St. Francis Xavier University

# CSCI-564
# CONSTRAINT PROCESSING AND HEURISTIC SEARCH

LECTURE 17 – ADVERSARY SEARCH (CONT’D)

Dr. Jean-Alexis Delamer

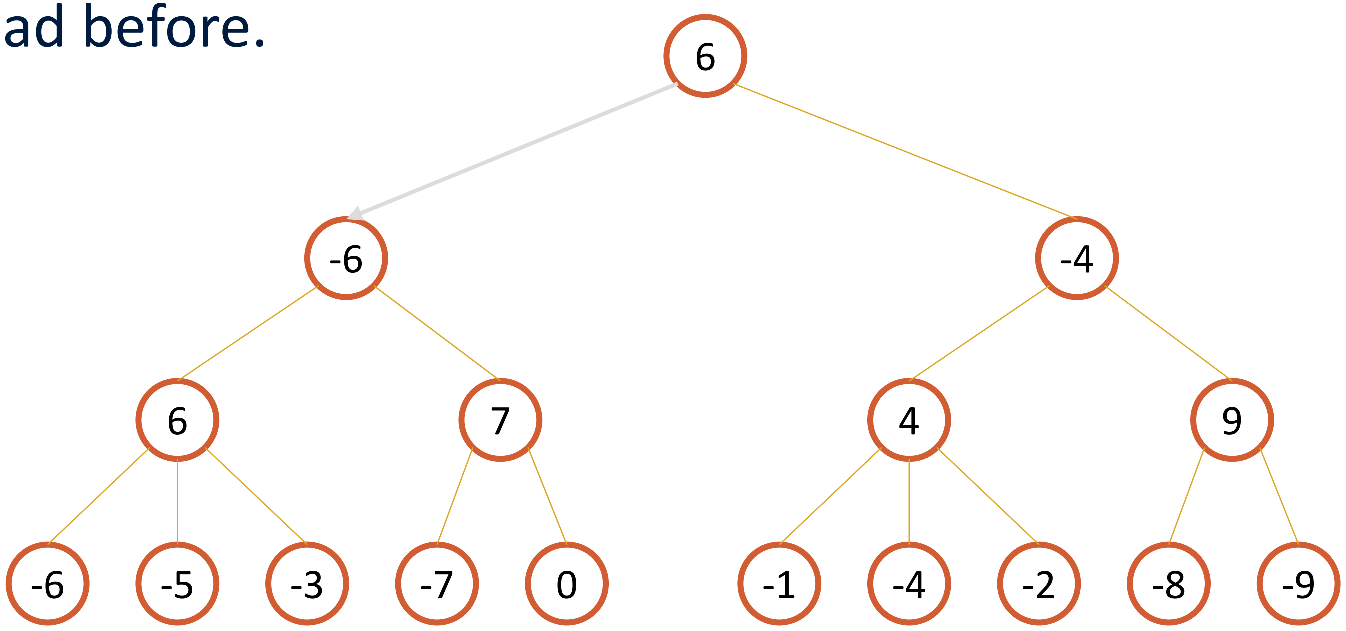
# Recap

- In Game Theory:
  - We want to calculate the optimal strategy.
  - A strategy (or policy) maps each state to an action.
- Tree Search:
  - We limit the search in depth.
  - More important to evaluate all the nodes before a certain depth.
- Two algorithms:
  - Minimax and negmax.
  - Similar ideas.

# $\alpha\beta$-Pruning

- Negmax performs a depth-first search of the game.
  - Explore the entire tree until a fixed depth.

- We fall in the same issue we had before.
  - The tree can grow very fast.

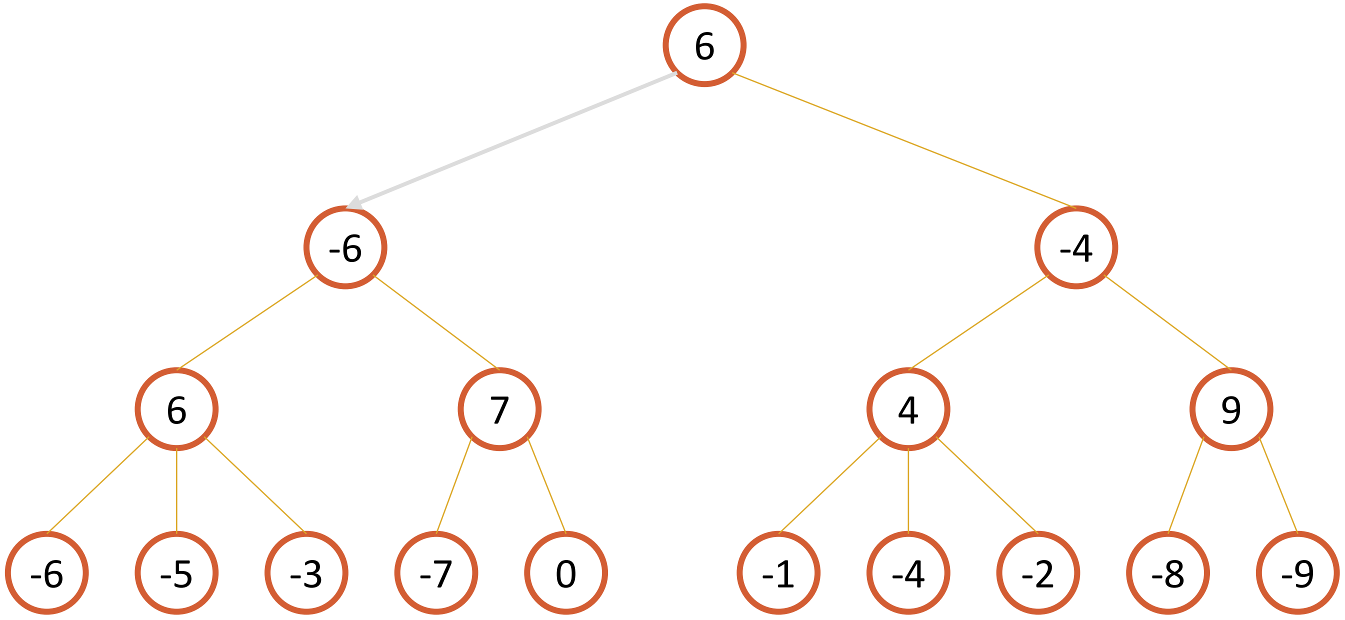



Negmax tree search

# $\alpha\beta$-Pruning

- $\alpha\beta$-pruning is a branch-and-bound method.
- The idea is simple:
  - Computing the root value while avoiding a part of the tree.

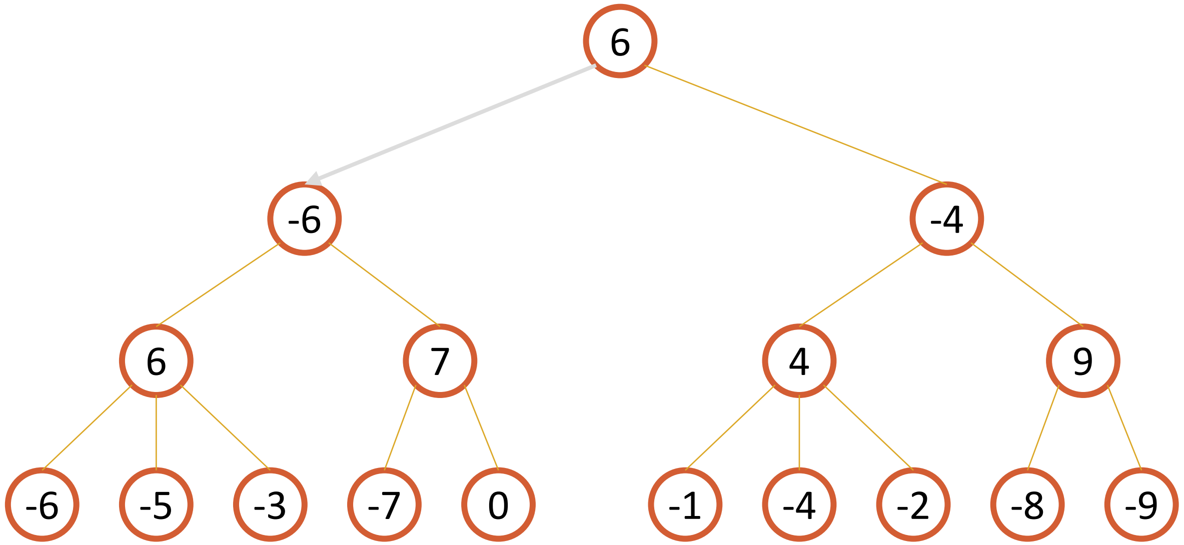
- Do you have an idea?




Negmax tree search

# $\alpha\beta$-Pruning

- $\alpha\beta$-pruning uses two bounds called $\alpha\beta$-window:
  - $\alpha$: The least value that the agent can achieve.
  - $\beta$: The maximum value that the opponent can achieve.
  - If the initial window is $(-\infty, \infty)$, it will determine the correct value




Negmax tree search

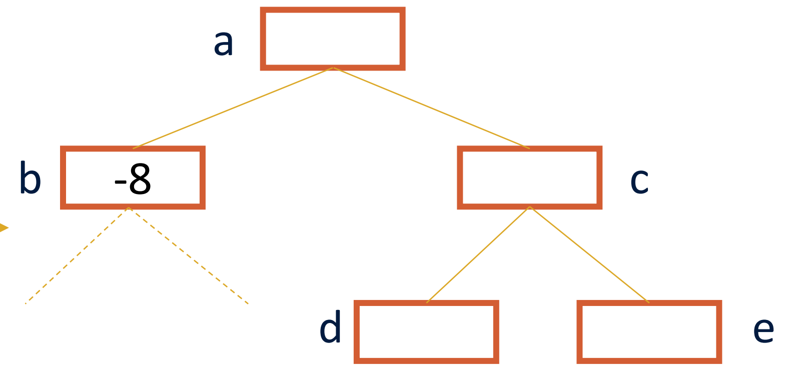
# $\alpha\beta$-Pruning

- It avoid nodes by achieving cut-offs.
- We can distinguish two types:
  - Shallow
  - Deep cut-offs

# $\alpha\beta$-Pruning

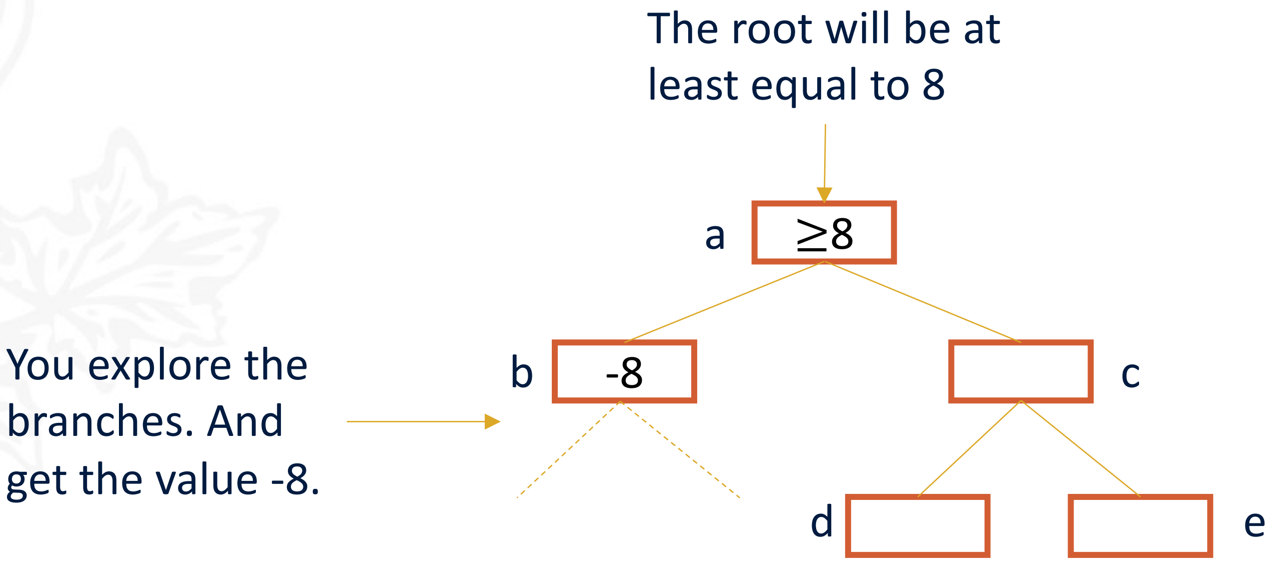
- Shallow cut-off.

You explore the branches. And get the value -8.



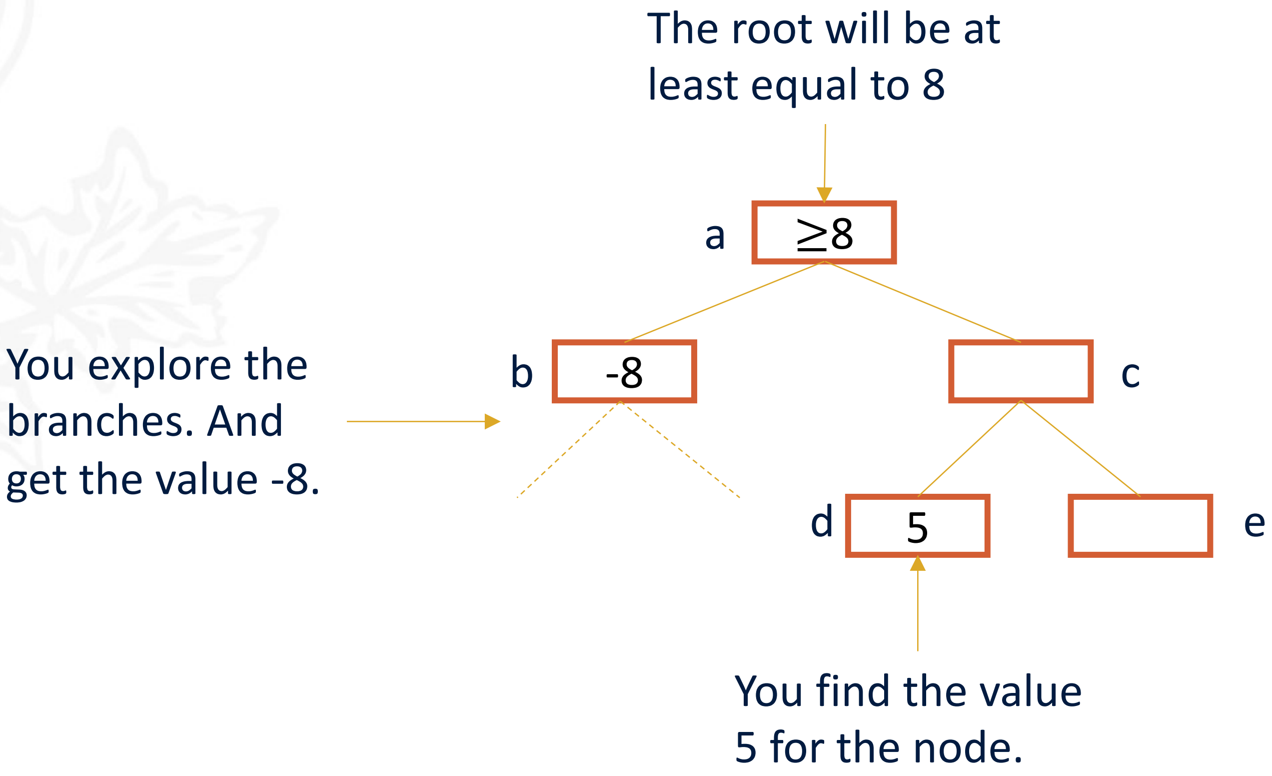
# $\alpha\beta$-Pruning

- Shallow cut-off.



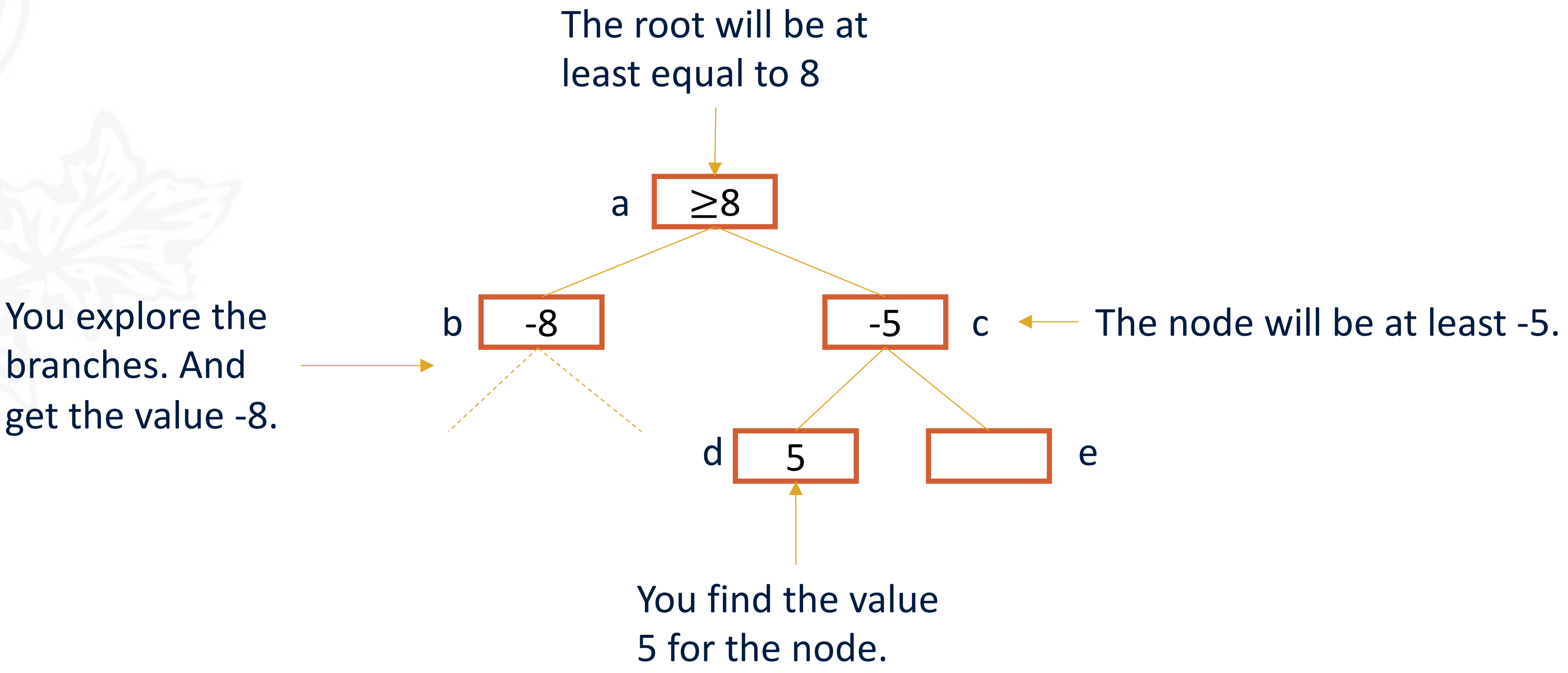
# $\alpha\beta$-Pruning

- Shallow cut-off.



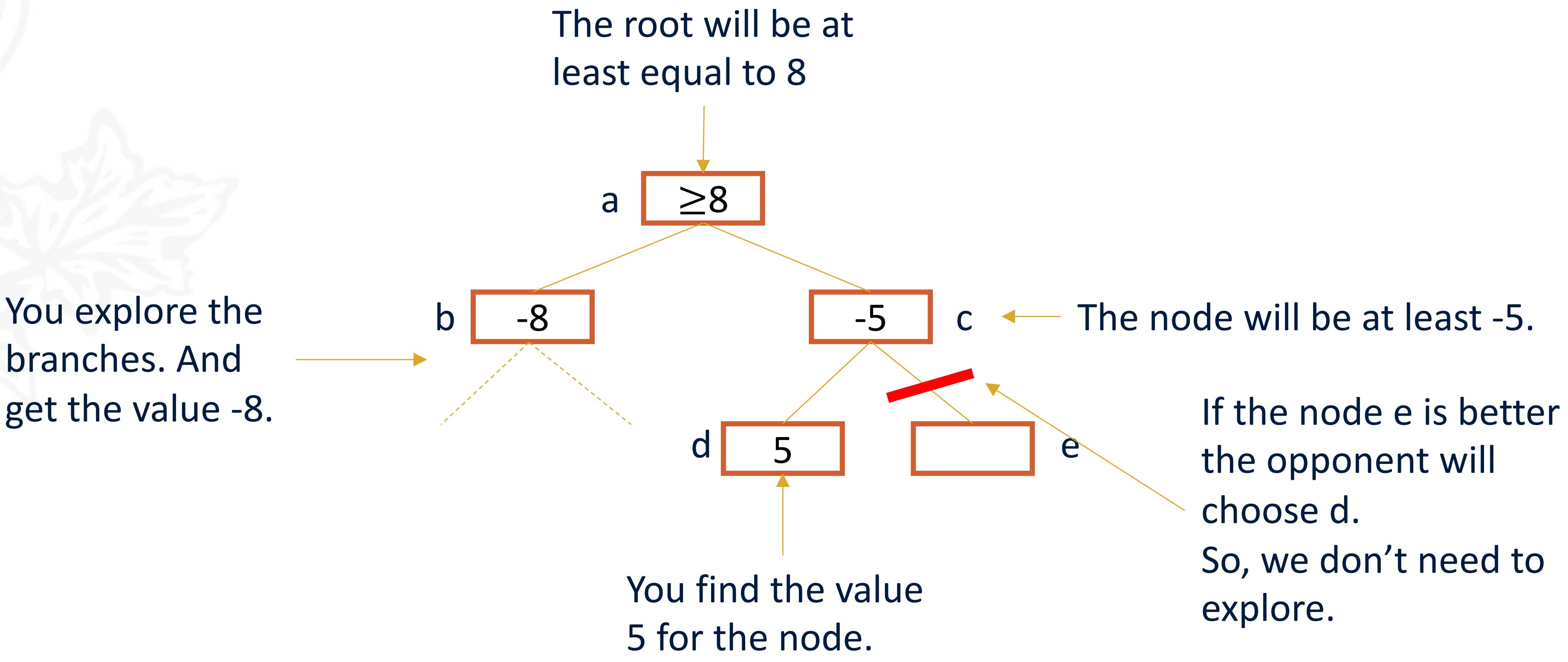
# $\alpha\beta$-Pruning

- Shallow cut-off.



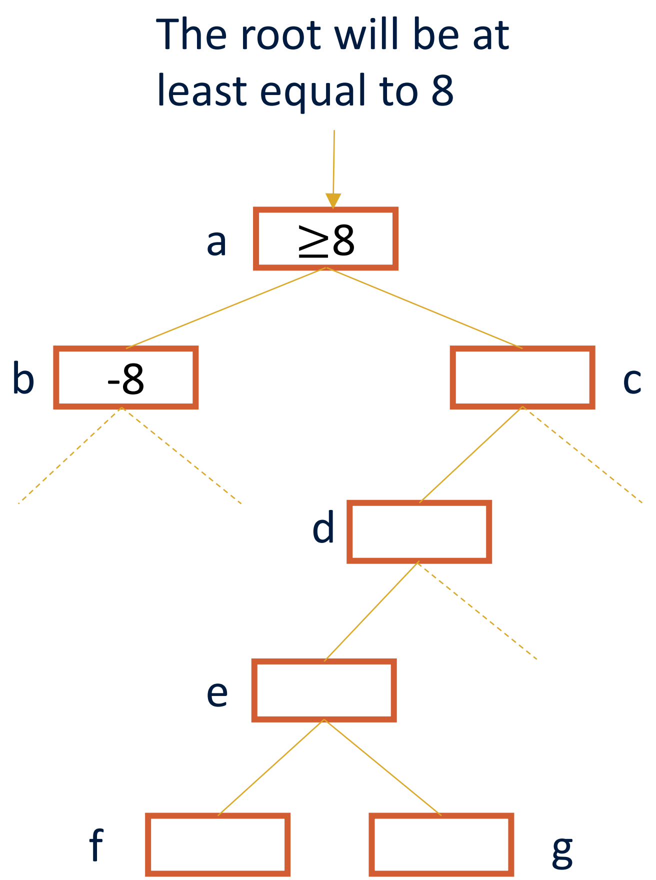
# $\alpha\beta$-Pruning

- Shallow cut-off.



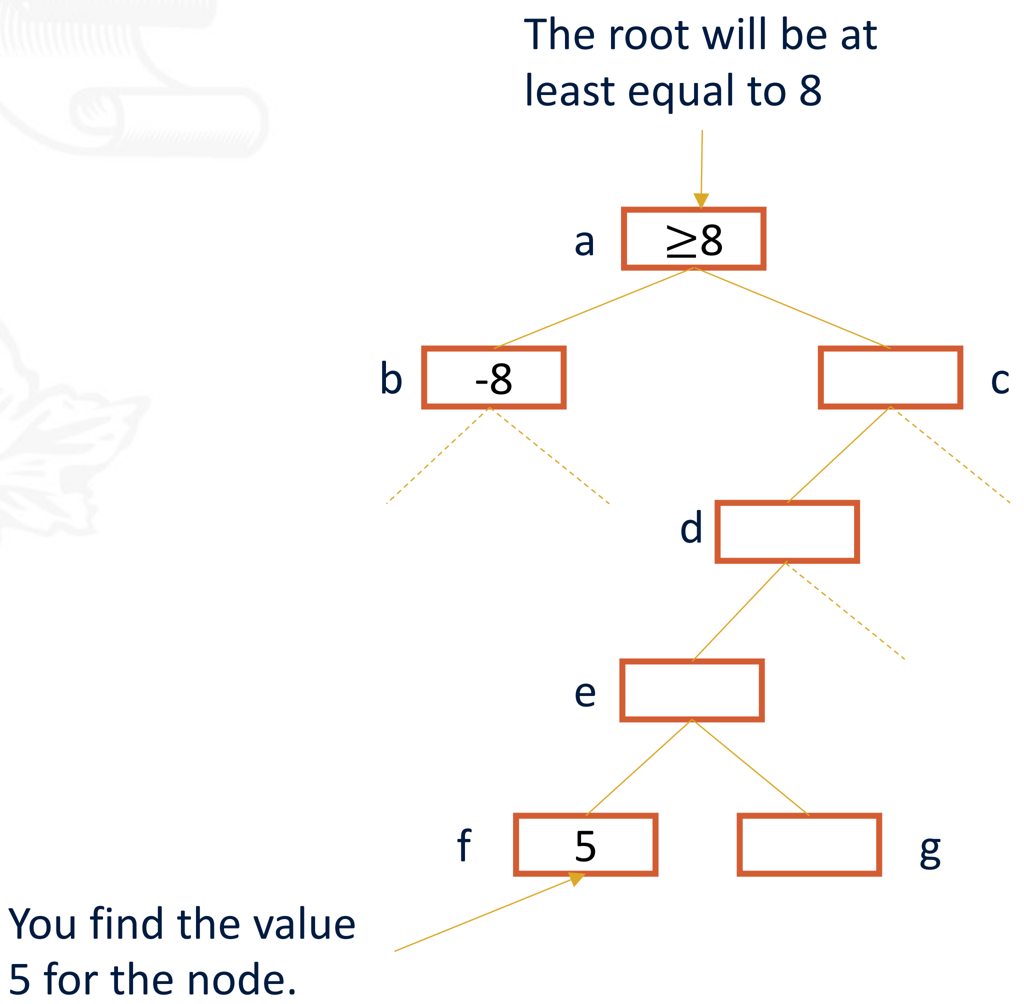
# $\alpha\beta$-Pruning

- Deep cut-off.



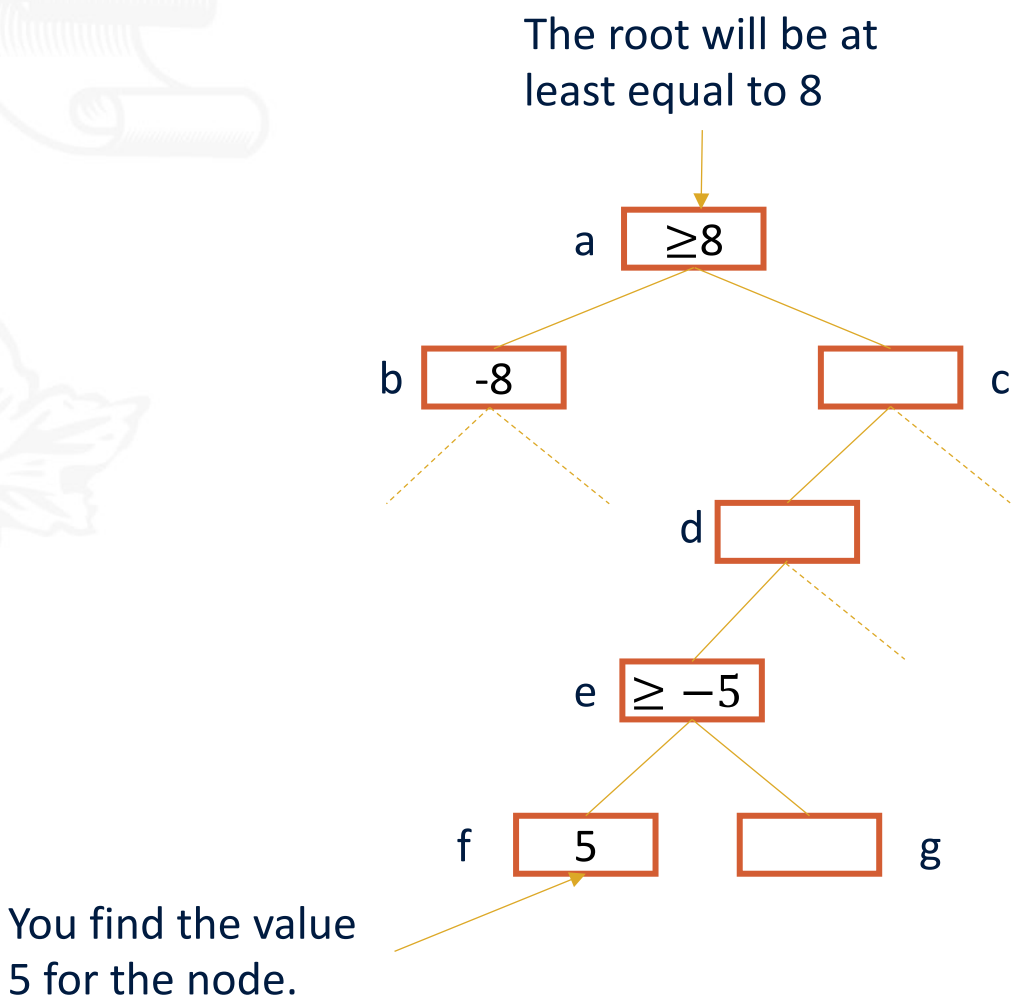
# $\alpha\beta$-Pruning

- Deep cut-off.



# $\alpha\beta$-Pruning

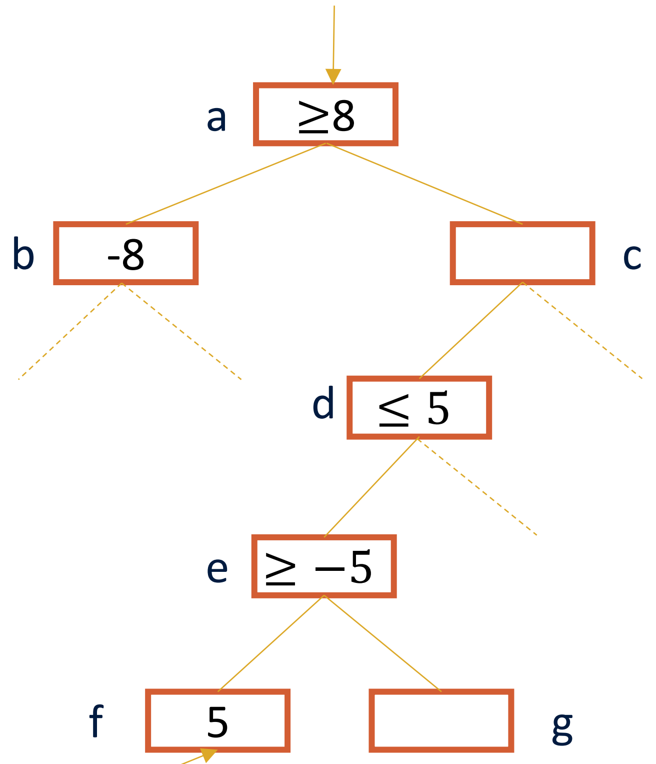
- Deep cut-off.



# $\alpha\beta$-Pruning

- Deep cut-off.

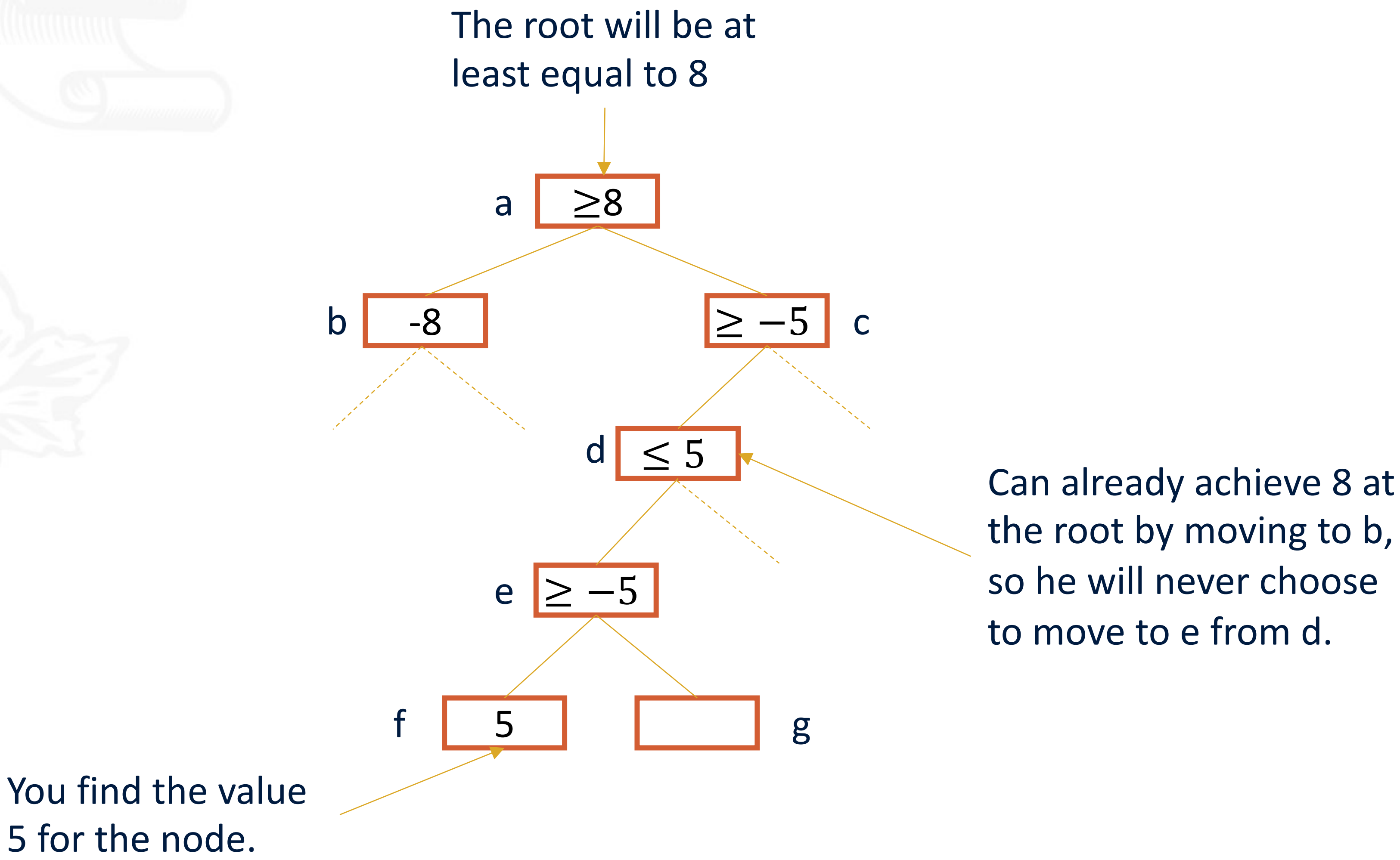
The root will be at least equal to 8




You find the value 5 for the node.

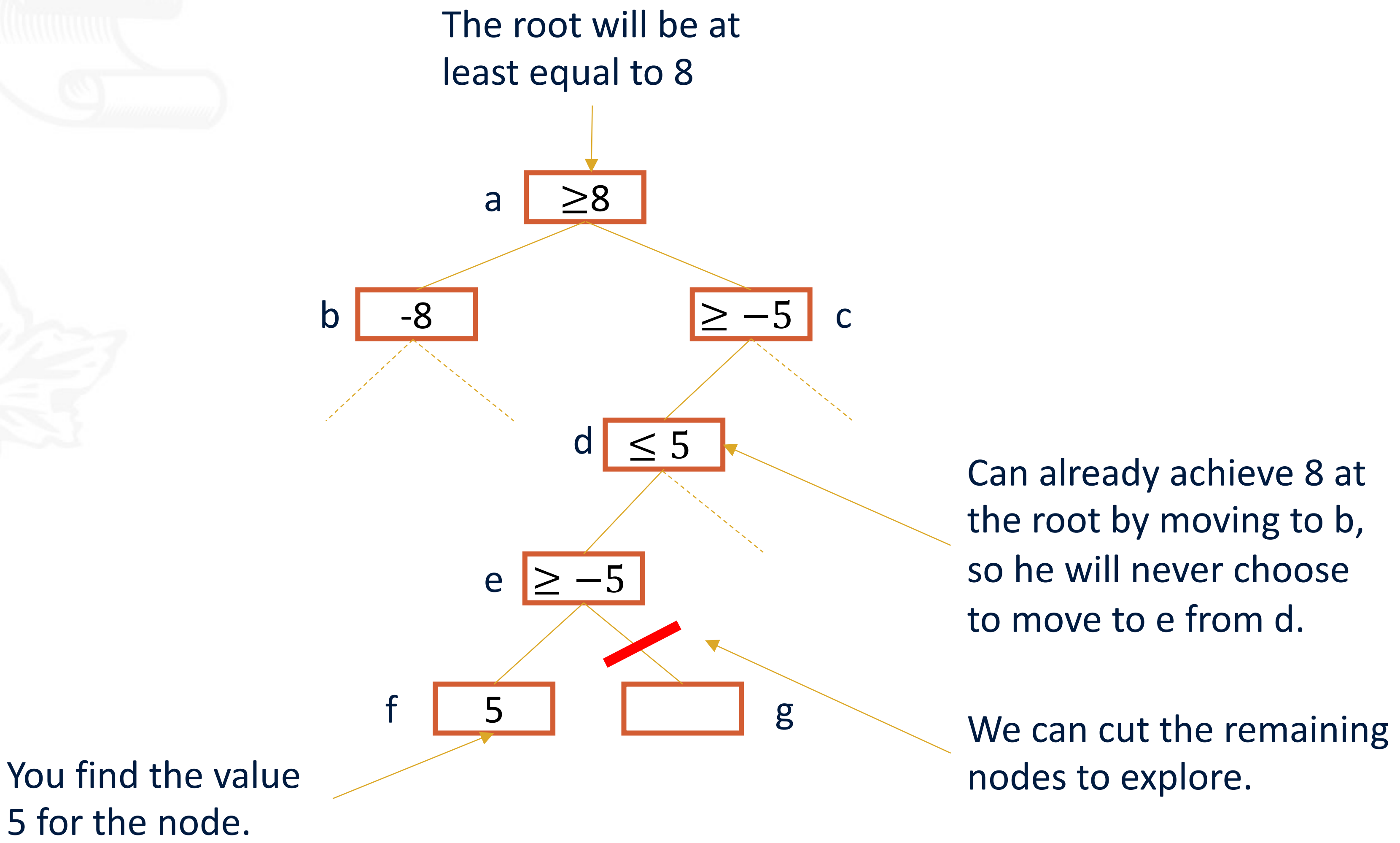
# $\alpha\beta$-Pruning

- Deep cut-off.



# $\alpha\beta$-Pruning

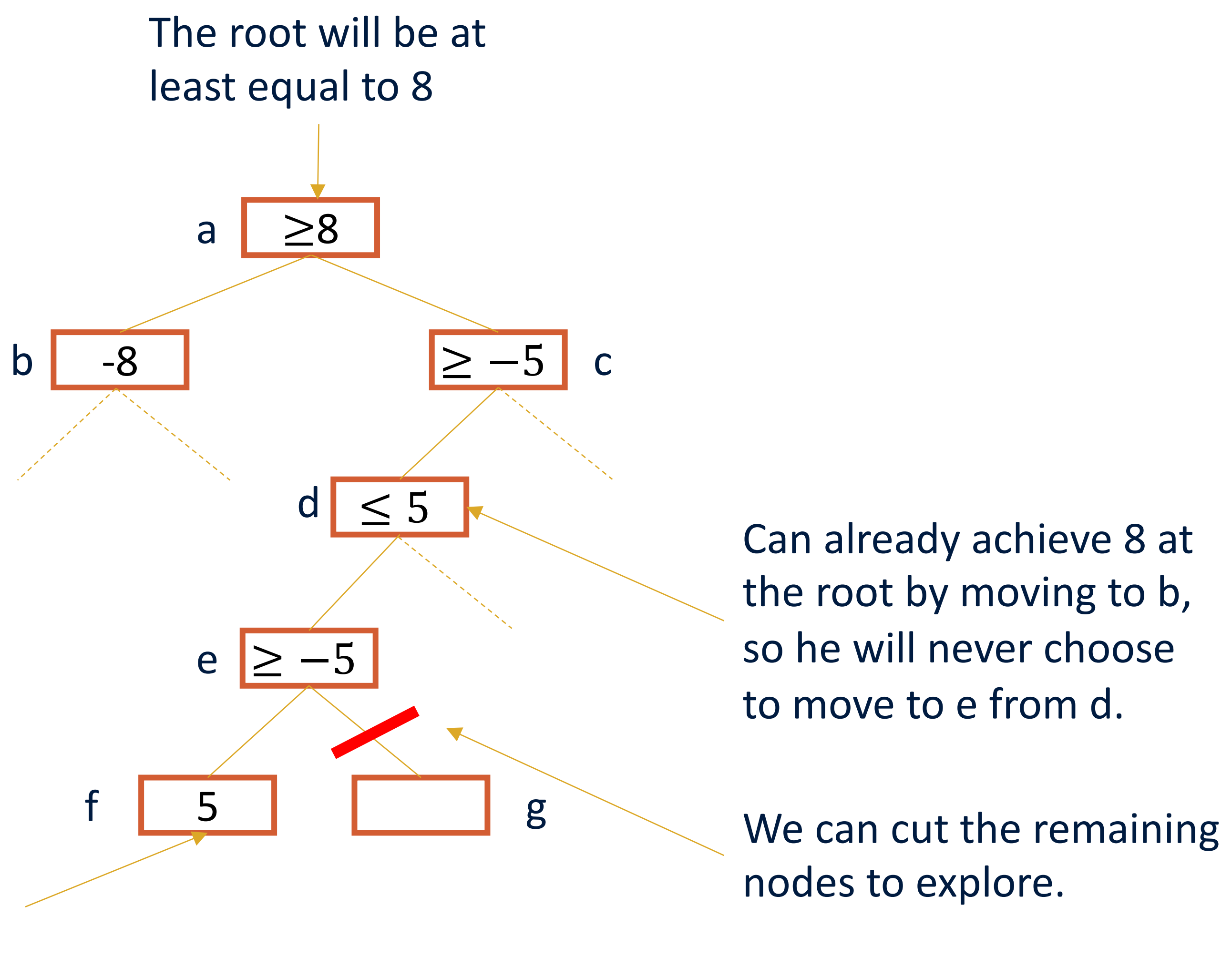
- Deep cut-off.



# $\alpha\beta$-Pruning

- Deep cut-off.

In deep pruning, the bound used for the cut-off can stem not only from the parent, but from any ancestor node.



# $\alpha\beta$-Pruning

**Procedure NegmaxAlphaBeta**
**Input:** Position $u$, bounds $\alpha$, $\beta$
**Output:** Value at root

| | |
|---|---|
| **if** $(leaf(u))$ **return** $Eval(u)$ | ;; No successor, static evaluation |
| $res \leftarrow \alpha$ | ;; Initialize value *res* for current frame |
| **for each** $v \in Succ(u)$ | ;; Traverse successor list |
|     $val \leftarrow -NegmaxAlphaBeta(v, -\beta, -res)$ | ;; Initialize cut-off value |
|     **if** $(val > res)$ $res \leftarrow val$ | ;; Update *res* |
|     **if** $(res \geq \beta)$ **return** *res* | ;; Perform cut-off |
| **return** *res* | ;; Return final evaluation |

# $\alpha\beta$-Pruning

- For any node $u$:
  - $\beta$ is the upper bound used to restrict the node below $u$.
  - A cut-off occurs when $u \geq \beta$.
- From the opponent point of view:
  - Can choose a move that avoids $u$ with a value no greater then $\beta$.
  - The alternate move is no worse than $u$, so searching below $p$ is not necessary.

**Procedure NegmaxAlphaBeta**
**Input:** Position $u$, bounds $\alpha$, $\beta$
**Output:** Value at root

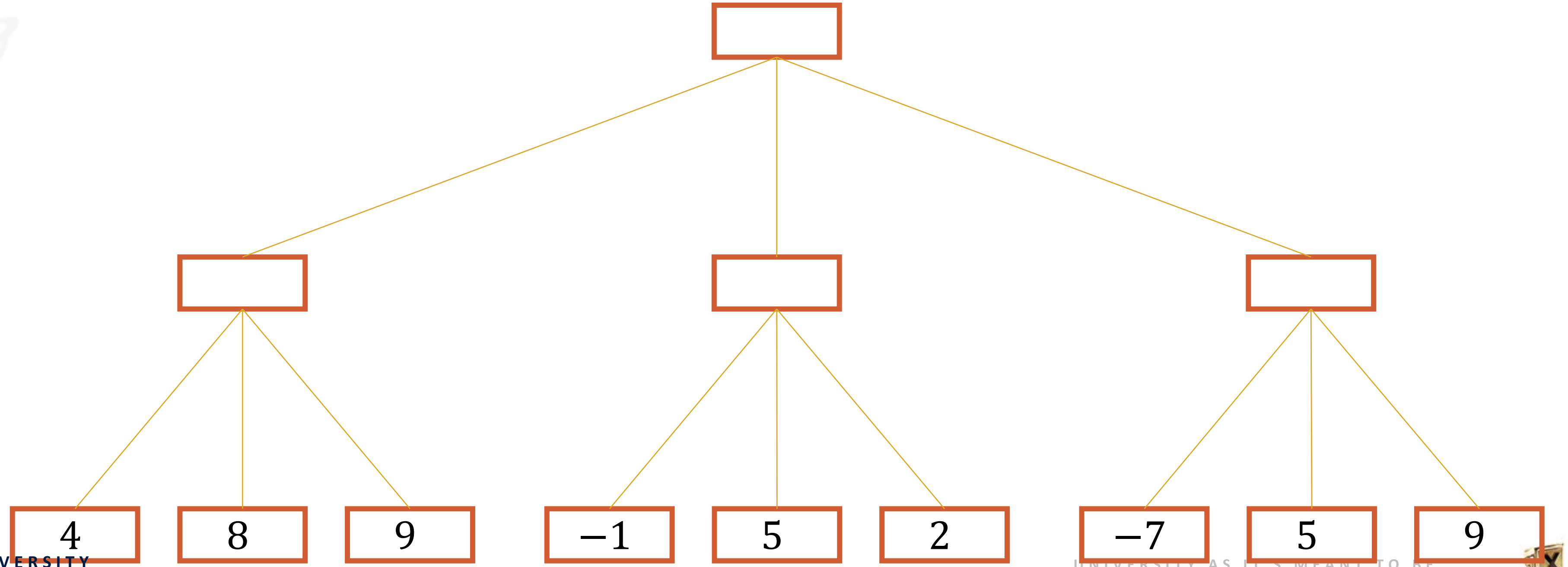
**if** $(leaf(u))$ **return** $Eval(u)$ ;; No successor, static evaluation
$res \leftarrow \alpha$ ;; Initialize value *res* for current frame
**for each** $v \in Succ(u)$ ;; Traverse successor list
    $val \leftarrow -NegmaxAlphaBeta(v, -\beta, -res)$ ;; Initialize cut-off value
    **if** $(val > res)$ $res \leftarrow val$ ;; Update *res*
    **if** $(res \geq \beta)$ **return** *res* ;; Perform cut-off
**return** *res* ;; Return final evaluation

# $\alpha\beta$-Pruning

```
Procedure MinimaxAlphaBeta
Input: Position u, value α, value β
Output: Value at root

if (leaf(u)) return Eval(p)                              ;; No successor, return evaluation
if (max-node(u))                                         ;; MAX node
   res ← α                                               ;; Initialize result value
   for each v ∈ Succ(u)                                  ;; Traverse successor list
         val ← MinimaxAlphaBeta(v, res, β)               ;; Recursion for α
         res ← max{res, val}                             ;; Take maximal value
         if (res ≥ β)                                    ;; Result exceeds threshold
            return res                                   ;; Propagate value
else                                                     ;; MIN node
   res ← β                                               ;; Initialize result value
   for each v ∈ Succ(u)                                  ;; Traverse successor list
         val ← MinimaxAlphaBeta(v, α, res)               ;; Recursion for β
         res ← min{res, val}                             ;; Take minimal value
         if (res ≤ α)                                    ;; Result exceeds threshold
            return res                                   ;; Propagate value
return res                                               ;; Propagate value
```

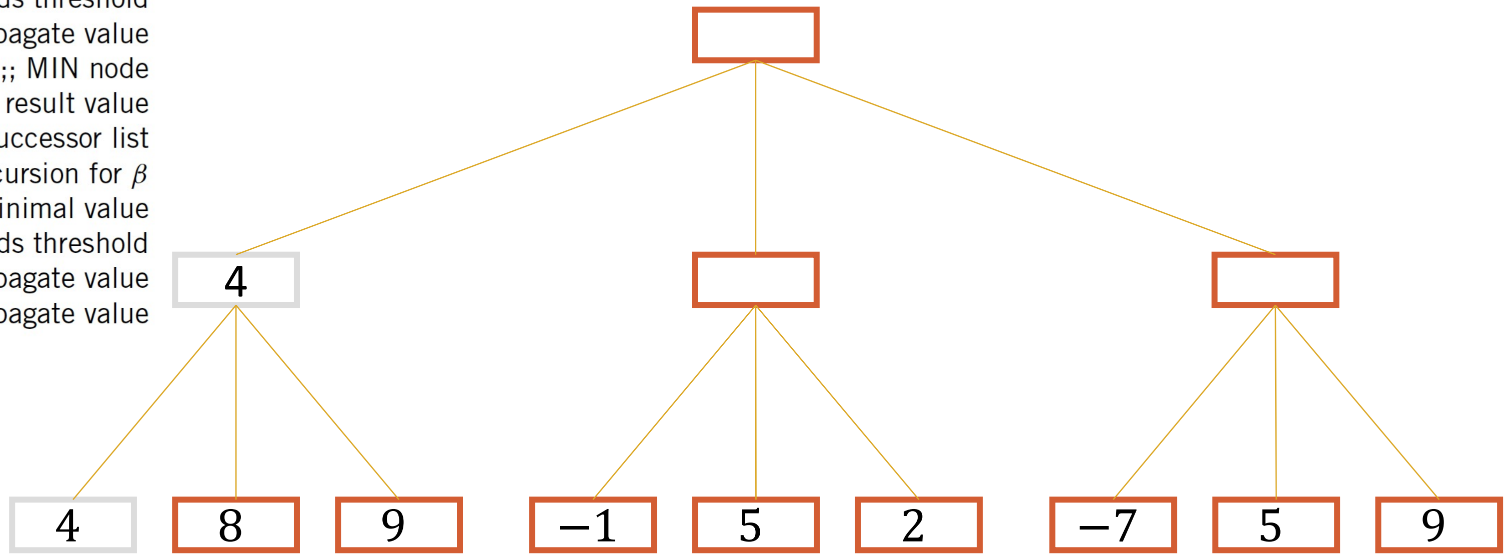
# $\alpha\beta$-Pruning



# $\alpha\beta$-Pruning

**Procedure MinimaxAlphaBeta**
**Input:** Position $u$, value $\alpha$, value $\beta$
**Output:** Value at root

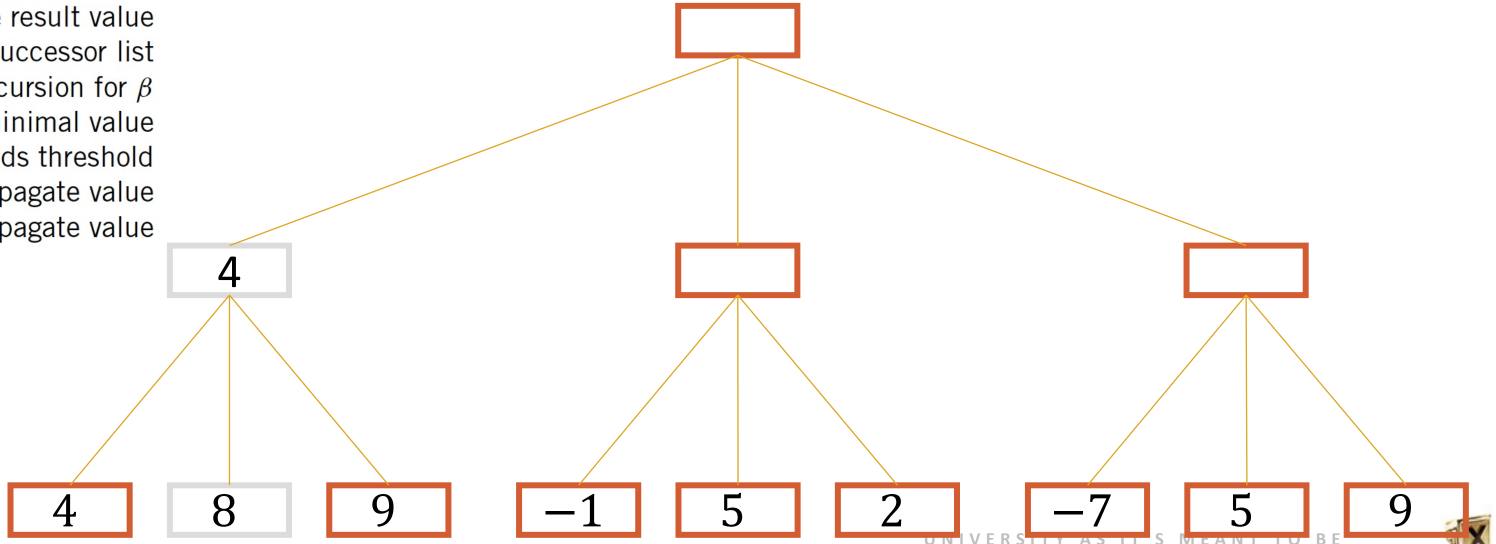
```
if (leaf(u)) return Eval(p)                     ;; No successor, return evaluation
if (max-node(u))                                ;; MAX node
   res ← α                                      ;; Initialize result value
   for each v ∈ Succ(u)                         ;; Traverse successor list
        val ← MinimaxAlphaBeta(v, res, β)       ;; Recursion for α
        res ← max{res, val}                     ;; Take maximal value
        if (res ≥ β)                            ;; Result exceeds threshold
           return res                           ;; Propagate value
else                                            ;; MIN node
   res ← β                                      ;; Initialize result value
   for each v ∈ Succ(u)                         ;; Traverse successor list
        val ← MinimaxAlphaBeta(v, α, res)       ;; Recursion for β
        res ← min{res, val}                     ;; Take minimal value
        if (res ≤ α)                            ;; Result exceeds threshold
           return res                           ;; Propagate value
return res                                      ;; Propagate value
```



# αβ-Pruning

**Procedure MinimaxAlphaBeta**
**Input:** Position $u$, value $\alpha$, value $\beta$
**Output:** Value at root

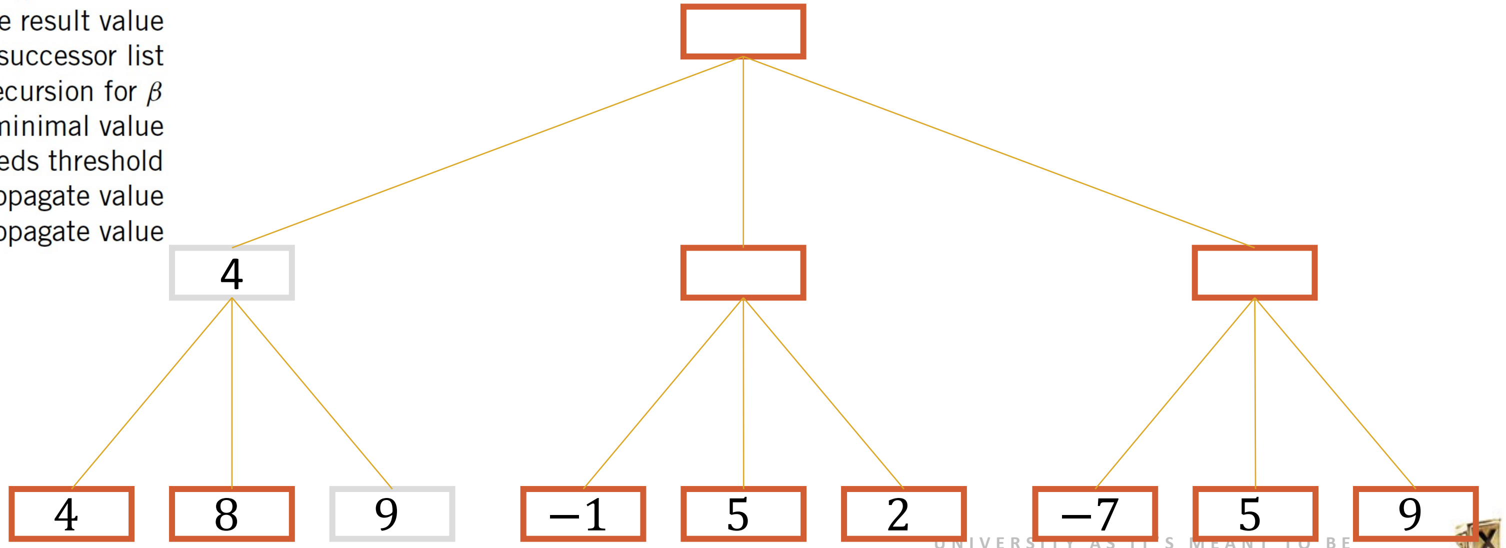
```
if (leaf(u)) return Eval(p)                          ;; No successor, return evaluation
if (max-node(u))                                     ;; MAX node
   res ← α                                           ;; Initialize result value
   for each v ∈ Succ(u)                              ;; Traverse successor list
         val ← MinimaxAlphaBeta(v, res, β)           ;; Recursion for α
         res ← max{res, val}                         ;; Take maximal value
         if (res ≥ β)                                ;; Result exceeds threshold
            return res                               ;; Propagate value
else                                                 ;; MIN node
   res ← β                                           ;; Initialize result value
   for each v ∈ Succ(u)                              ;; Traverse successor list
         val ← MinimaxAlphaBeta(v, α, res)           ;; Recursion for β
         res ← min{res, val}                         ;; Take minimal value
         if (res ≤ α)                                ;; Result exceeds threshold
            return res                               ;; Propagate value
return res                                           ;; Propagate value
```



# $\alpha\beta$-Pruning

**Procedure MinimaxAlphaBeta**
**Input:** Position $u$, value $\alpha$, value $\beta$
**Output:** Value at root

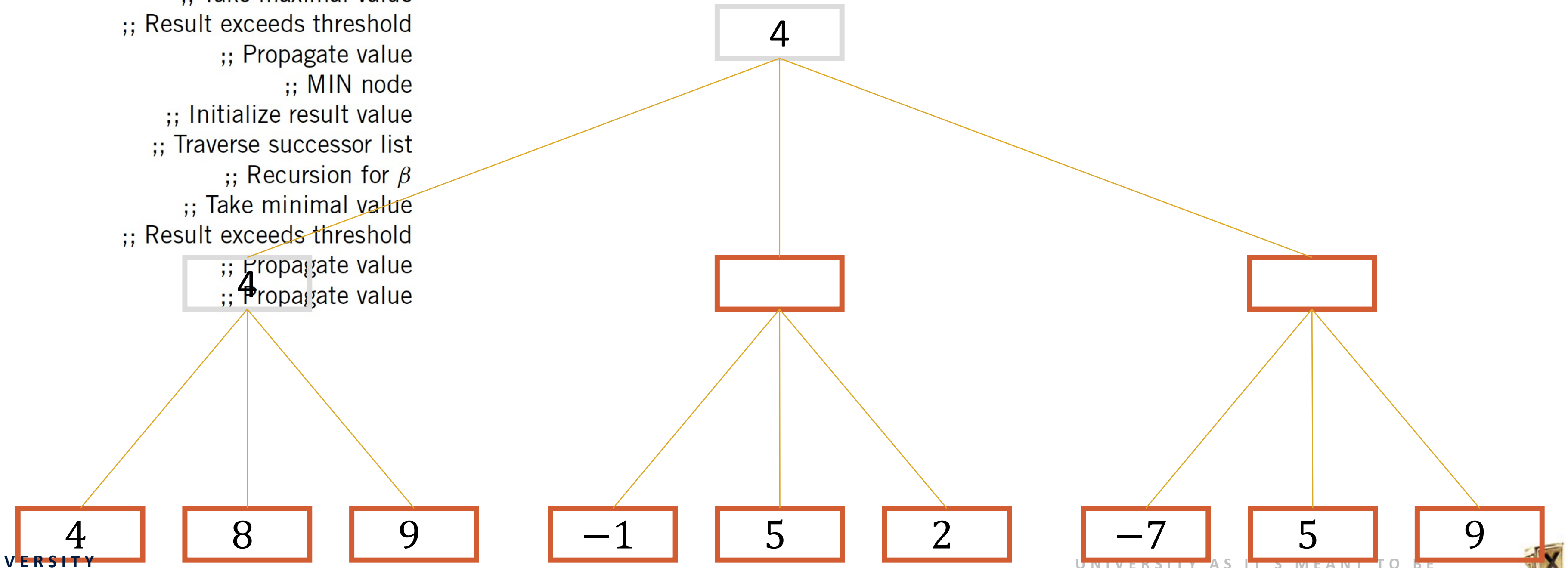
```
if (leaf(u)) return Eval(p)                    ;; No successor, return evaluation
if (max-node(u))                               ;; MAX node
   res ← α                                     ;; Initialize result value
   for each v ∈ Succ(u)                        ;; Traverse successor list
        val ← MinimaxAlphaBeta(v, res, β)      ;; Recursion for α
        res ← max{res, val}                    ;; Take maximal value
        if (res ≥ β)                           ;; Result exceeds threshold
           return res                          ;; Propagate value
else                                           ;; MIN node
   res ← β                                     ;; Initialize result value
   for each v ∈ Succ(u)                        ;; Traverse successor list
        val ← MinimaxAlphaBeta(v, α, res)      ;; Recursion for β
        res ← min{res, val}                    ;; Take minimal value
        if (res ≤ α)                           ;; Result exceeds threshold
           return res                          ;; Propagate value
return res                                     ;; Propagate value
```



# $\alpha\beta$-Pruning

**Procedure MinimaxAlphaBeta**
**Input:** Position $u$, value $\alpha$, value $\beta$
**Output:** Value at root

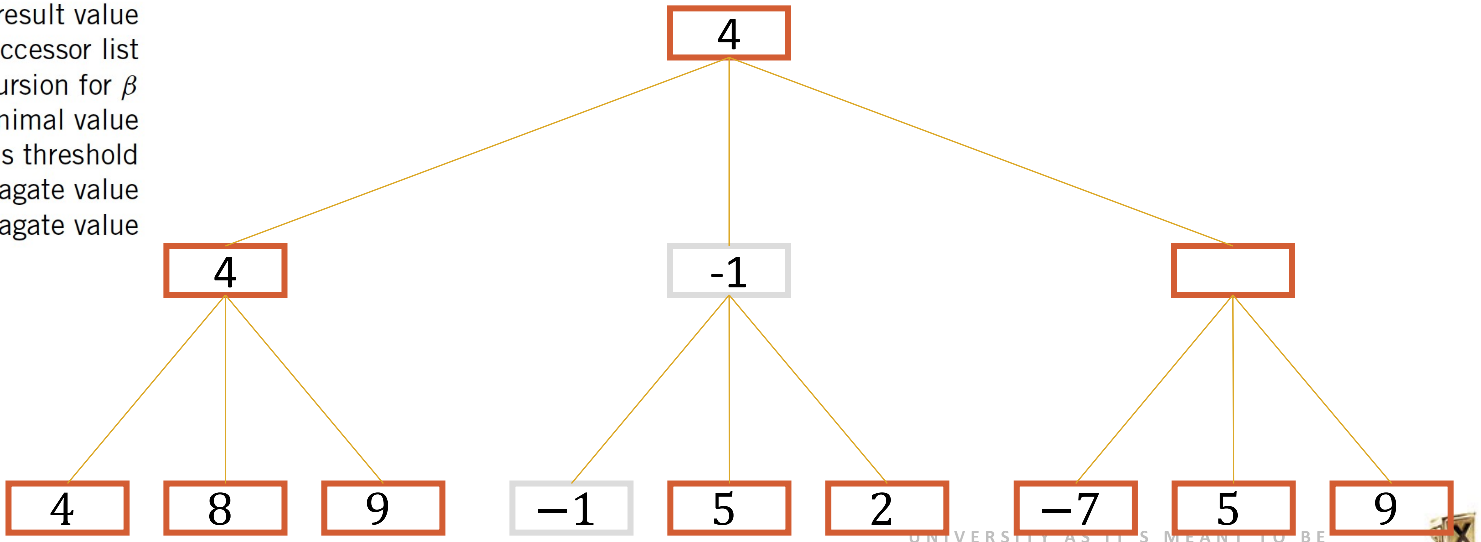
```
if (leaf(u)) return Eval(p)                 ;; No successor, return evaluation
if (max-node(u))                            ;; MAX node
   res ← α                                  ;; Initialize result value
   for each v ∈ Succ(u)                     ;; Traverse successor list
      val ← MinimaxAlphaBeta(v, res, β)     ;; Recursion for α
      res ← max{res, val}                   ;; Take maximal value
      if (res ≥ β)                          ;; Result exceeds threshold
         return res                         ;; Propagate value
else                                        ;; MIN node
   res ← β                                  ;; Initialize result value
   for each v ∈ Succ(u)                     ;; Traverse successor list
      val ← MinimaxAlphaBeta(v, α, res)     ;; Recursion for β
      res ← min{res, val}                   ;; Take minimal value
      if (res ≤ α)                          ;; Result exceeds threshold
         return res                         ;; Propagate value
return res                                  ;; Propagate value
```



# $\alpha\beta$-Pruning

**Procedure MinimaxAlphaBeta**
**Input:** Position $u$, value $\alpha$, value $\beta$
**Output:** Value at root

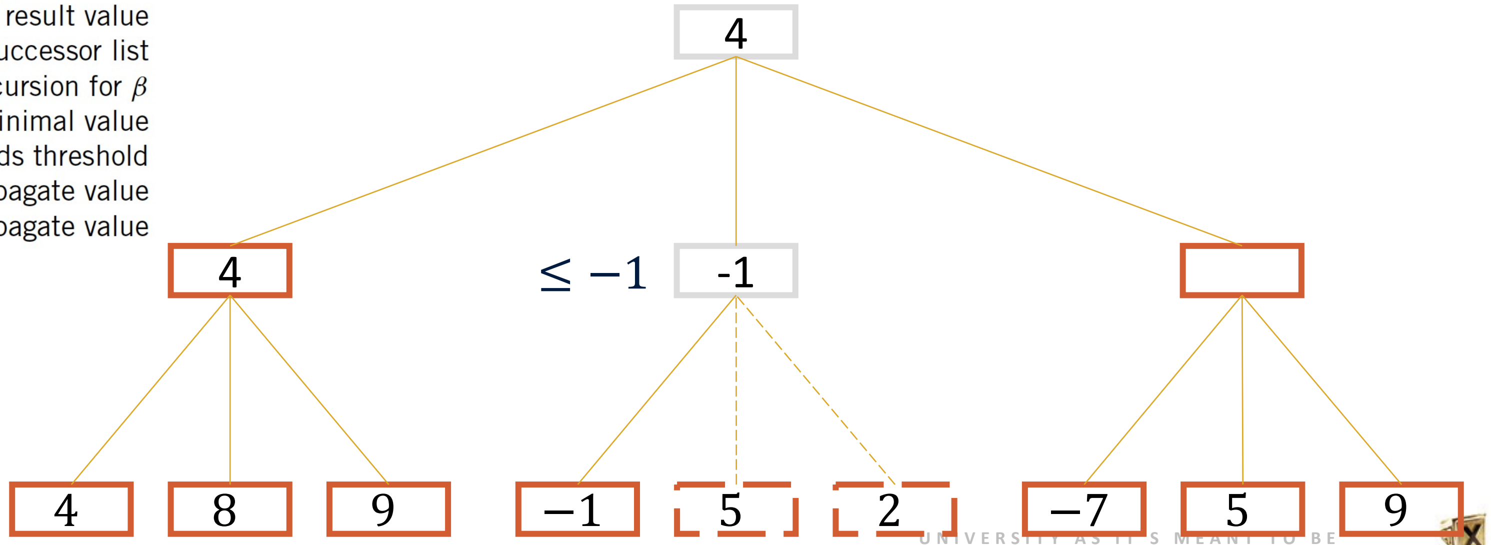
```
if (leaf(u)) return Eval(p)                       ;; No successor, return evaluation
if (max-node(u))                                  ;; MAX node
   res ← α                                        ;; Initialize result value
   for each v ∈ Succ(u)                           ;; Traverse successor list
       val ← MinimaxAlphaBeta(v, res, β)          ;; Recursion for α
       res ← max{res, val}                        ;; Take maximal value
       if (res ≥ β)                               ;; Result exceeds threshold
          return res                              ;; Propagate value
else                                              ;; MIN node
   res ← β                                        ;; Initialize result value
   for each v ∈ Succ(u)                           ;; Traverse successor list
       val ← MinimaxAlphaBeta(v, α, res)          ;; Recursion for β
       res ← min{res, val}                        ;; Take minimal value
       if (res ≤ α)                               ;; Result exceeds threshold
          return res                              ;; Propagate value
return res                                        ;; Propagate value
```



# $\alpha\beta$-Pruning

**Procedure MinimaxAlphaBeta**
**Input:** Position $u$, value $\alpha$, value $\beta$
**Output:** Value at root

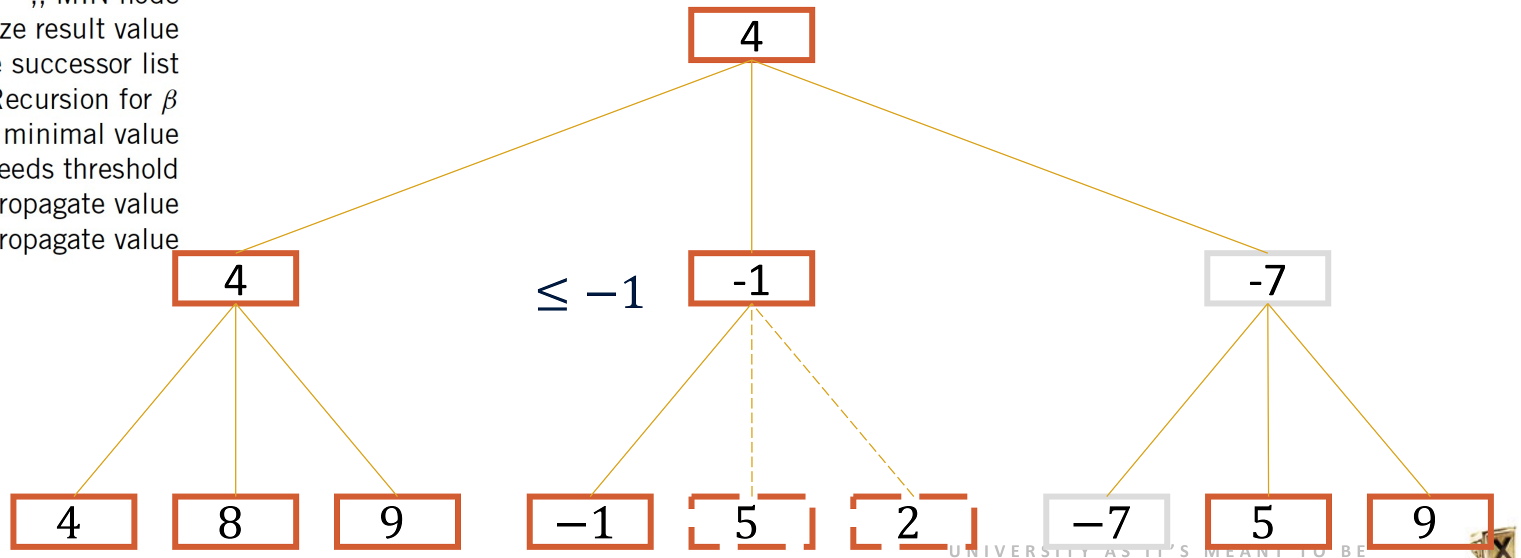
```
if (leaf(u)) return Eval(p)                   ;; No successor, return evaluation
if (max-node(u))                              ;; MAX node
   res ← α                                    ;; Initialize result value
   for each v ∈ Succ(u)                       ;; Traverse successor list
         val ← MinimaxAlphaBeta(v, res, β)    ;; Recursion for α
         res ← max{res, val}                  ;; Take maximal value
         if (res ≥ β)                         ;; Result exceeds threshold
            return res                        ;; Propagate value
else                                          ;; MIN node
   res ← β                                    ;; Initialize result value
   for each v ∈ Succ(u)                       ;; Traverse successor list
         val ← MinimaxAlphaBeta(v, α, res)    ;; Recursion for β
         res ← min{res, val}                  ;; Take minimal value
         if (res ≤ α)                         ;; Result exceeds threshold
            return res                        ;; Propagate value
return res                                    ;; Propagate value
```



# $\alpha\beta$-Pruning

**Procedure MinimaxAlphaBeta**
**Input:** Position $u$, value $\alpha$, value $\beta$
**Output:** Value at root

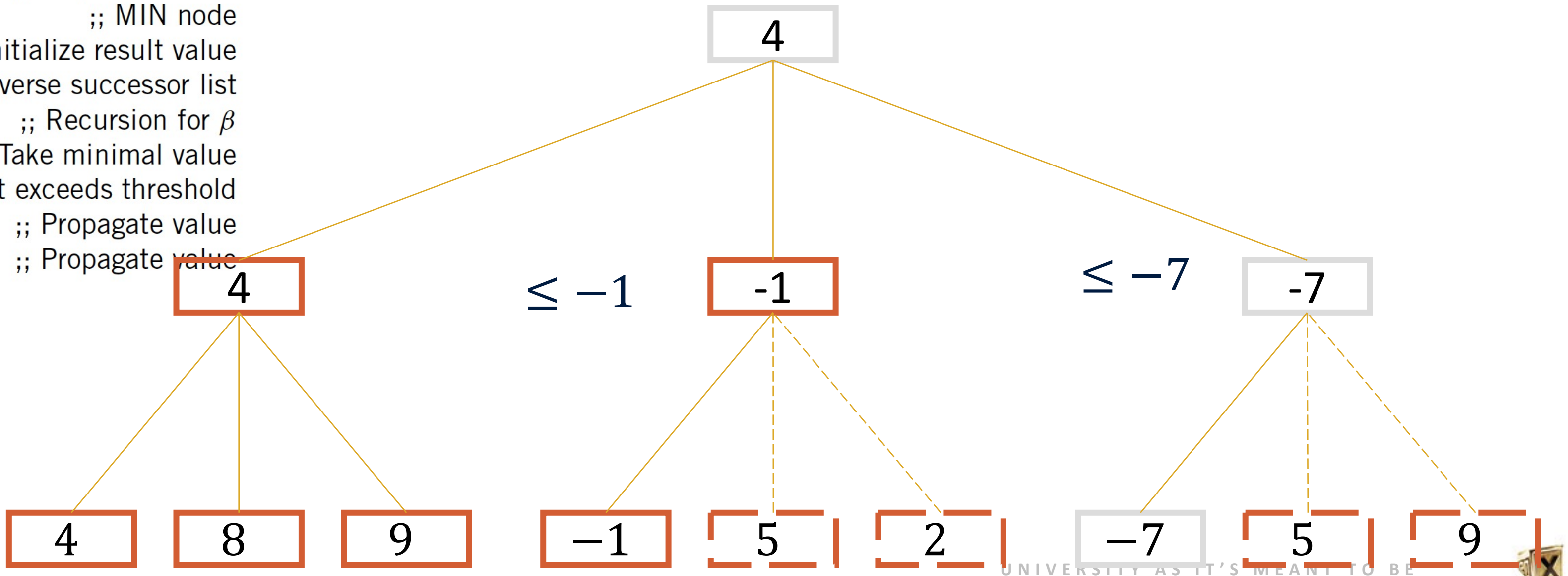
```
if (leaf(u)) return Eval(p)                       ;; No successor, return evaluation
if (max-node(u))                                  ;; MAX node
   res ← α                                        ;; Initialize result value
   for each v ∈ Succ(u)                           ;; Traverse successor list
        val ← MinimaxAlphaBeta(v, res, β)         ;; Recursion for α
        res ← max{res, val}                       ;; Take maximal value
        if (res ≥ β)                              ;; Result exceeds threshold
           return res                             ;; Propagate value
else                                              ;; MIN node
   res ← β                                        ;; Initialize result value
   for each v ∈ Succ(u)                           ;; Traverse successor list
        val ← MinimaxAlphaBeta(v, α, res)         ;; Recursion for β
        res ← min{res, val}                       ;; Take minimal value
        if (res ≤ α)                              ;; Result exceeds threshold
           return res                             ;; Propagate value
return res                                        ;; Propagate value
```



# $\alpha\beta$-Pruning

```
Procedure MinimaxAlphaBeta
Input: Position u, value α, value β
Output: Value at root

if (leaf(u)) return Eval(p)                     ;; No successor, return evaluation
if (max-node(u))                                ;; MAX node
   res ← α                                      ;; Initialize result value
   for each v ∈ Succ(u)                         ;; Traverse successor list
        val ← MinimaxAlphaBeta(v, res, β)       ;; Recursion for α
        res ← max{res, val}                     ;; Take maximal value
        if (res ≥ β)                            ;; Result exceeds threshold
           return res                           ;; Propagate value
else                                            ;; MIN node
   res ← β                                      ;; Initialize result value
   for each v ∈ Succ(u)                         ;; Traverse successor list
        val ← MinimaxAlphaBeta(v, α, res)       ;; Recursion for β
        res ← min{res, val}                     ;; Take minimal value
        if (res ≤ α)                            ;; Result exceeds threshold
           return res                           ;; Propagate value
return res                                      ;; Propagate value
```



# Performance $\alpha\beta$-Pruning

- To obtain an upper bound on the performance we need to prove that there is a minimal tree that must be searched.
  - This tree is called critical tree.
  - And the nodes are critical nodes.

- The number of leaves on the critical tree is:

$$b^{\lceil d/2 \rceil} + b^{\lfloor d/2 \rfloor} - 1$$

  - Where $b$ is the number of children of a node and $d$ is the height.

# Performance $\alpha\beta$-Pruning

- The number of leaves on the critical tree is:

$$b^{\lceil d/2 \rceil} + b^{\lfloor d/2 \rfloor} - 1$$

  - Where $b$ is the number of children of a node and $d$ is the height.

- Motivation:
  - To prove that the value of the root is at least $v$, $b^{\lceil d/2 \rceil}$ leaves must be inspected.
    - One move must be considered on each agent level
    - And all moves on each opponent level.
  - $b^{\lfloor d/2 \rfloor} - 1$ leaf nodes must be generated to see that the value is at most $v$.
    - Opponent strategy.

# Performance $\alpha\beta$-Pruning

- The number of leaves on the critical tree is:

$$b^{\lceil d/2 \rceil} + b^{\lfloor d/2 \rfloor} - 1$$

  - Where $b$ is the number of children of a node and $d$ is the height.
- Any remarks?
  - $b^{\lceil d/2 \rceil} + b^{\lfloor d/2 \rfloor} - 1 \approx 2\sqrt{b^d}$
  - Which is the number of leaves in the whole unpruned tree.
  - Consequently $\alpha\beta$-pruning could double the search.
- If the tree is searched in the best-first order.
  - The best move is always chosen first for exploration.
  - Then $\alpha\beta$ will search only the critical tree.