Alpha-Beta Pruning Implemenation

```
AlphaBeta(pos,alpha,beta): #return best move for player(pos)
                           #and MAX's value for pos
  best_move = None
  if terminal(pos):
    return best_move, utility(pos)
  if player(pos) == MAX: value = -infinity
  if player(pos) == MIN: value = infinity
  for move in actions(pos):
    nxt_pos = result(pos, move)
    nxt_val,nxt_move = AlphaBeta(nxt_pos, alpha, beta)
    if player == MAX:
      if value < nxt_val: value, best_move = nxt_val, move
      if value >= beta: return best_move, value
      alpha = max(alpha, value)
    if player == MIN:
      if value > nxt_value: value, best_move = nxt_val, move
      if value <= alpha: return best_move, value
      beta = min(beta, value)
  return best_move, value
```

Alpha-Beta Pruning Implemenation

```
AlphaBeta(pos,alpha,beta): #return best move for player(pos)
                           #and MAX's value for pos
```

```
for move in actions(pos):
  nxt_pos = result(pos, move)
  nxt_val,nxt_move = AlphaBeta(nxt_pos, alpha, beta)
```

First child node gets passed the same alpha and beta values.

Alpha-Beta Pruning Implemenation

```
AlphaBeta(pos,alpha,beta): #return best move for player(pos)
                           #and MAX's value for pos
```

```
for move in actions(pos):
  nxt_pos = result(pos, move)
  nxt_val,nxt_move = AlphaBeta(nxt_pos, alpha, beta)
  if player == MAX:
    if value < nxt_val: value, best_move = nxt_val, move
    if value >= beta: return best_move, value
    alpha = max(alpha, value)
```

As we iterate through children of MAX node only alpha gets updated (so children get passed updated alpha values)

Alpha-Beta Pruning Implemenation

```
AlphaBeta(pos,alpha,beta): #return best move for player(pos)
                           #and MAX's value for pos
```

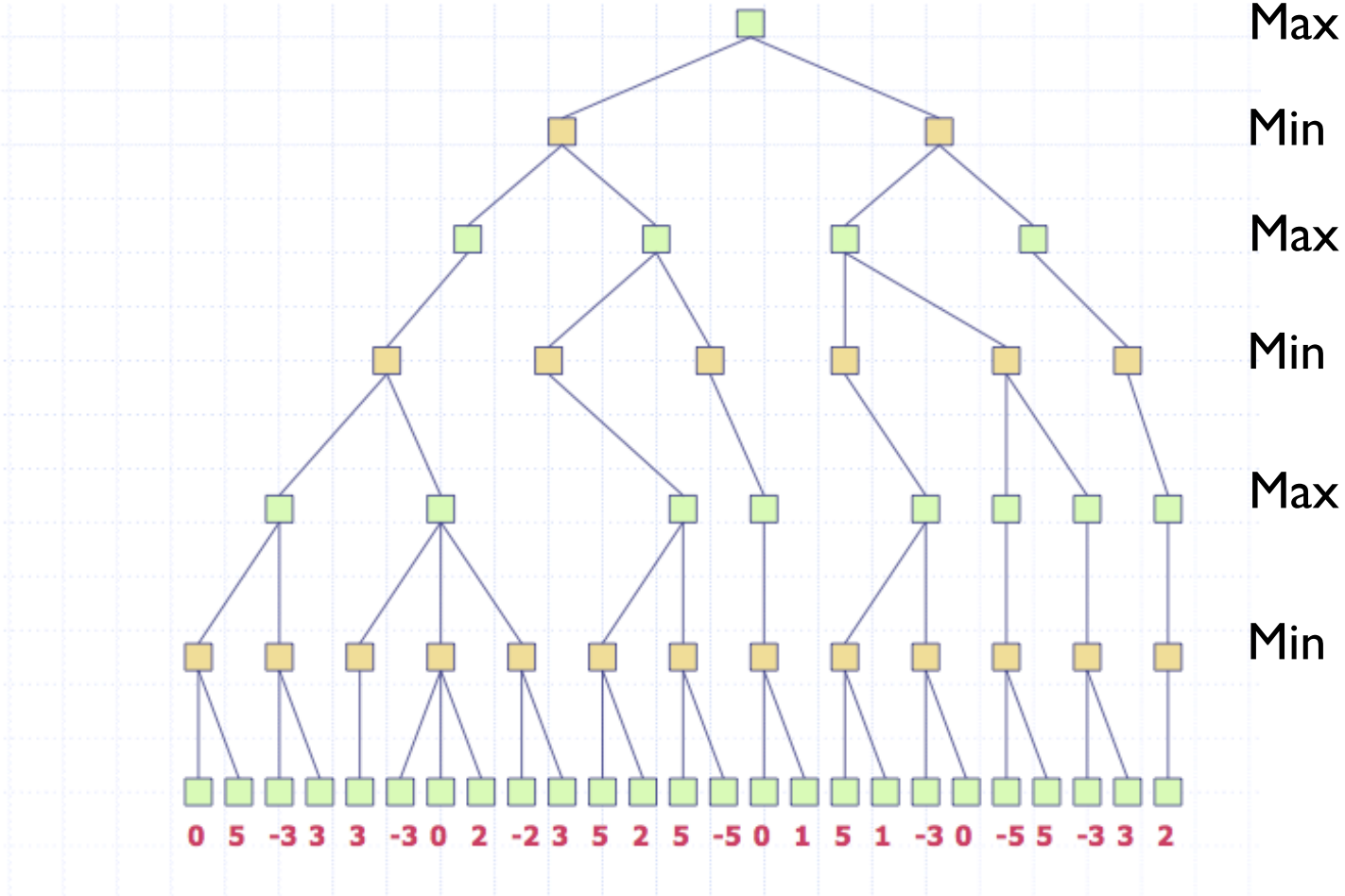
```
  for move in actions(pos):
    nxt_pos = result(pos, move)
    nxt_val,nxt_move = AlphaBeta(nxt_pos, alpha, beta)
```

As we iterate through children of MIN node only beta gets updated (so children get passed updated beta values)

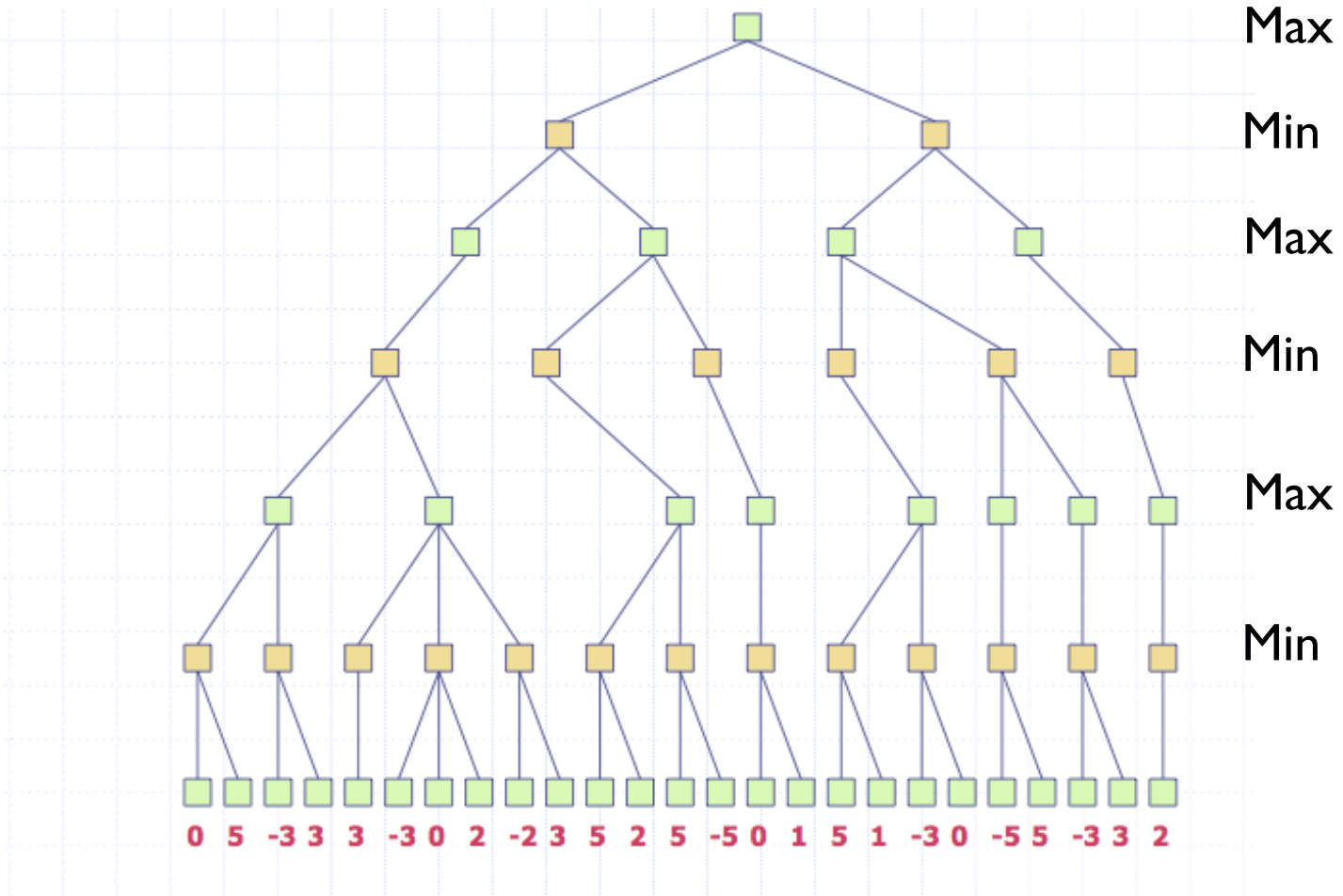
```
    if player == MIN:
      if value > nxt_value: value, best_move = nxt_val, move
      if value <= alpha: return best_move, value
      beta = min(beta, value)
  return best_move, value
```

Alpha-Beta Cuts

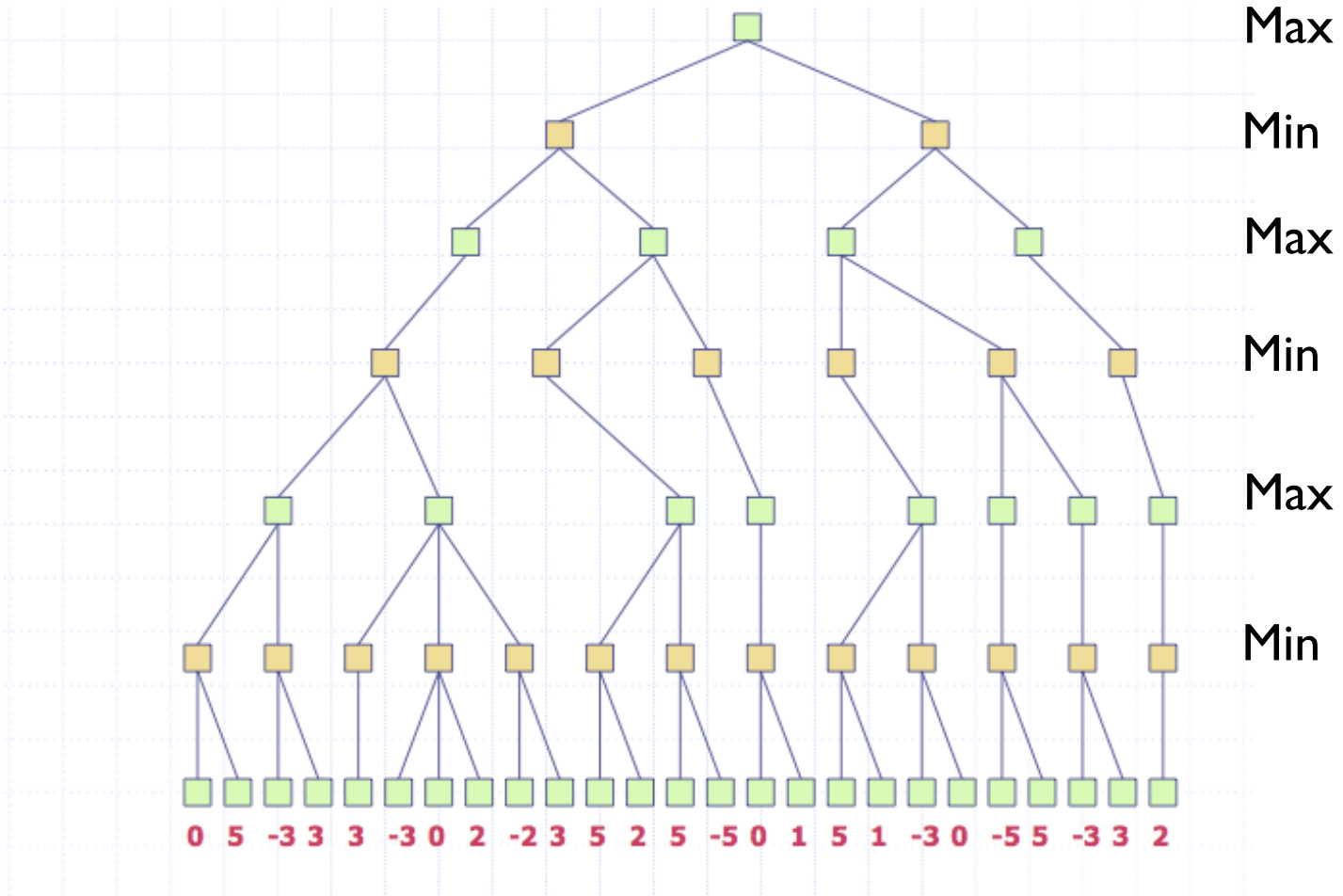
- What cuts will alpha-beta make?
- To see this perhaps easier to consider one call, and not considering the internal details of the recursive calls.
- Of course we would have to execute the recursive calls to obtain the values needed in deciding the cuts!
- But for understanding the cuts we can cheat by backing up the terminal values so that we know what value the recursive call would return.
- Of course, in an implementation, the recursive call would compute the values—we wouldn't back them up.


Max
Min
Max
Min
Max
Min
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

First we manually backup the minimax values so we will know what the recursive calls would return (if we had done it).

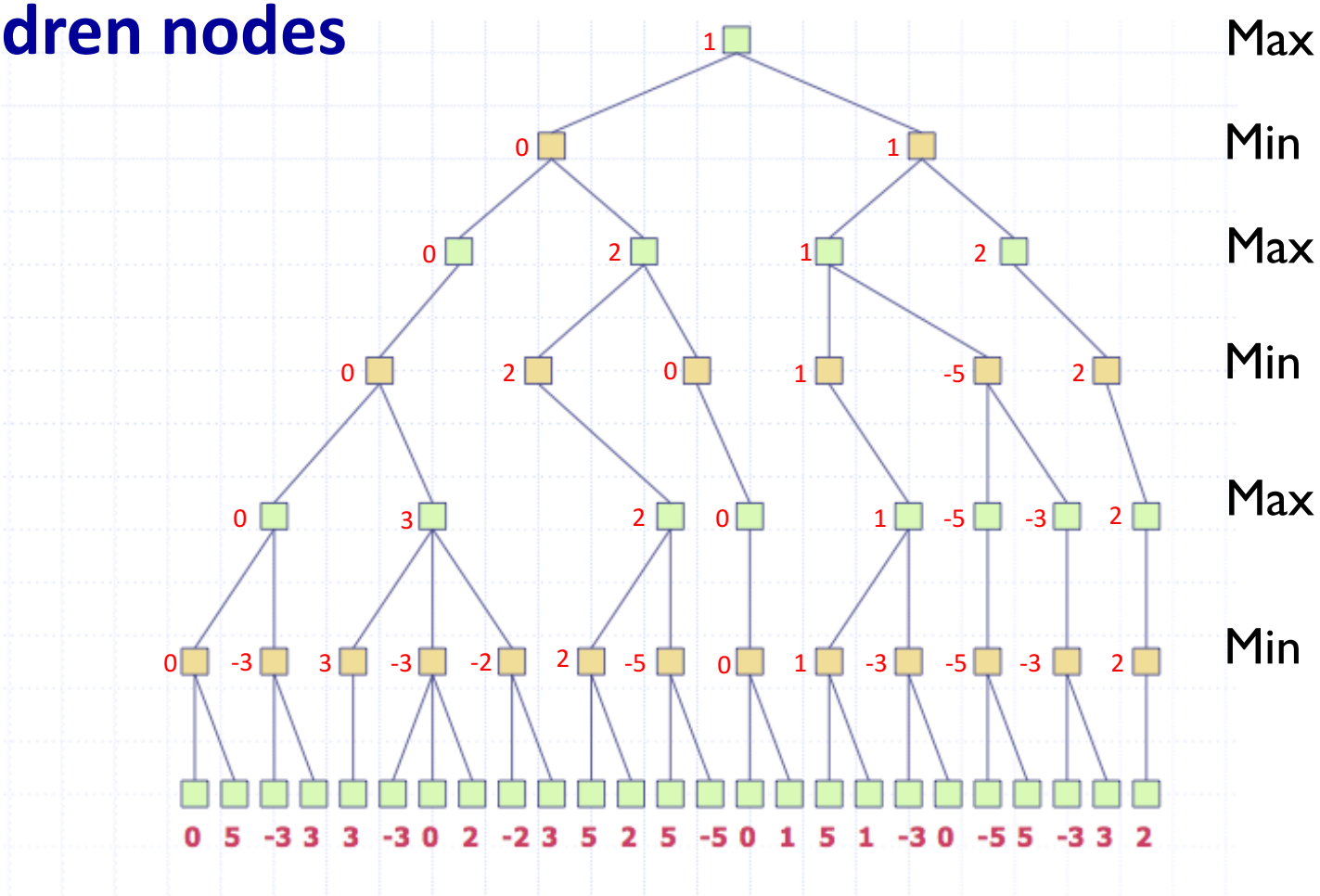


Terminal Nodes have a utility specific to the game

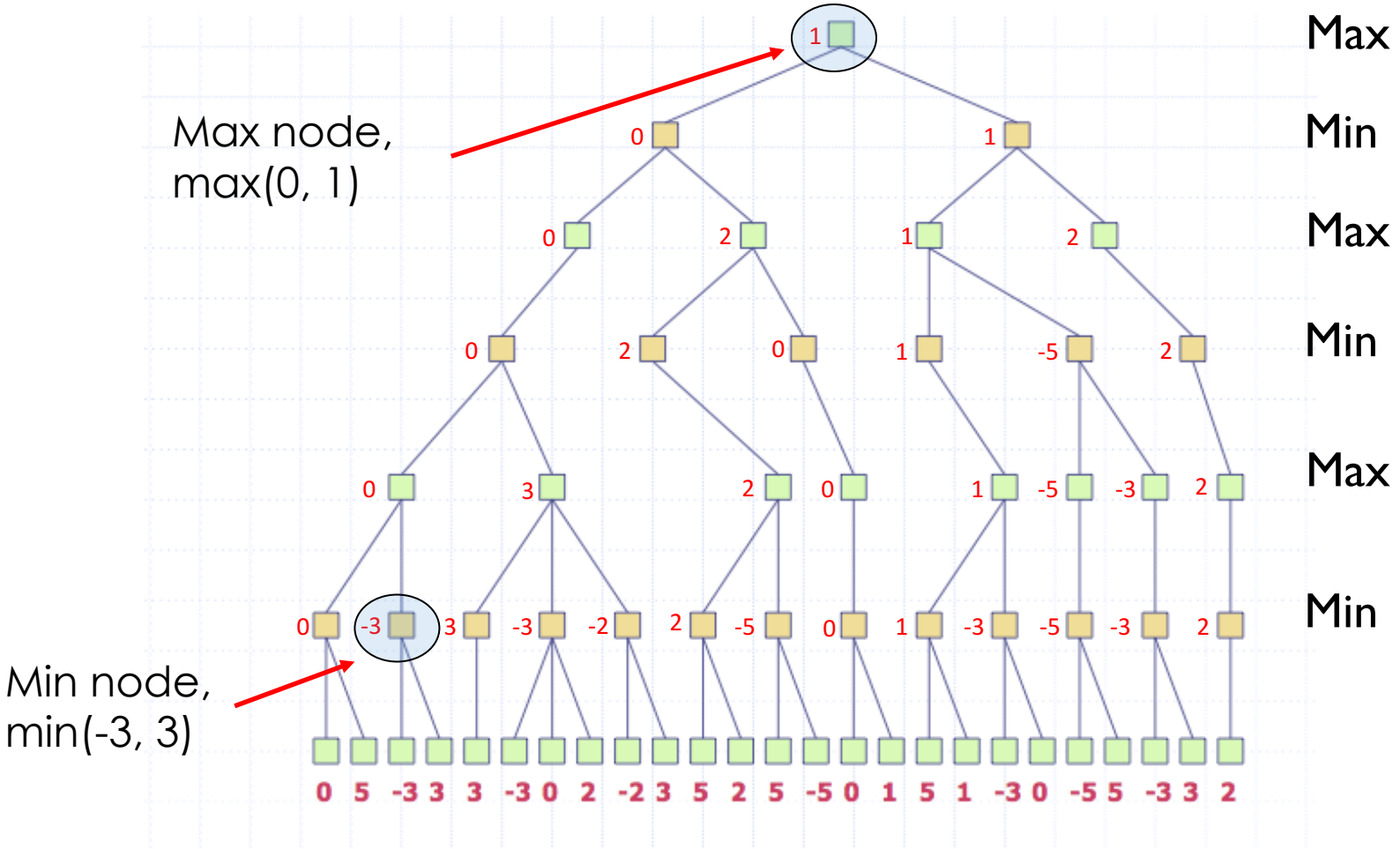


Min nodes have a value equal to the min of their children nodes

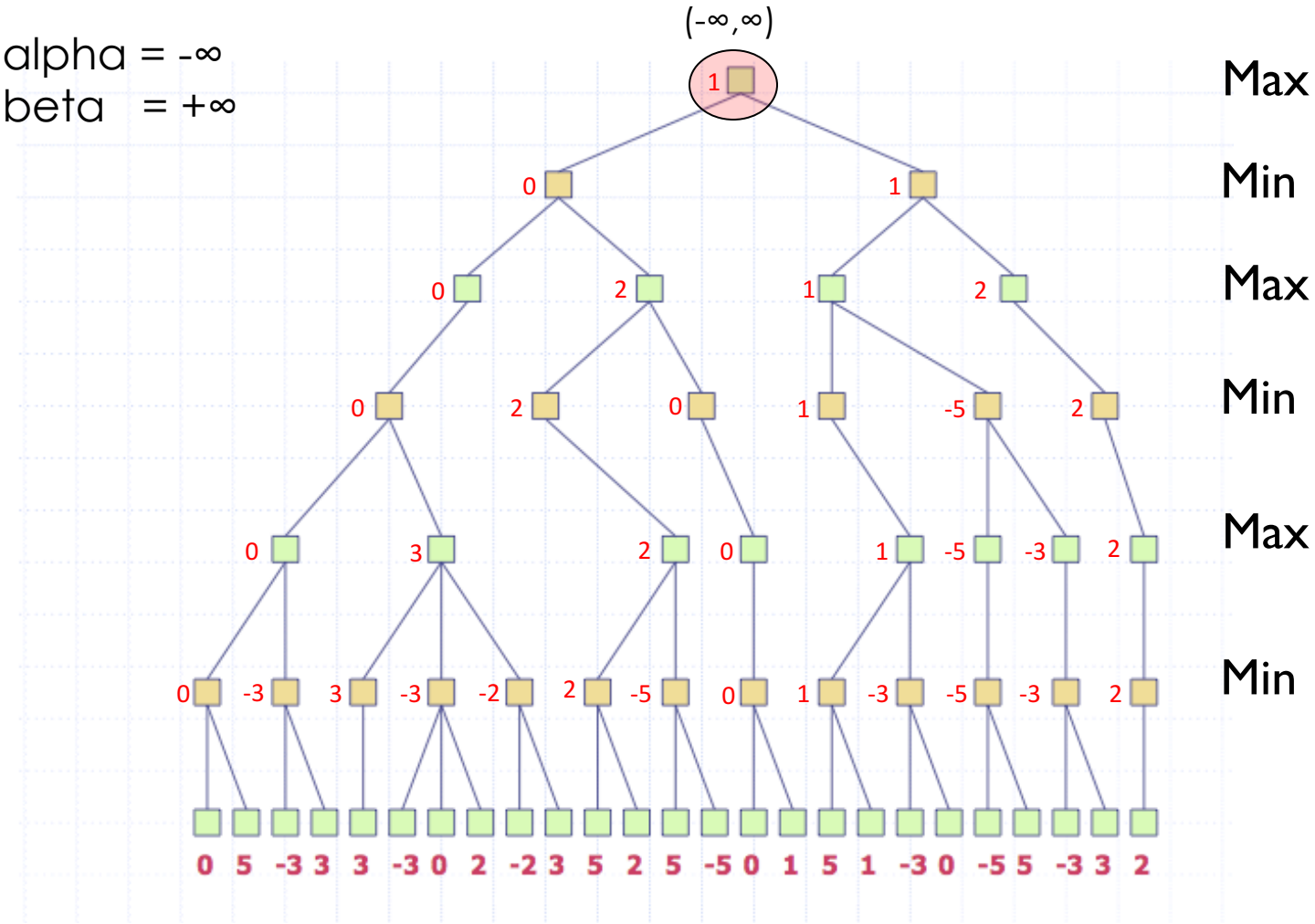
Max nodes have a value equal to the max of their children nodes



Min nodes have a value equal to the min of their children nodes
Max nodes have a value equal to the max of their children nodes



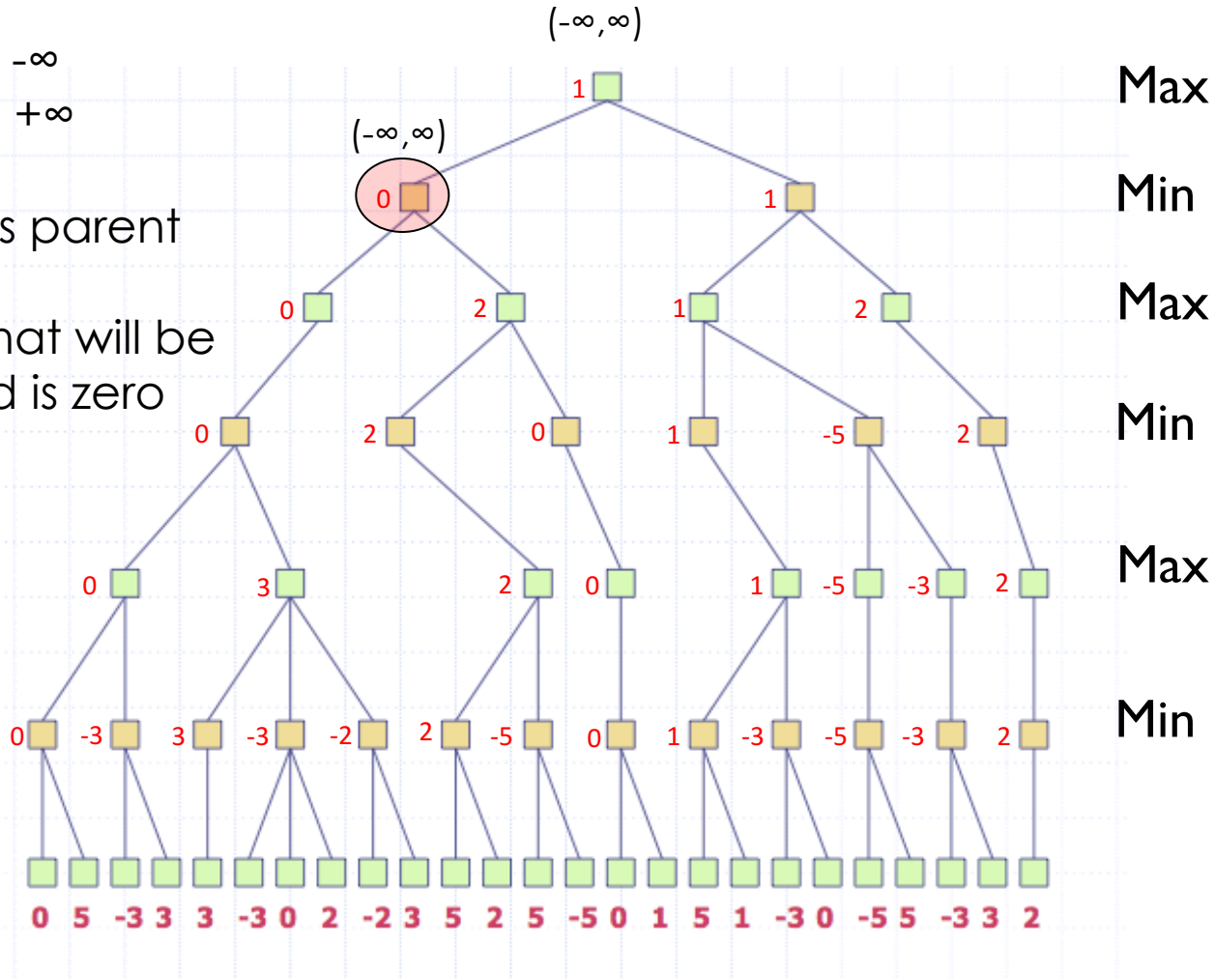
alpha = -∞
beta = +∞



alpha = -∞
beta = +∞

Same as parent

Value that will be returned is zero

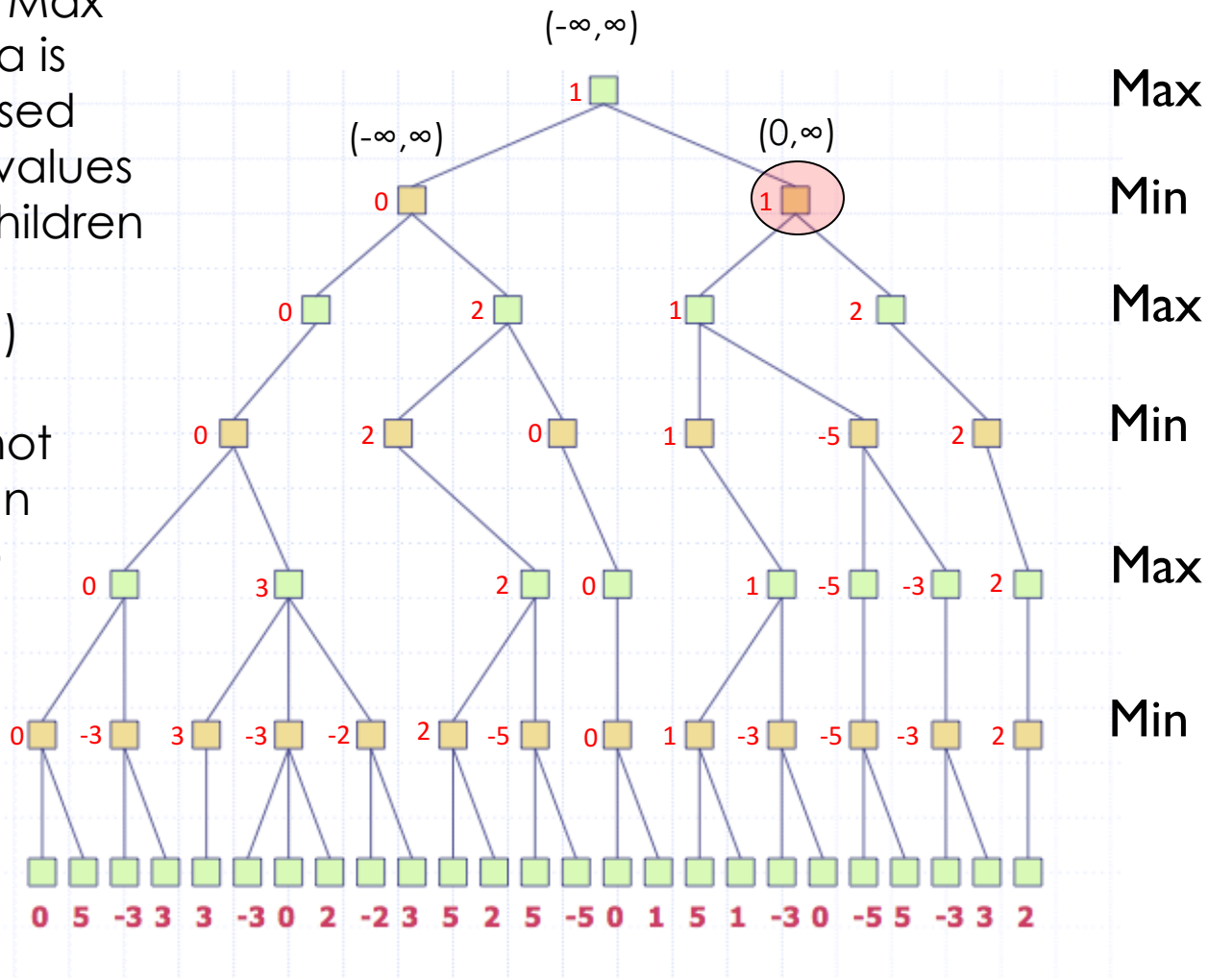


alpha = 0
beta = +∞

Children of Max node, alpha is max of passed value and values of solved children

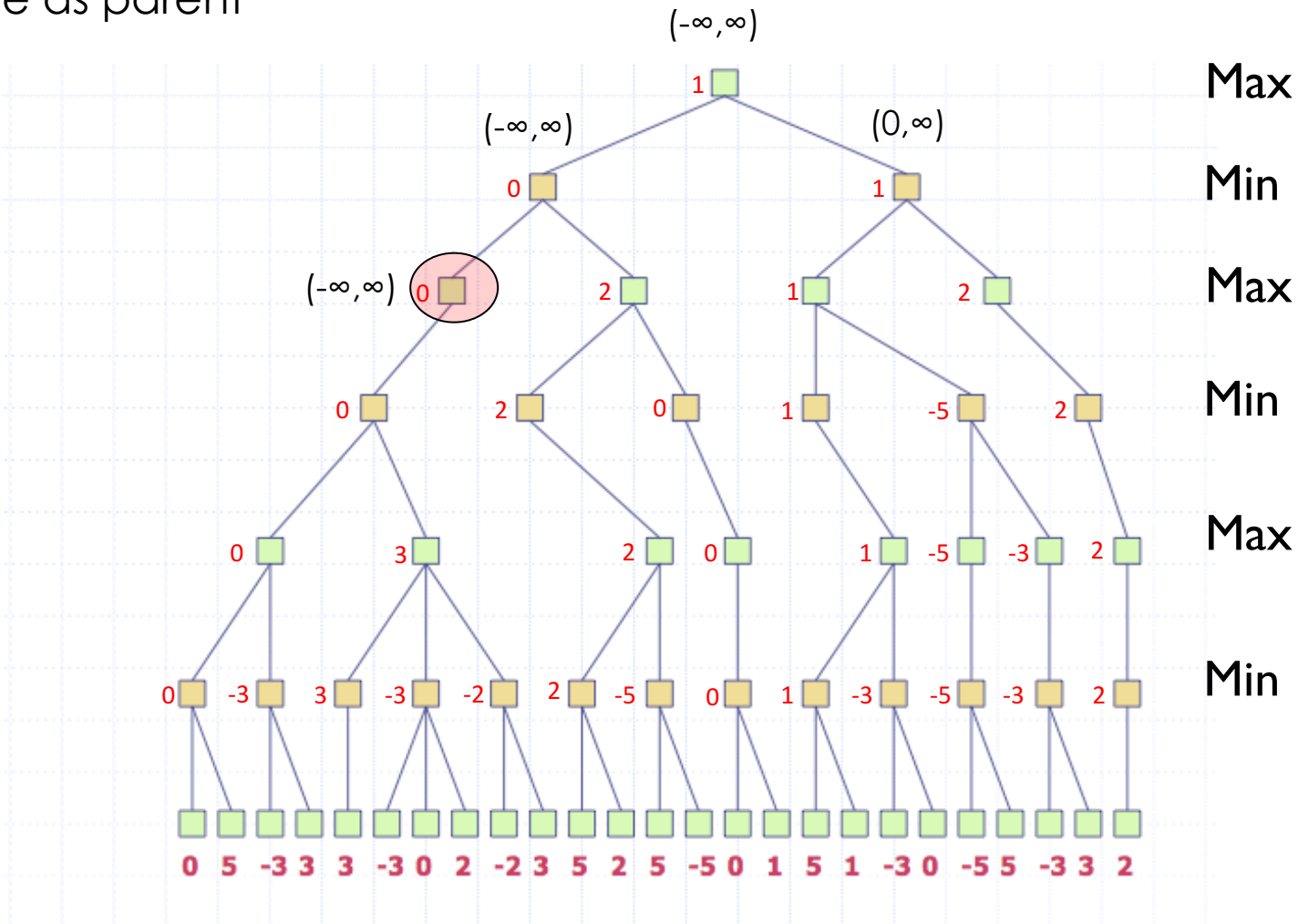
= max(0, -∞)

value = 0, not greater than beta, so no pruning



alpha = -∞
beta = +∞

same as parent

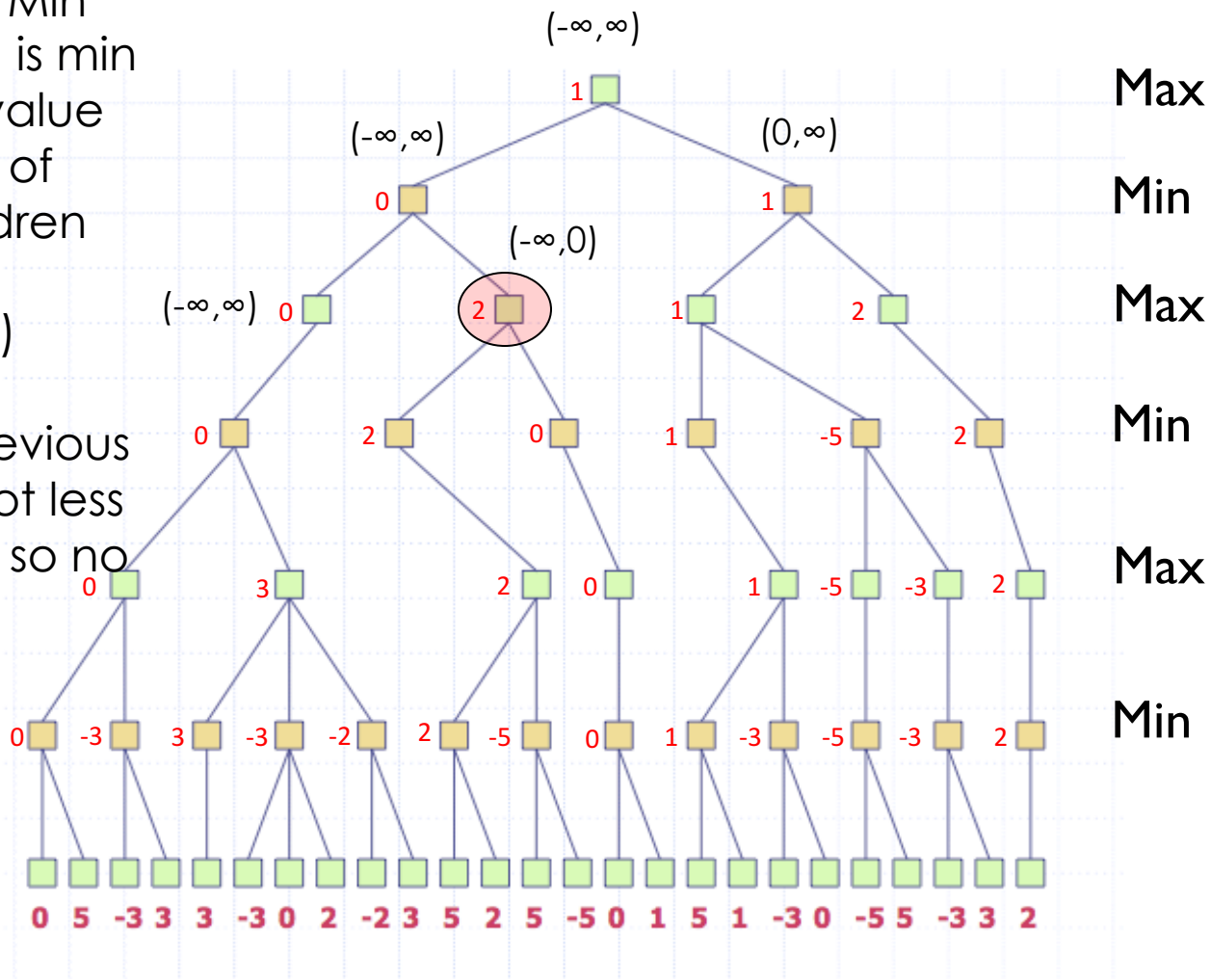


alpha = -∞
beta = 0

Children of Min node, beta is min of passed value and values of solved children

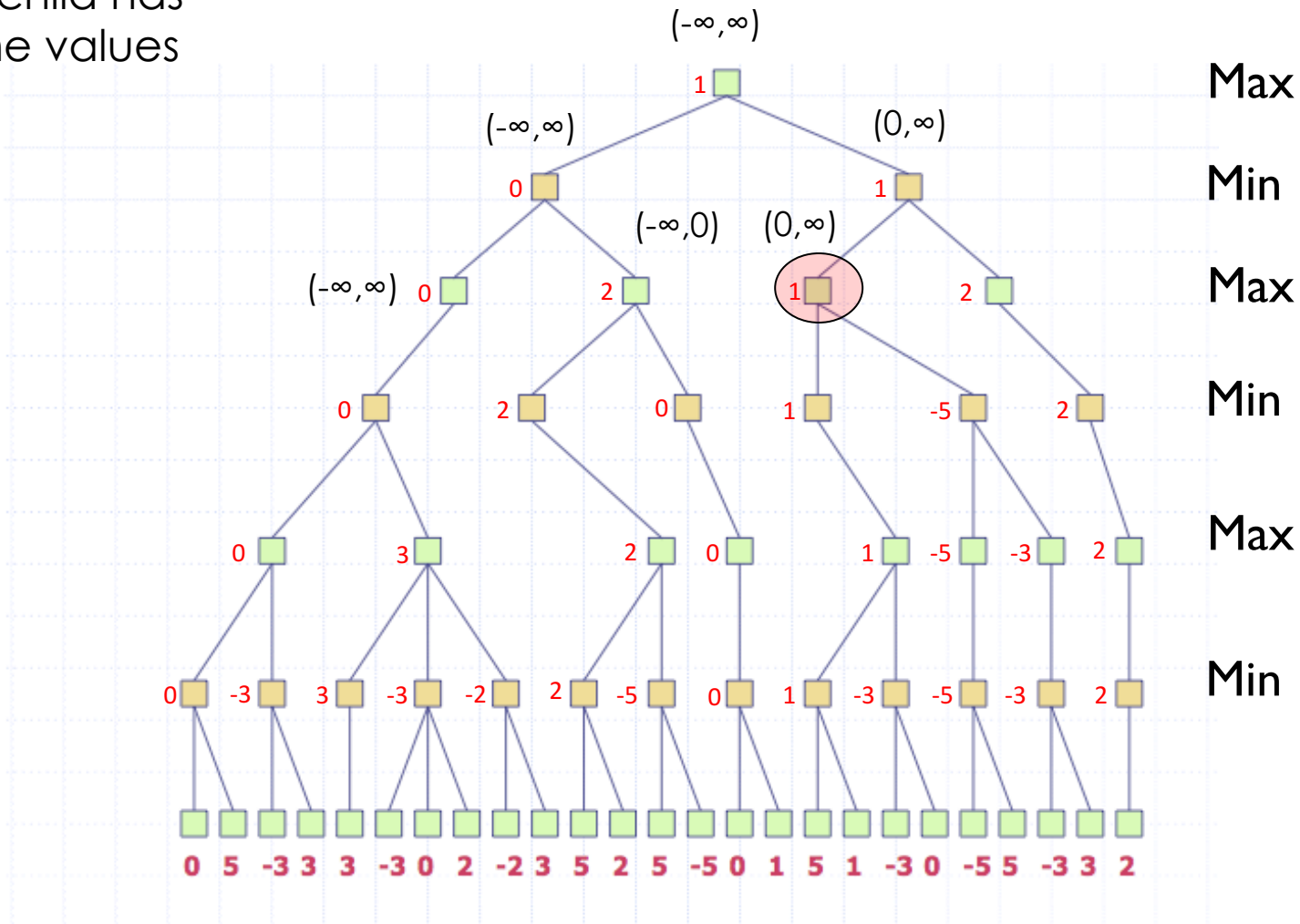
= min(0, +∞)

value of previous child = 0, not less than alpha so no pruning



alpha = 0
beta = ∞

first child has same values

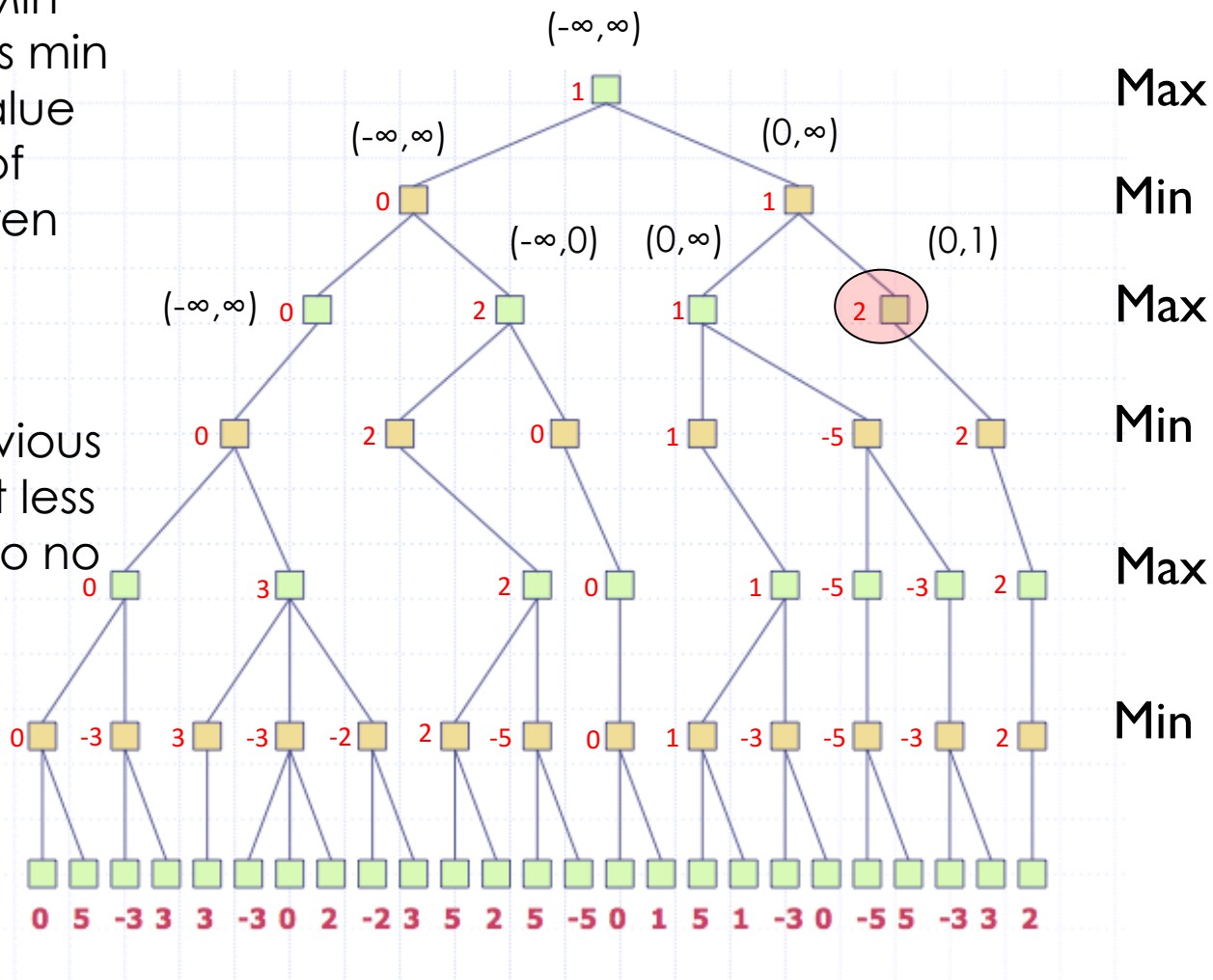


alpha = 0
beta = 1

Children of Min node, beta is min of passed value and values of solved children

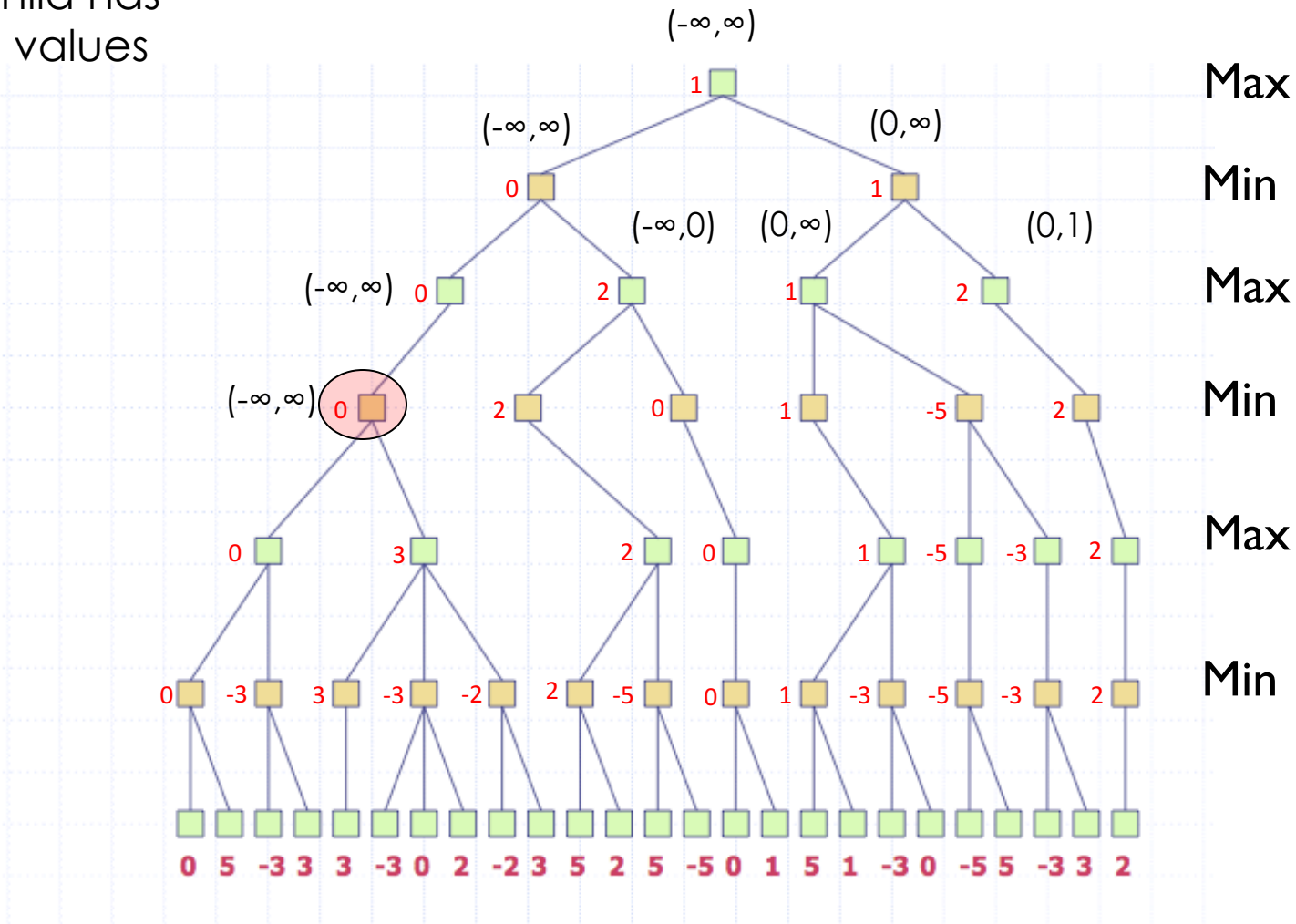
= min(1, +∞)

value of previous child = 1, not less than alpha so no pruning



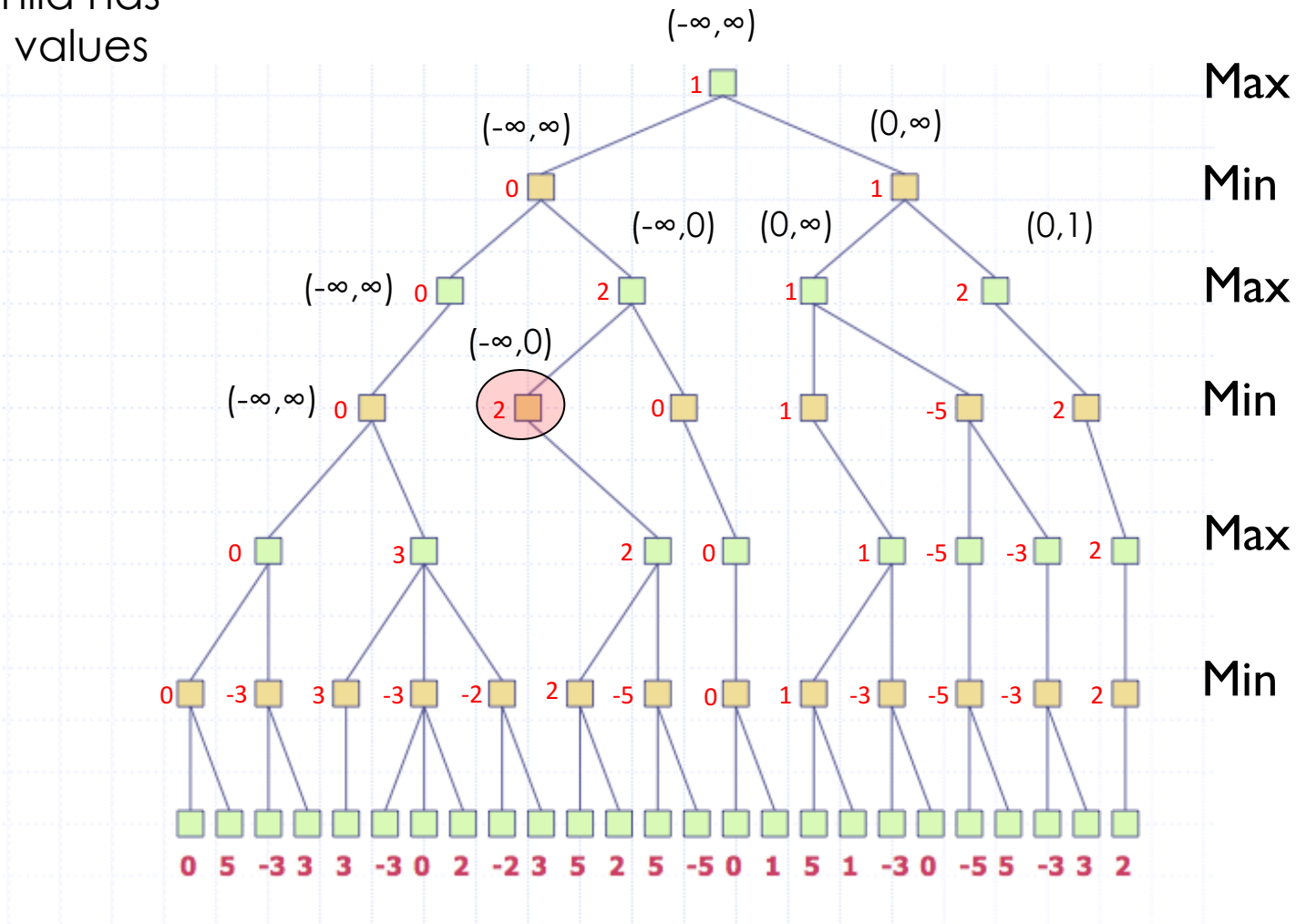
alpha = -∞
beta = ∞

First child has same values



alpha = -∞
beta = 0

First child has same values

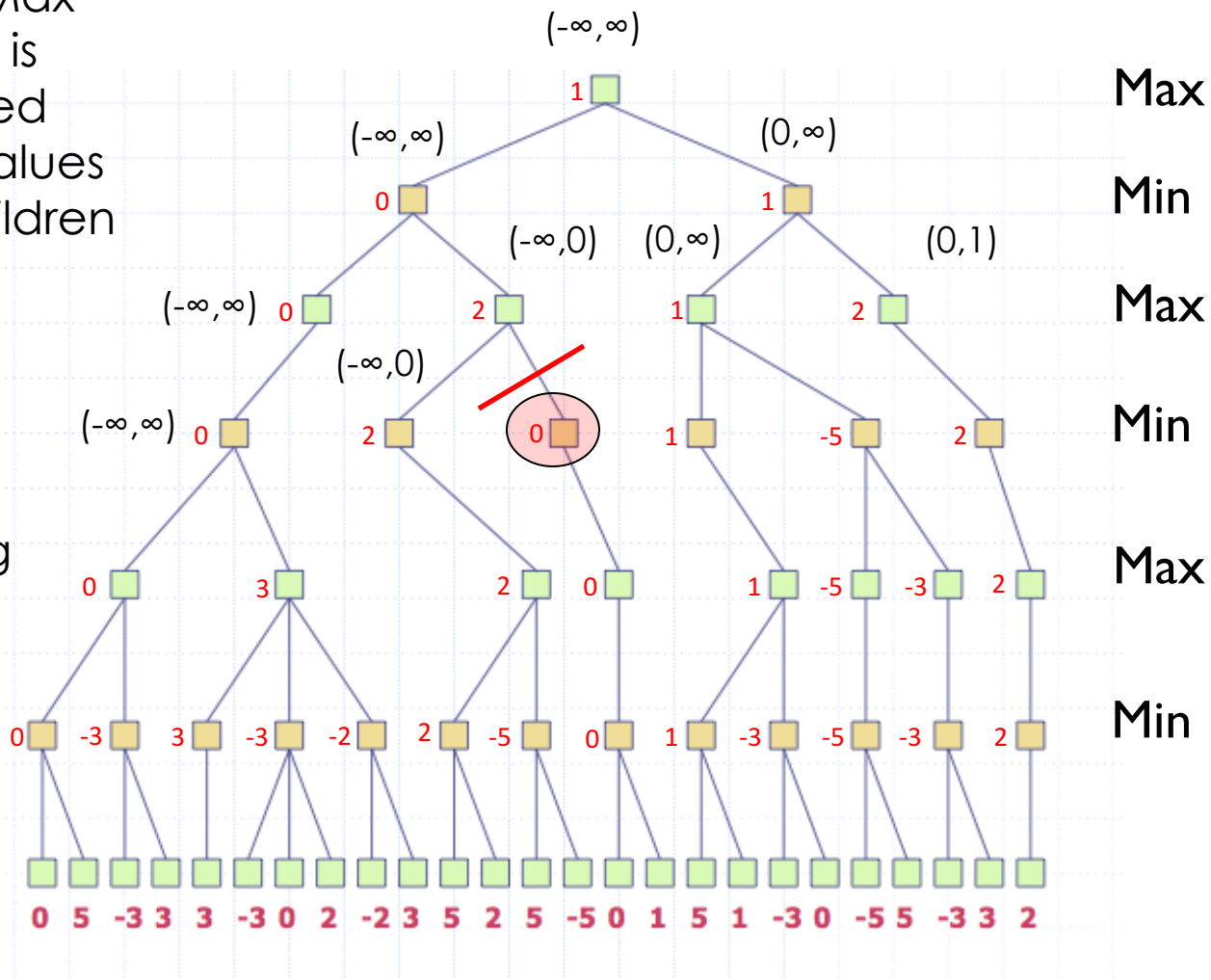


alpha = 2
beta = 0

Children of Max node, alpha is max of passed value and values of solved children

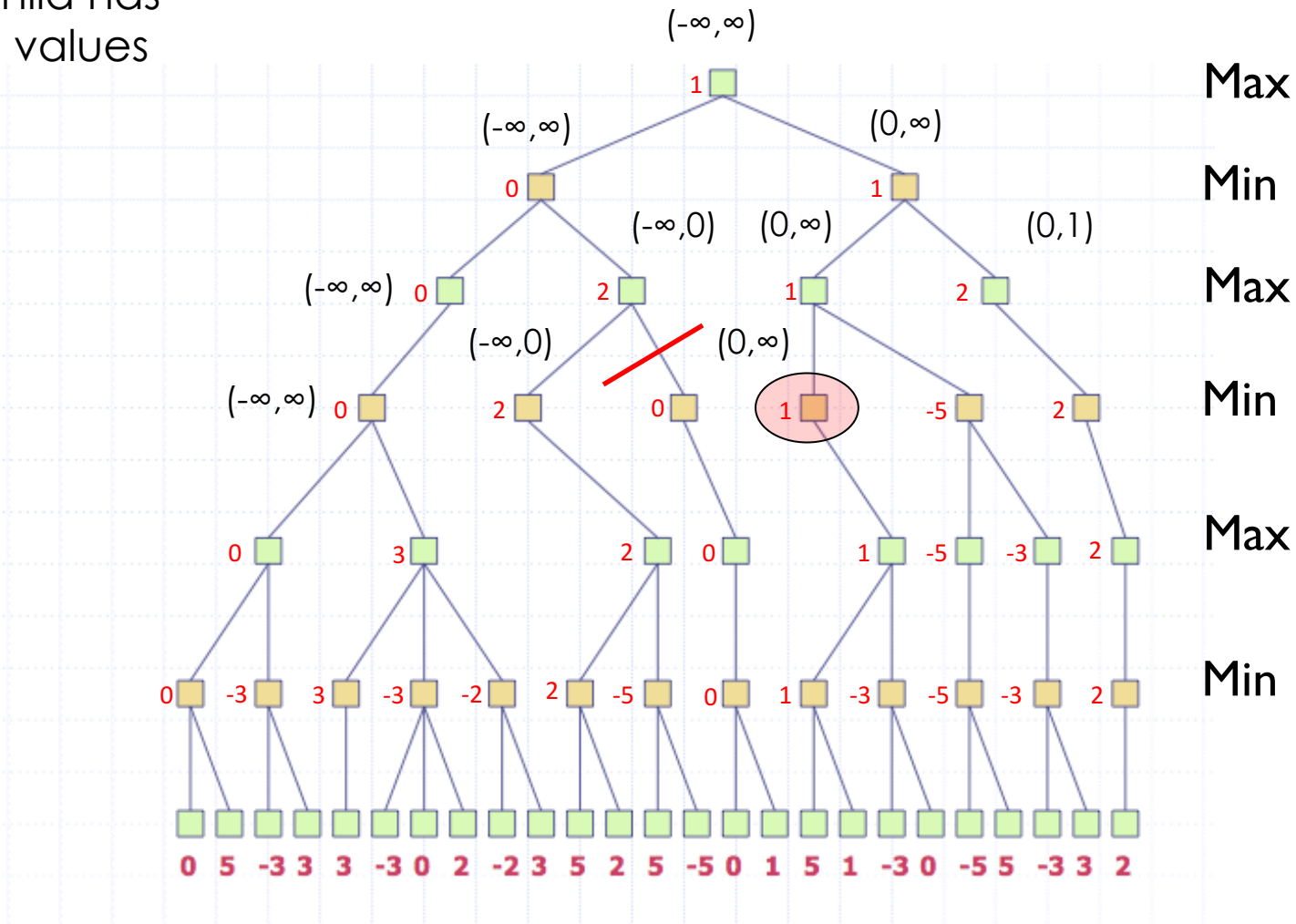
= max(2, -∞)

value = 2, is greater than beta pruning



alpha = 0
beta = ∞

First child has same values

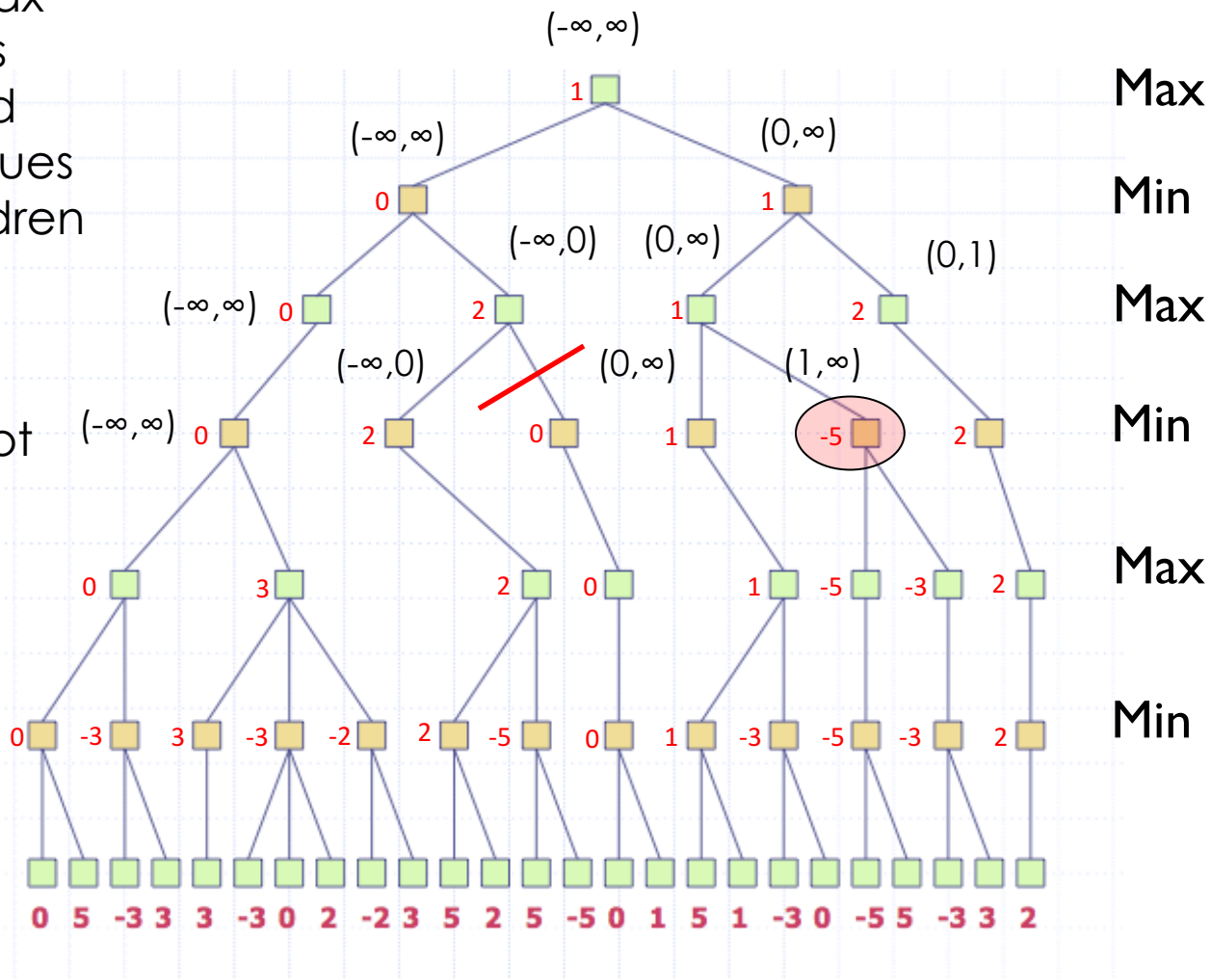


alpha = 1
beta = ∞

Children of Max node, alpha is max of passed value and values of solved children

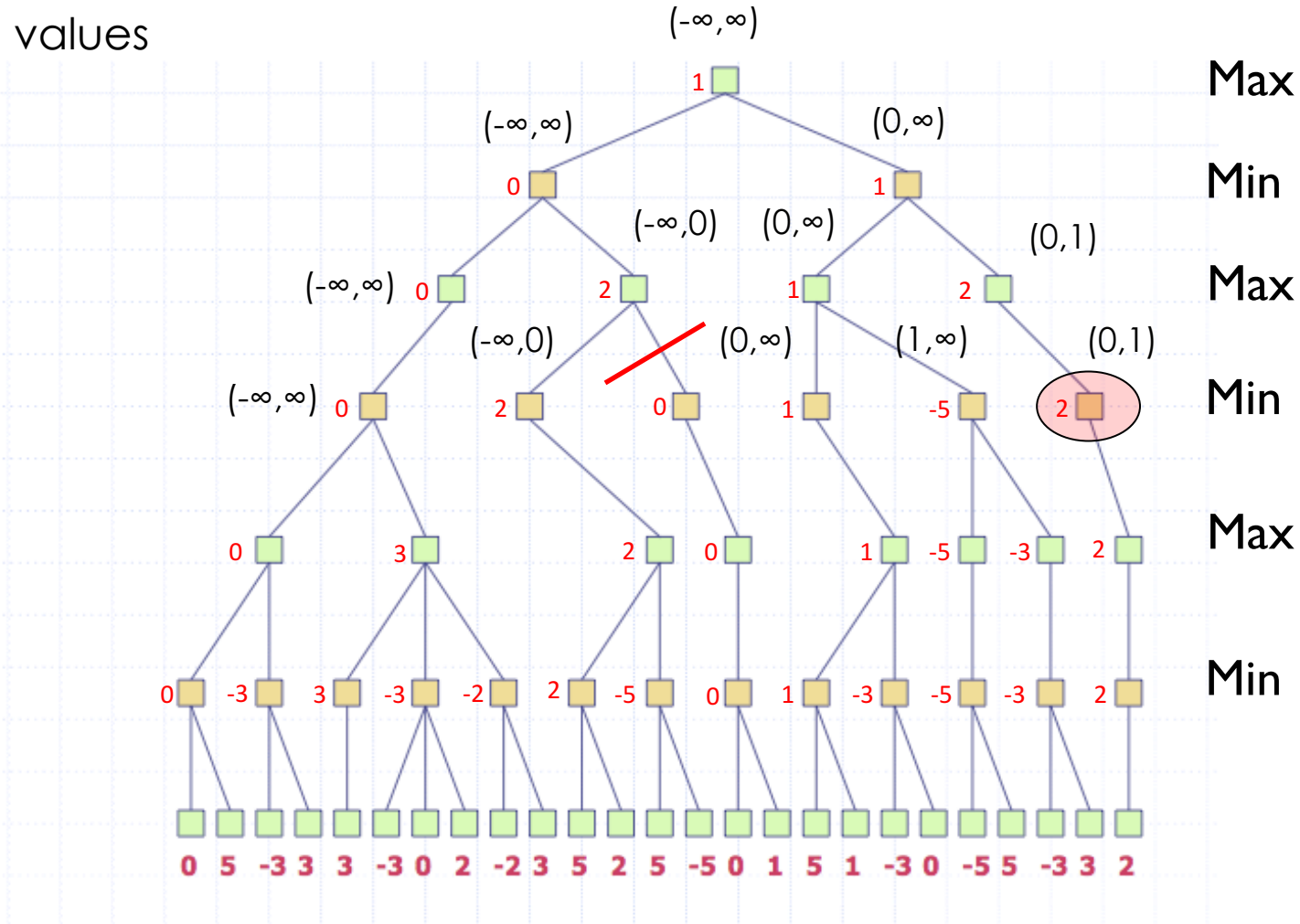
= max(1, 0)

value = 1, is not greater than beta so no pruning



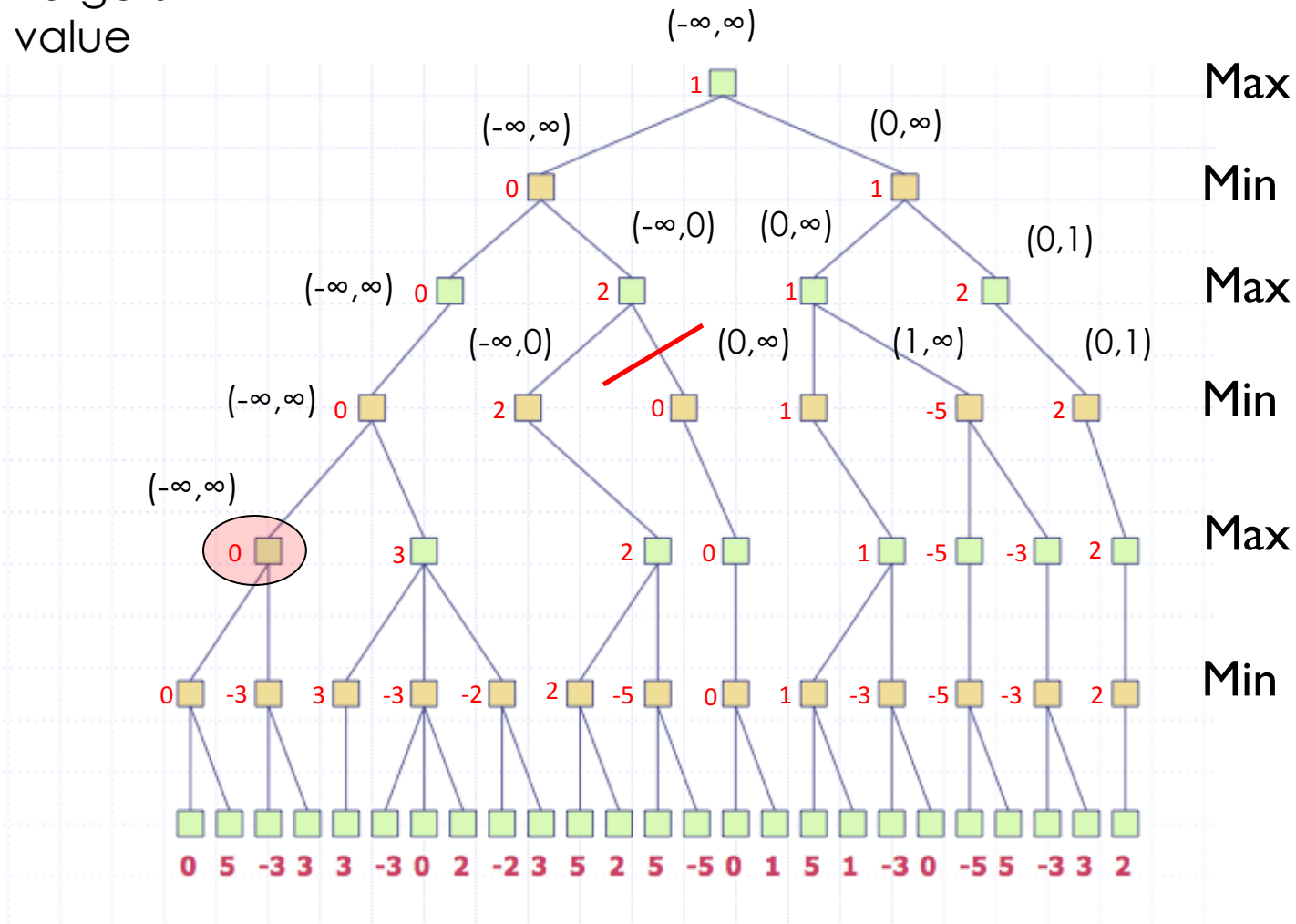
alpha = 0
beta = 1

First child has same values



alpha = -∞
beta = ∞

first child gets
same value

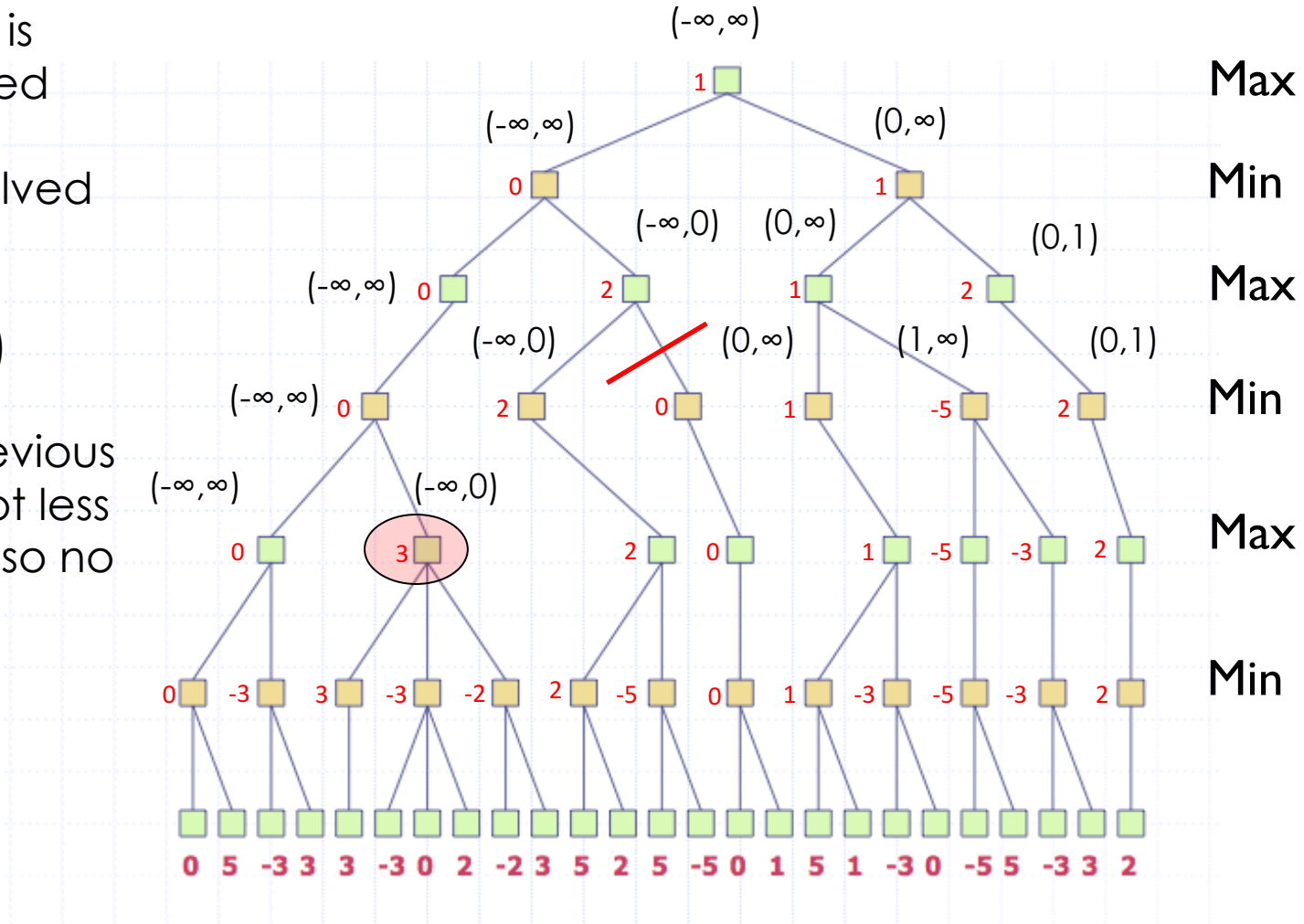


alpha = -∞
beta = 0

Children of Min node, beta is min of passed value and values of solved children

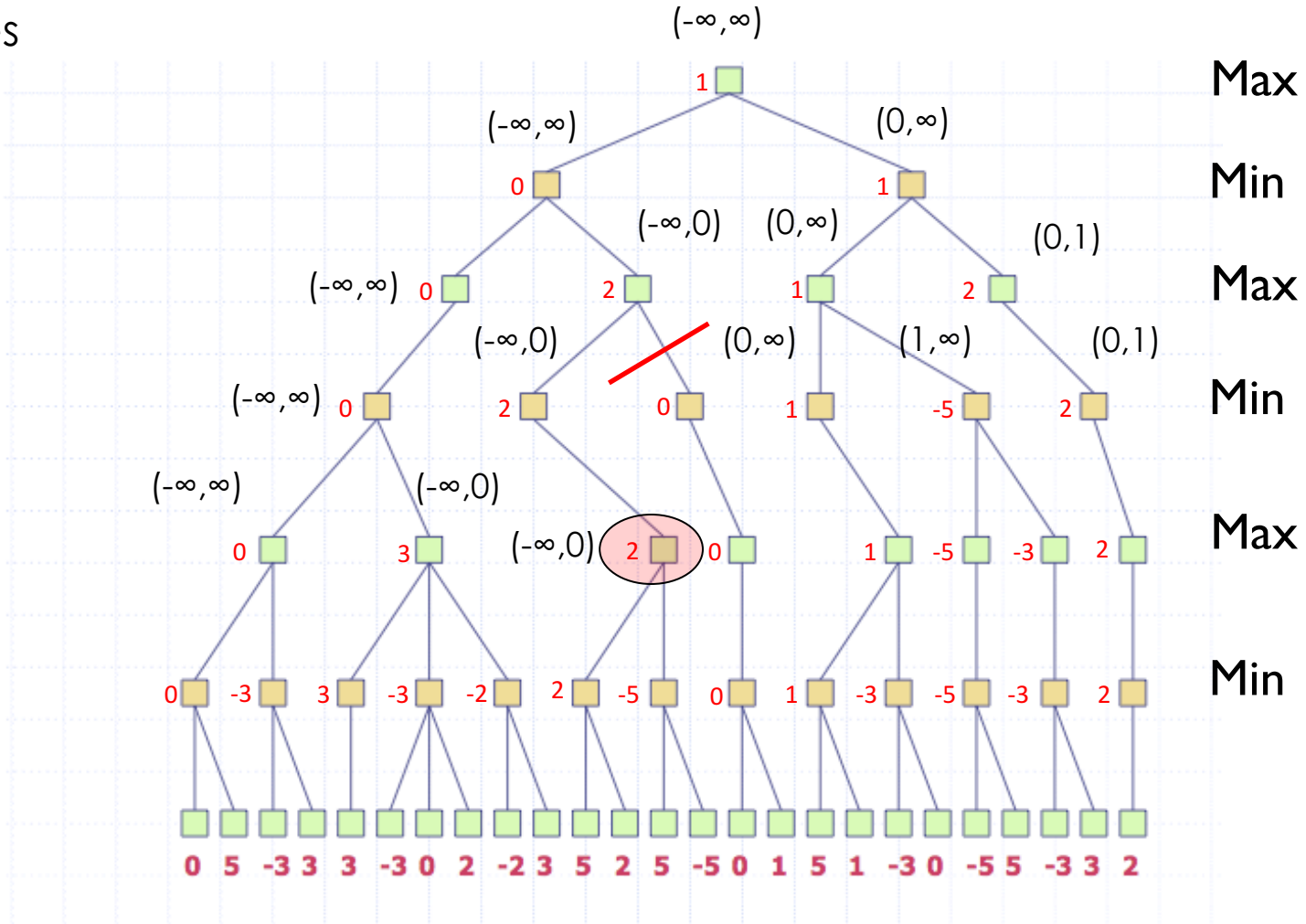
= min(0, +∞)

value of previous child = 0, not less than alpha so no pruning



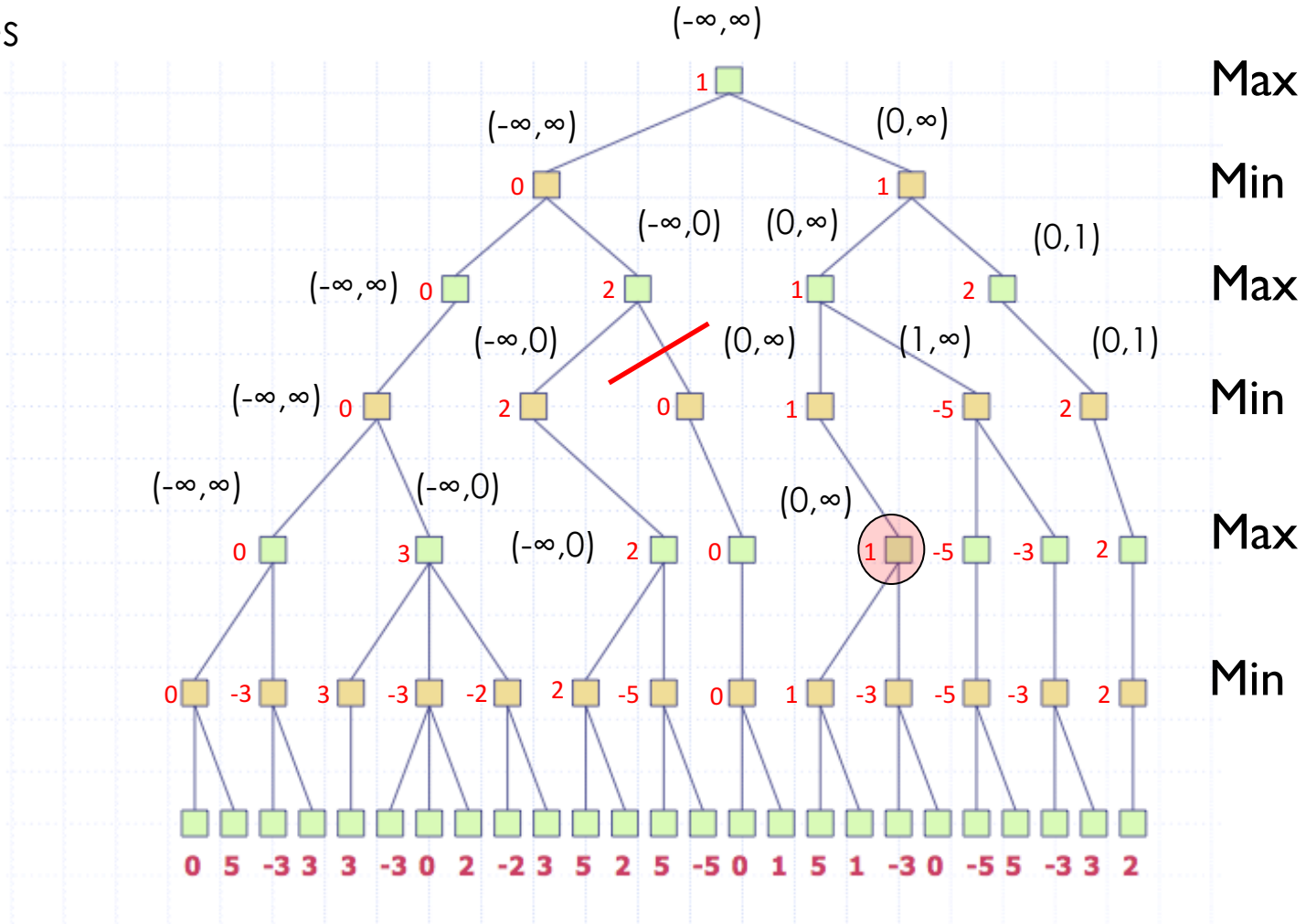
alpha = -∞
beta = 0

First child has same values



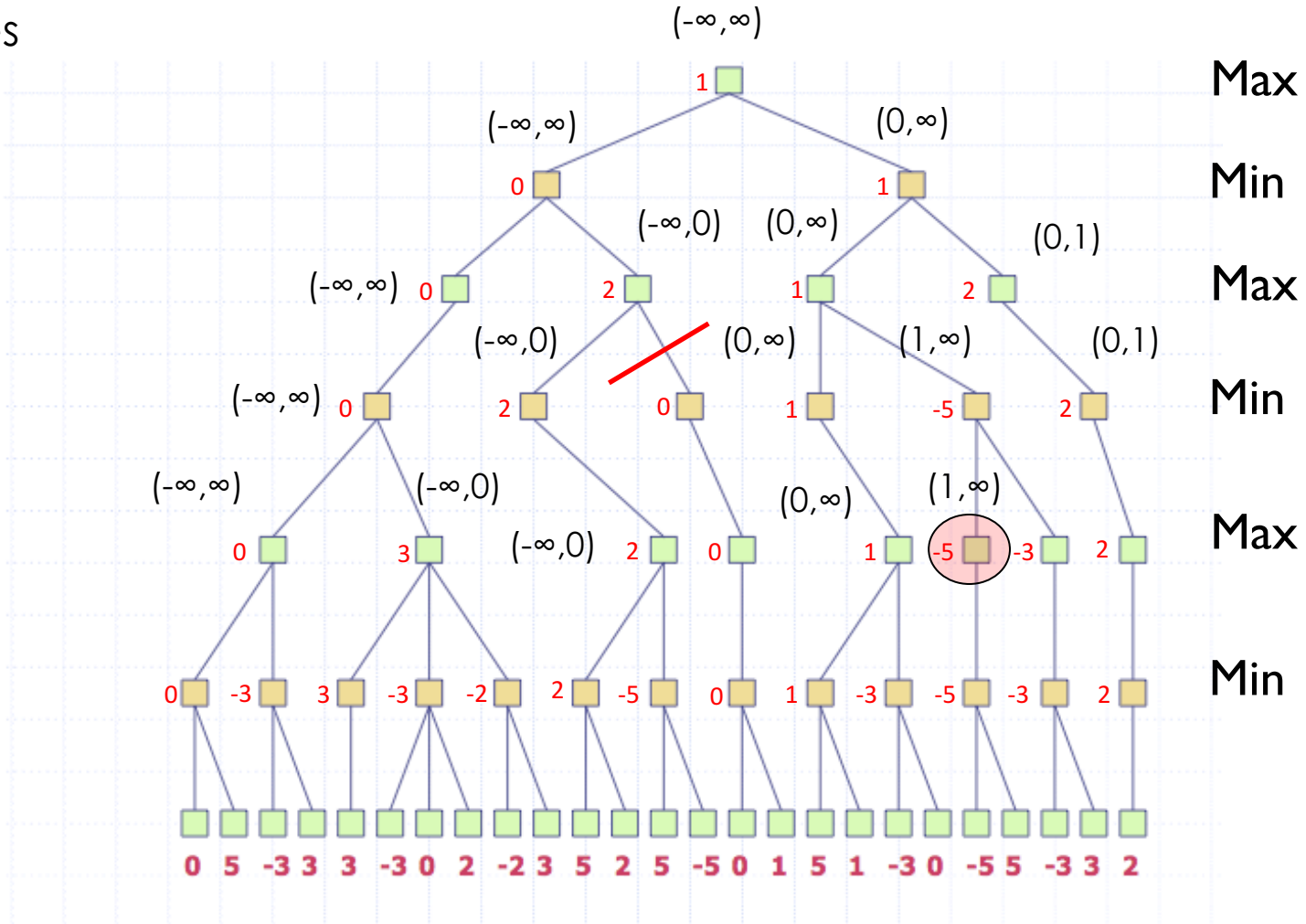
alpha = 0
beta = ∞

First child has same values



alpha = 1
beta = ∞

First child has same values

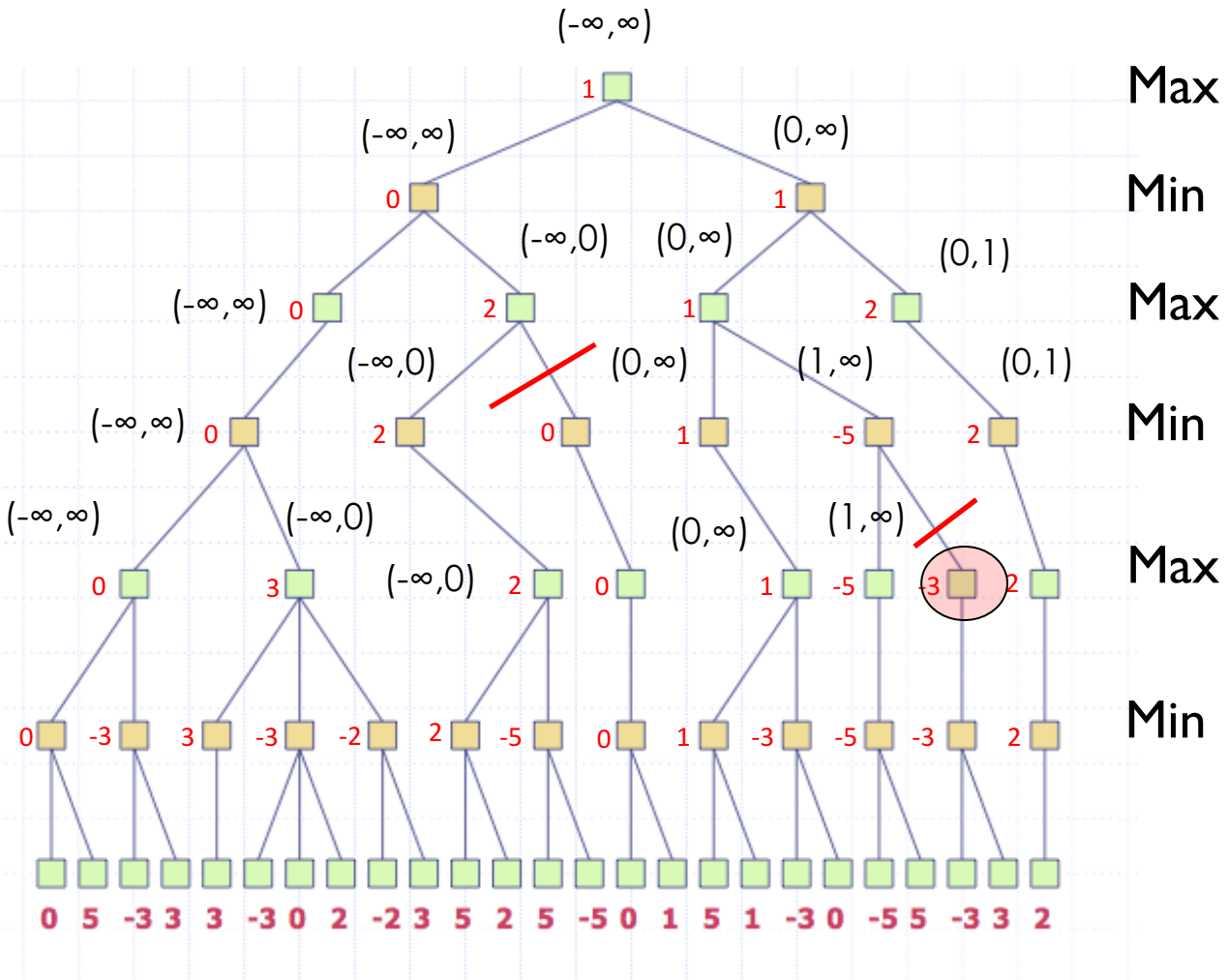


alpha = 1
beta = ∞

Children of Min node, beta is min of passed value and values of solved children

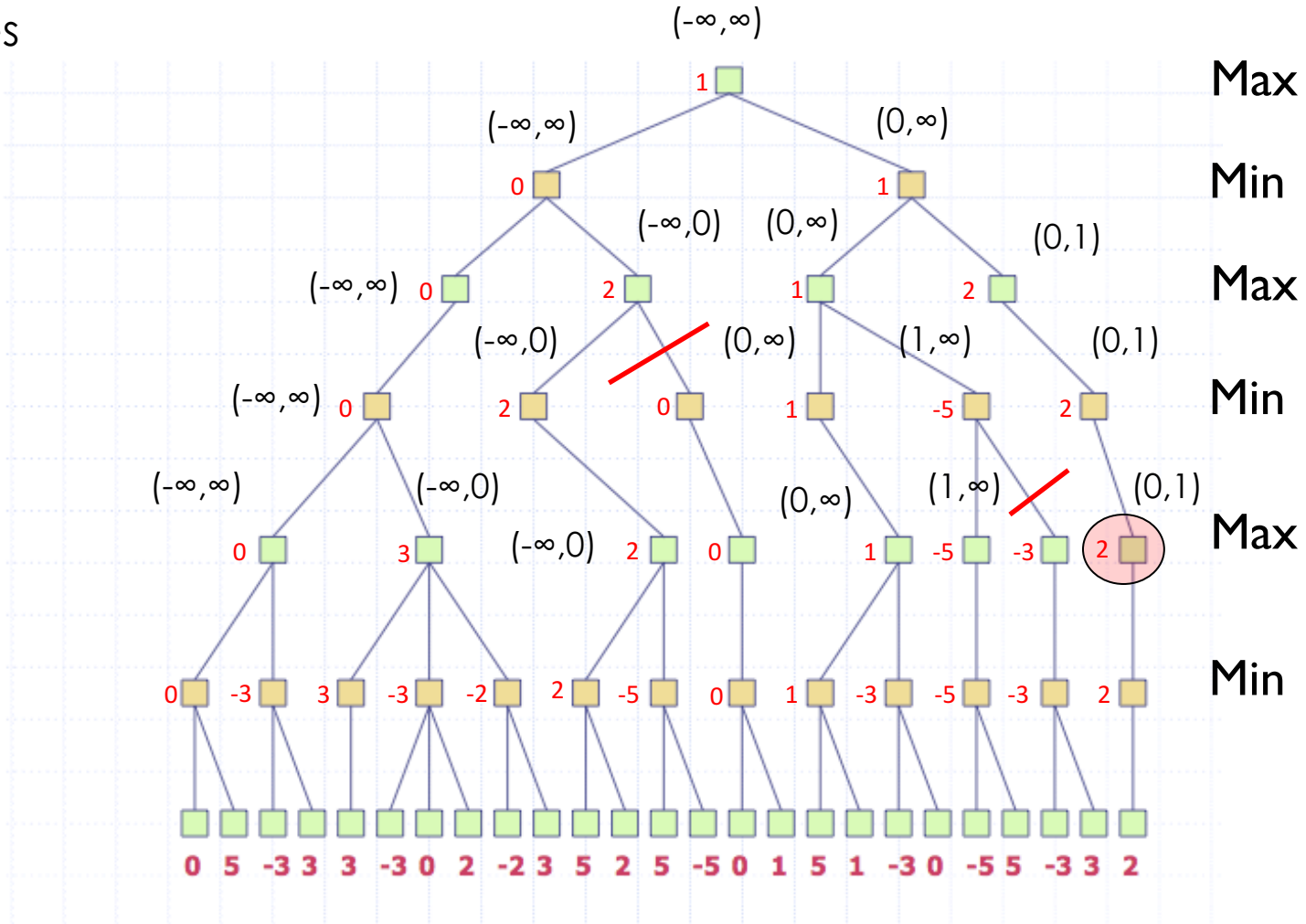
= min(-5, +∞)

value of previous child = -5, is less than alpha so pruning



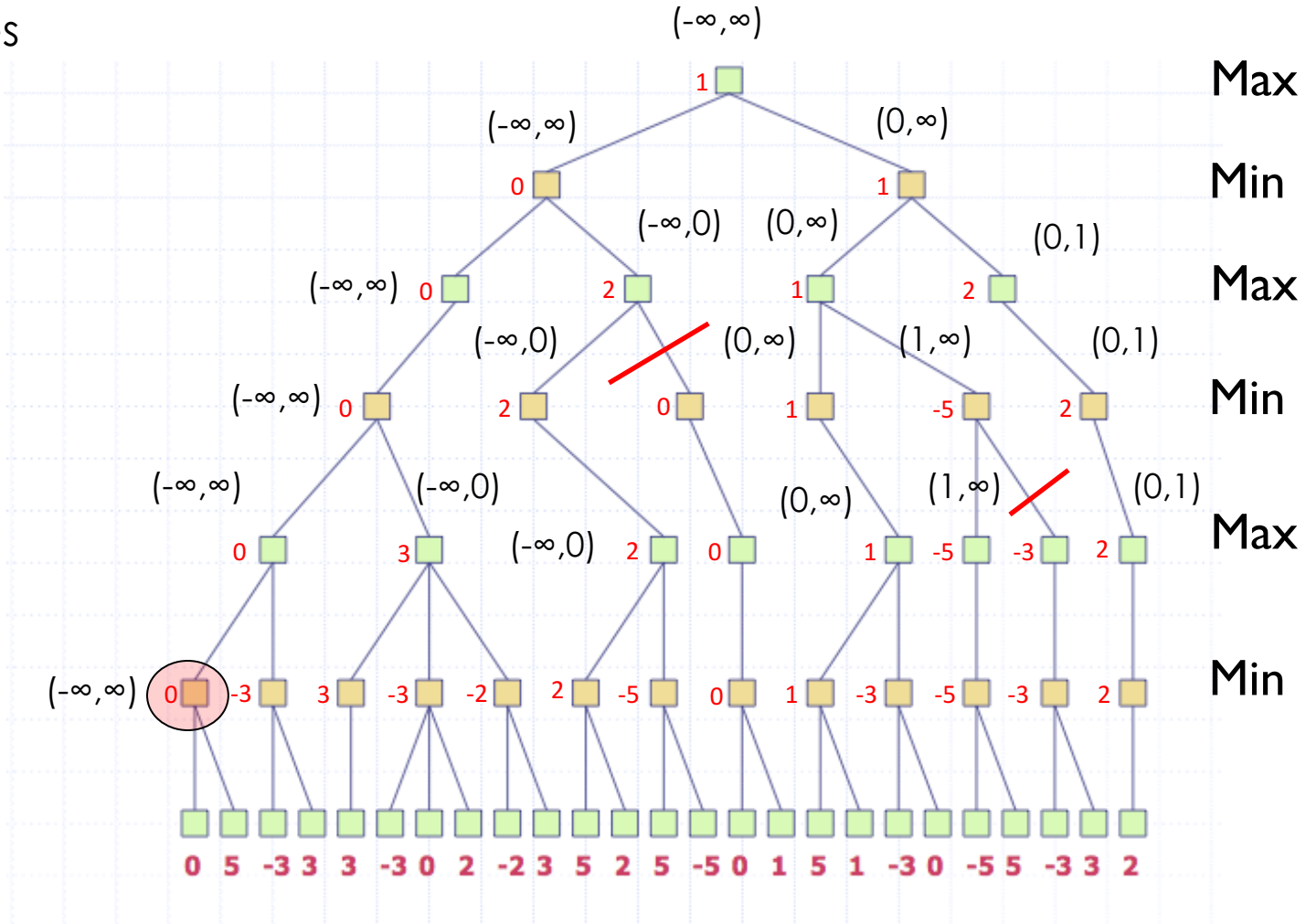
alpha = 0
beta = 1

First child gets same values



alpha = -∞
beta = ∞

First child gets same values

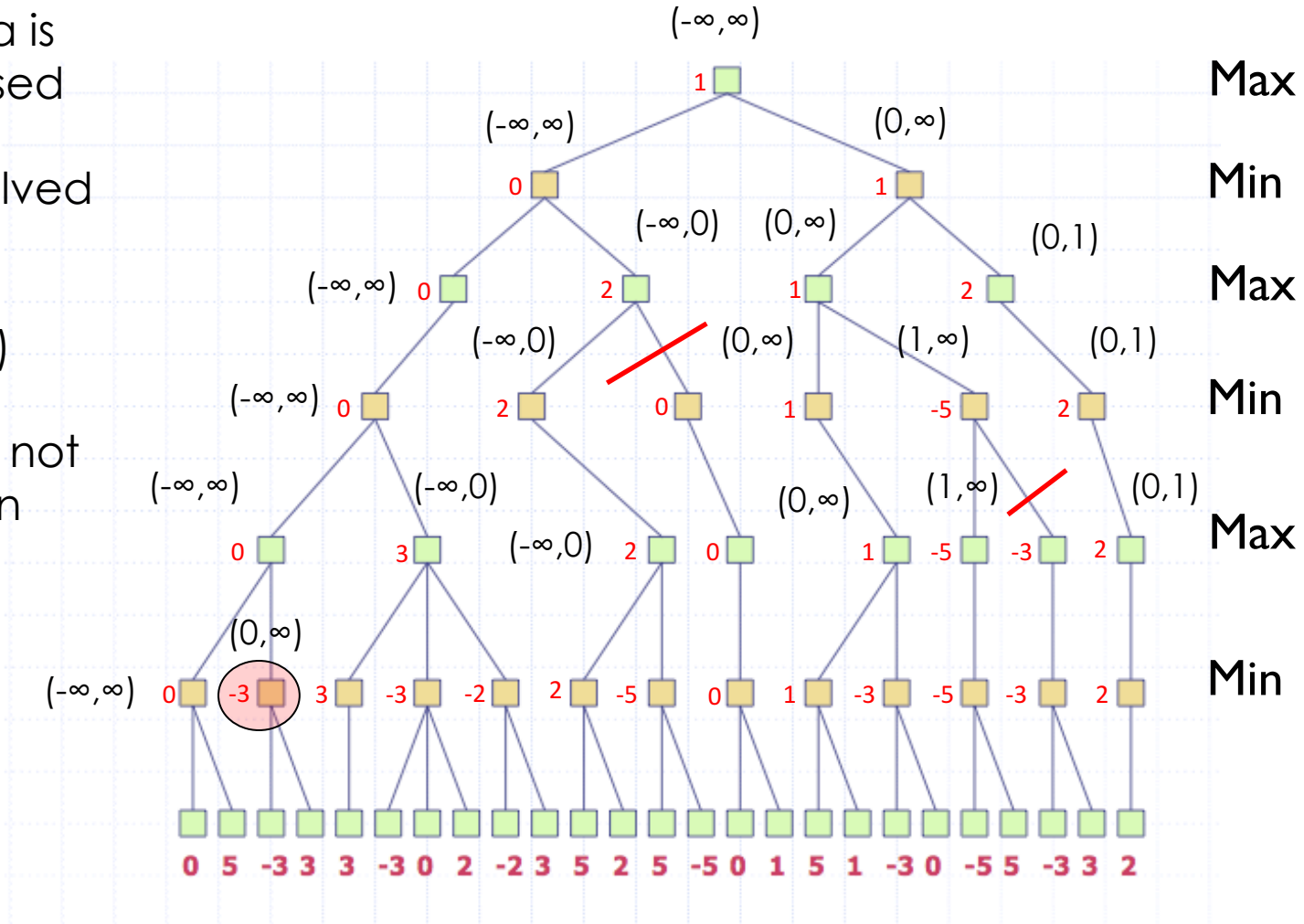


alpha = 0
beta = ∞

Children of Max node, alpha is max of passed value and values of solved children

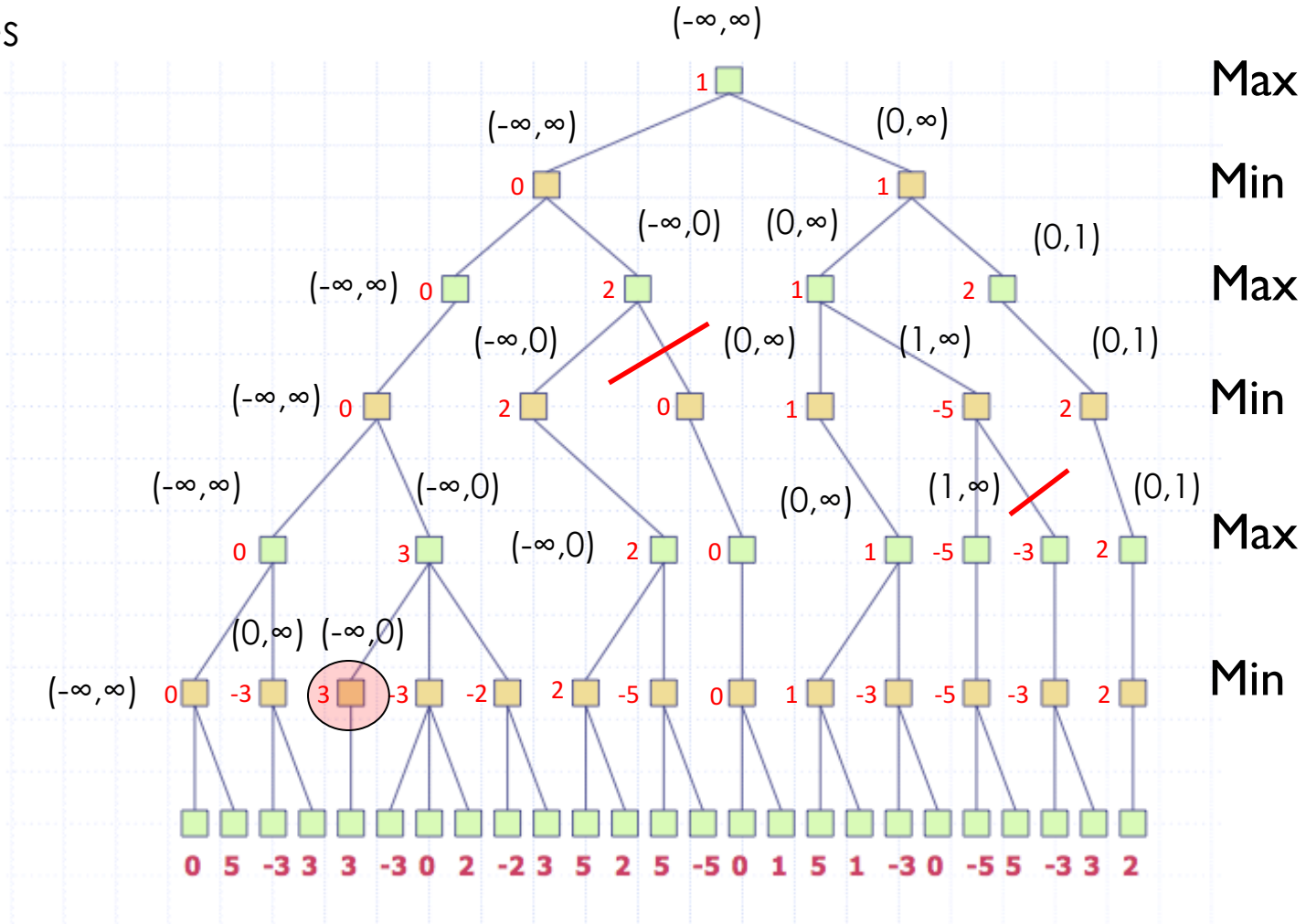
= max(0, -∞)

value = 0, is not greater than beta so no pruning



alpha = -∞
beta = 0

First child gets same values

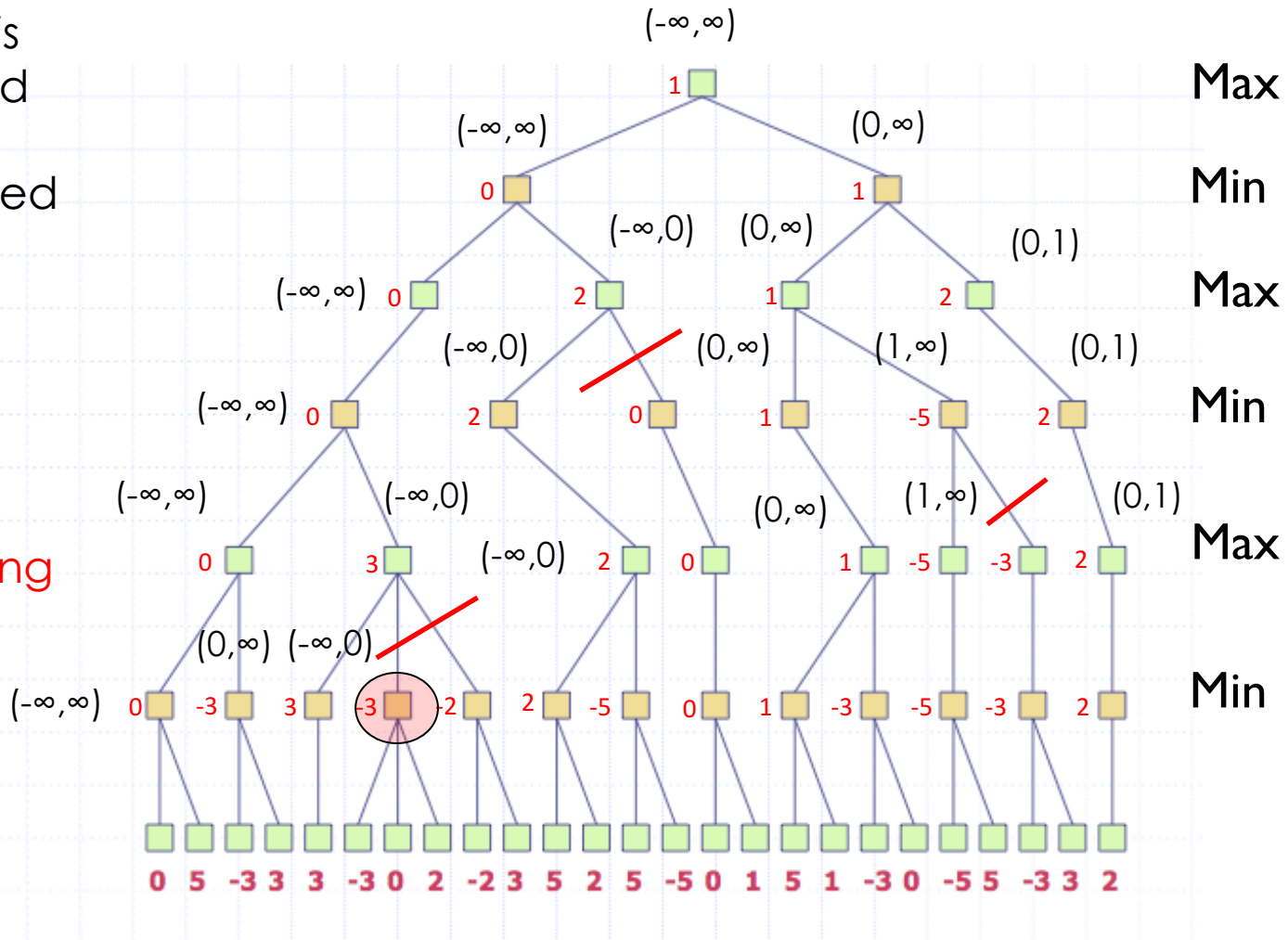


alpha = -∞
beta = 0

Children of Max node, alpha is max of passed value and values of solved children

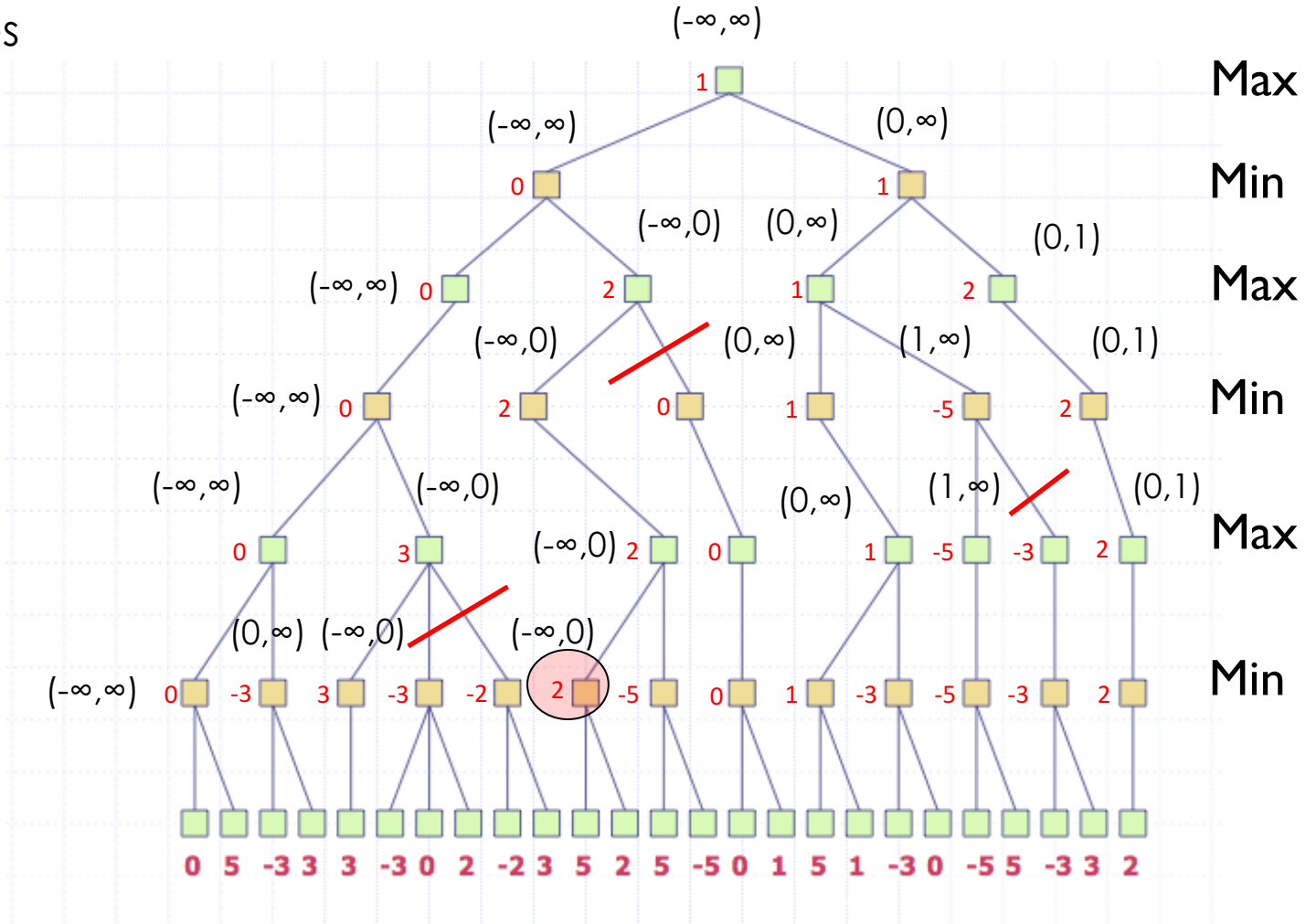
= max(3, -∞)

value = 3, is greater than beta so pruning



alpha = -∞
beta = 0

First child gets same values

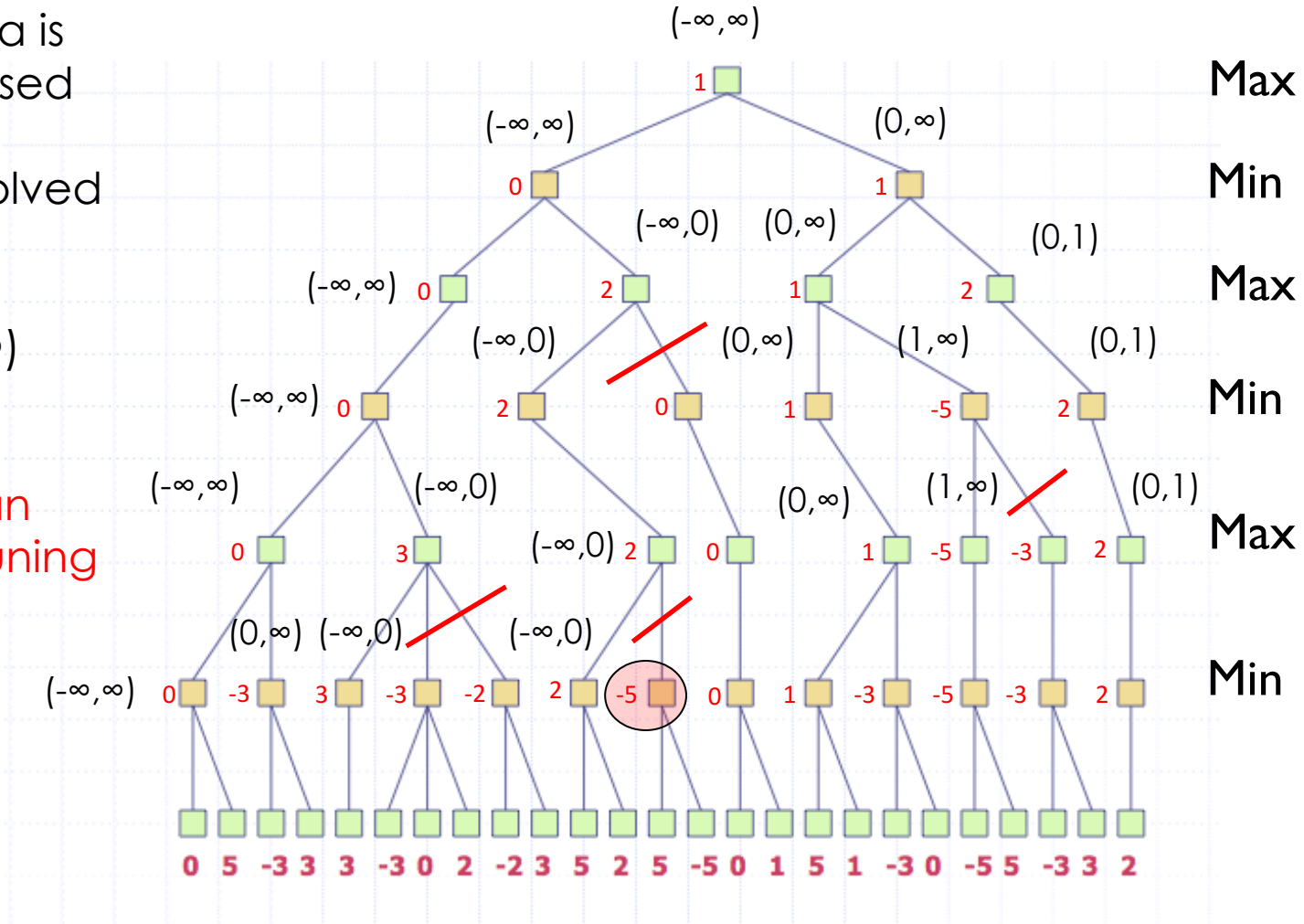


alpha = 2
beta = 0

Children of Max node, alpha is max of passed value and values of solved children

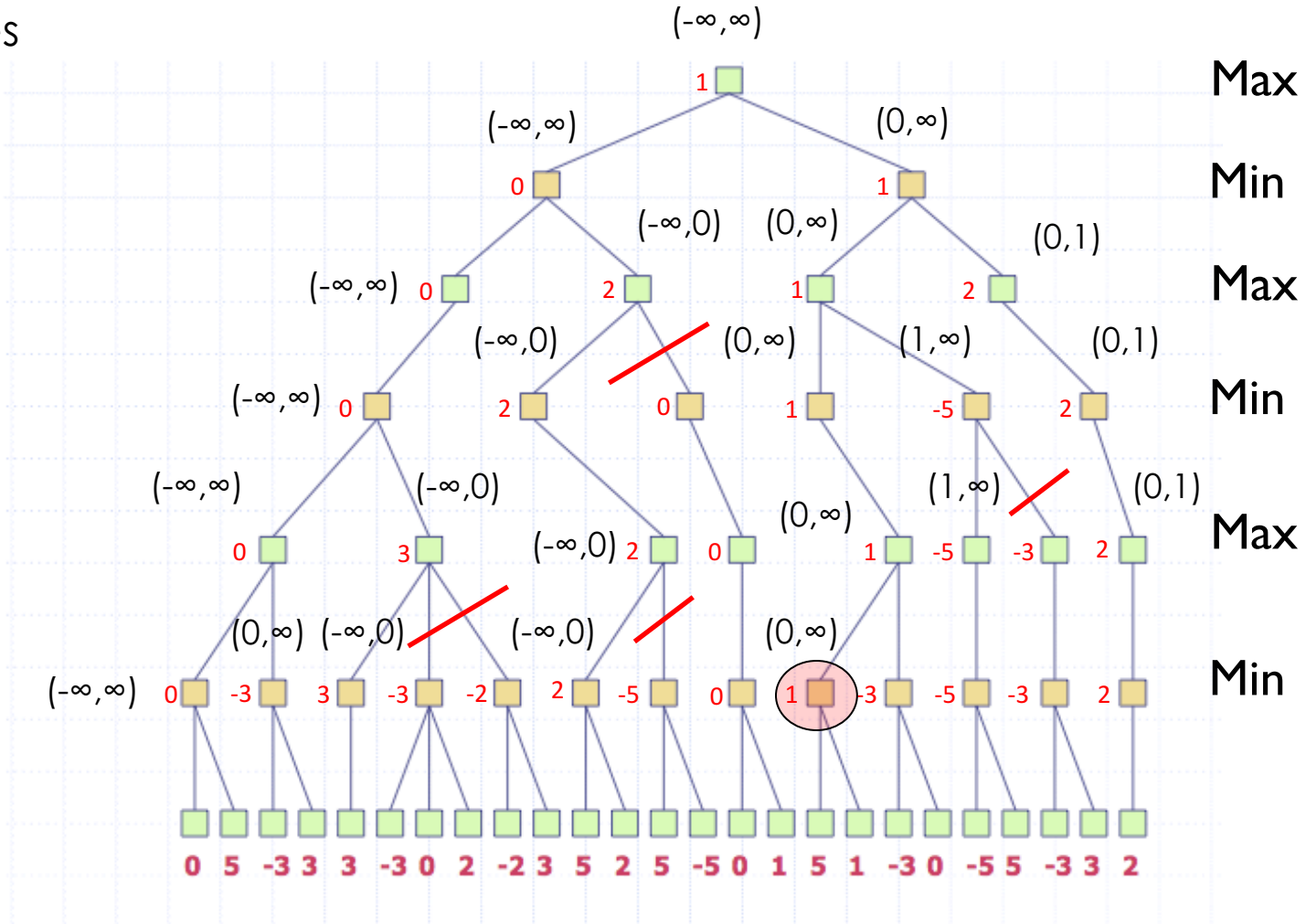
= max(2, -∞)

value = 2, greater than beta so pruning



alpha = 0
beta = ∞

First child gets same values

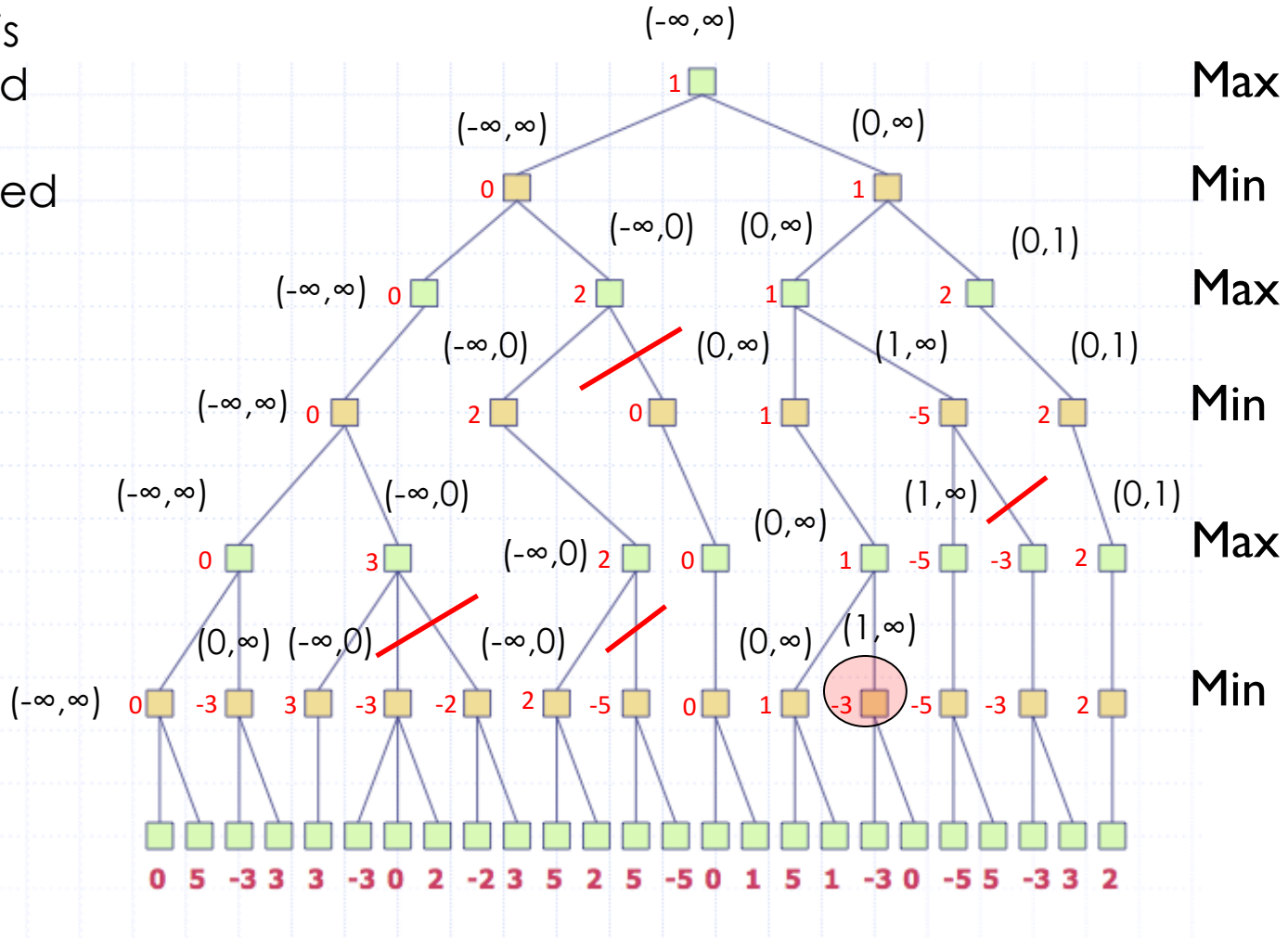


alpha = 1
beta = ∞

Children of Max node, alpha is max of passed value and values of solved children

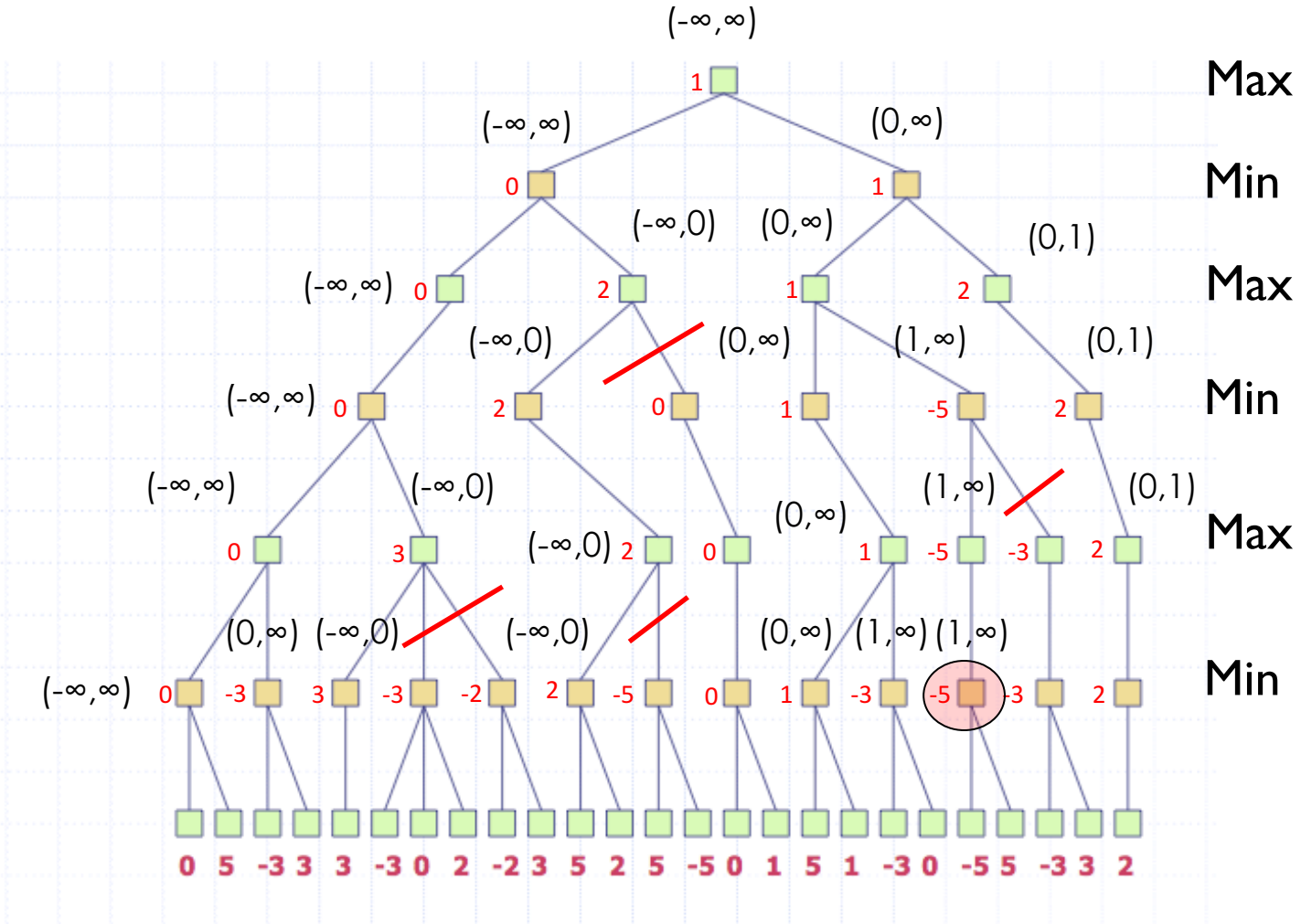
= max(1, -∞)

value = 1 not greate than beta so no pruning



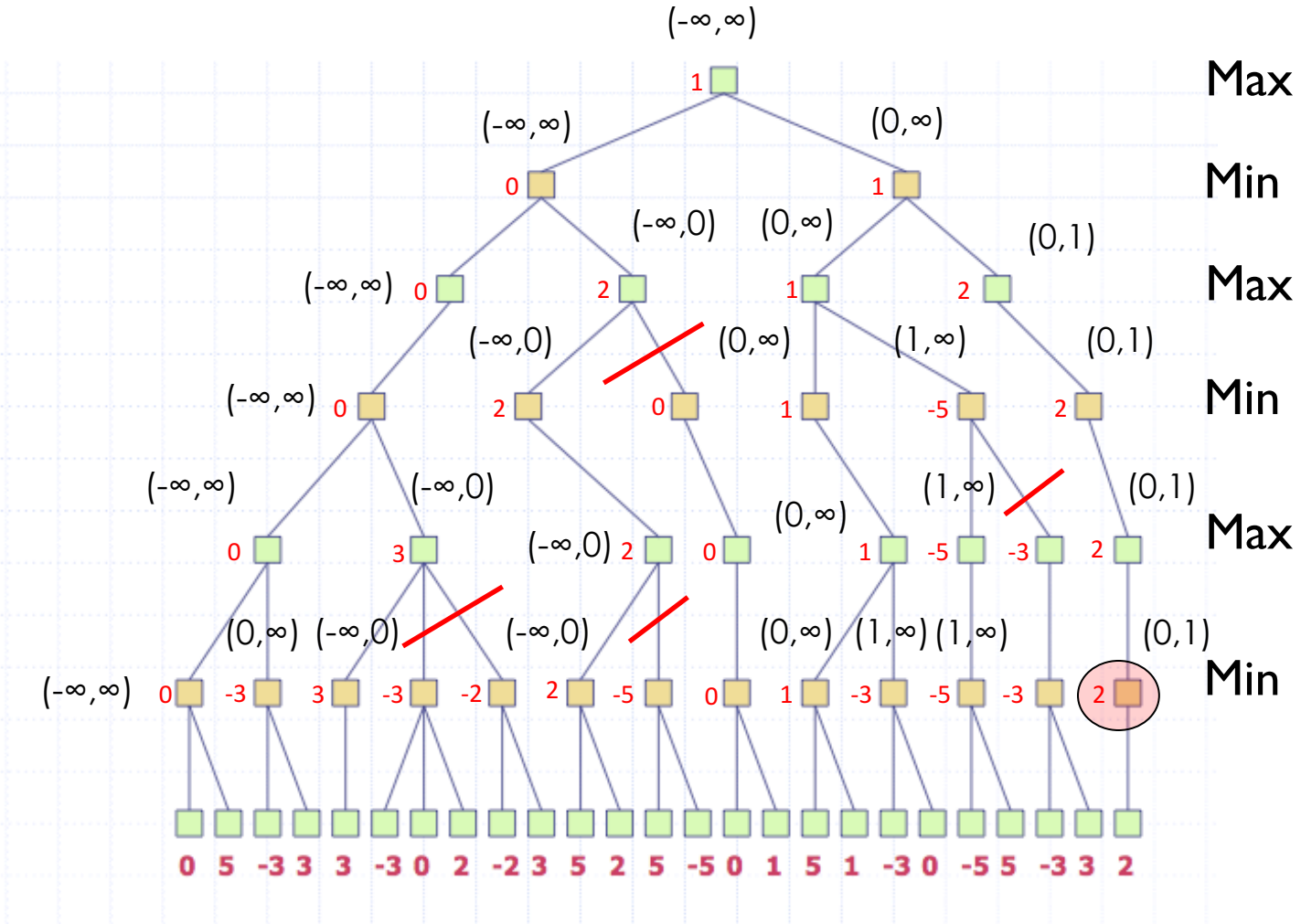
alpha = 1
beta = ∞

First child gets same value



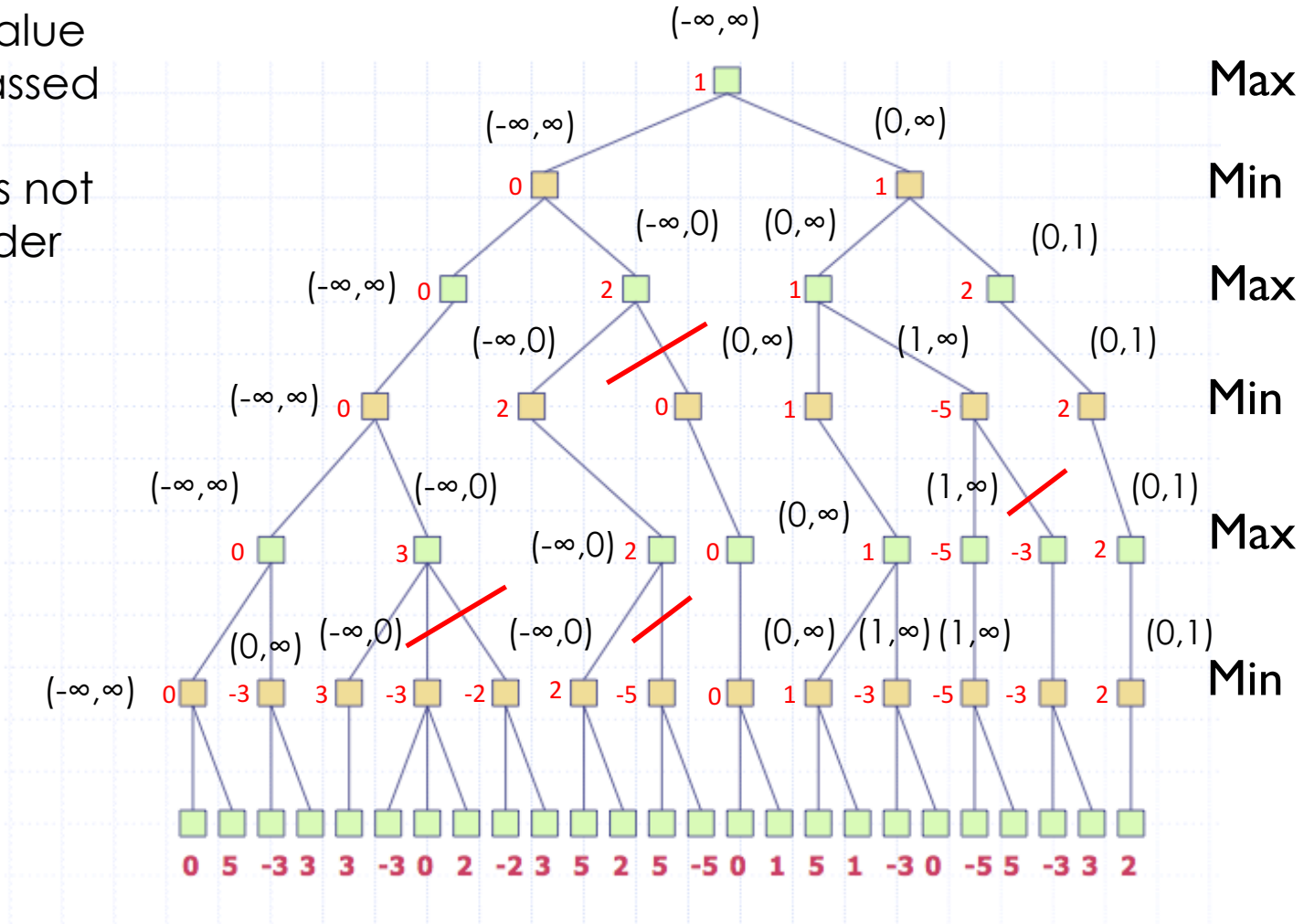
alpha = 0
beta = 1

First child gets same value



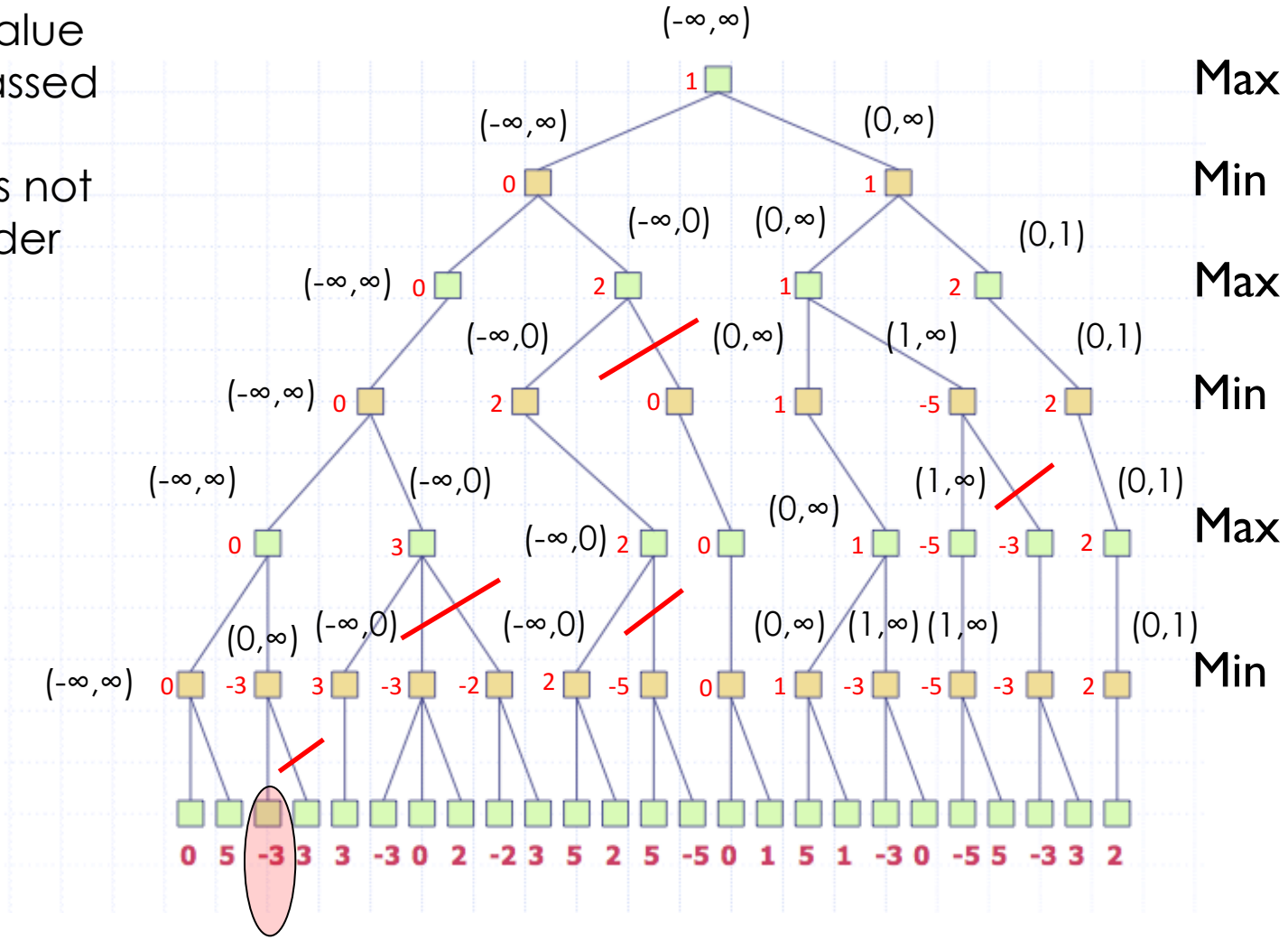
Last layer is MIN layer.

Prune after first child with value less than passed alpha
(alpha does not change under min node)



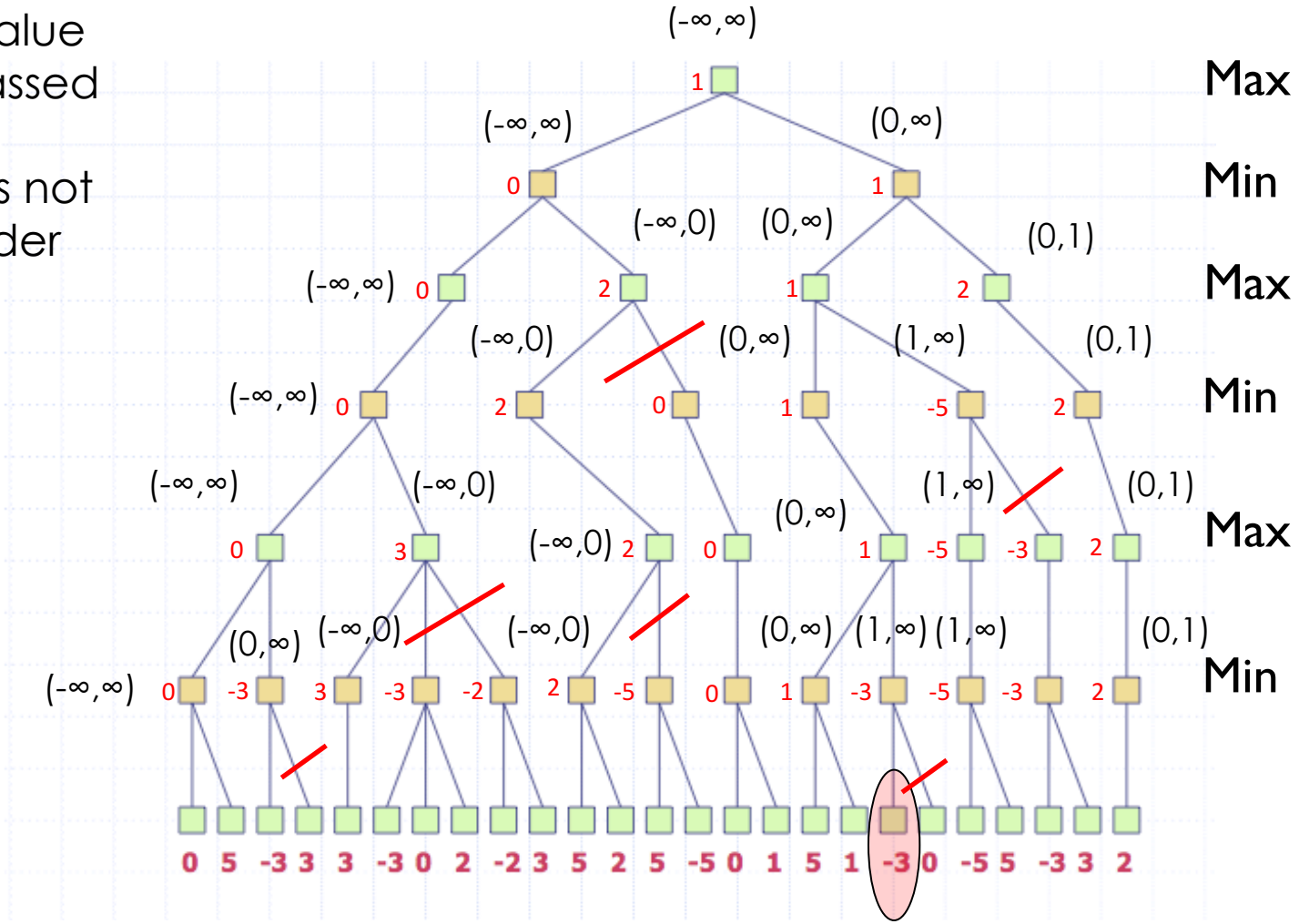
Last layer is MIN layer.

Prune after first child with value less than passed alpha
(alpha does not change under min node)



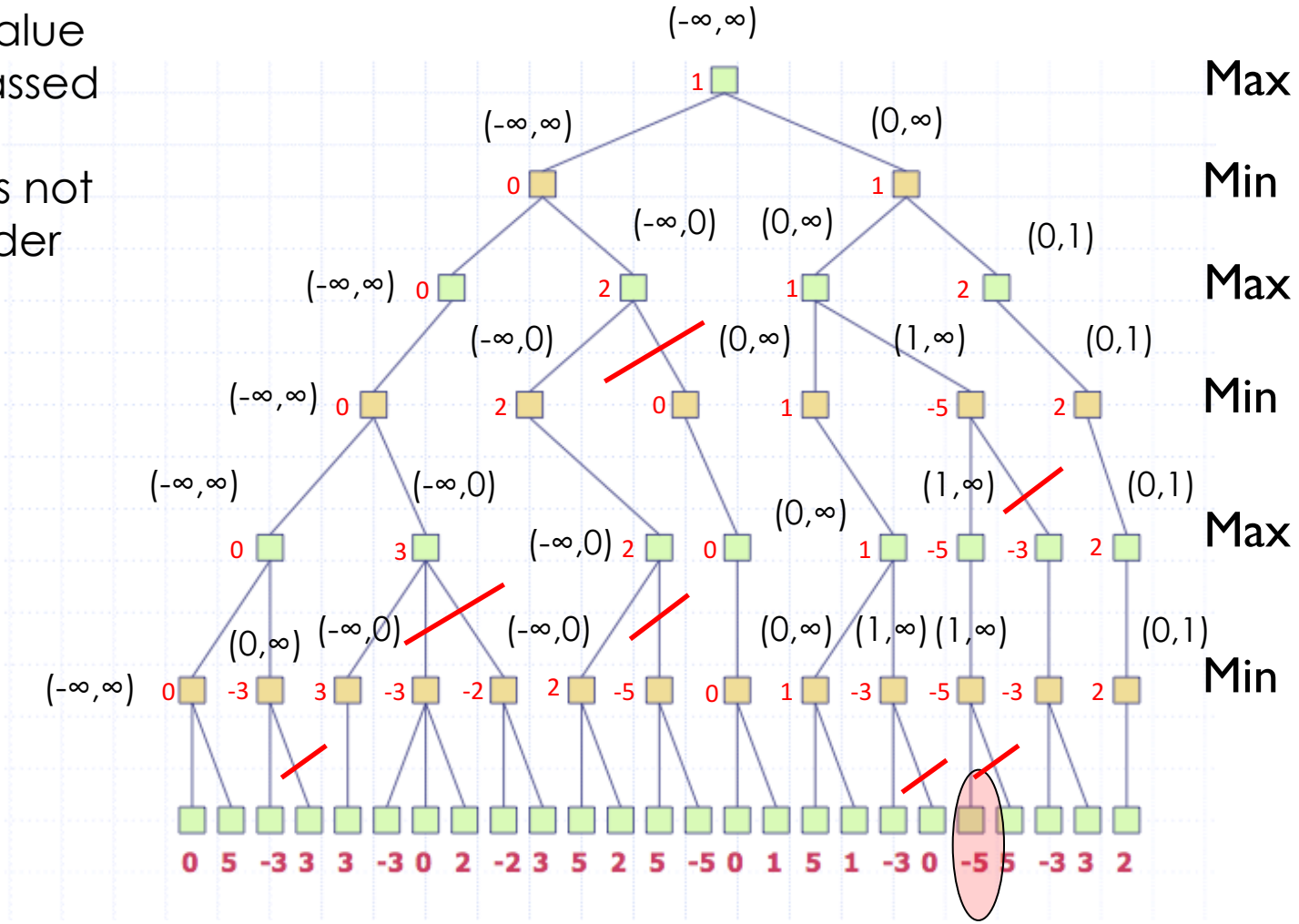
Last layer is MIN layer.

Prune after first child with value less than passed alpha
(alpha does not change under min node)

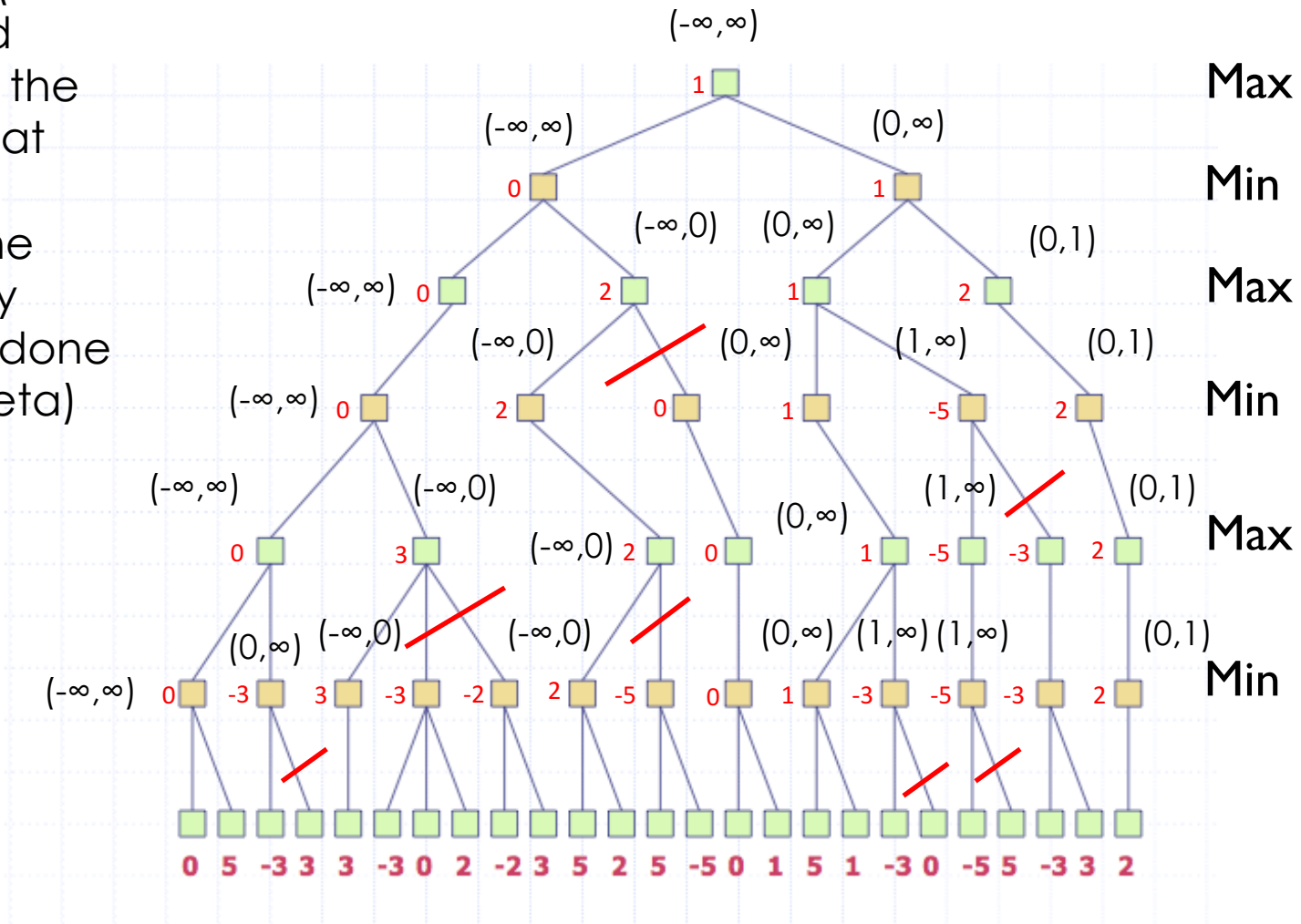


Last layer is MIN layer.

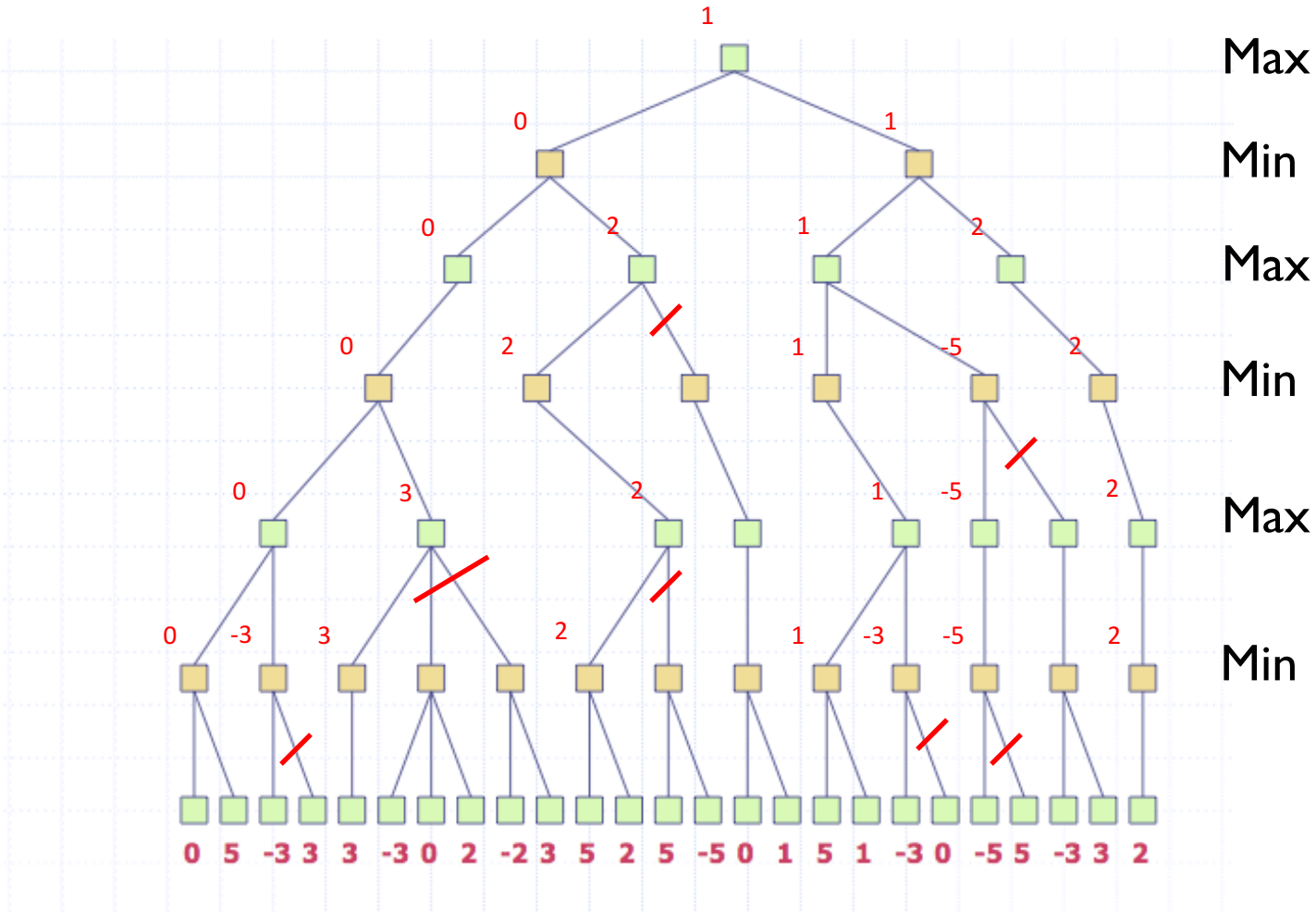
Prune after first child with value less than passed alpha
(alpha does not change under min node)



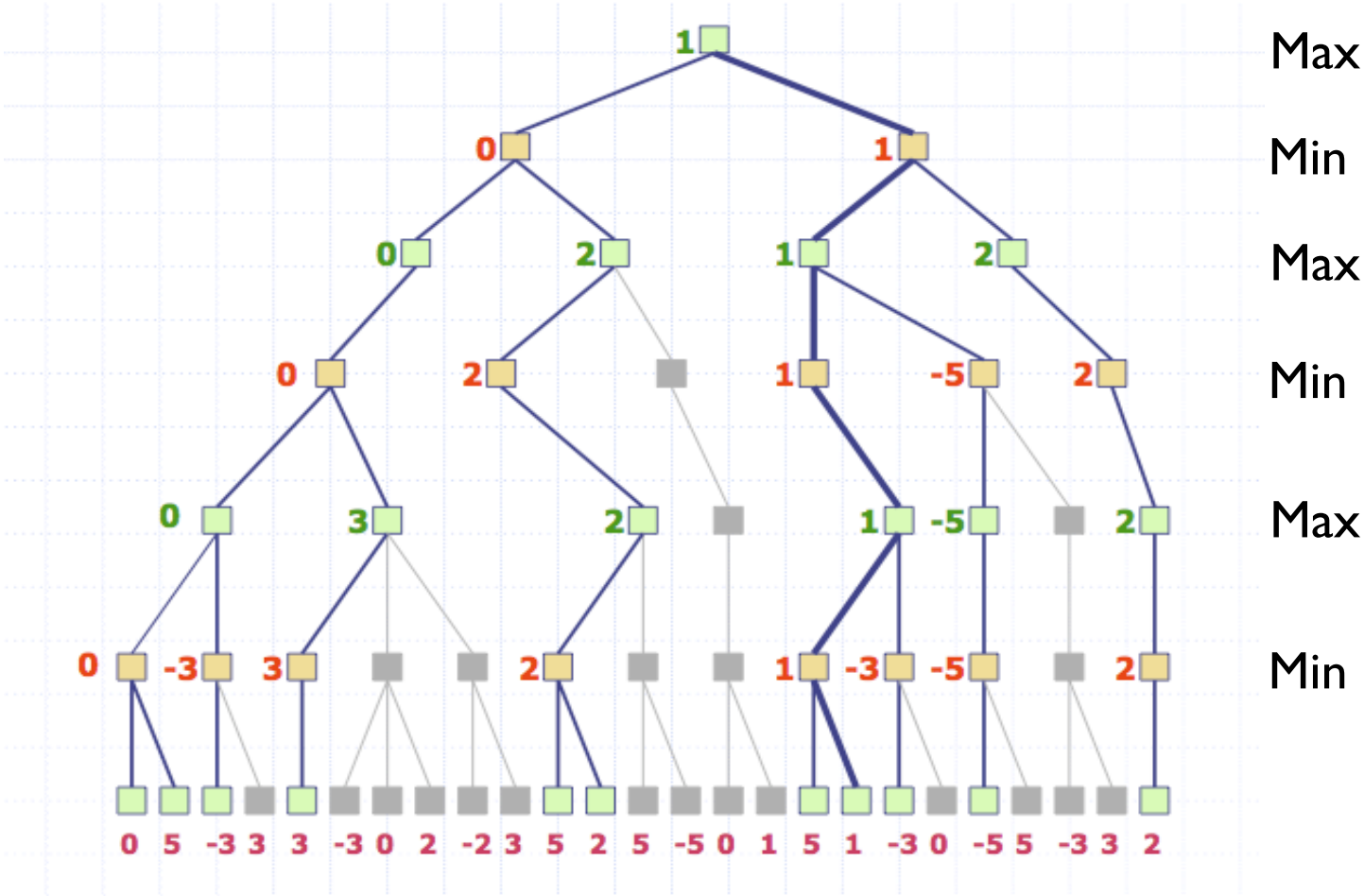
Note values of nodes in pruned subtrees are not computed (the work we did backing up the values so that we could compute the cuts level by level, is not done by alpha-beta)



Example


Max
Min
Max
Min
Max
Min
1
0 1
0 2 1 2
0 2 1 -5 2
0 3 2 1 -5 2
0 -3 3 2 1 -3 -5 2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2

Example


1
0
1
0
2
1
2
0
2
1
-5
2
0
3
2
1
-5
2
0
-3
3
2
1
-3
-5
2
0 5 -3 3 3 -3 0 2 -2 3 5 2 5 -5 0 1 5 1 -3 0 -5 5 -3 3 2
Max
Min
Max
Min
Max
Min