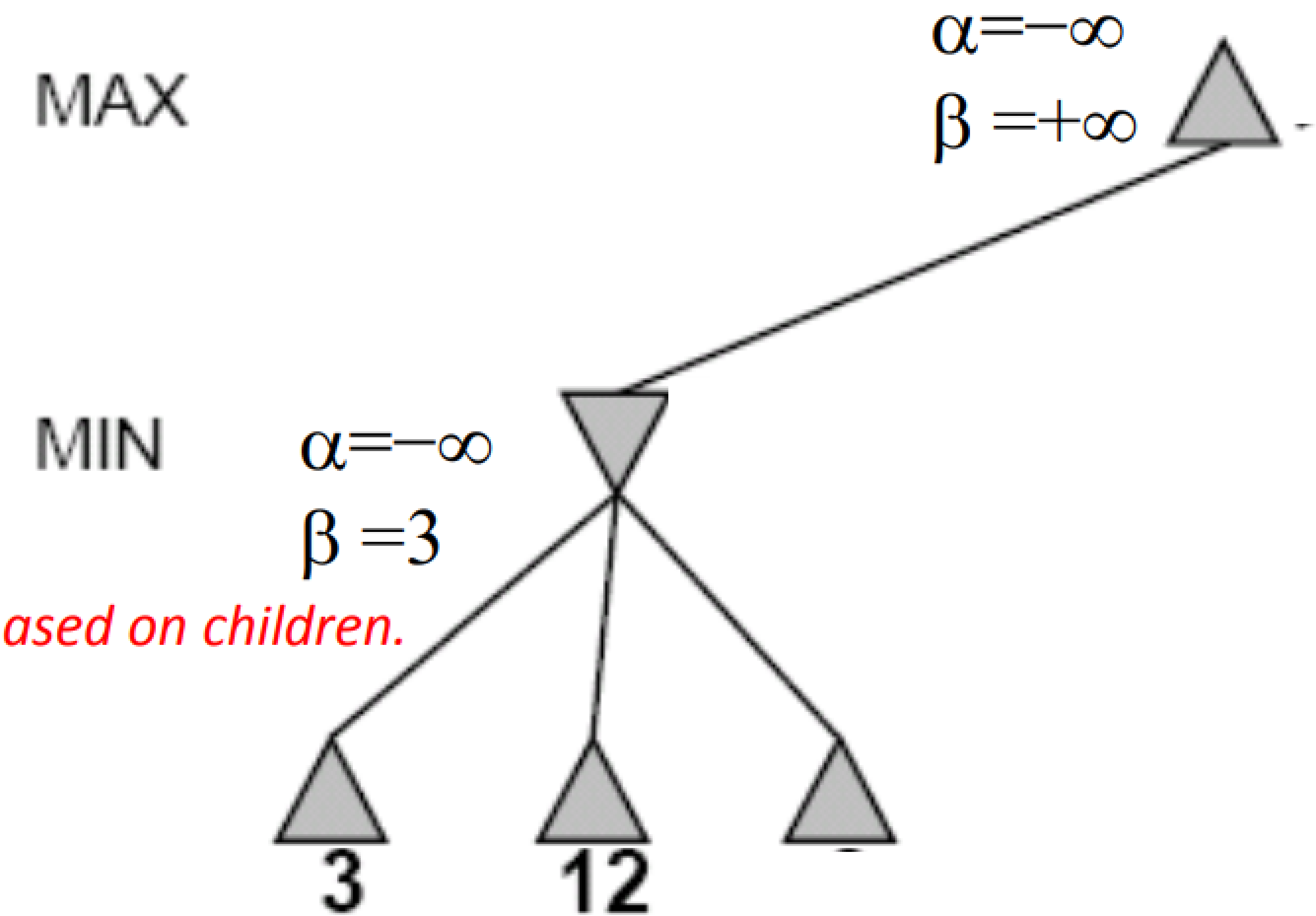
Do DF-search until first leaf



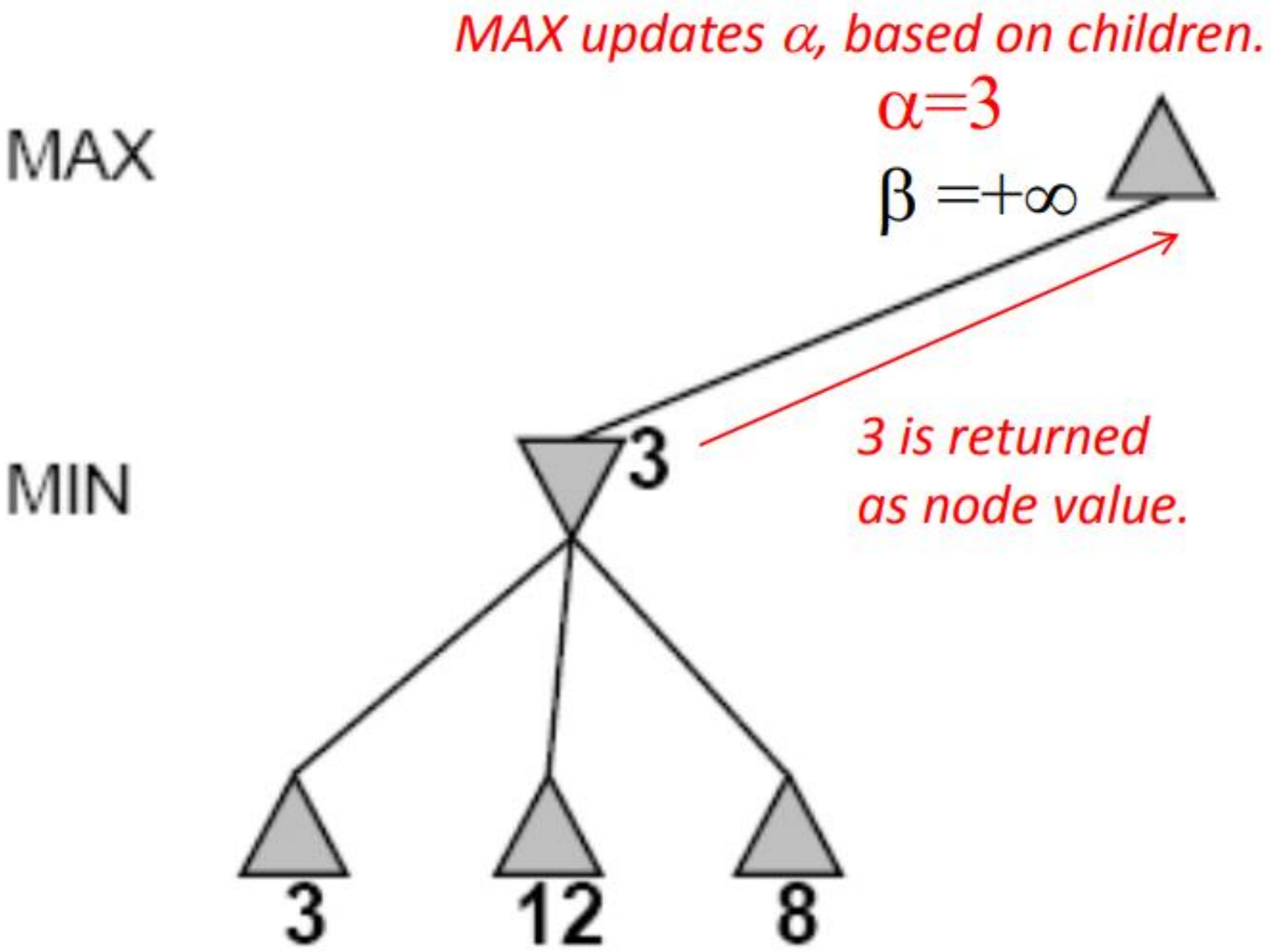
α, β, initial values
$\alpha=-\infty$
$\beta=+\infty$
MAX
α, β, passed to children
MIN
$\alpha=-\infty$
$\beta=+\infty$

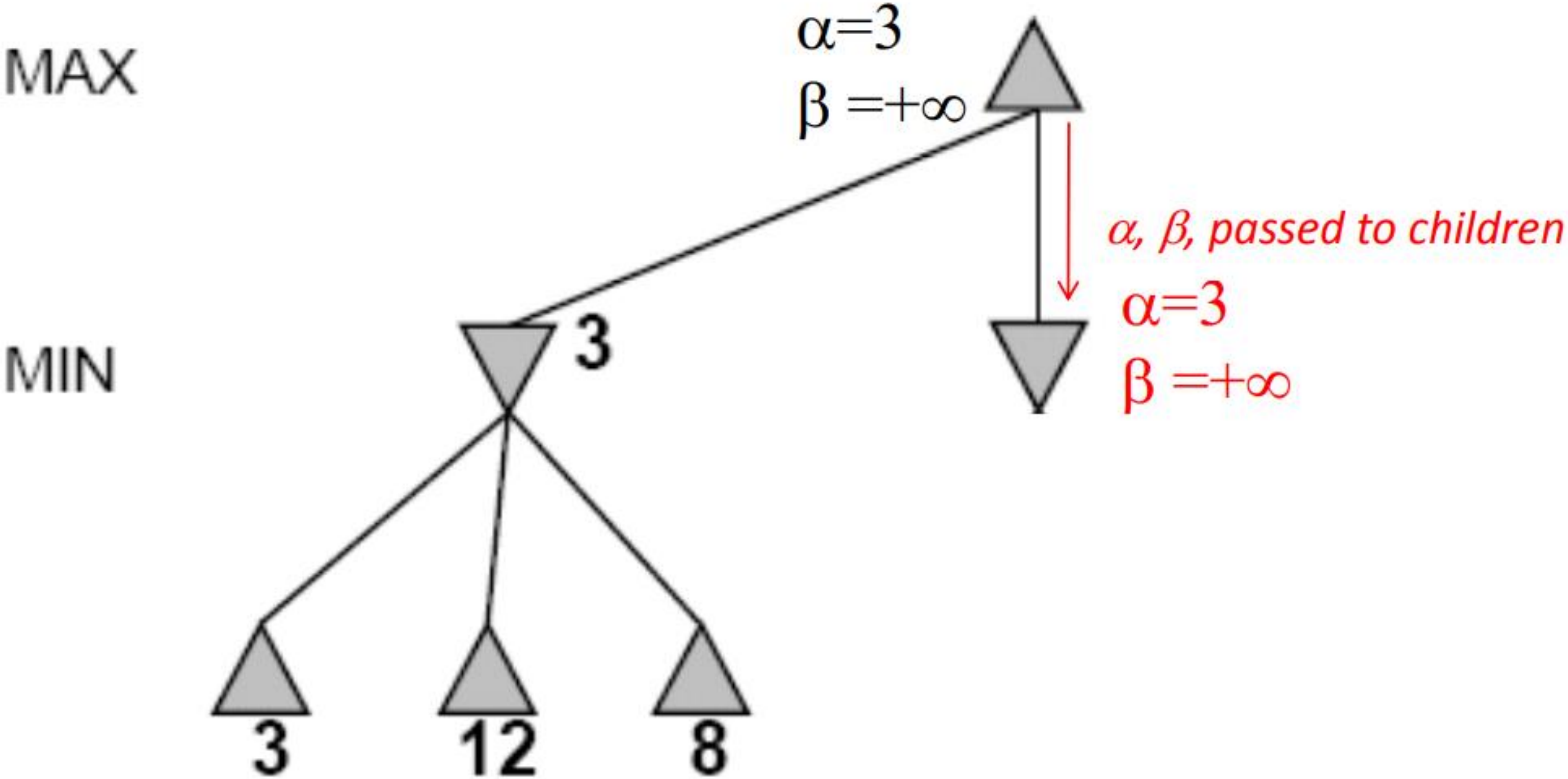


MIN updates β, based on children

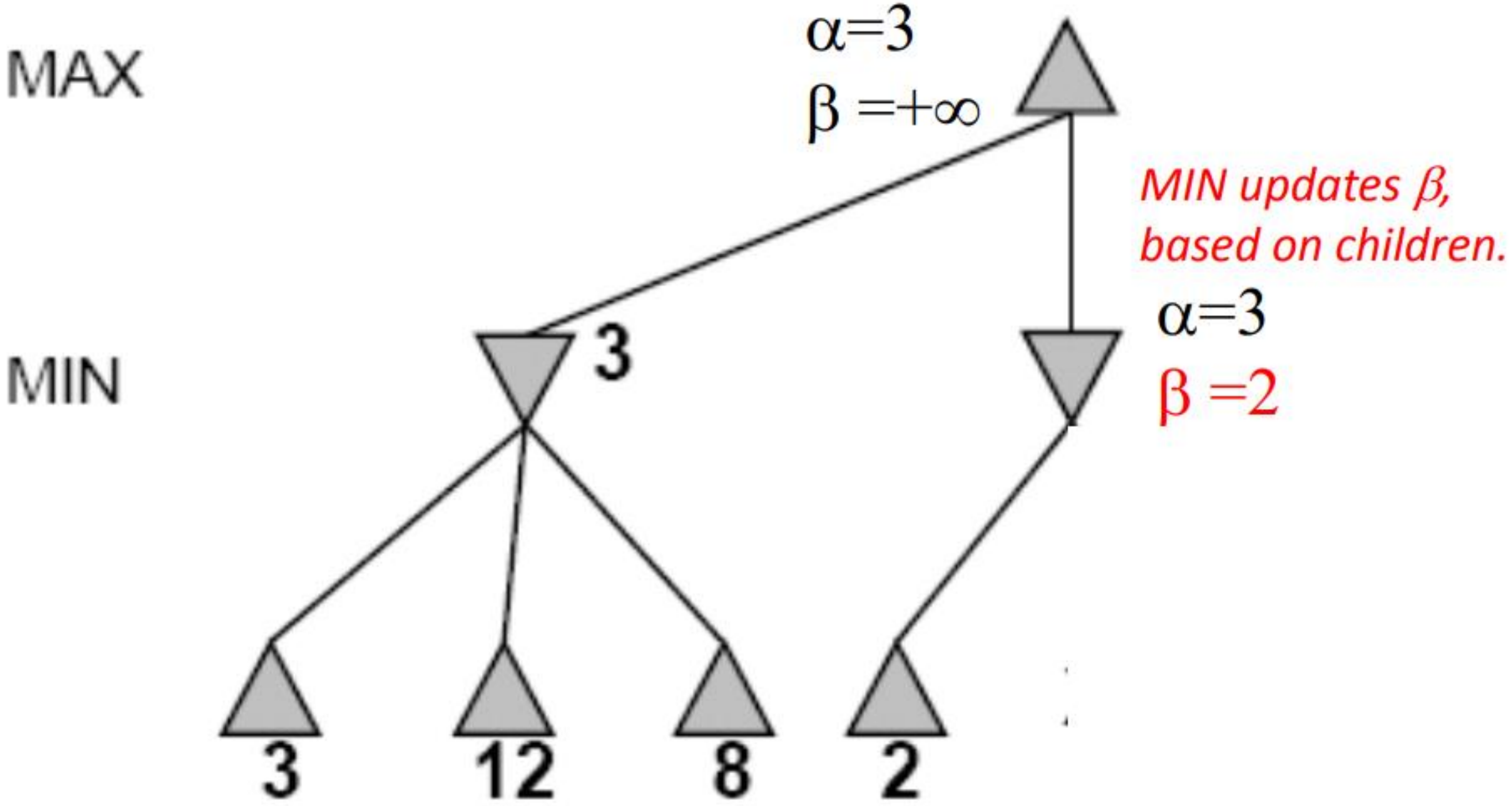


MIN updates β, based on children.
No change.

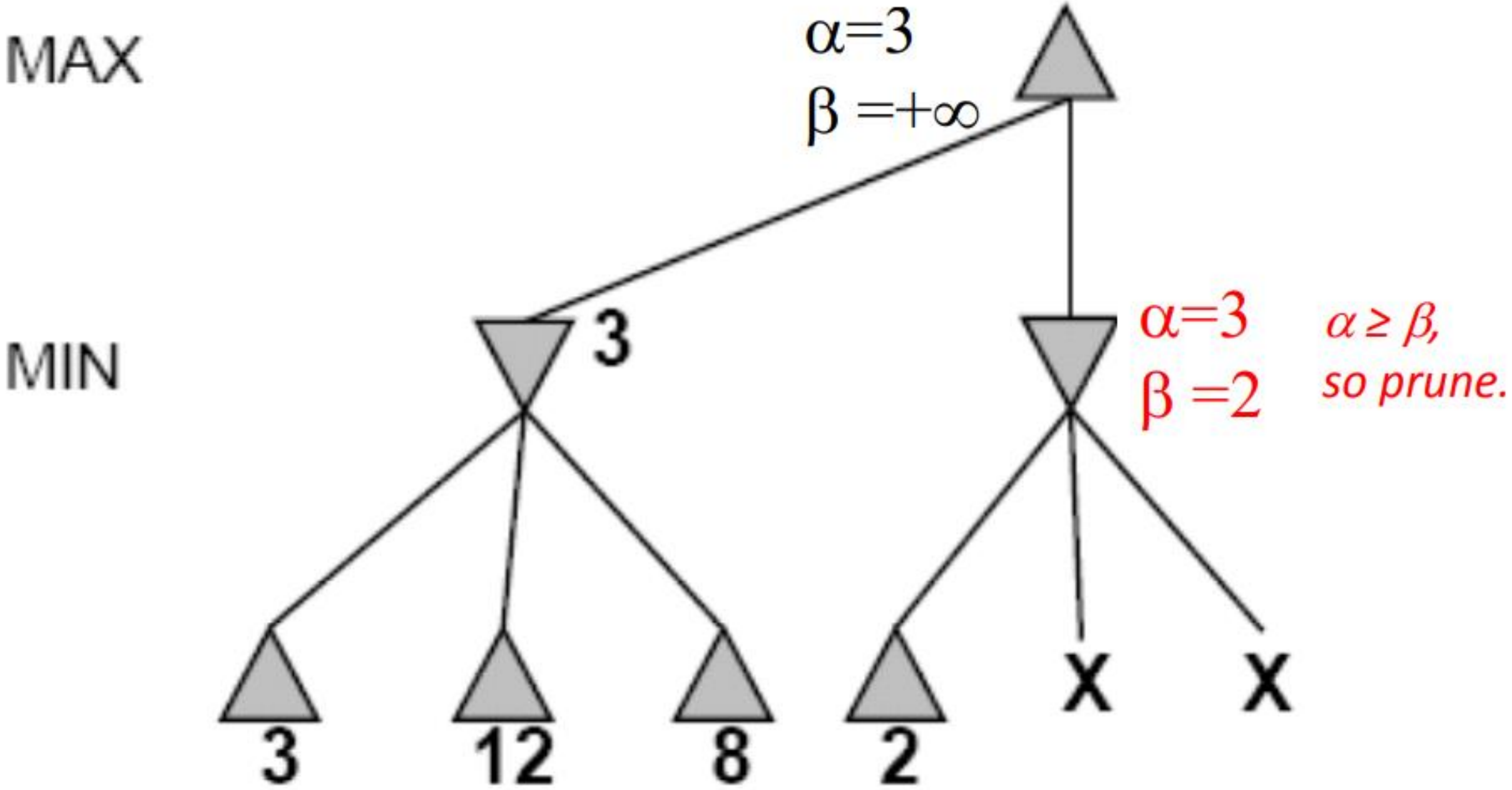

MAX updates α, based on children.
α=3
β =+∞
MAX
MIN
3
3 is returned
as node value.
3
12
8

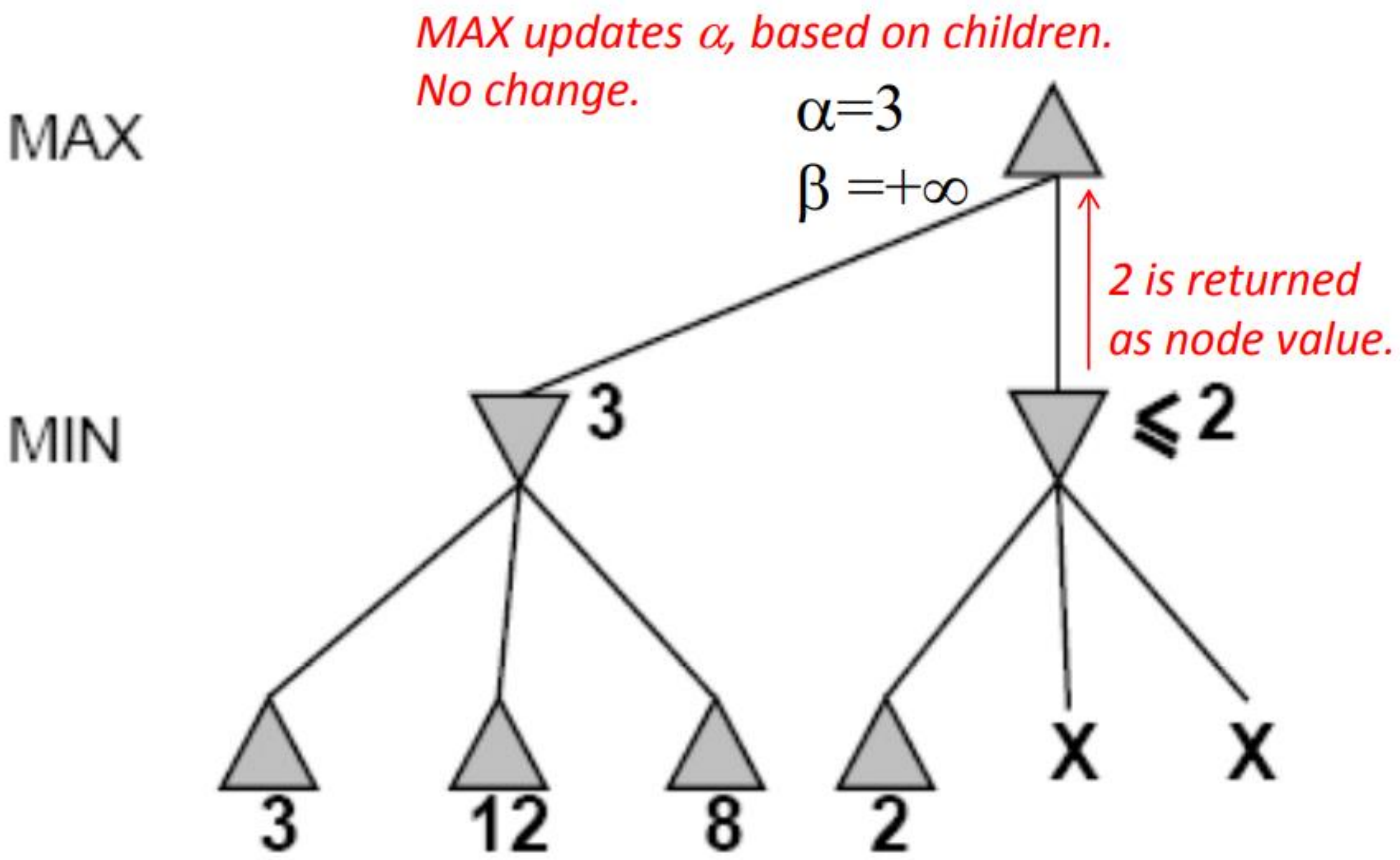


α, β, passed to children



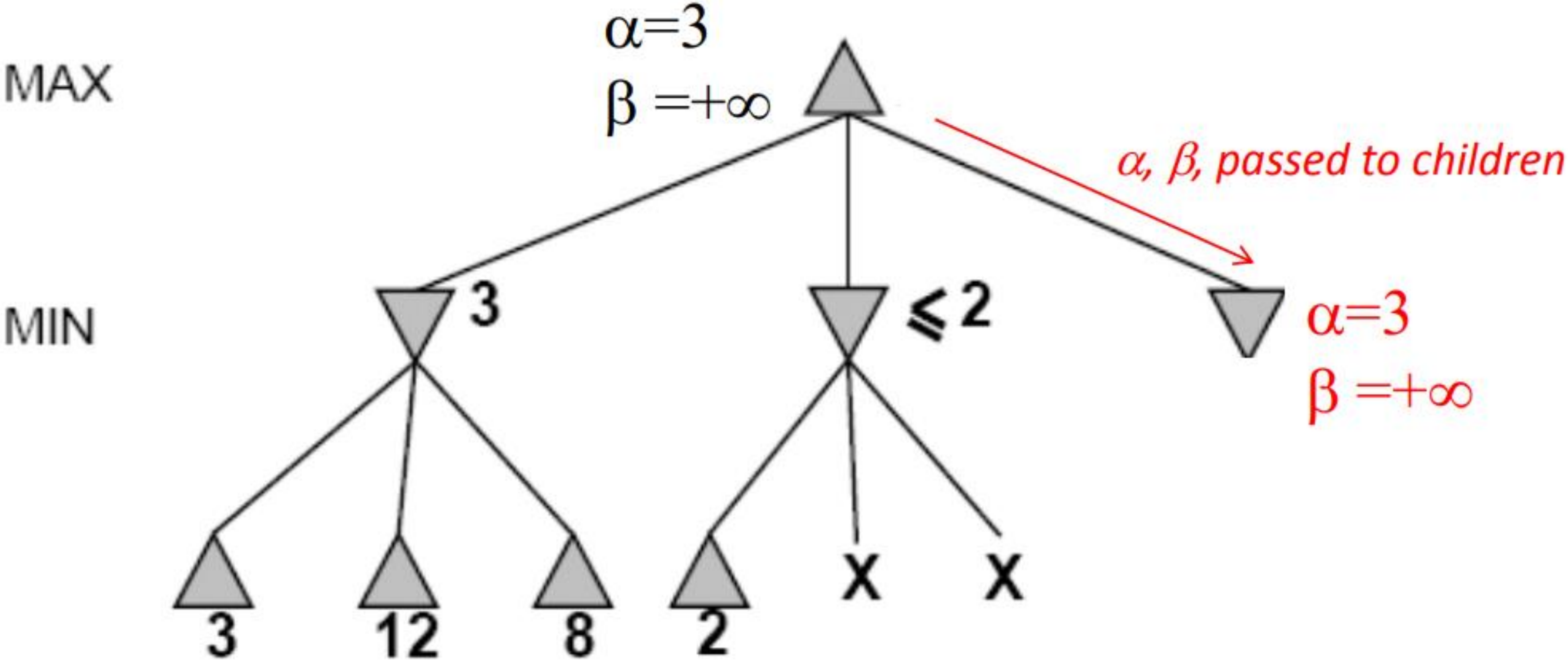
MIN updates β, based on children.


MAX
α=3
β =+∞
MIN
3
α=3
β =2
α ≥ β,
so prune.
3
12
8
2
X
X

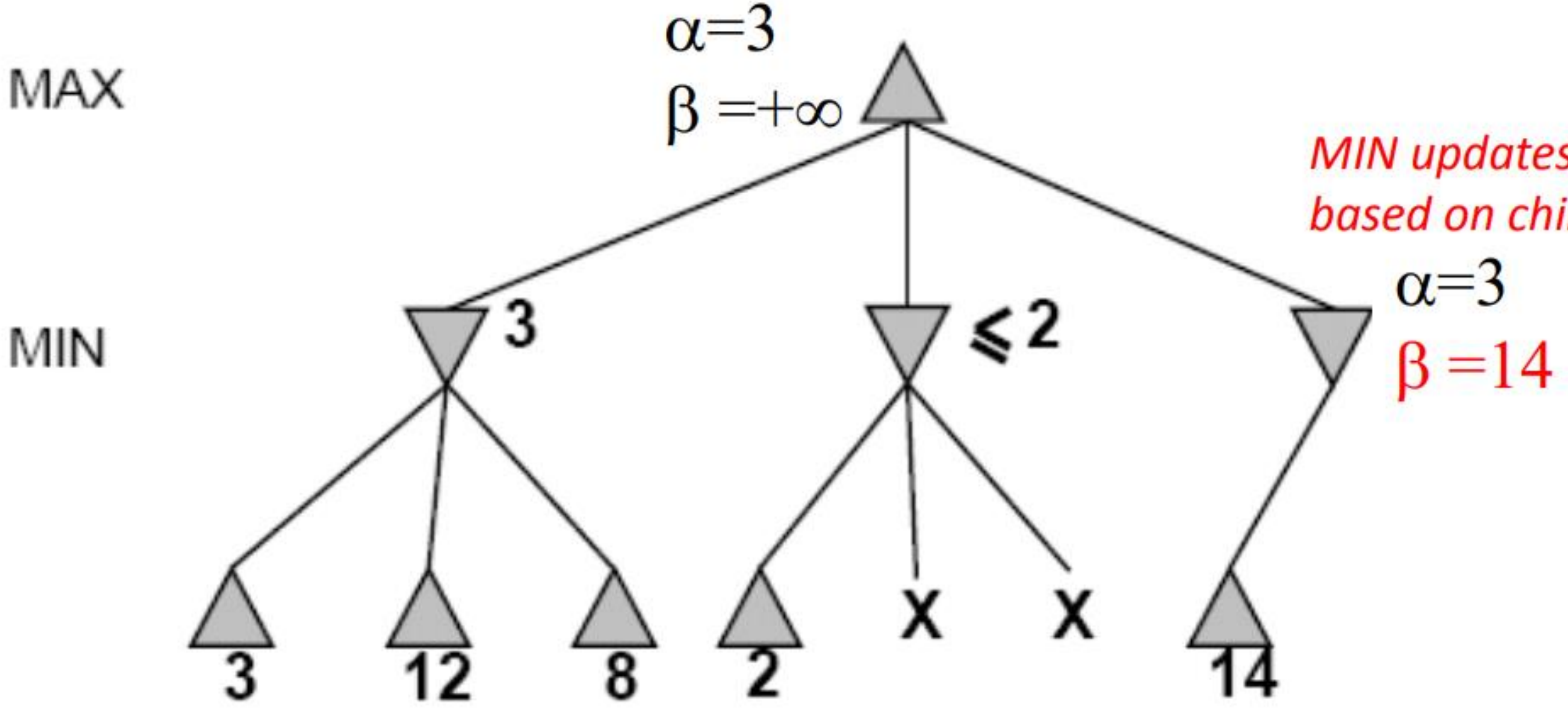


MAX updates α, based on children. No change.

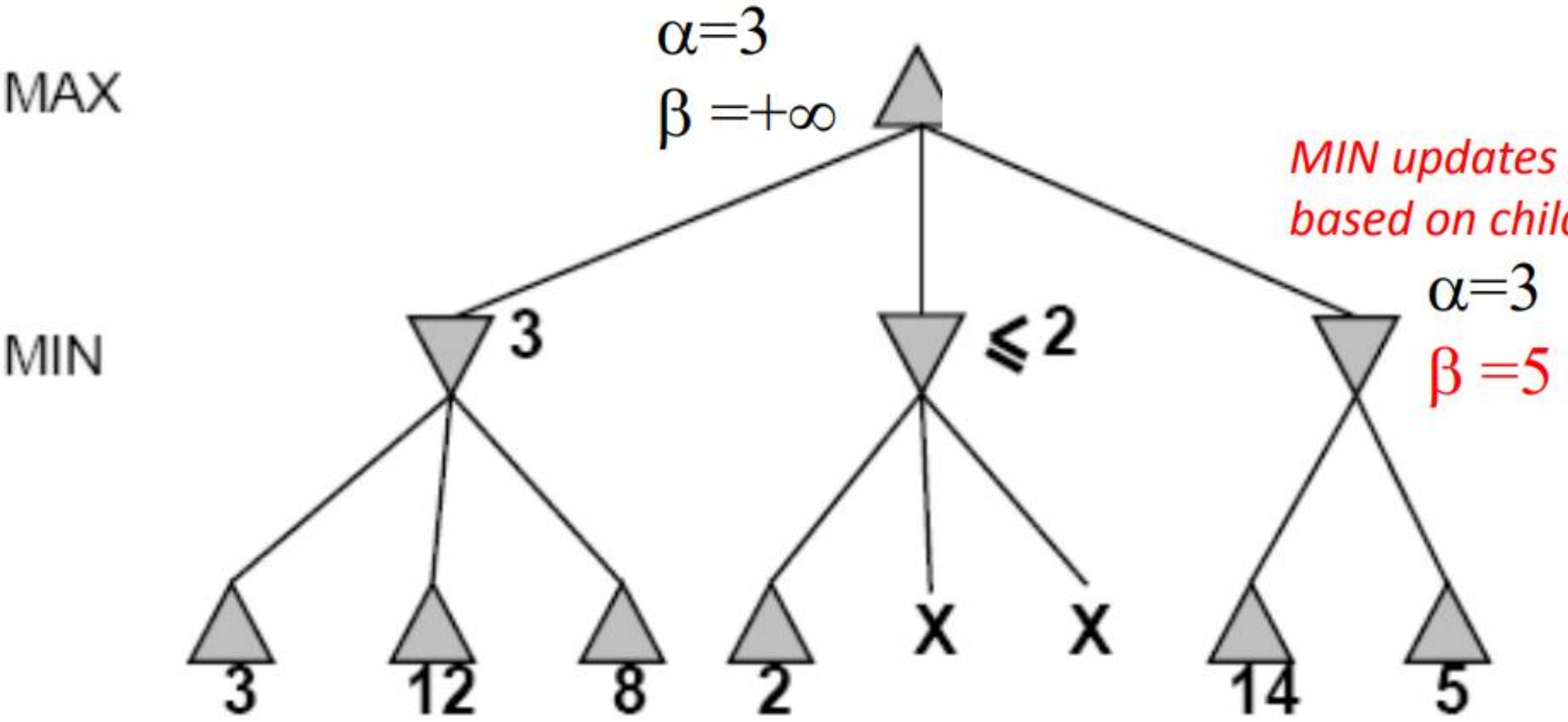
2 is returned as node value.



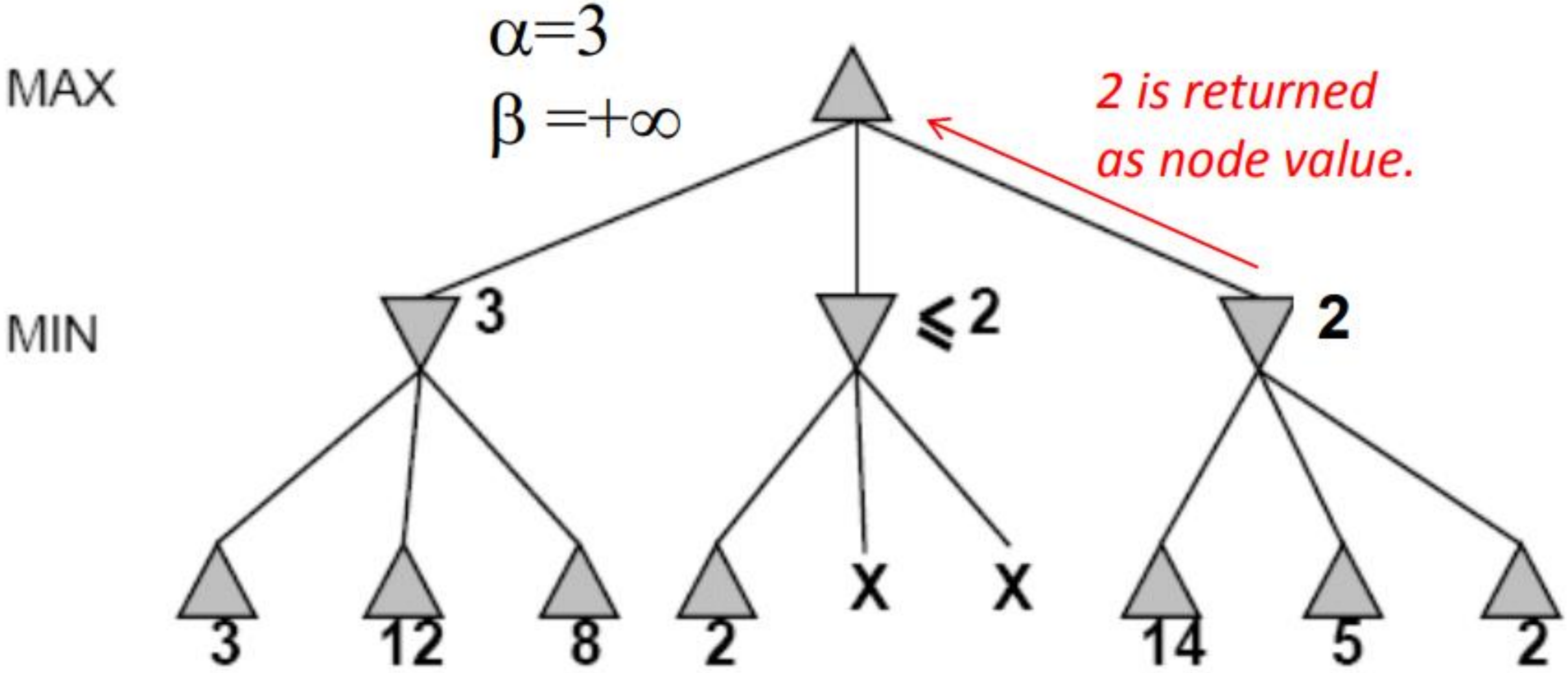
α, β, passed to children



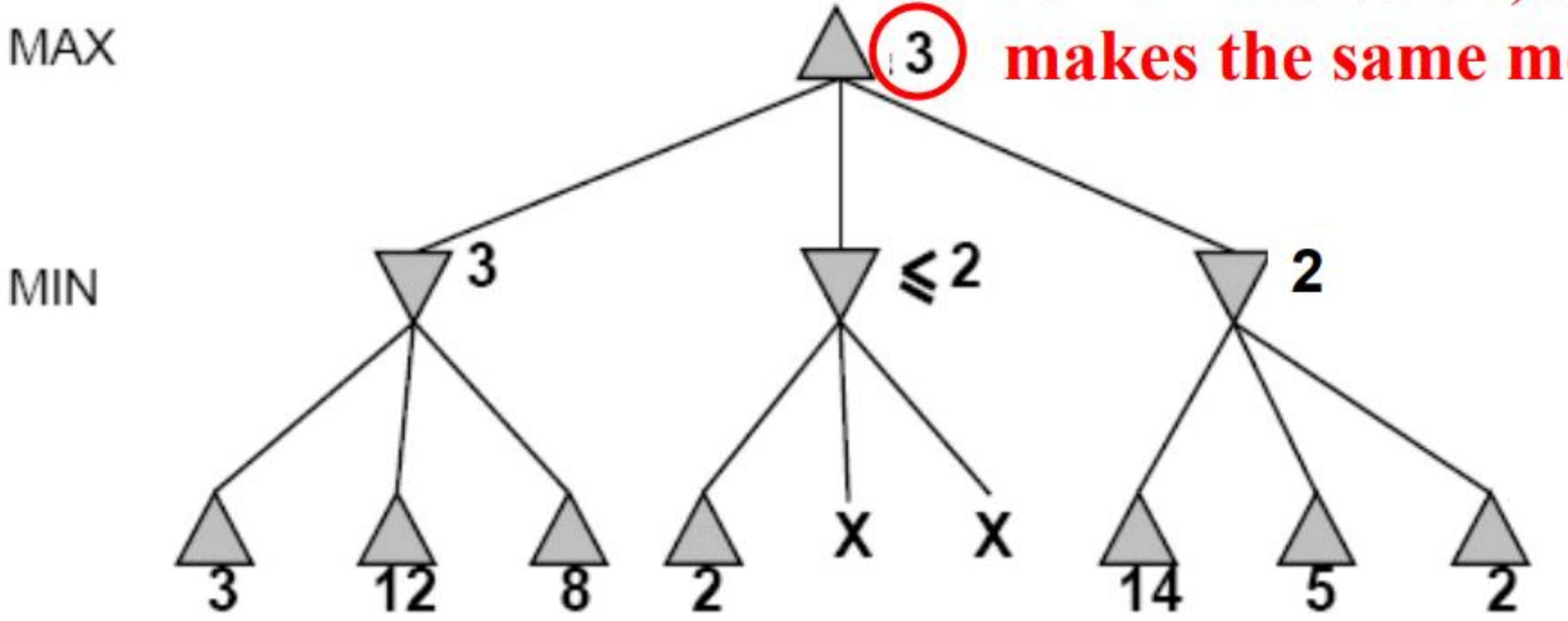
MIN updates β, based on children.



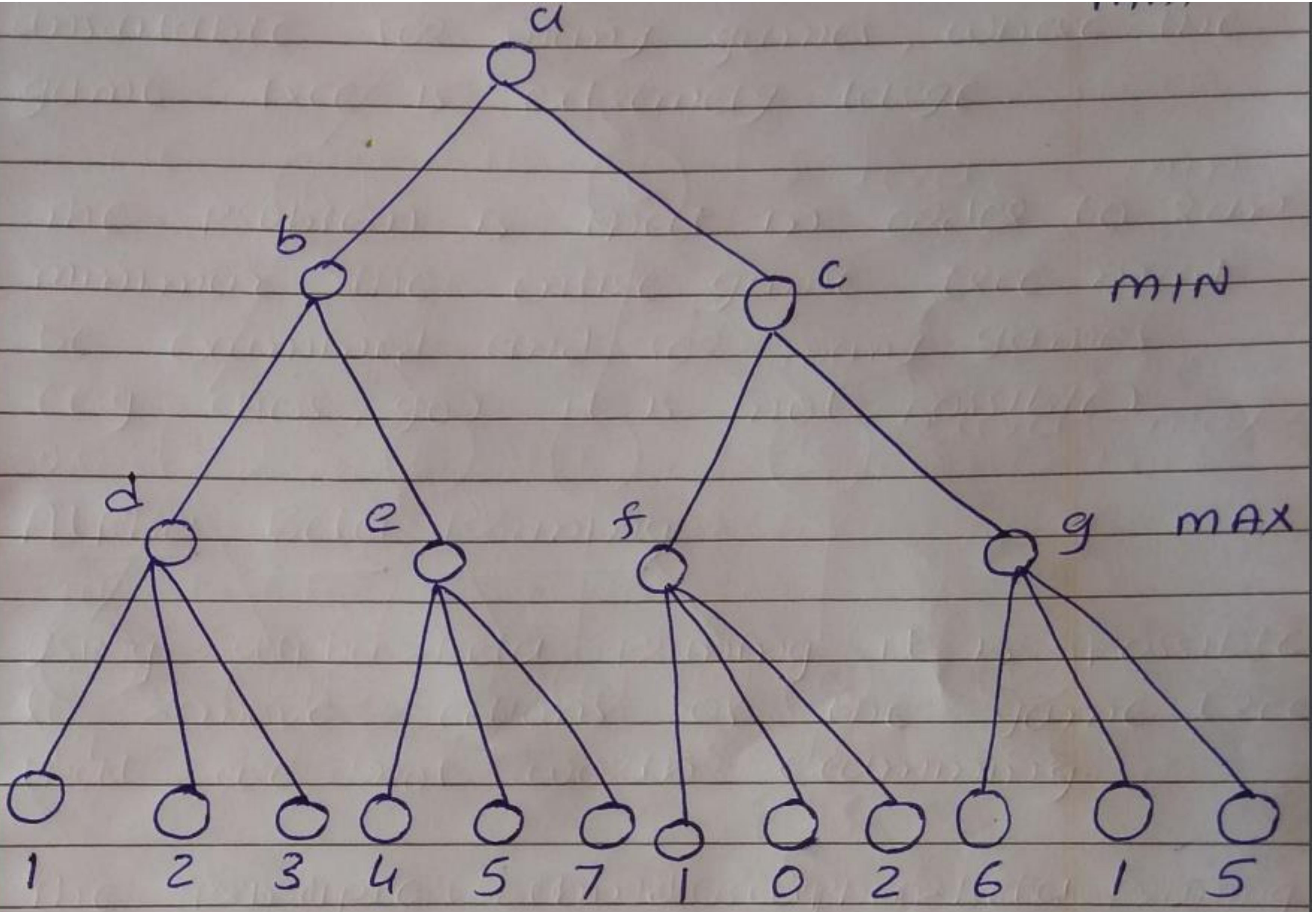
MIN updates β, based on children.

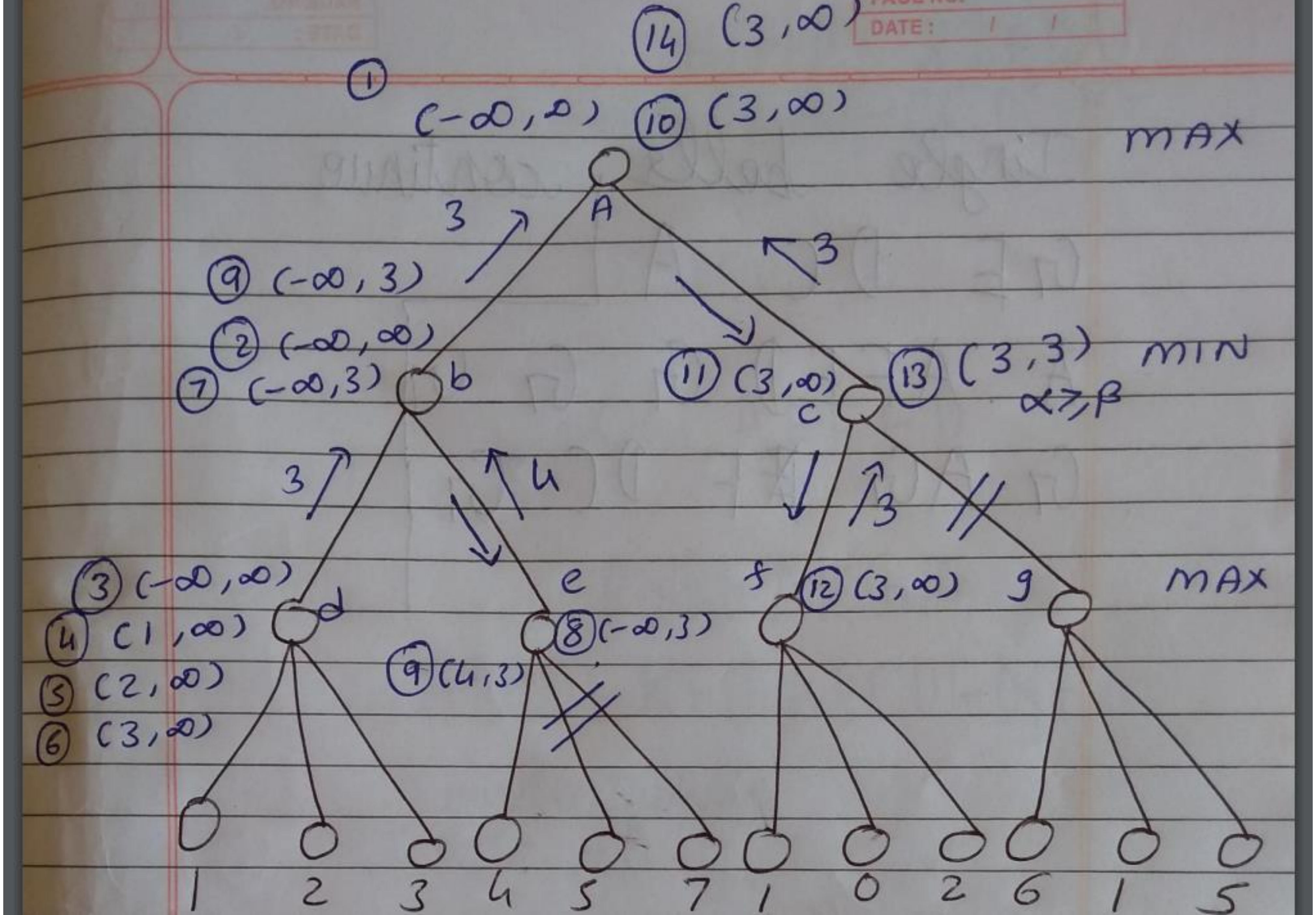


2 is returned as node value.

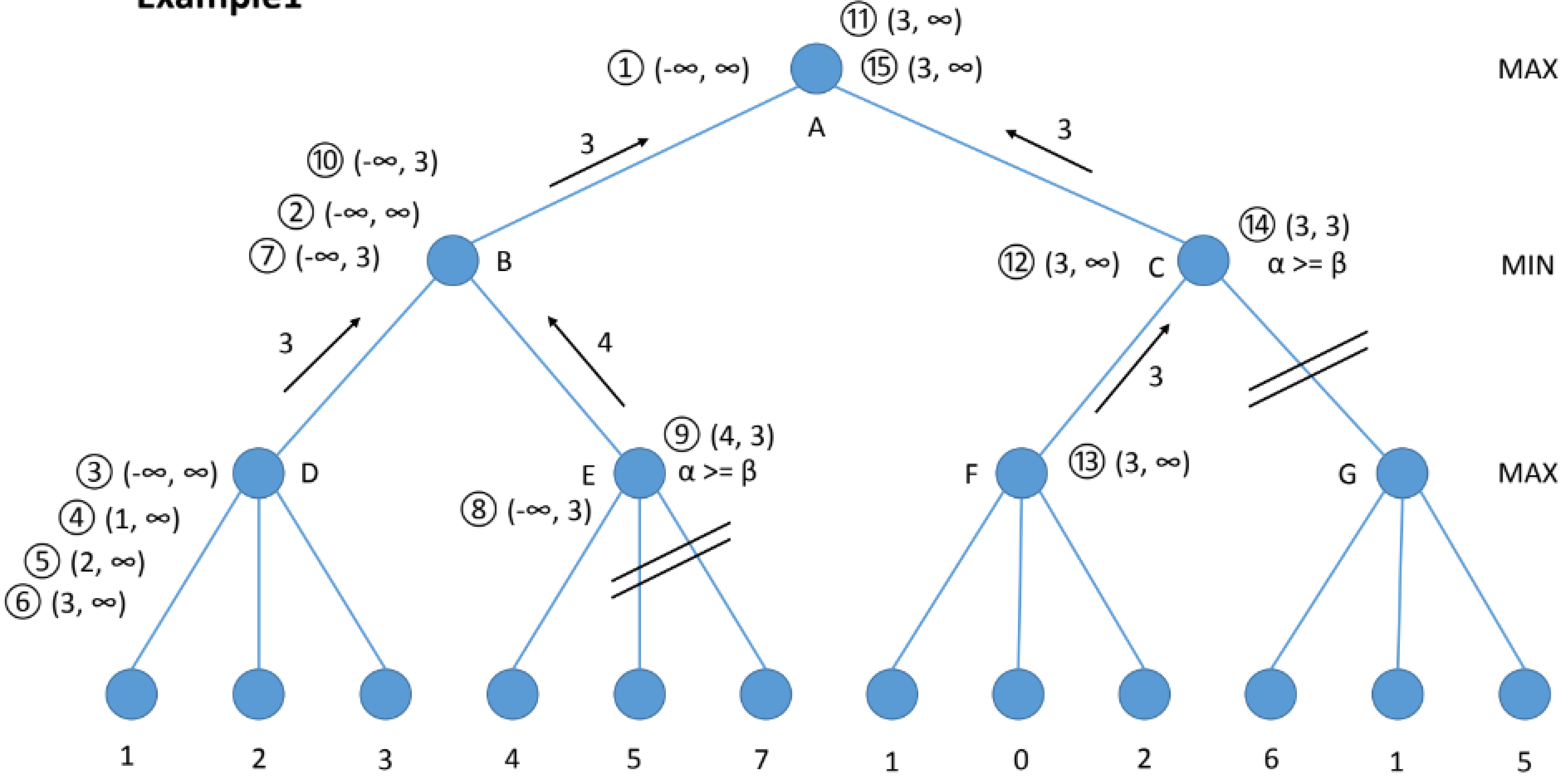


Max calculates the same node value, and makes the same move!

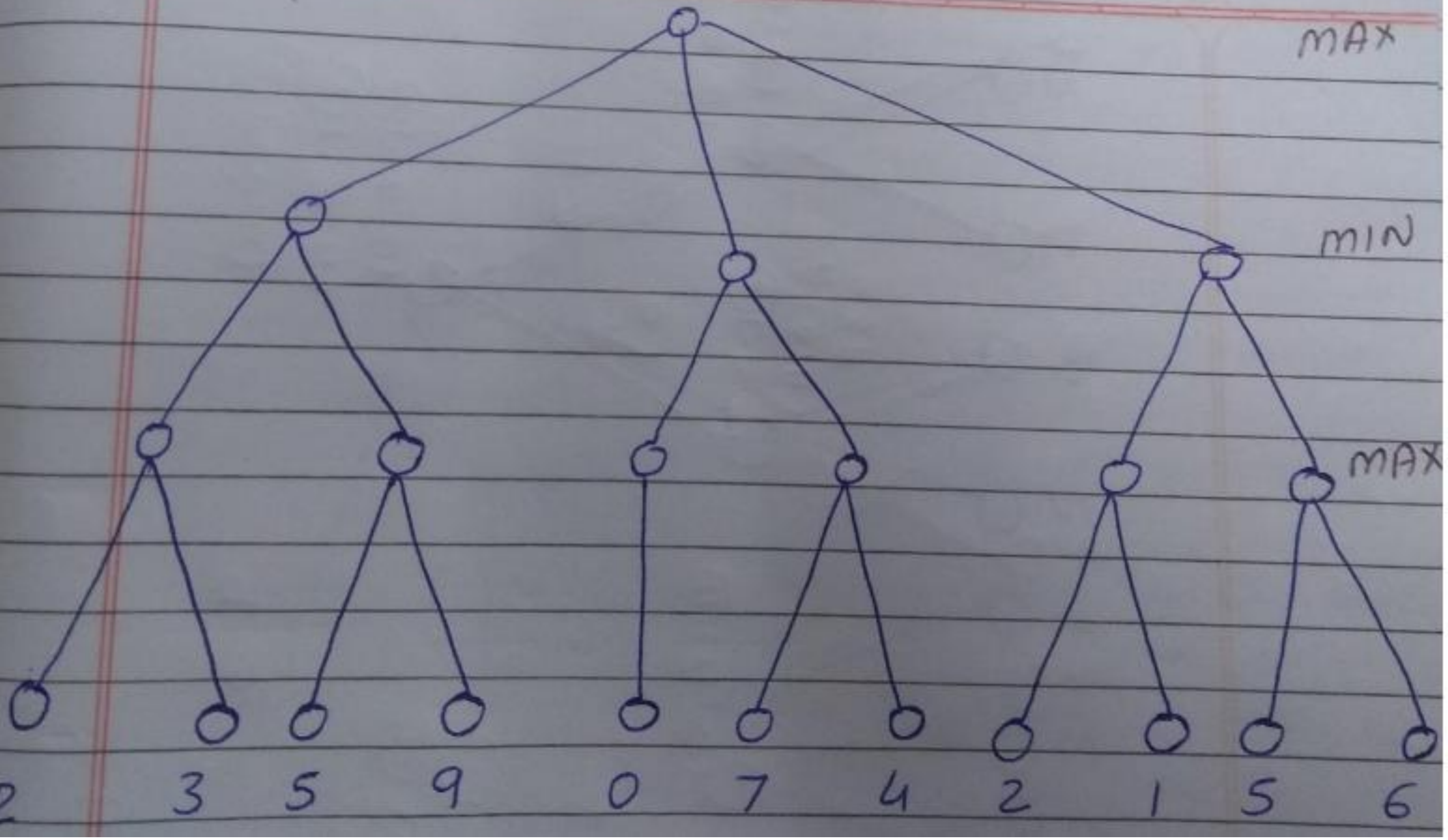

a
b
c
MIN
d
e
f
g
MAX
1
2
3
4
5
7
1
0
2
6
1
5


(14) (3,∞)
(1) (−∞,∞)
(10) (3,∞)
MAX
A
3
3
(9) (−∞,3)
(2) (−∞,∞)
(7) (−∞,3)
b
(11) (3,∞)
c
(13) (3,3)
α≥β
MIN
3
4
3
(3) (−∞,∞)
(4) (1,∞)
(5) (2,∞)
(6) (3,∞)
d
e
(8) (−∞,3)
(9) (4,3)
f
(12) (3,∞)
g
MAX
1 2 3 4 5 7 1 0 2 6 1 5

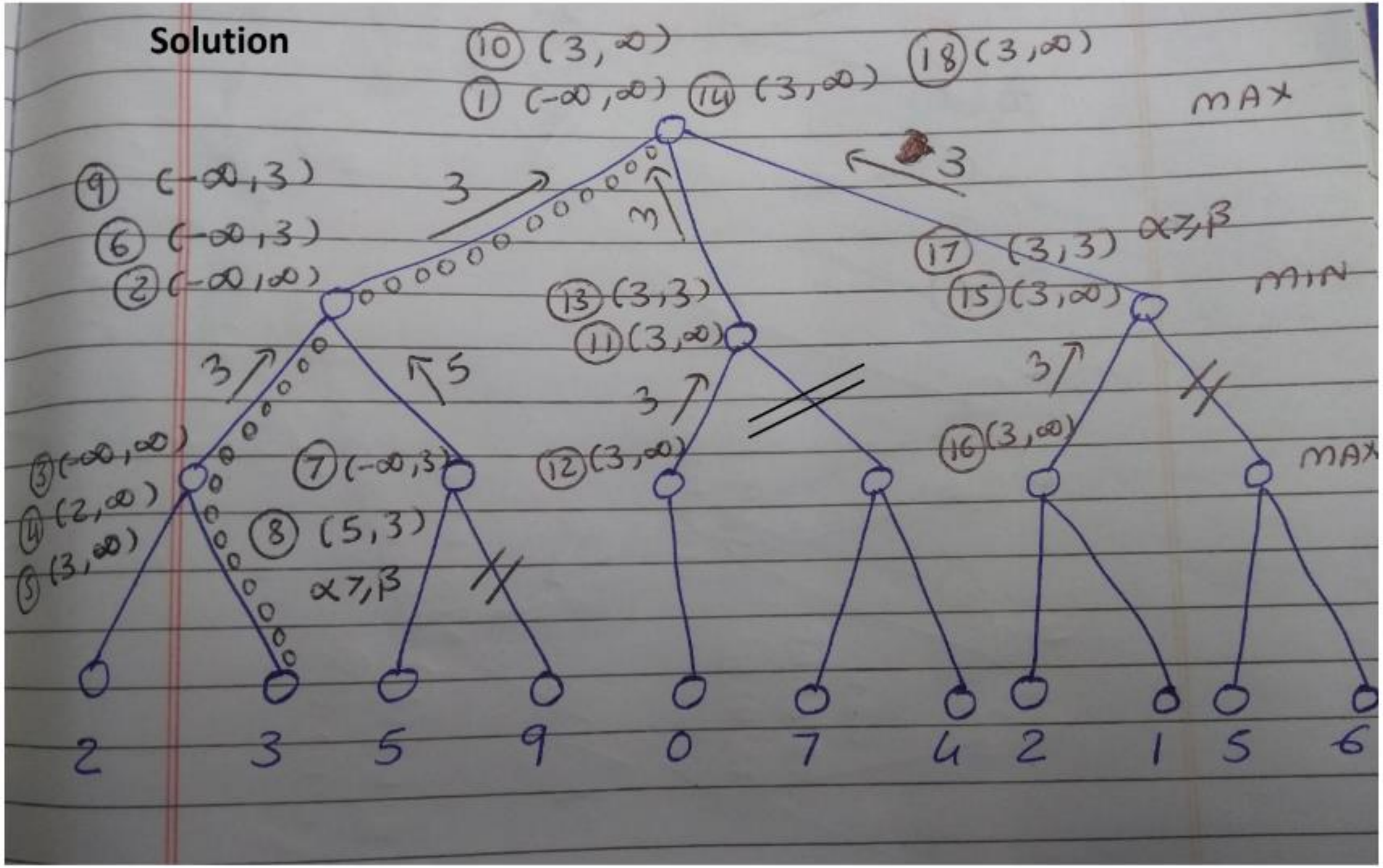
Example1


⑪ (3, ∞)
① (-∞, ∞)
⑮ (3, ∞)
MAX
A
3
3
⑩ (-∞, 3)
② (-∞, ∞)
⑦ (-∞, 3)
B
⑭ (3, 3)
⑫ (3, ∞)
C
α >= β
MIN
3
4
3
⑨ (4, 3)
α >= β
③ (-∞, ∞)
D
E
F
⑬ (3, ∞)
G
MAX
④ (1, ∞)
⑧ (-∞, 3)
⑤ (2, ∞)
⑥ (3, ∞)
1
2
3
4
5
7
1
0
2
6
1
5

Example-2

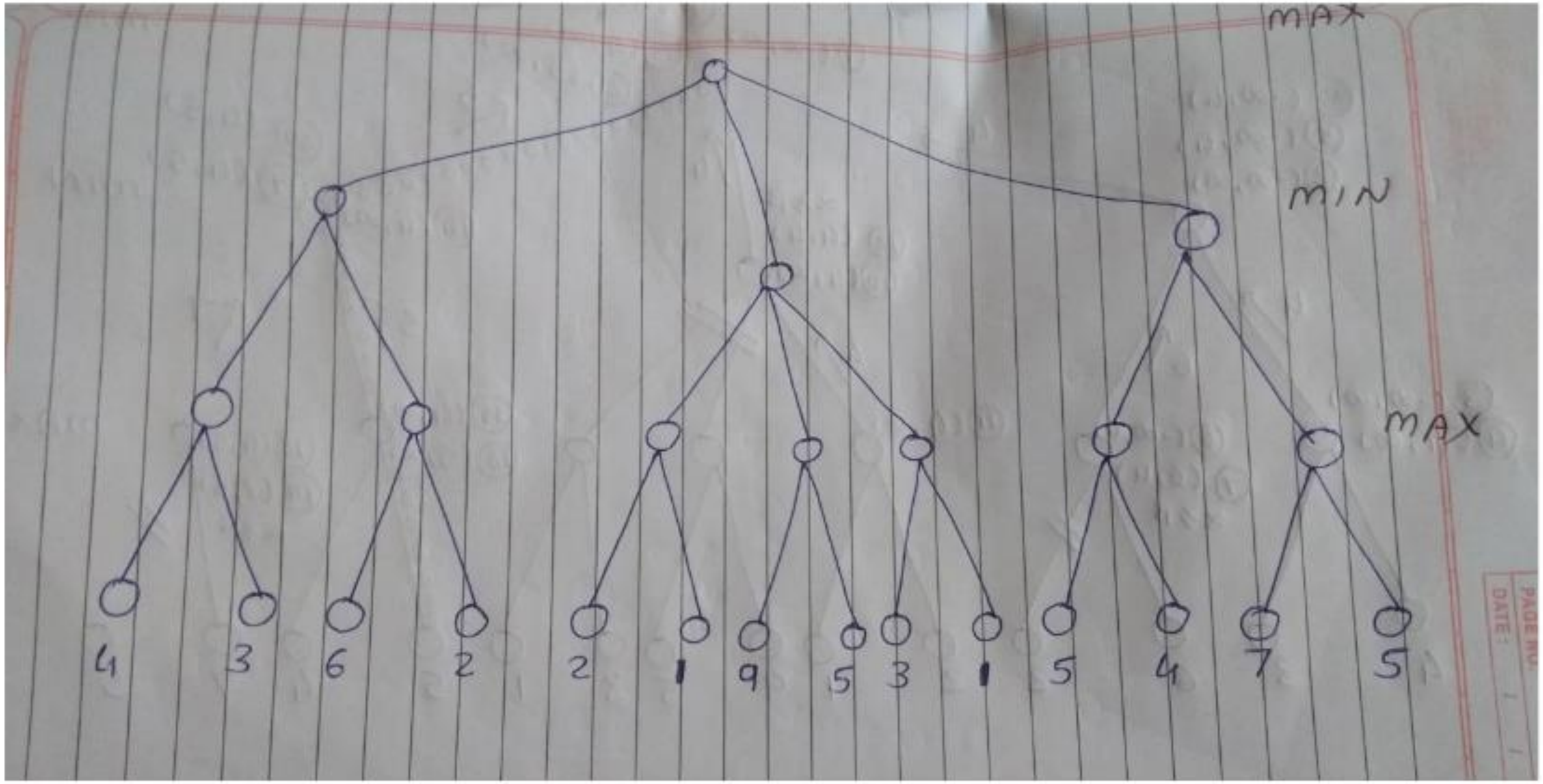


Solution

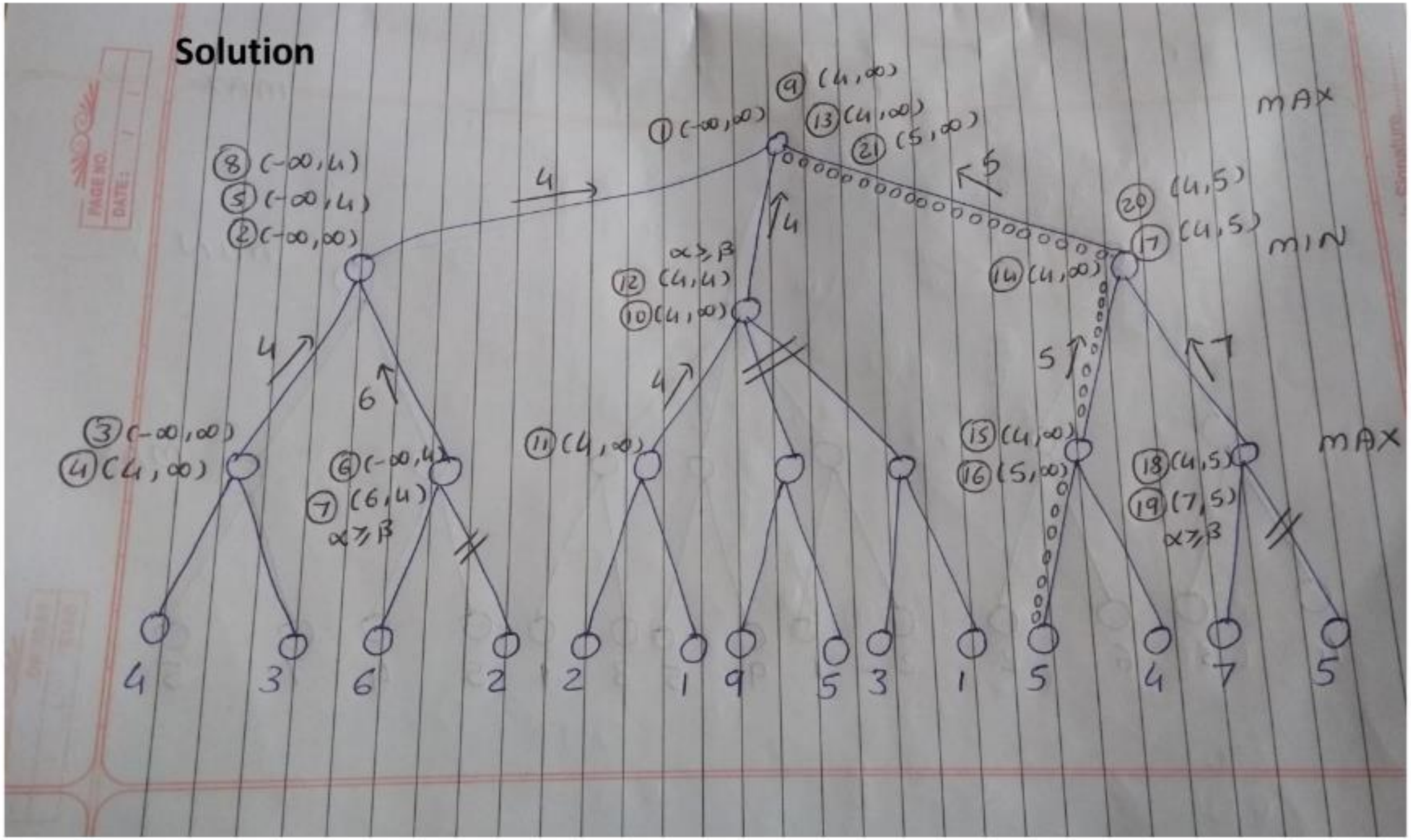


⑩ (3, ∞)
① (-∞, ∞)
⑭ (3, ∞)
⑱ (3, ∞)
MAX
⑨ (-∞, 3)
⑥ (-∞, 3)
② (-∞, ∞)
3
3
3
⑰ (3, 3) α≥β
⑮ (3, ∞)
MIN
⑬ (3, 3)
⑪ (3, ∞)
3
5
3
3
③ (-∞, ∞)
④ (2, ∞)
⑤ (3, ∞)
⑦ (-∞, 3)
⑧ (5, 3)
α≥β
⑫ (3, ∞)
⑯ (3, ∞)
MAX
2
3
5
9
0
7
4
2
1
5
6

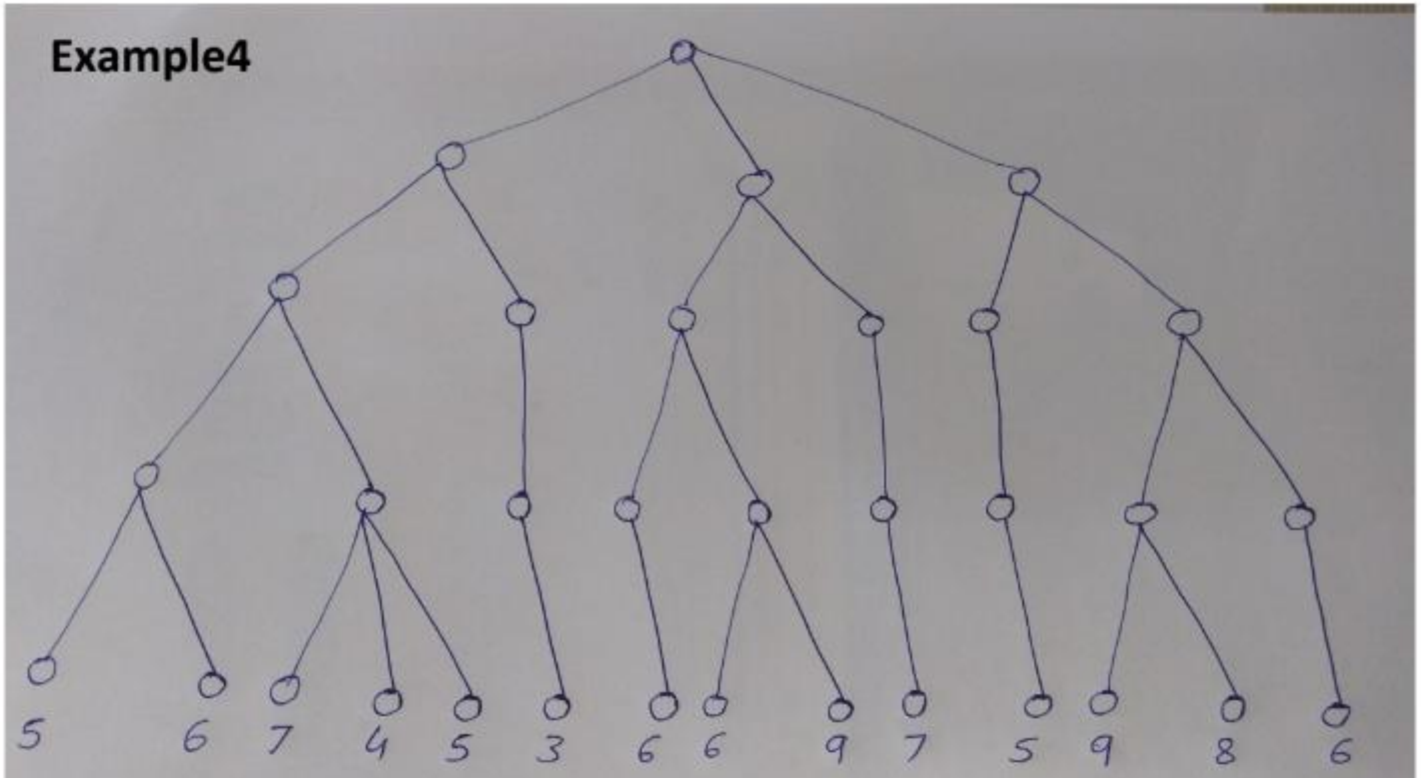
Example3



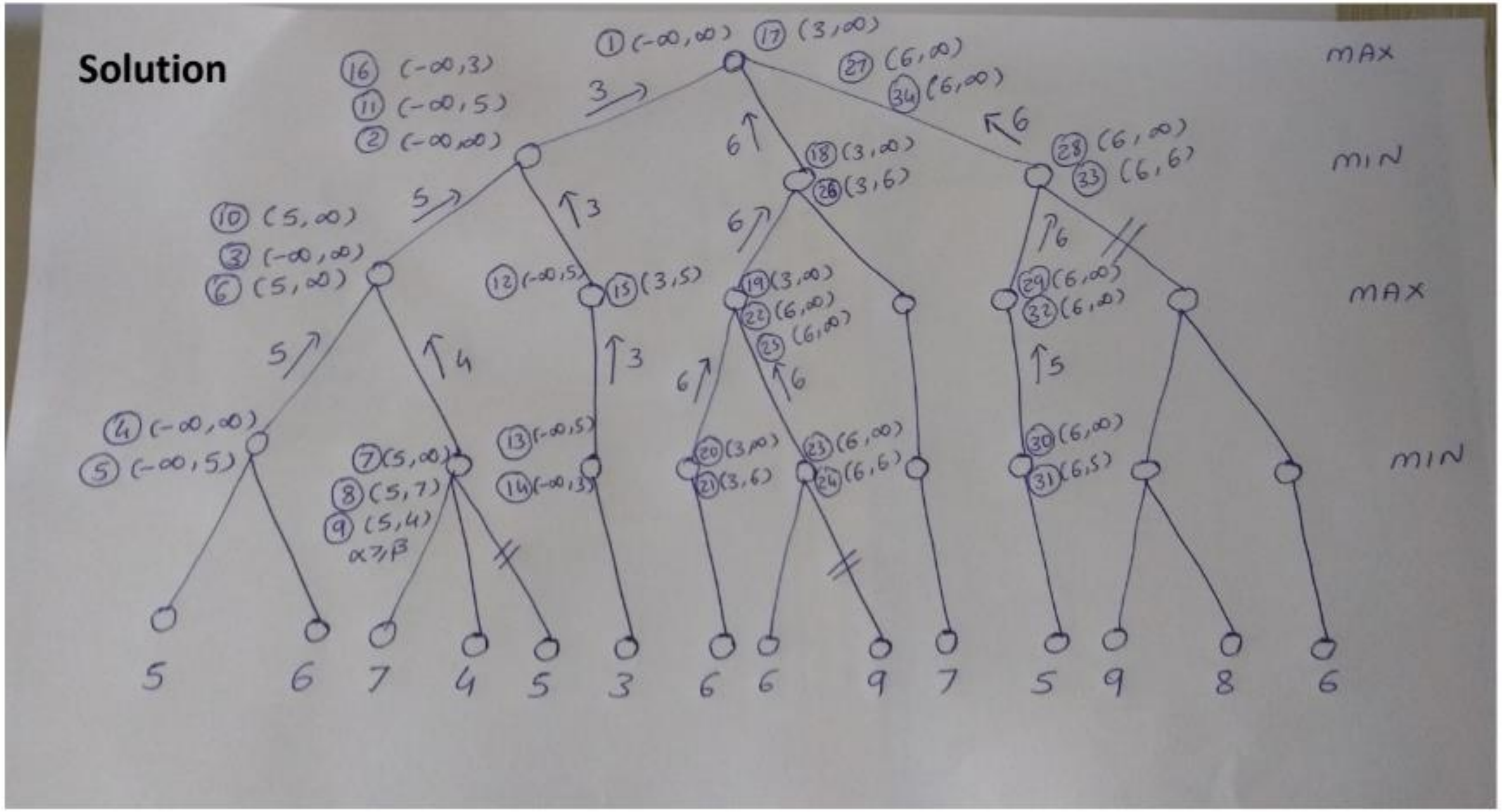
Solution

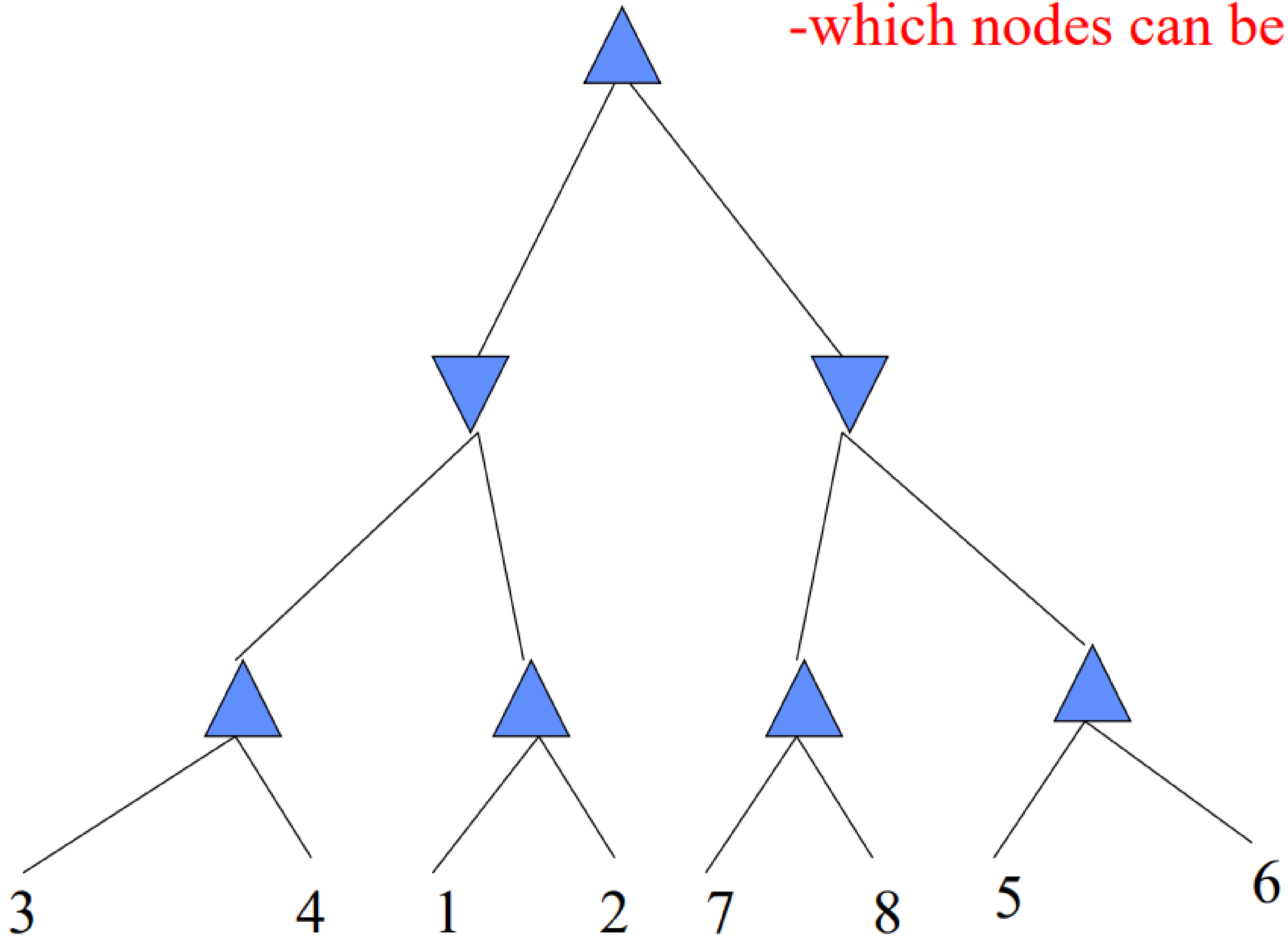

MAX
MIN
MAX
① (−∞,∞)
⑨ (4,∞)
⑬ (4,∞)
㉑ (5,∞)
⑧ (−∞,4)
⑤ (−∞,4)
② (−∞,∞)
4 →
↑4
↖5
α≥β
⑫ (4,4)
⑩ (4,∞)
⑳ (4,5)
⑰ (4,5)
⑭ (4,∞)
4↗
6↑
4↗
5↑
↖7
③ (−∞,∞)
④ (4,∞)
⑥ (−∞,4)
⑦ (6,4)
α≥β
⑪ (4,∞)
⑮ (4,∞)
⑯ (5,∞)
⑱ (4,5)
⑲ (7,5)
α≥β
4
3
6
2
2
1
9
5
3
1
5
4
7
5

Example4



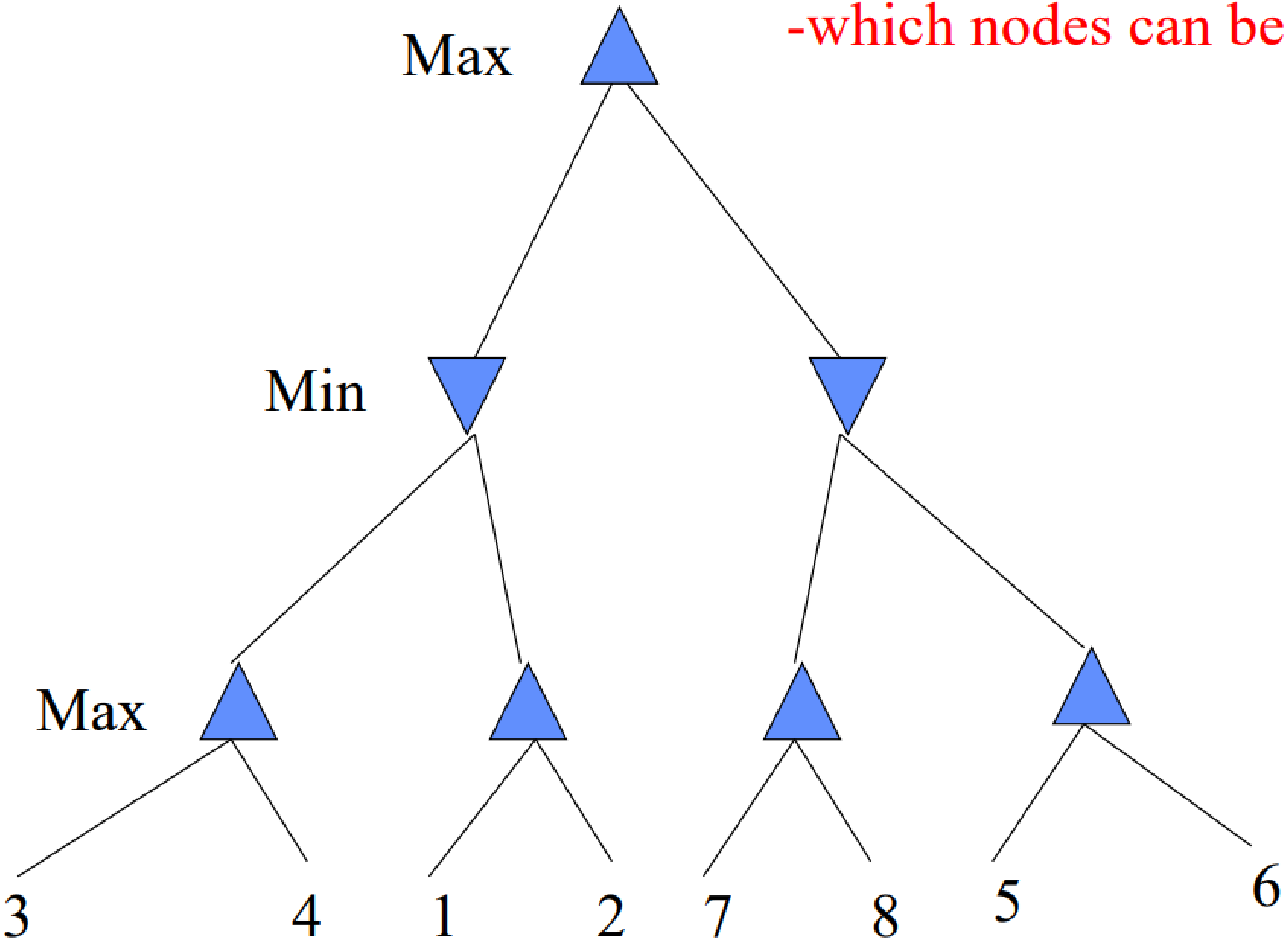
Solution


1 (-∞,∞)
17 (3,∞)
MAX
16 (-∞,3)
11 (-∞,5)
2 (-∞,∞)
3 →
27 (6,∞)
34 (6,∞)
6 ↑
↖ 6
18 (3,∞)
26 (3,6)
28 (6,∞)
33 (6,6)
MIN
10 (5,∞)
3 (-∞,∞)
6 (5,∞)
5 →
↑ 3
6 ↗
↗ 6
12 (-∞,5)
15 (3,5)
19 (3,∞)
22 (6,∞)
25 (6,∞)
29 (6,∞)
32 (6,∞)
MAX
5 ↗
↑ 4
↑ 3
6 ↗
↑ 6
↑ 5
4 (-∞,∞)
5 (-∞,5)
7 (5,∞)
8 (5,7)
9 (5,4)
α ≥ β
13 (-∞,5)
14 (-∞,3)
20 (3,∞)
21 (3,6)
23 (6,∞)
24 (6,6)
30 (6,∞)
31 (6,5)
MIN
5
6
7
4
5
3
6
6
9
7
5
9
8
6



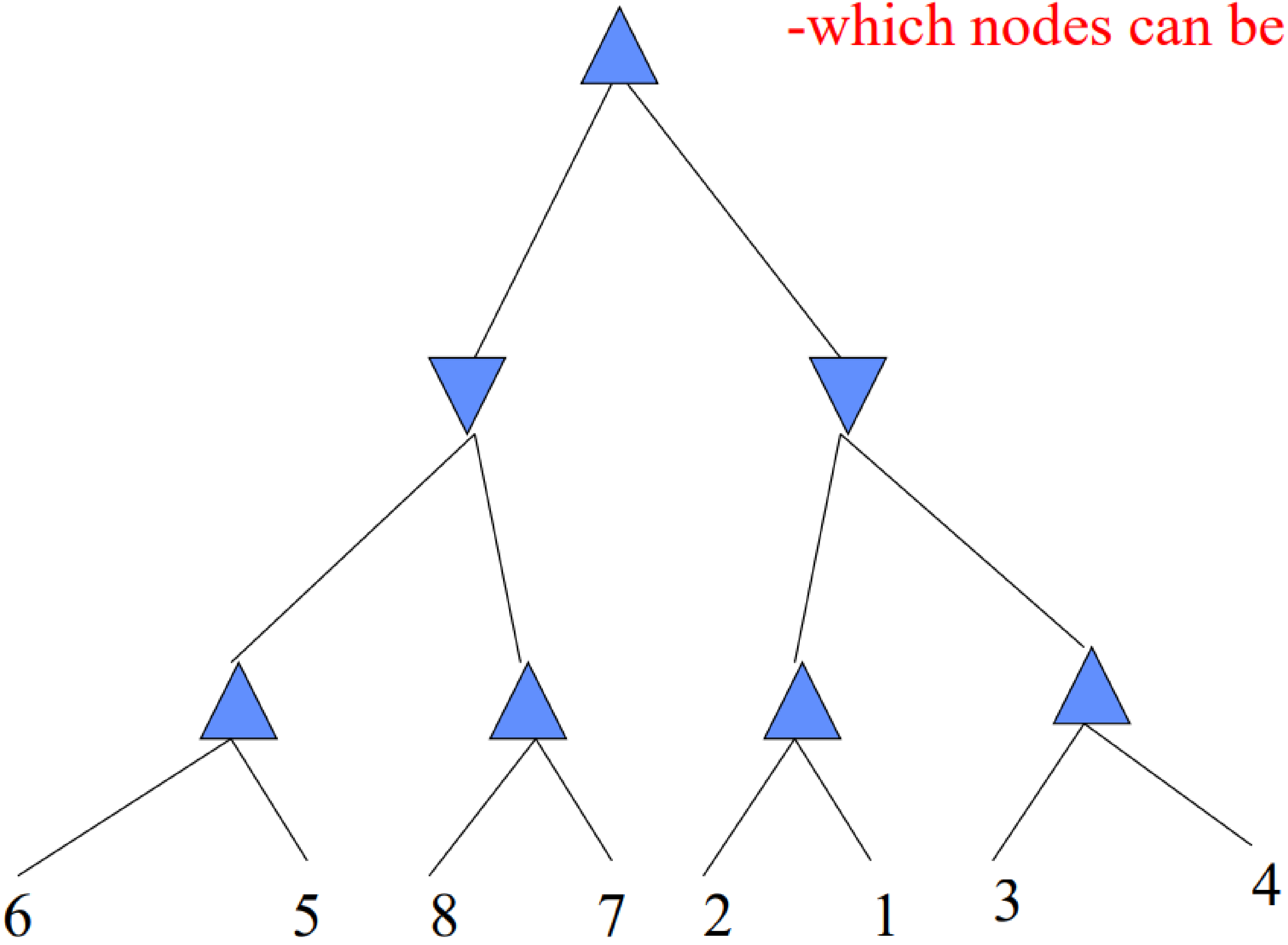
-which nodes can be pruned?

-which nodes can be pruned?

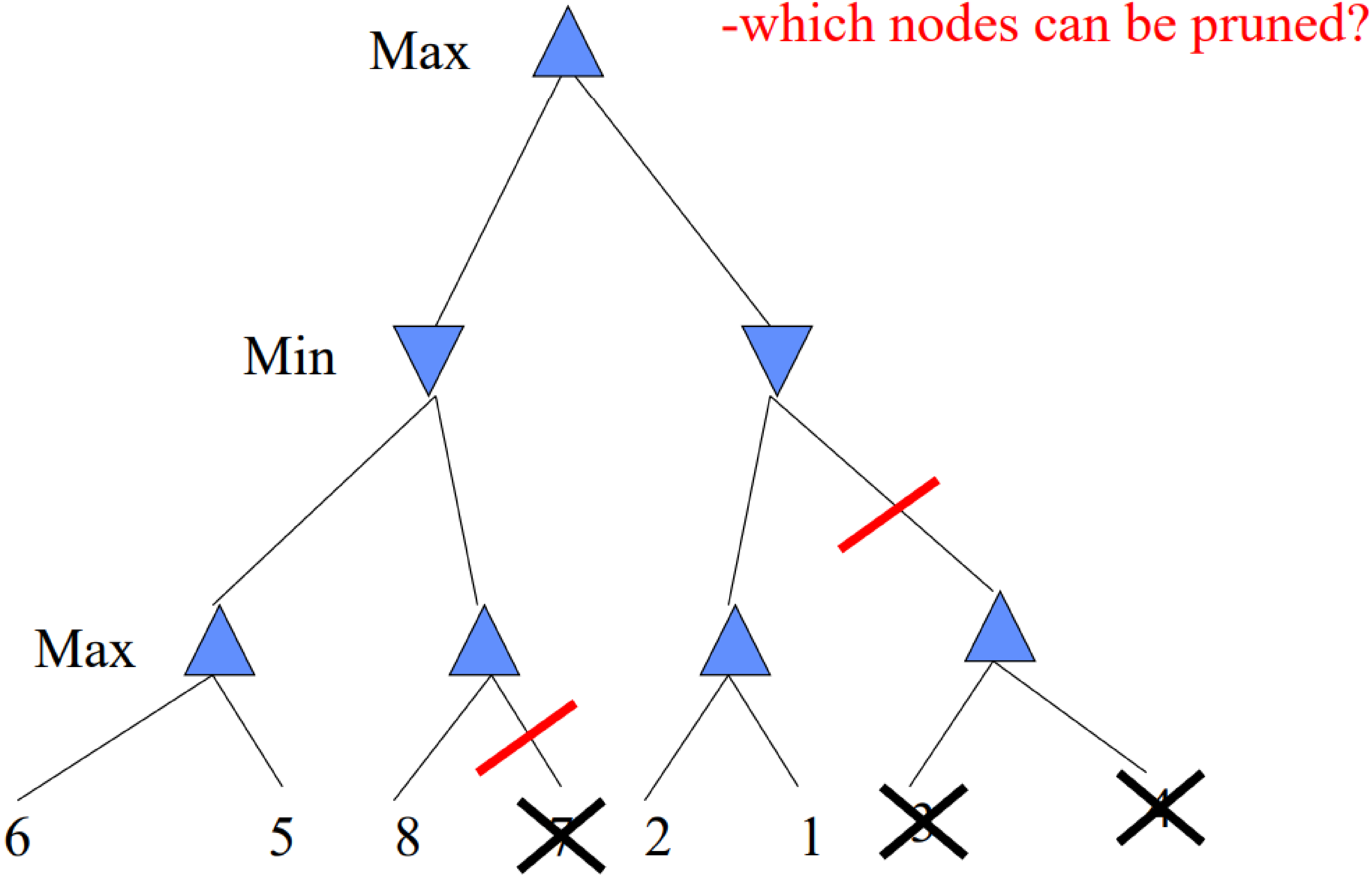


Answer: NONE! Because the most favorable nodes for both are explored last (i.e., in the diagram, are on the right-hand side).

(the exact mirror image of the first example)



-which nodes can be pruned?



-which nodes can be pruned?

Answer: LOTS! Because the most favorable nodes for both are explored first (i.e., in the diagram, are on the left-hand side).