Max Min Algorithms, Alpha Beta Pruning, Constraint Search Problems

Dr. Latesh Malik

Associate Professor & Head

Government College of Engineering Nagpur

Contents

Games vs Search Problems

- "Unpredictable" opponent : specifying a move for every possible opponent reply
- Time limits : unlikely to find goal, must approximate

Game Playing Strategy

- Maximize winning possibility assuming that opponent will try to minimize (Minimax Algorithm)
- Ignore the unwanted portion of the search tree (Alpha Beta Pruning)
- Evaluation(Utility) Function
  - A measure of winning possibility of the player

Tic-Tac-Toe

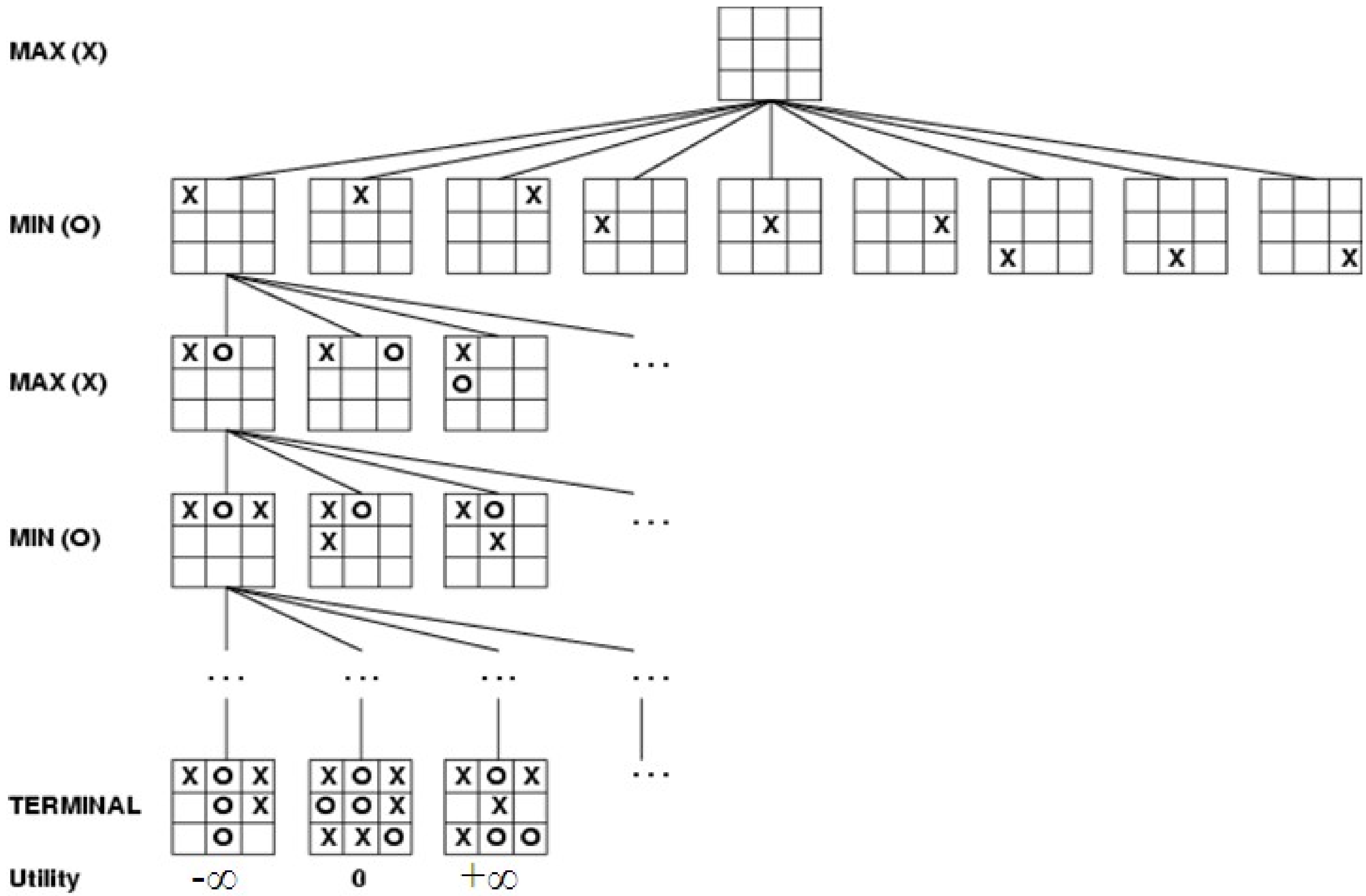
X	**O**	

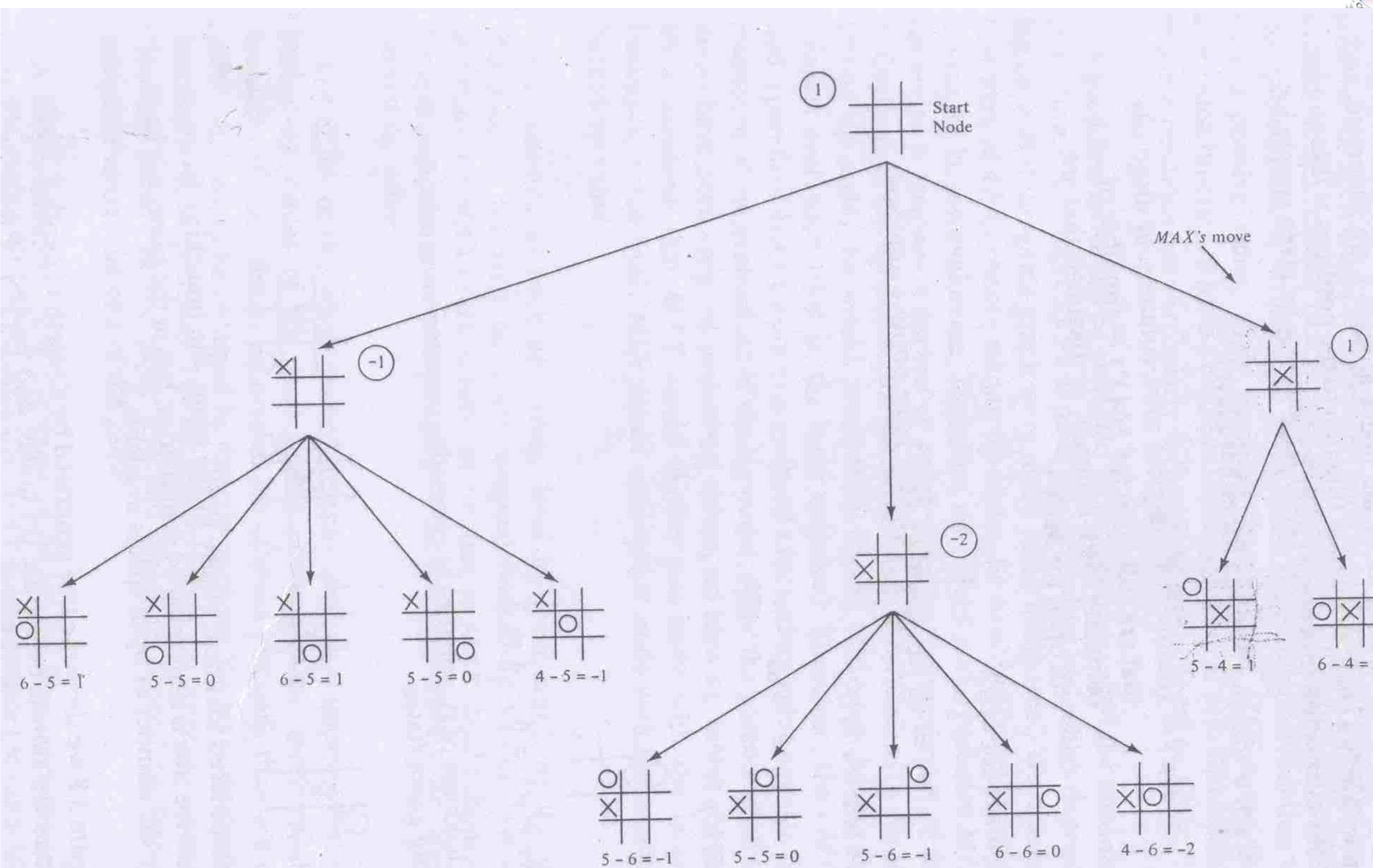
e(p) = 6 - 5 = 1

- **Initial State**: Board position of 3x3 matrix with 0 and X.
- **Operators:** Putting 0’s or X’s in vacant positions alternatively
- **Terminal test:** Which determines game is over
- **Utility function**:

  e(p) = (No. of complete rows, columns or diagonals are still open for player) – (No. of complete rows, columns or diagonals are still open for opponent)

Game tree for Tic-Tac-Toe




1
Start Node
MAX's move
-1
-2
1
6 - 5 = 1
5 - 5 = 0
6 - 5 = 1
5 - 5 = 0
4 - 5 = -1
5 - 4 = 1
6 - 4 = 2
5 - 6 = -1
5 - 5 = 0
5 - 6 = -1
6 - 6 = 0
4 - 6 = -2

Max Min Algorithm

Mini-max algorithm is a recursive or backtracking algorithm which is used in decision-making and game theory.

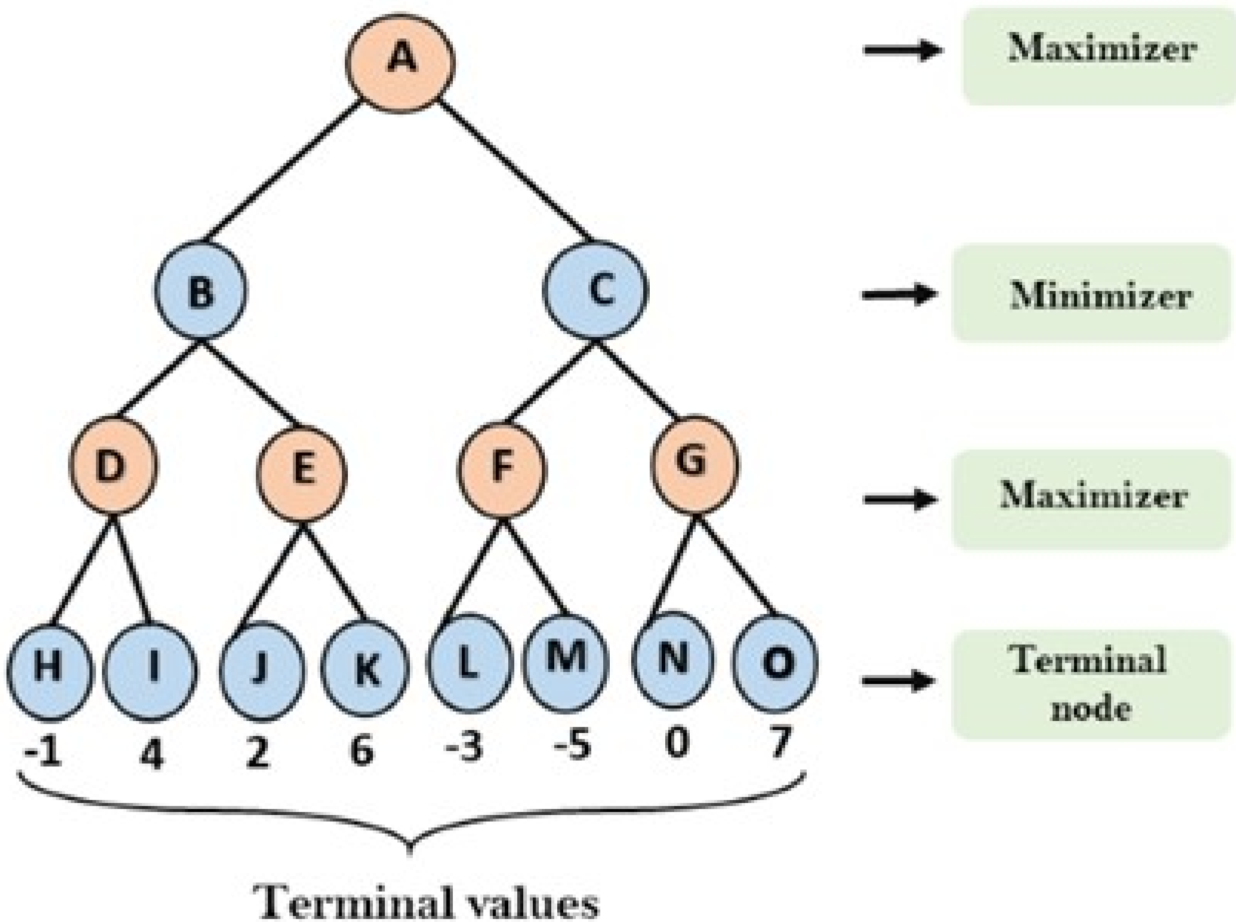
It provides an optimal move for the player assuming that opponent is also playing optimally.

- Mini-Max algorithm uses recursion to search through the game-tree.
- Min-Max algorithm is mostly used for game playing in AI. Such as Chess, Checkers, tic-tac-toe, go, and various tow-players game. This Algorithm computes the minimax decision for the current state.
- In this algorithm two players play the game, one is called MAX and other is called MIN.

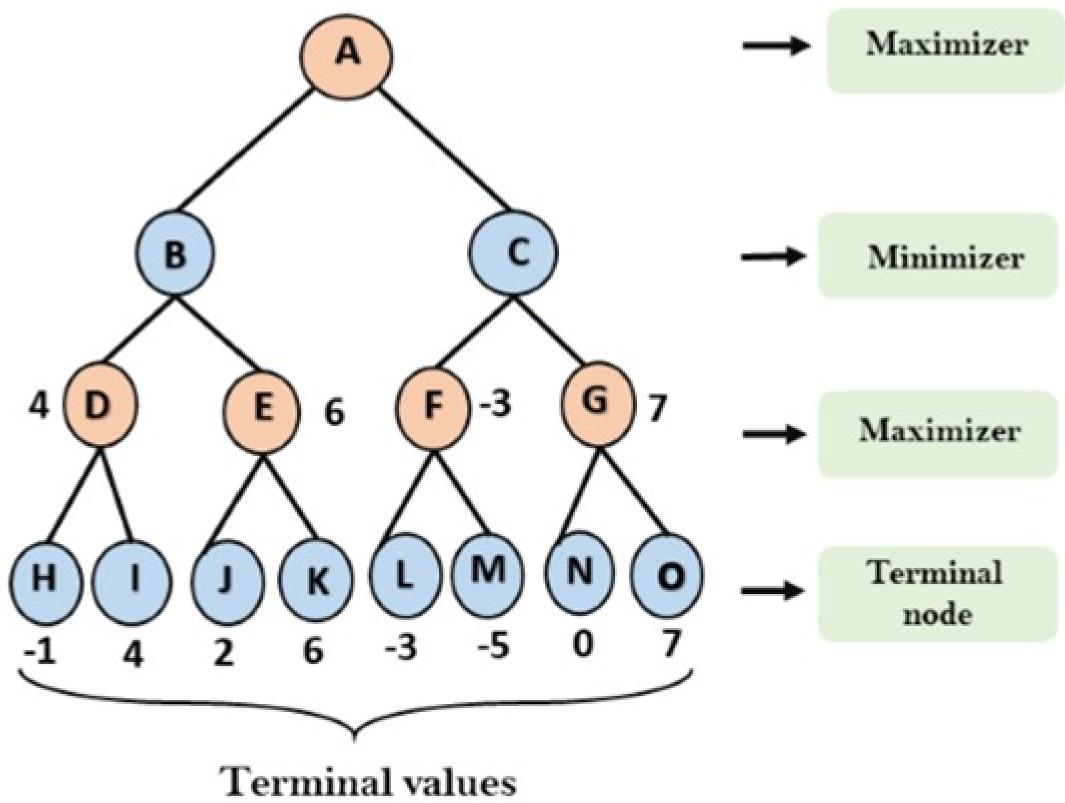
- Both the players fight it as the opponent player gets the minimum benefit while they get the maximum benefit.
- Both Players of the game are opponent of each other, where MAX will select the maximized value and MIN will select the minimized value.
- The minimax algorithm performs a depth-first search algorithm for the exploration of the complete game tree.
- The minimax algorithm proceeds all the way down to the terminal node of the tree, then backtrack the tree as the recursion.

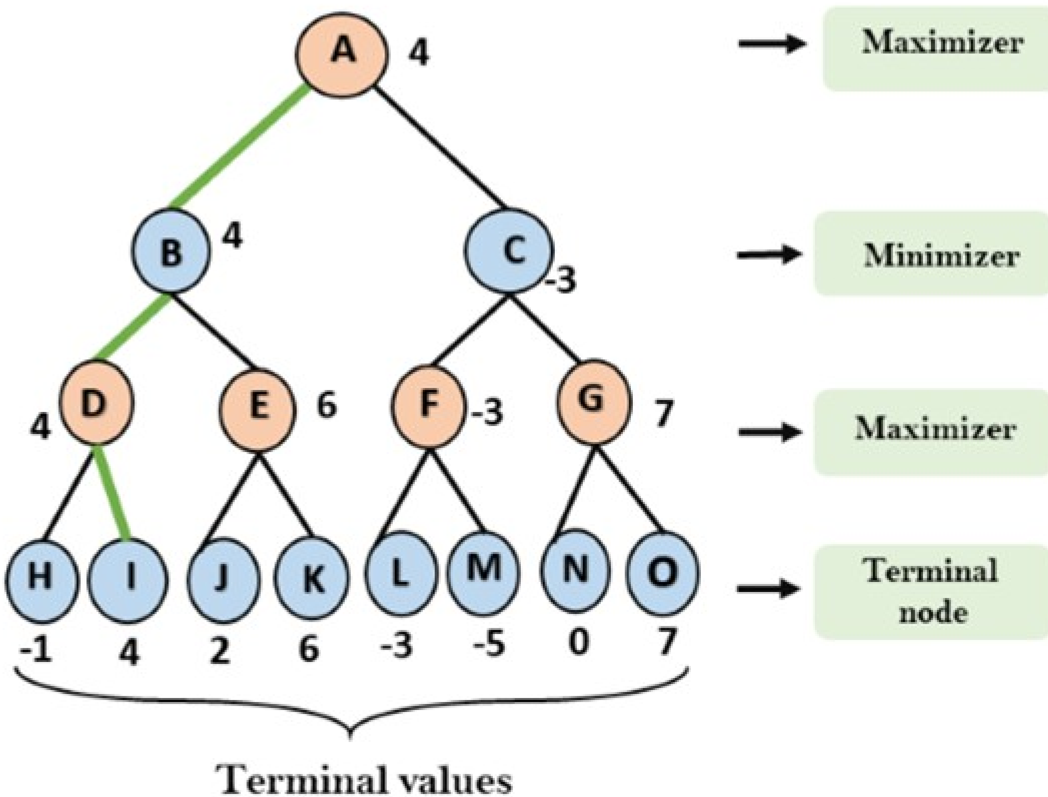
Pseudo-code for MinMax Algorithm:

```
function minimax(node, depth, maximizingPlayer) is
if depth ==0 or node is a terminal node then
return static evaluation of node

if MaximizingPlayer then      // for Maximizer Player
maxEva= -infinity
 for each child of node do
 eva= minimax(child, depth-1, false)
maxEva= max(maxEva,eva)      //gives Maximum of the values
return maxEva

else                         // for Minimizer player
 minEva= +infinity
 for each child of node do
 eva= minimax(child, depth-1, true)
 minEva= min(minEva, eva)     //gives minimum of the values
 return minEva
```


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node

- For node D max(-1,- -∞) => max(-1,4)= 4
- For Node E max(2, -∞) => max(2, 6)= 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) = max(0, 7) = 7


A
B
C
4 D
E 6
F -3
G 7
H
I
J
K
L
M
N
O
-1 4 2 6 -3 -5 0 7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node


A
4
B
4
C
-3
D
4
E
6
F
-3
G
7
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node

The main drawback of the **minimax algorithm is that it gets really slow for complex games such as Chess, go, etc**. **This type of games has a huge branching factor, and the player has lots of choices to decide**. This limitation of the minimax algorithm can be improved from **alpha-beta pruning** which we have discussed in the next topic.

Properties of Minimax

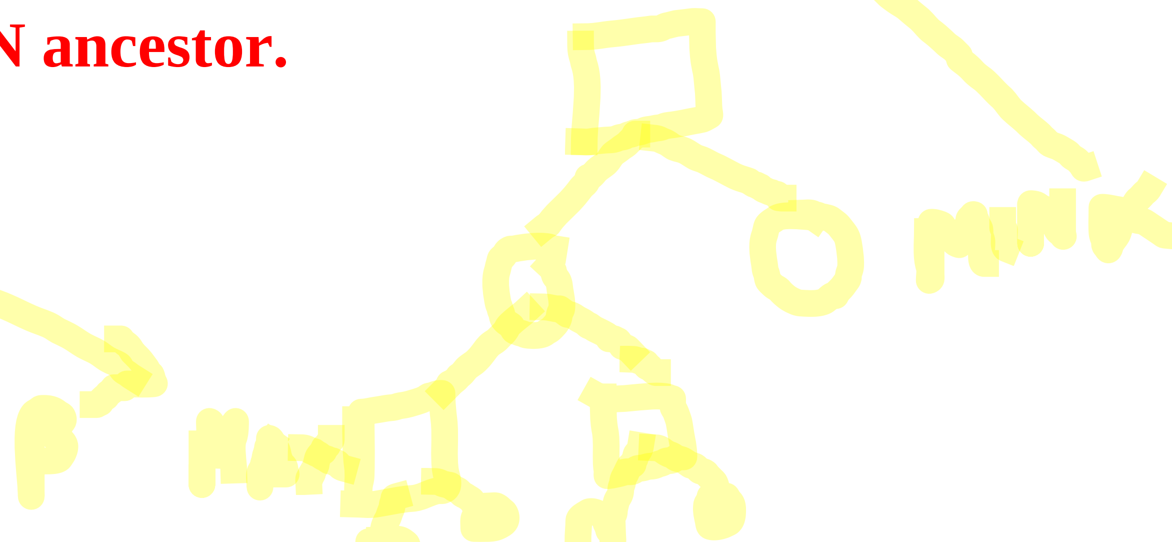
- Complete : Yes (if tree is finite)
- Time complexity : $O(b^d)$
- Space complexity : $O(bd)$ (depth-first exploration)

Alpha Beta Pruning

Alpha-Beta Pruning

Rules:

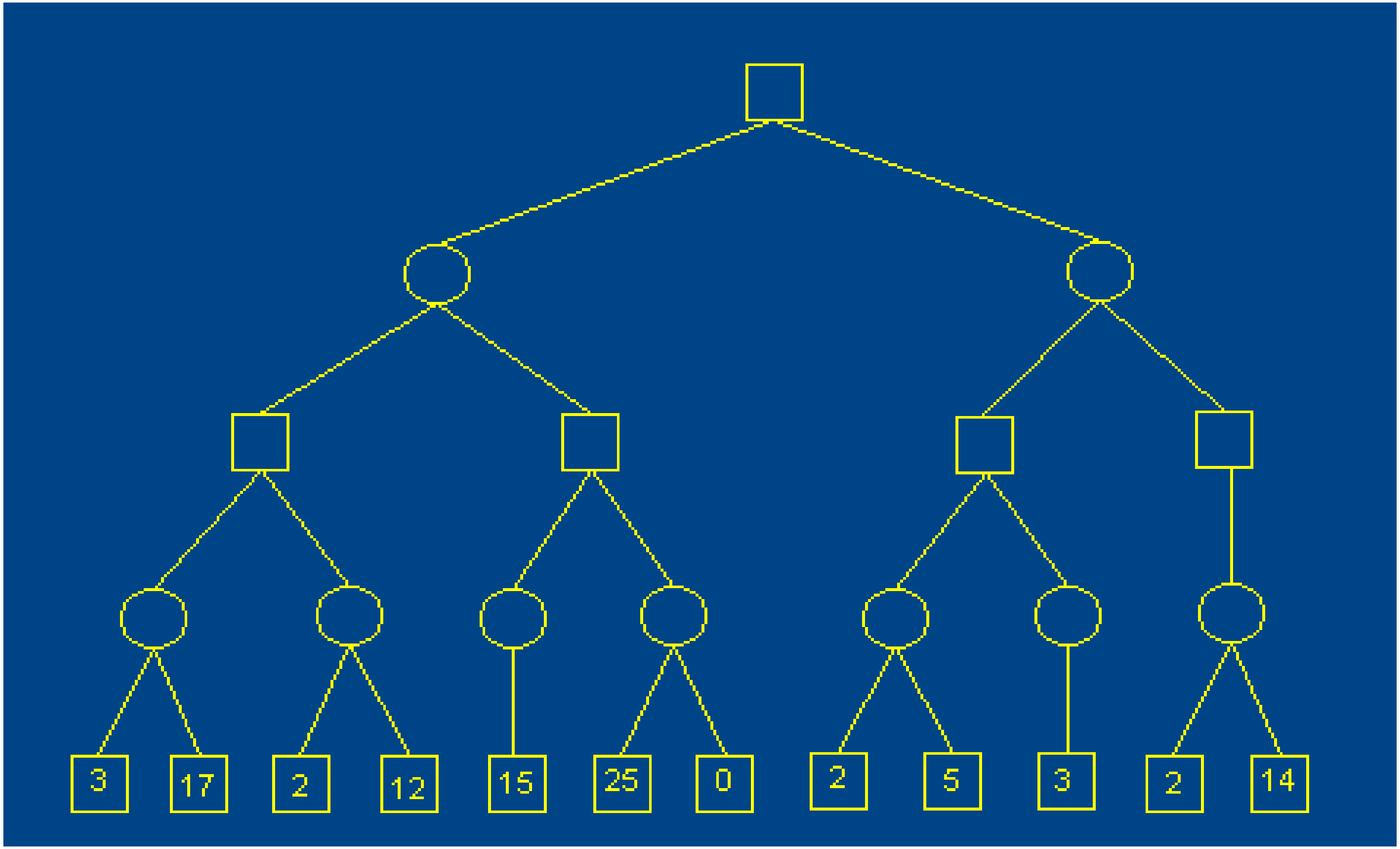
- α is the best (highest) found so far along the path for Max
- β is the best (lowest) found so far along the path for Min
- Search below a MIN node may be alpha-pruned if its β <= α of some MAX ancestor
- Search below a MAX node may be beta-pruned if its α >= β of some MIN ancestor.



Alpha-beta pruning

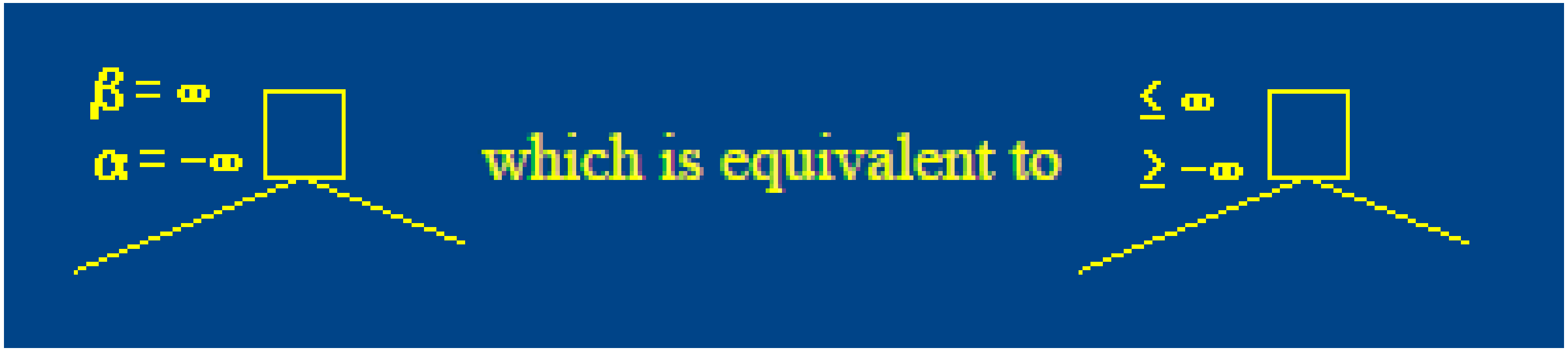
- *Alpha-beta pruning* is a way of finding the optimal minimax solution while avoiding searching subtrees of moves which won't be selected.
- Alpha-beta pruning gets its name from two parameters.
  - They describe bounds on the values that appear anywhere along the path under consideration:
    - α = the value of the best (i.e., highest value) choice found so far along the path for MAX
    - β = the value of the best (i.e., lowest value) choice found so far along the path for MIN

Example



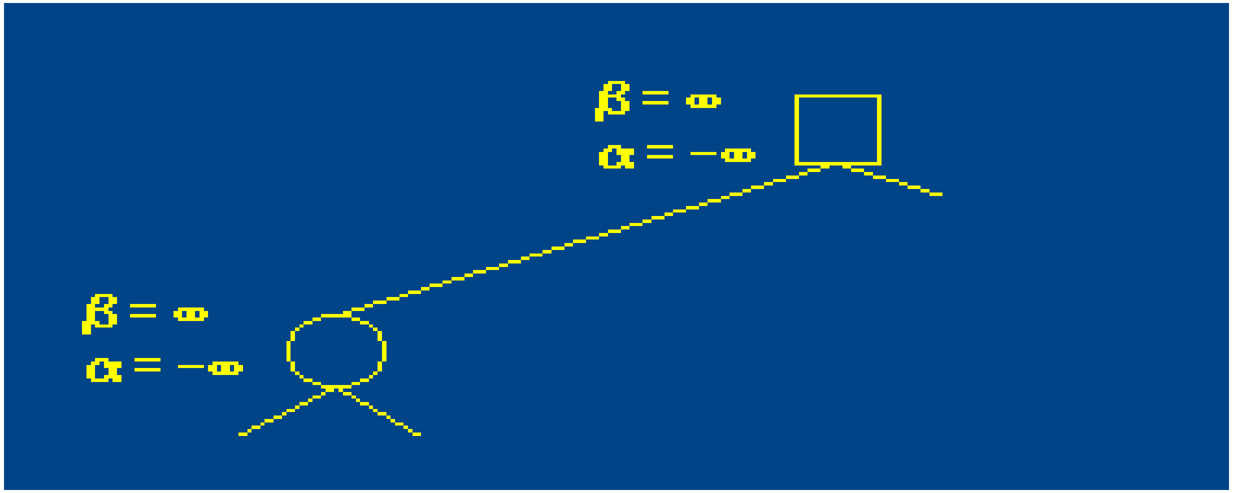
Initial Assumption for Alpha and Beta

- At the start of the problem, you see only the current state (i.e. the current position of pieces on the game board). As for upper and lower bounds, all you know is that it's a number less than infinity and greater than negative infinity. Thus, here's what the initial situation looks like:



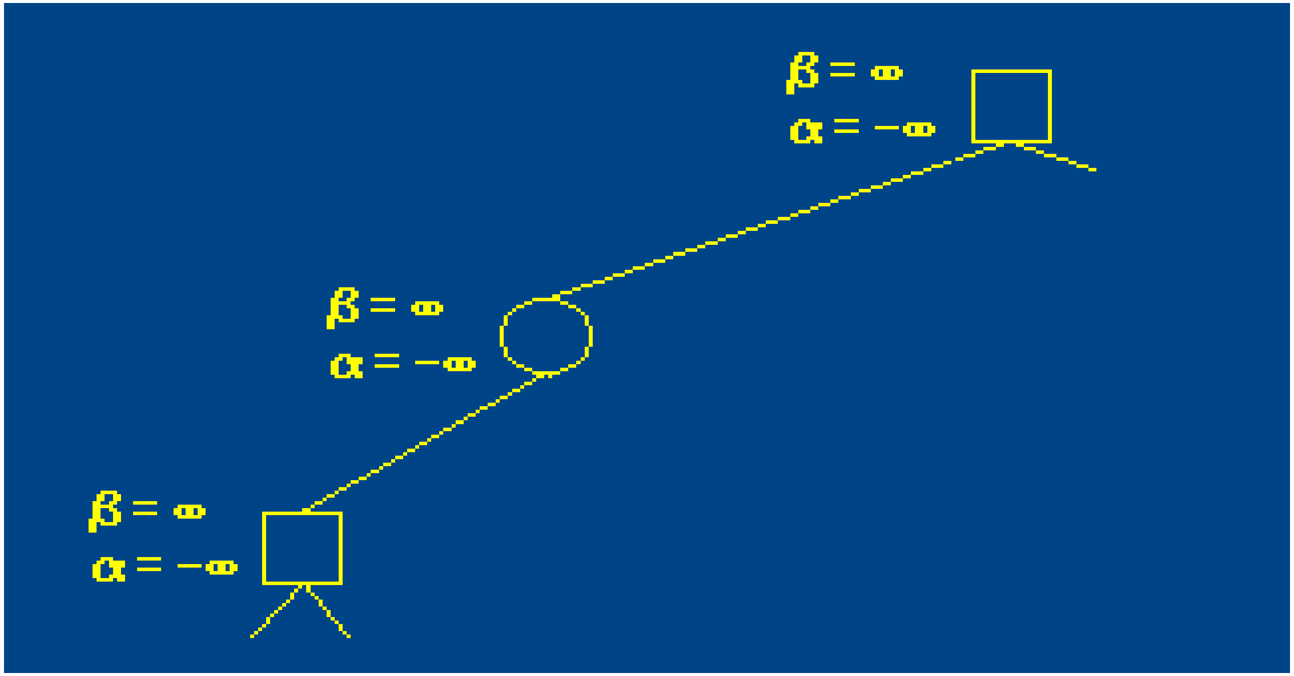
Example

- Since the bounds still contain a valid range, we start the problem by generating the first child state, and passing along the current set of bounds. At this point our search looks like this:



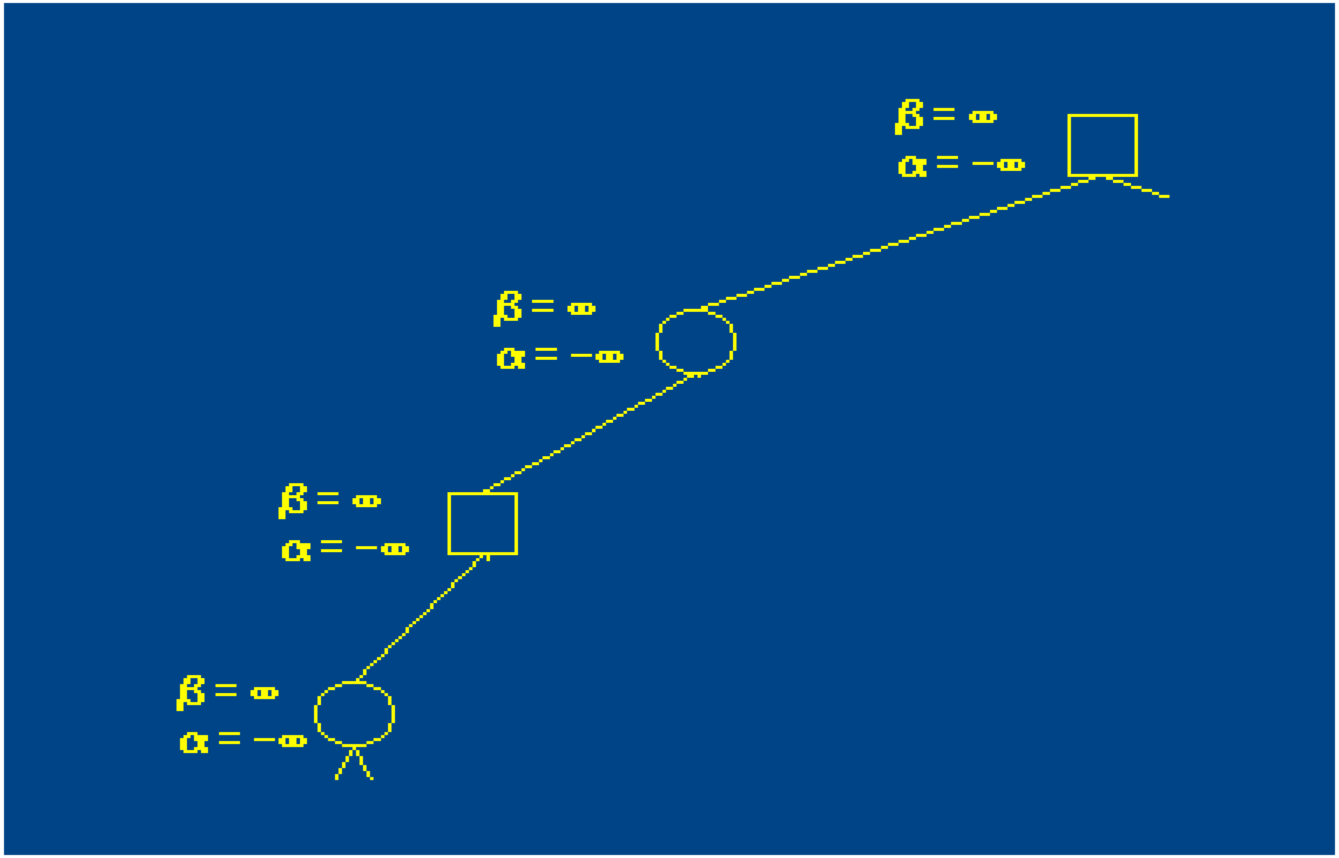
Example

- We're still not down to depth 4, so once again we generate the first child node and pass along our current alpha and beta values:



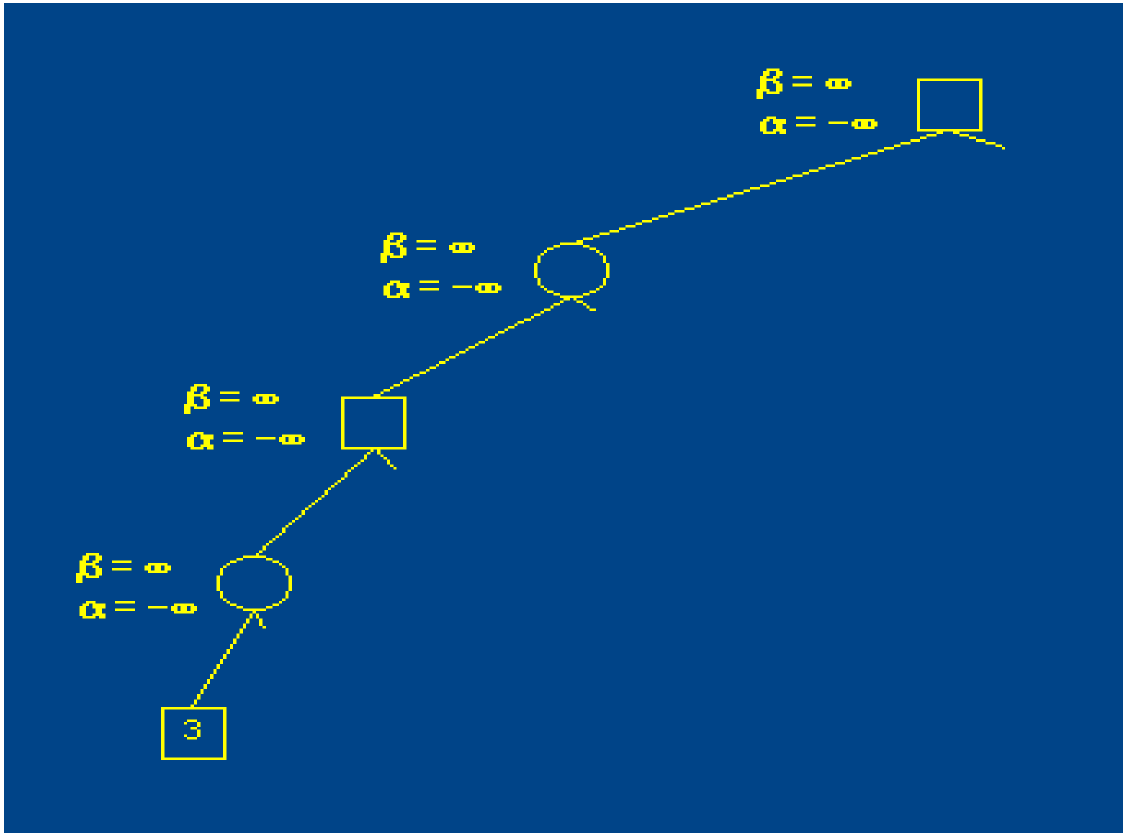
Example

- And one more time



Example

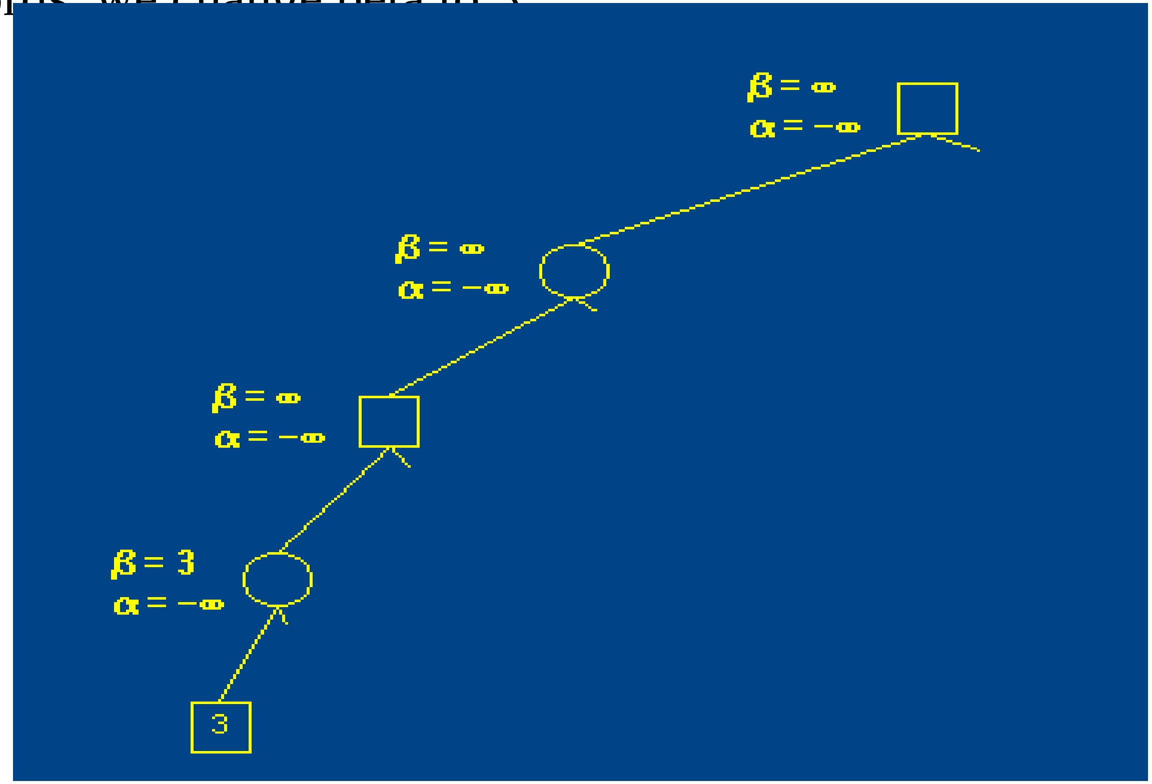
- When we get to the first node at depth 4, we run our evaluation function on the state, and get the value 3. Thus we have this:



Example

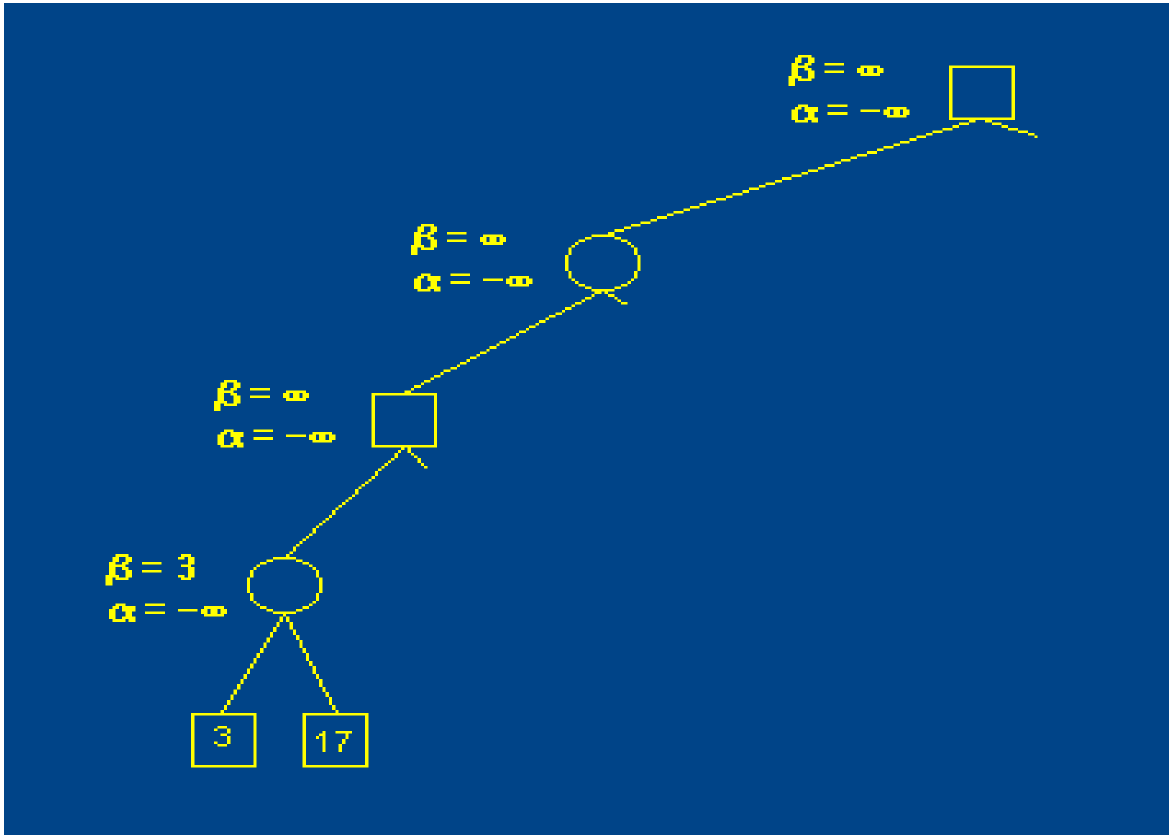
- We pass this node back to the min node above. Since this is a min node, we now know that the minimax value of this node must be less than or equal to

3. In other words, we change beta to 3.



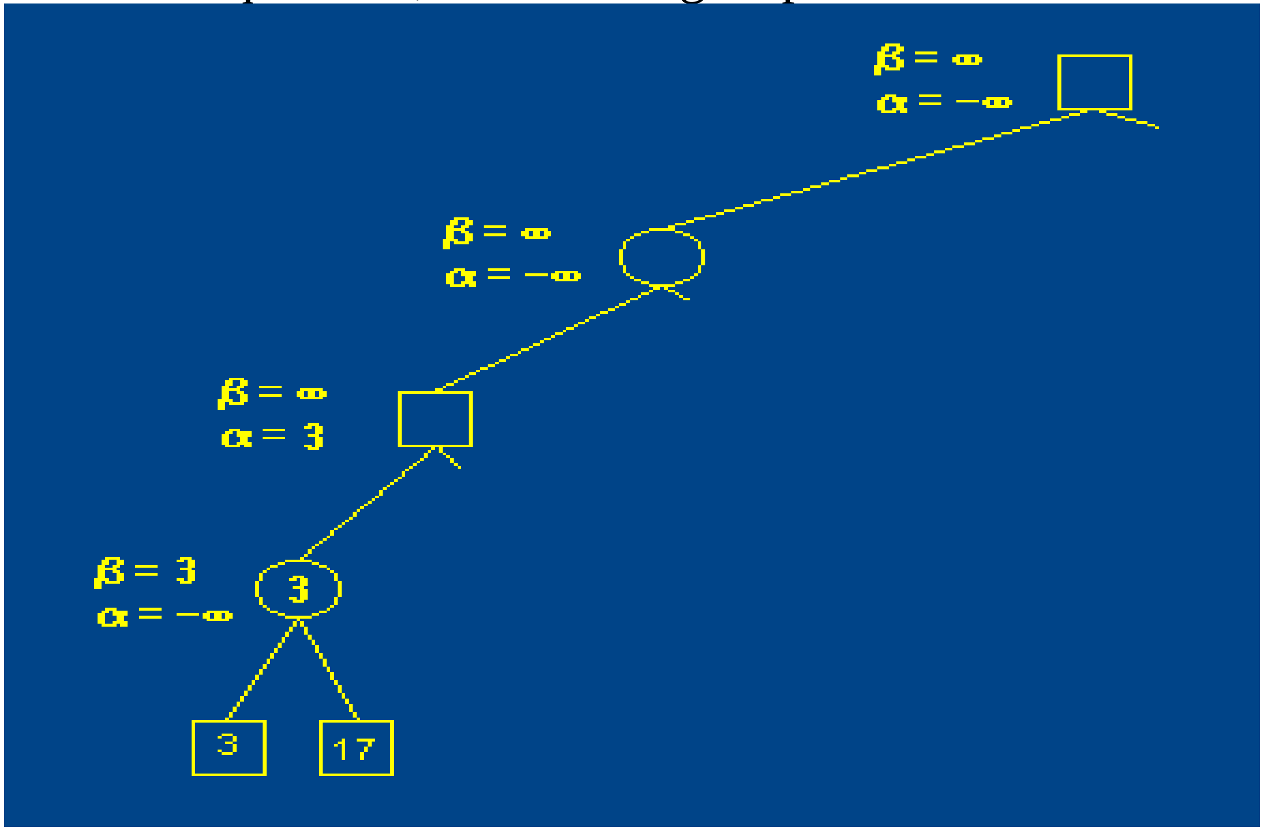
Example

- Next we generate the next child at depth 4, run our evaluation function, and return a value of 17 to the min node above:



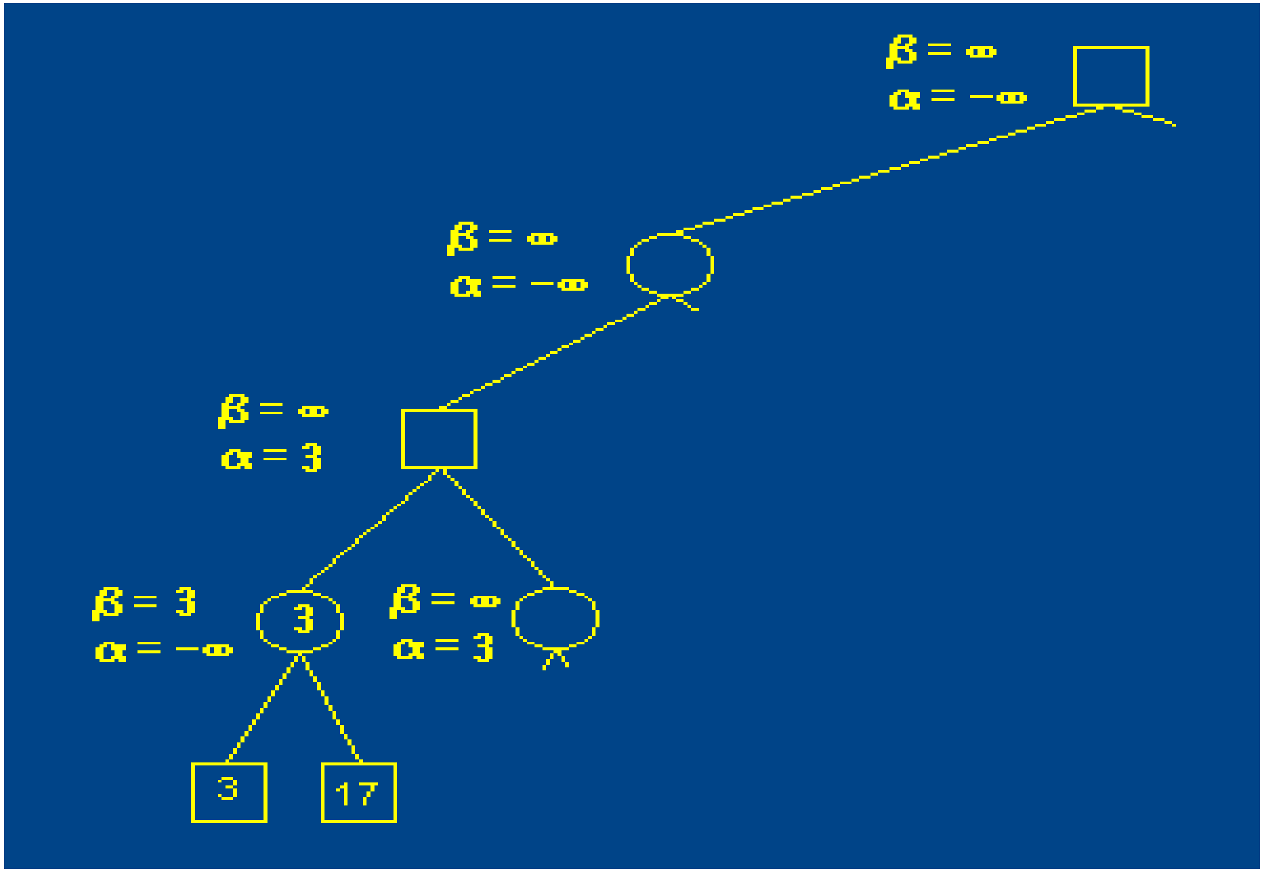
Example

- Since this is a min node and 17 is greater than 3, this child is ignored. Now we've seen all of the children of this min node, so we return the beta value to the max node above. Since it is a max node, we now know that it's value will be greater than or equal to 3, so we change alpha to 3:



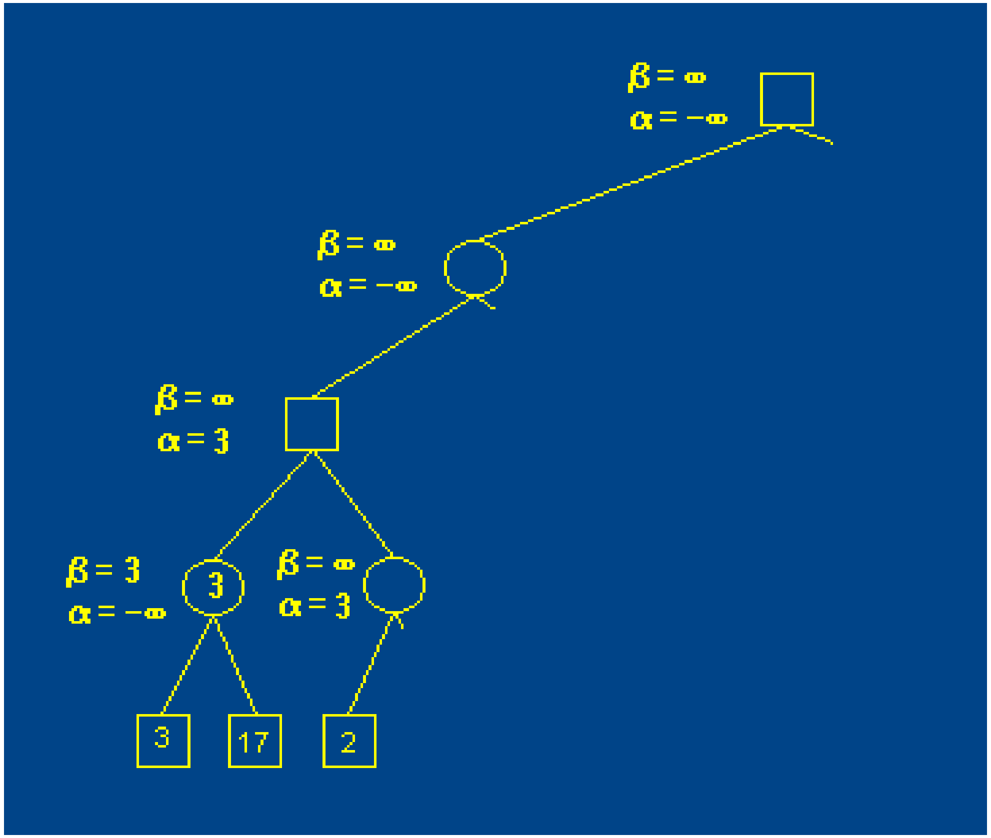
Exampl
e

- We generate the next child and pass the bounds along



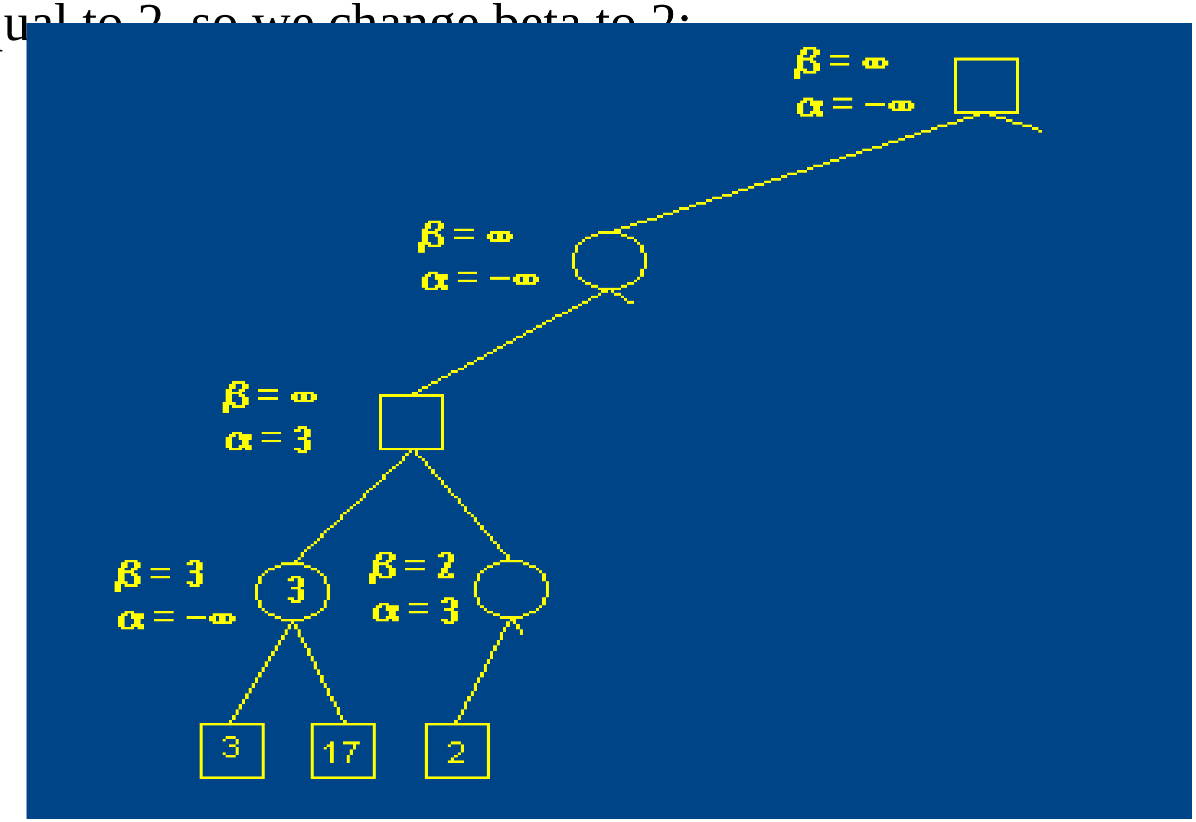
Example

- Since this node is not at the target depth, we generate its first child, run the evaluation function on that node, and return it's value



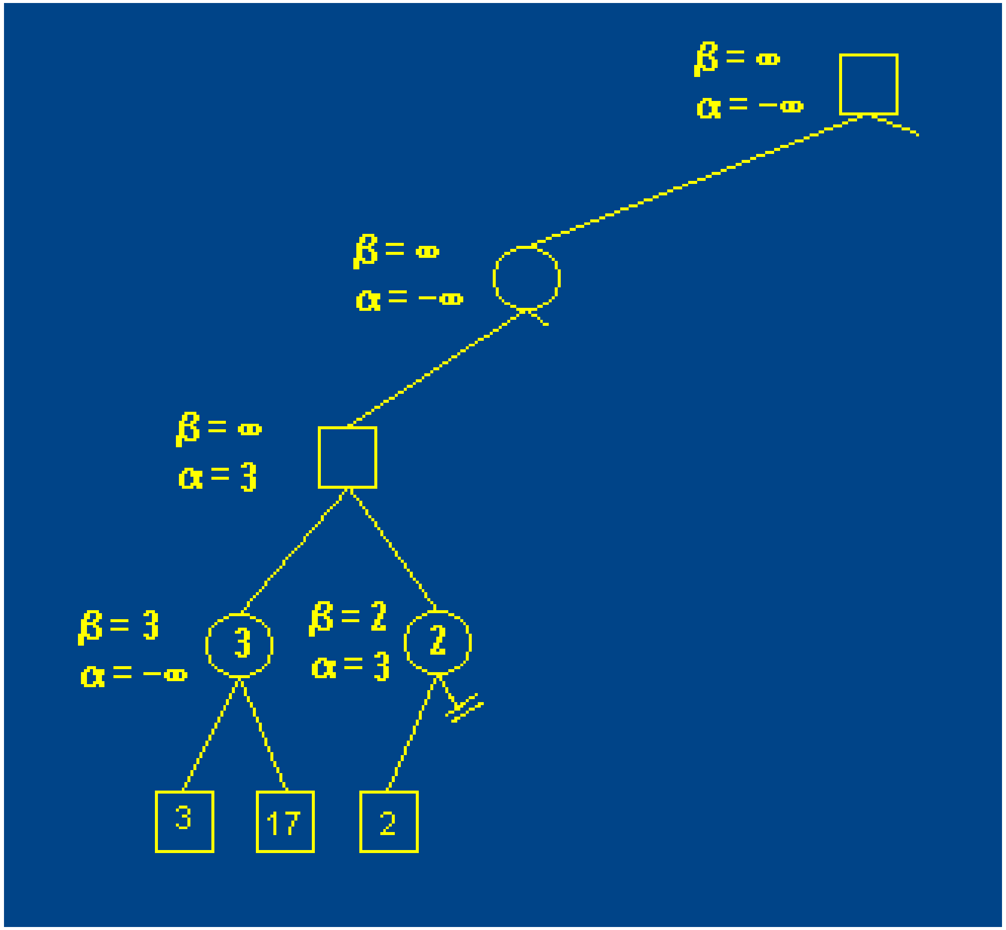
Example

- Since this is a min node, we now know that the value of this node will be
less than or equal to 2, so we change beta to 2:



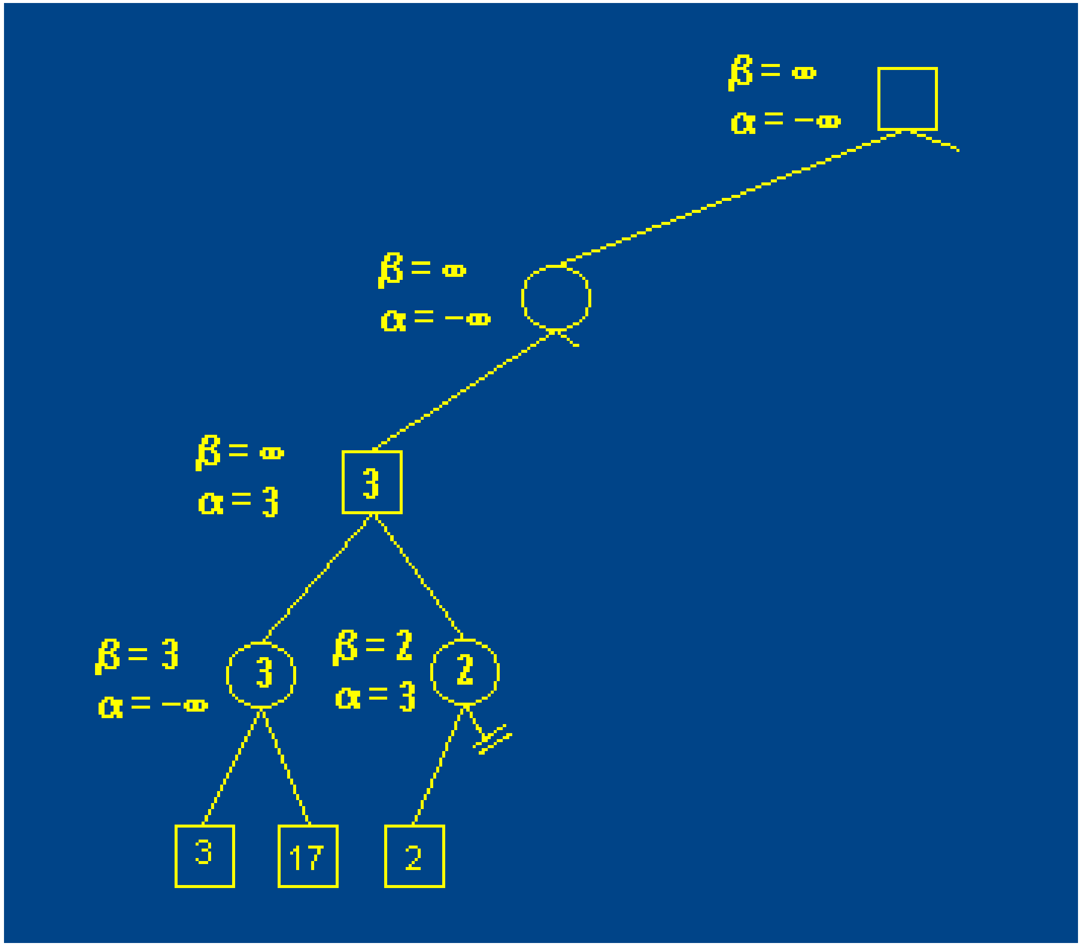
Example

- Admittedly, we don't know the actual value of the node. There could be a 1 or 0 or -100 somewhere in the other children of this node. But even if there was such a value, searching for it won't help us find the optimal solution in the search tree. The 2 alone is enough to make this subtree fruitless, so we can prune any other children and return it.
- **That's all there is to beta pruning!**



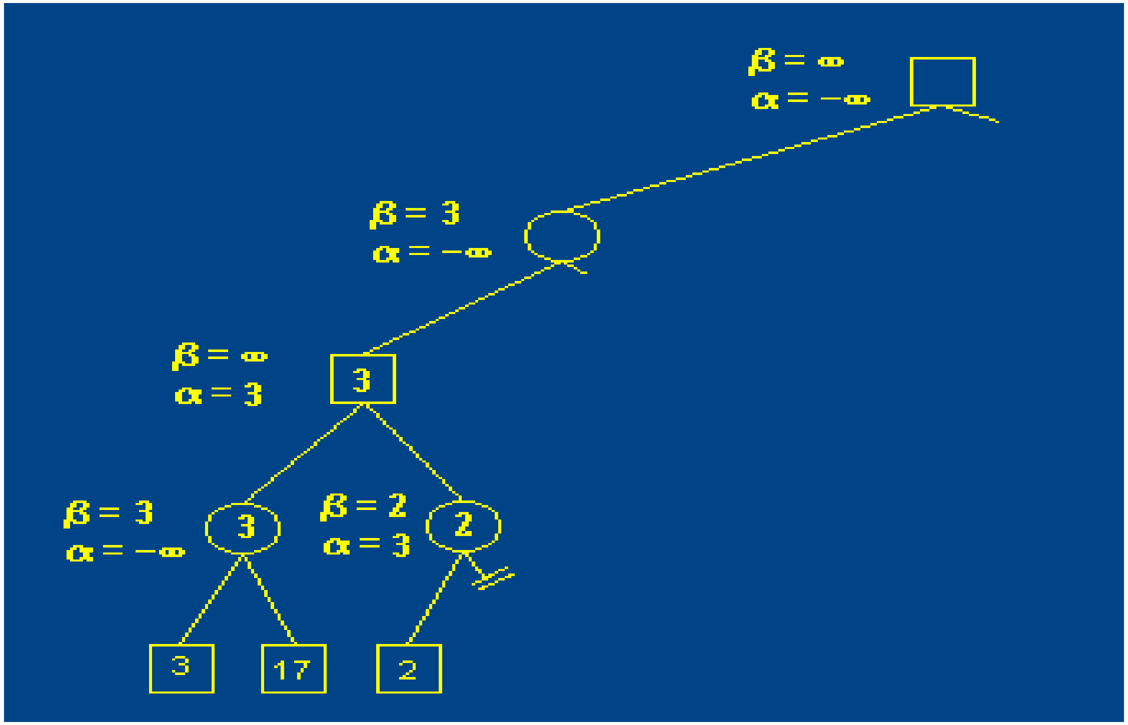
Example

- Back at the parent max node, our alpha value is already 3, which is more restrictive than 2, so we don't change it. At this point we've seen all the children of this max node, so we can set its value to the final alpha value:



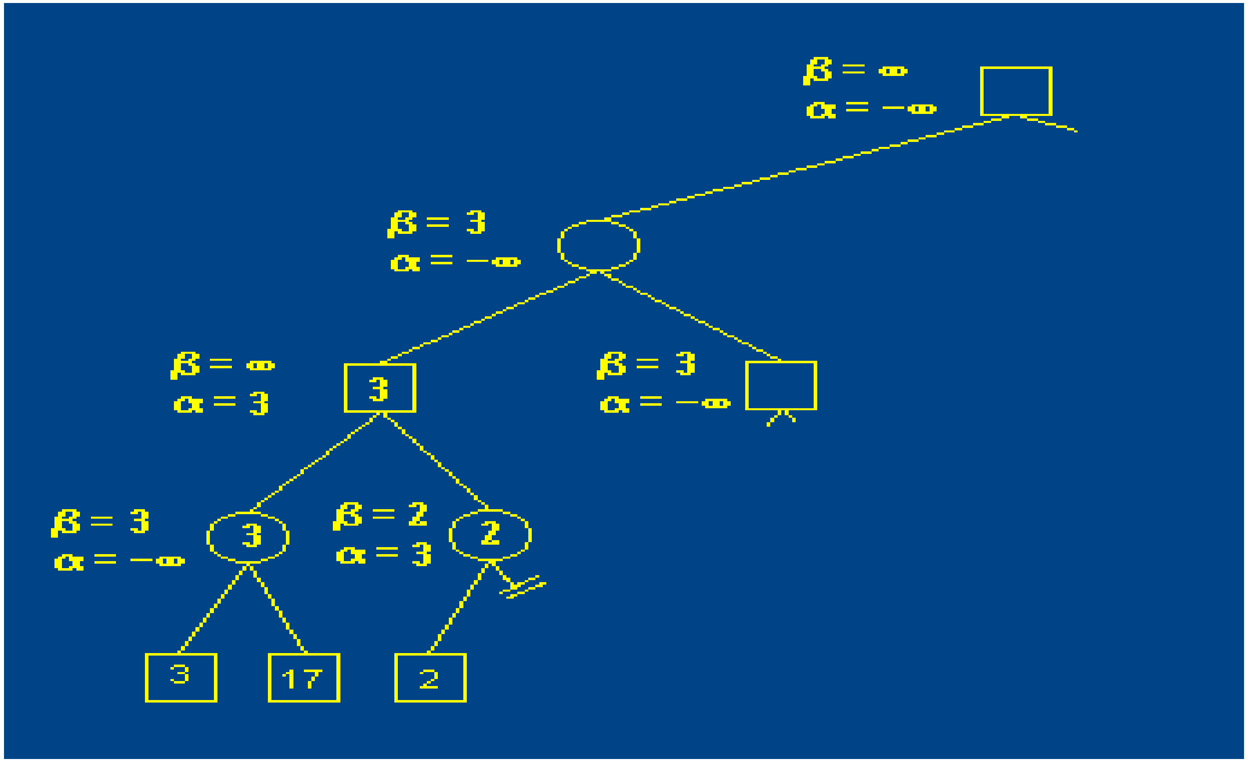
Example

- Now we move on to the parent min node. With the 3 for the first child value, we know that the value of the min node must be less than or equal to 3, thus we set beta to 3:



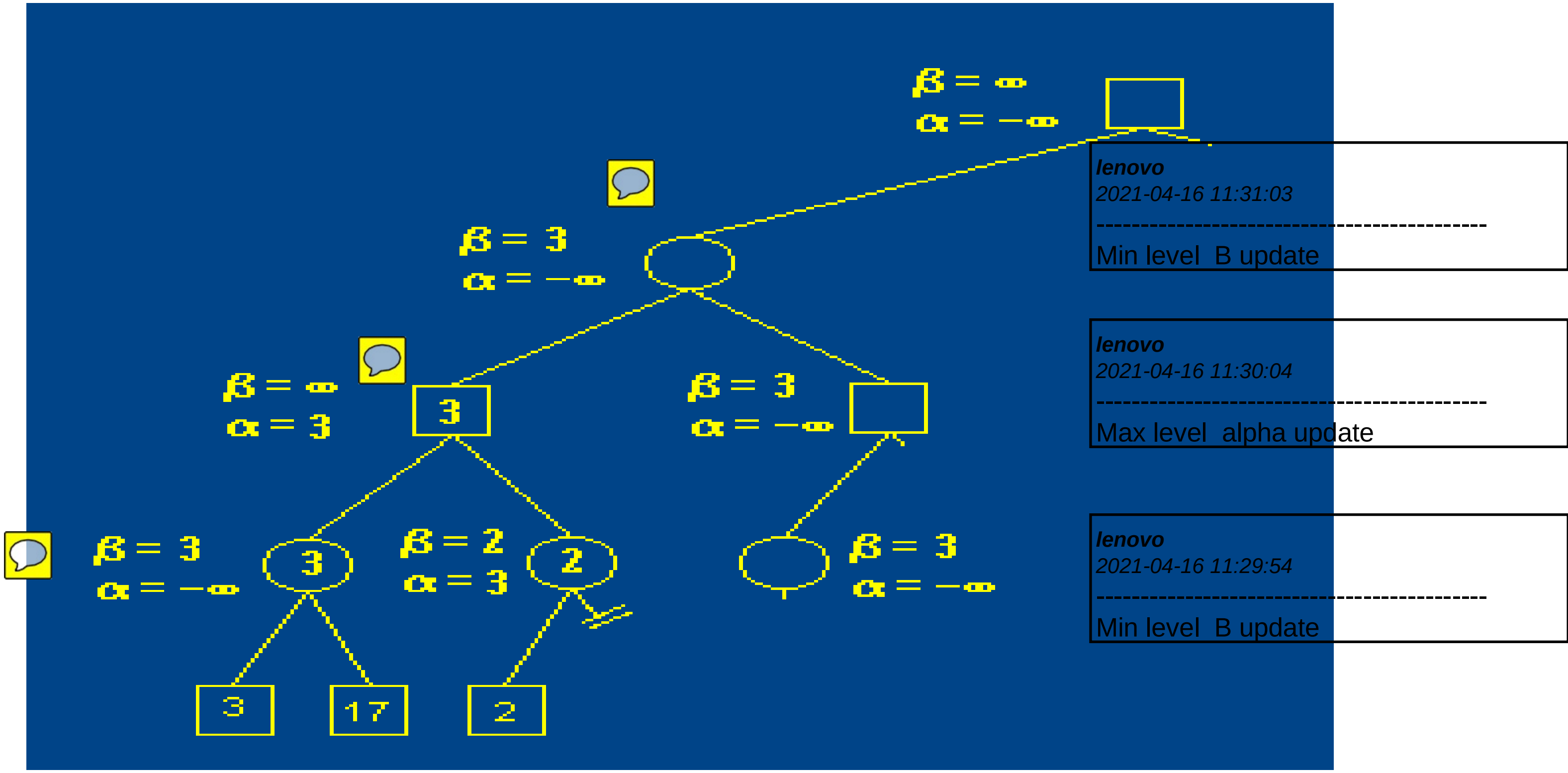
Example

- Since we still have a valid range, we go on to explore the next child. We generate the max node...



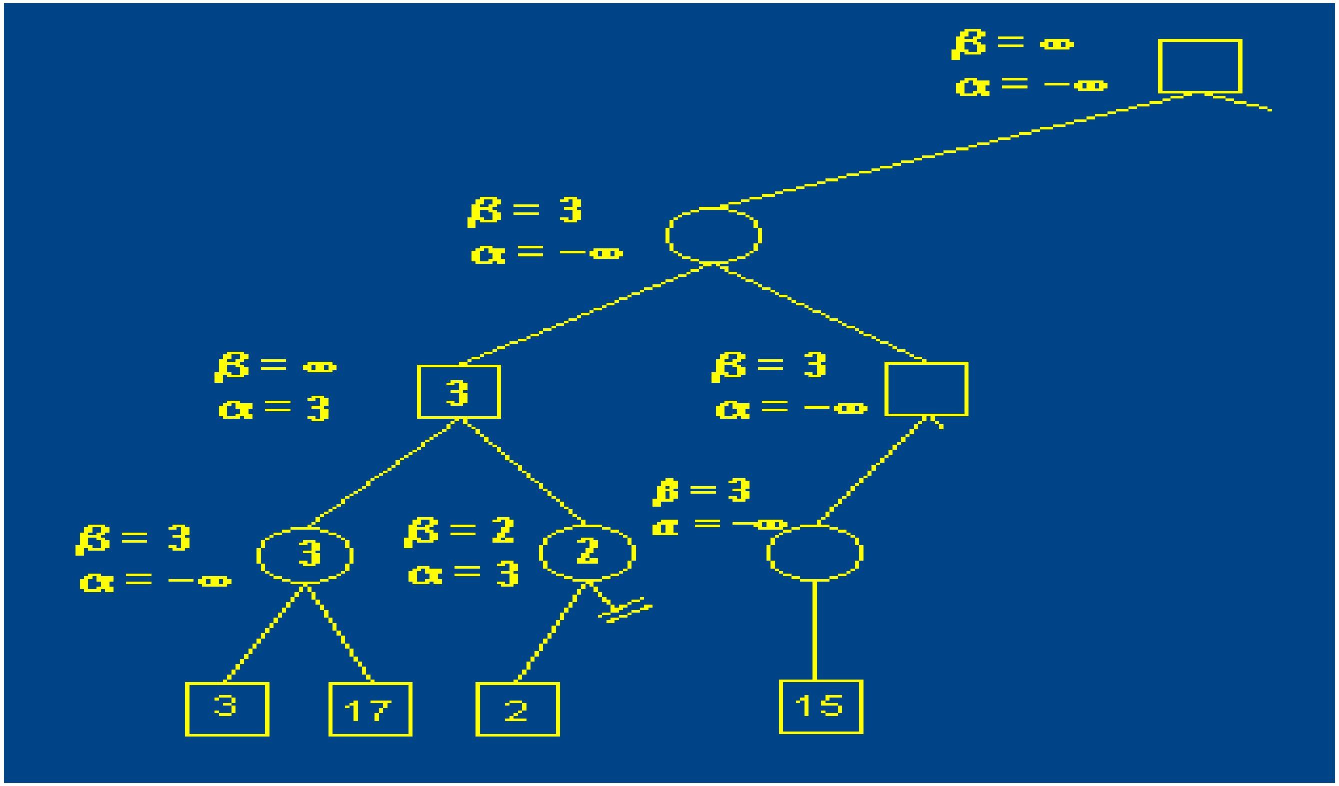
Example

- ... it's first child min node ...



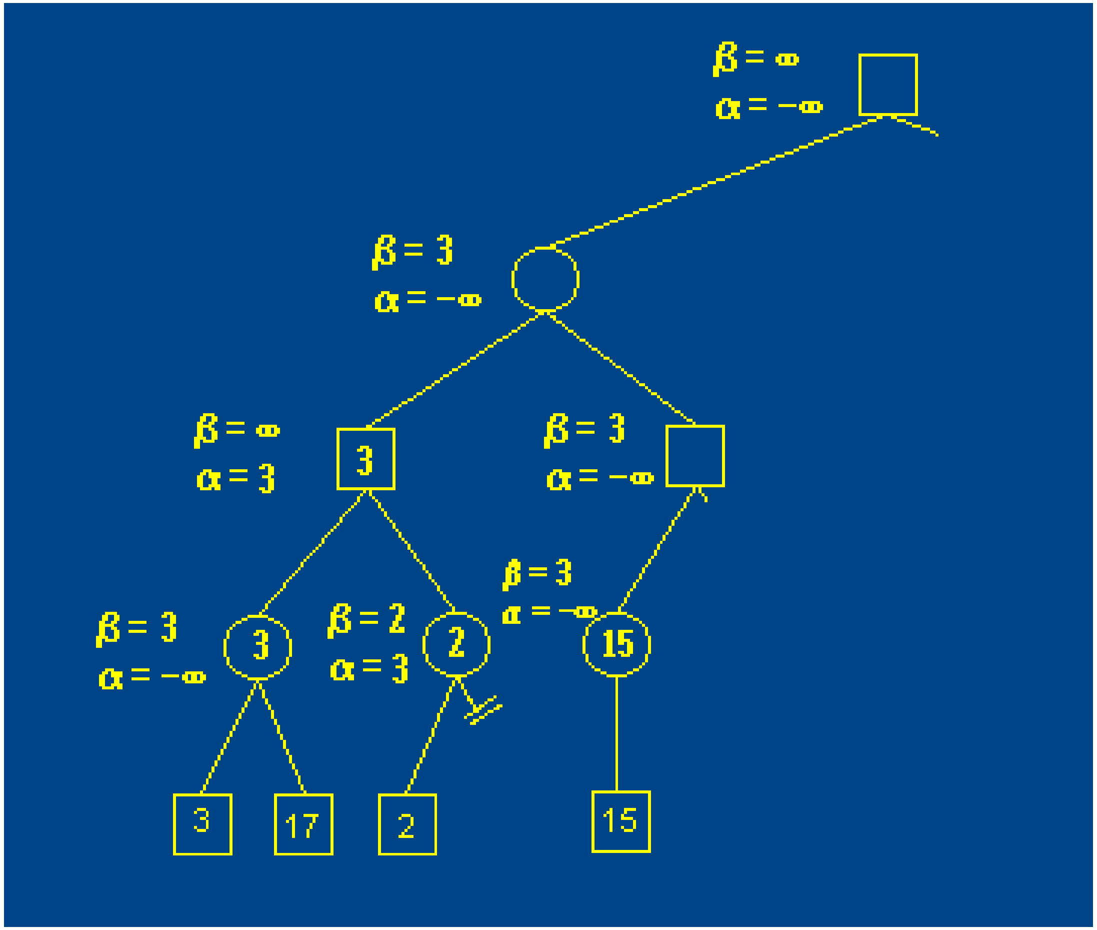
Example

- ... and finally the max node at the target depth. All along this path, we merely pass the alpha and beta bounds along.



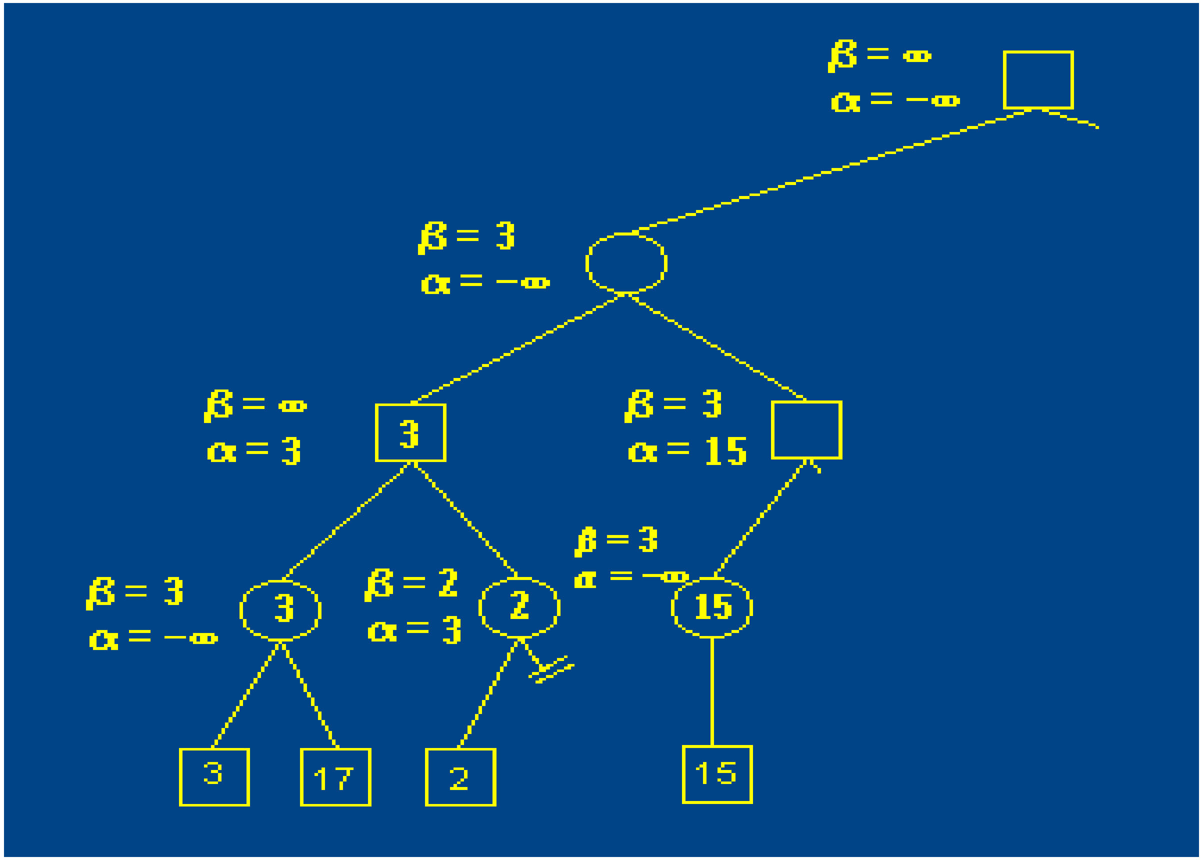
Example

- At this point, we've seen all of the children of the min node, and we haven't changed the beta bound. Since we haven't exceeded the bound, we should return the actual min value for the node. Notice that this is different than the case where we pruned, in which case you returned the beta value. The reason for this will become apparent shortly.



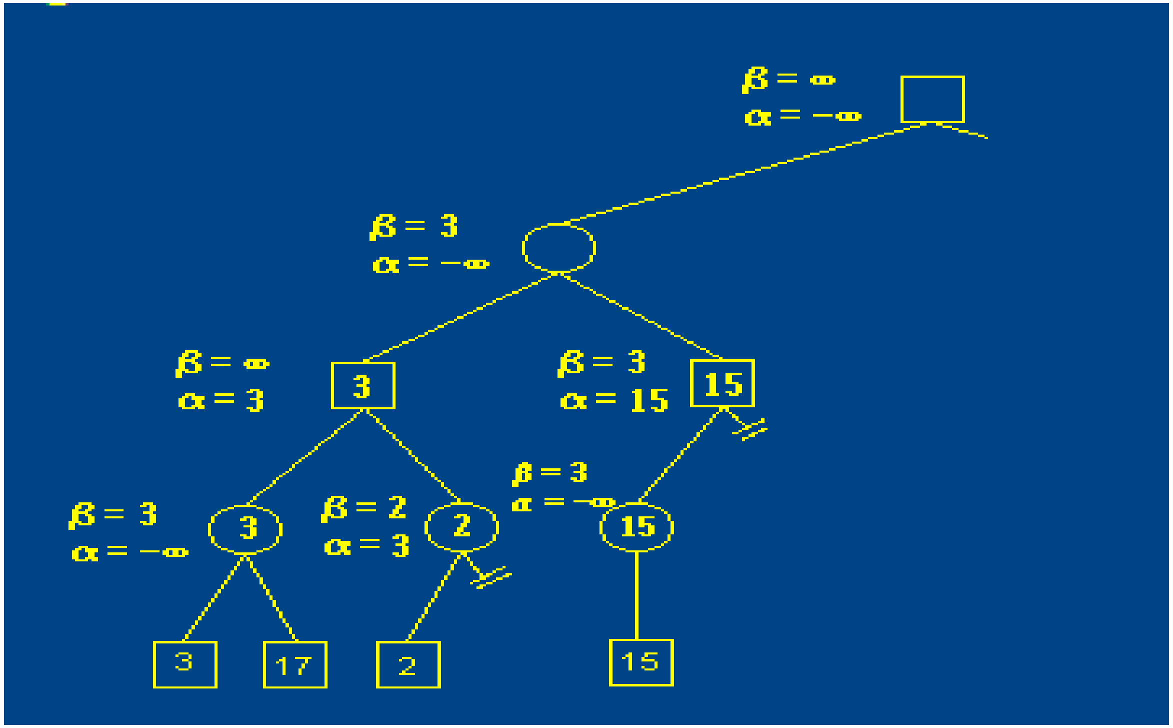
Example

- Now we return the value to the parent max node. Based on this value, we know that this max node will have a value of 15 or greater, so we set alpha to 15:



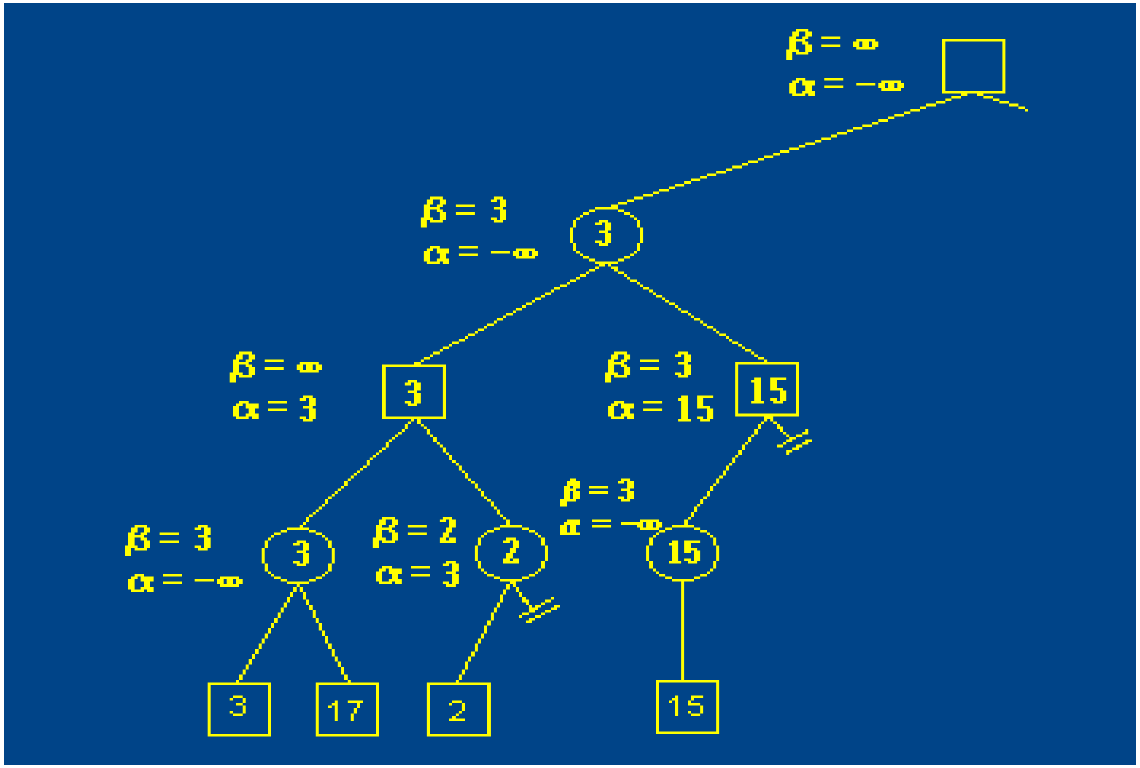
Example

- Once again the alpha and beta bounds have crossed, so we can prune the rest of this node's children and return the value that exceeded the bound (i.e. 15). Notice that if we had returned the beta value of the child min node (3) instead of the actual value (15), we wouldn't have been able to prune here.



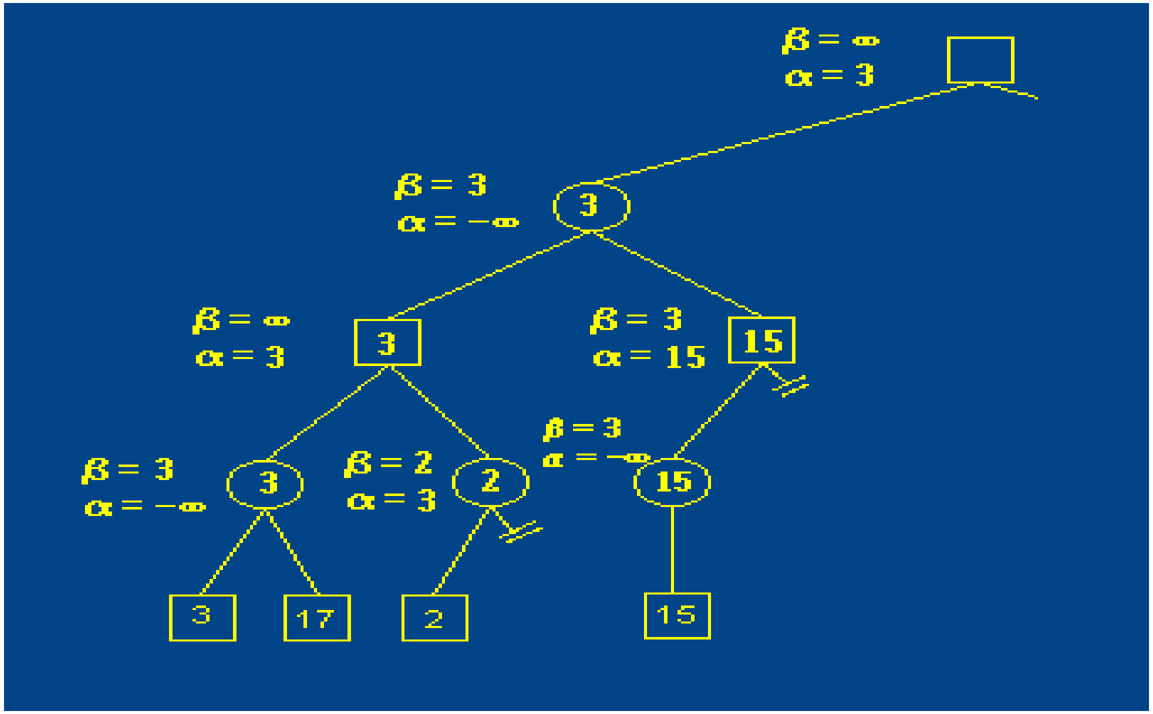
Example

- Now the parent min node has seen all of it's children, so it can select the minimum value of it's children (3) and return.



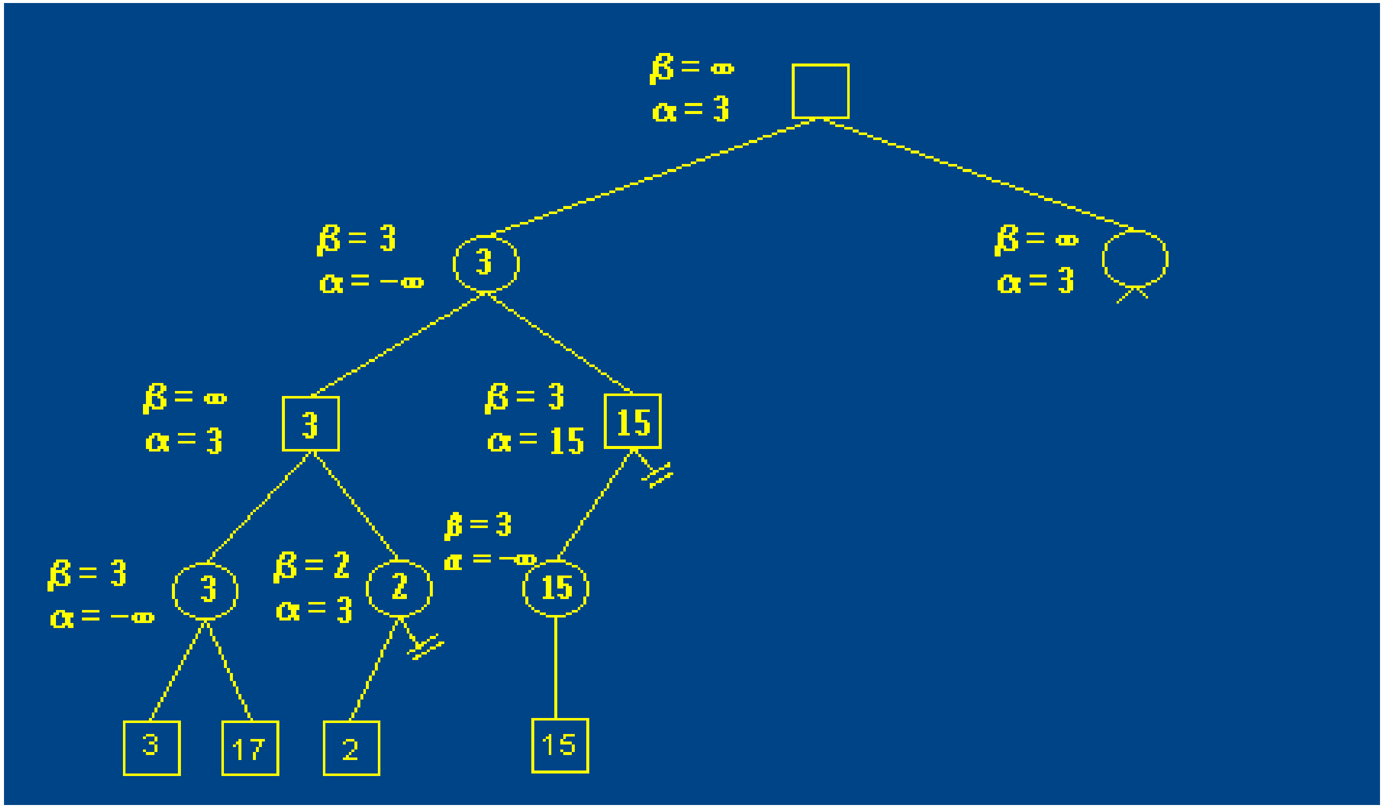
Example

- Finally we've finished with the first child of the root max node. We now know our solution will be at least 3, so we set the alpha value to 3 and go on to the second child.



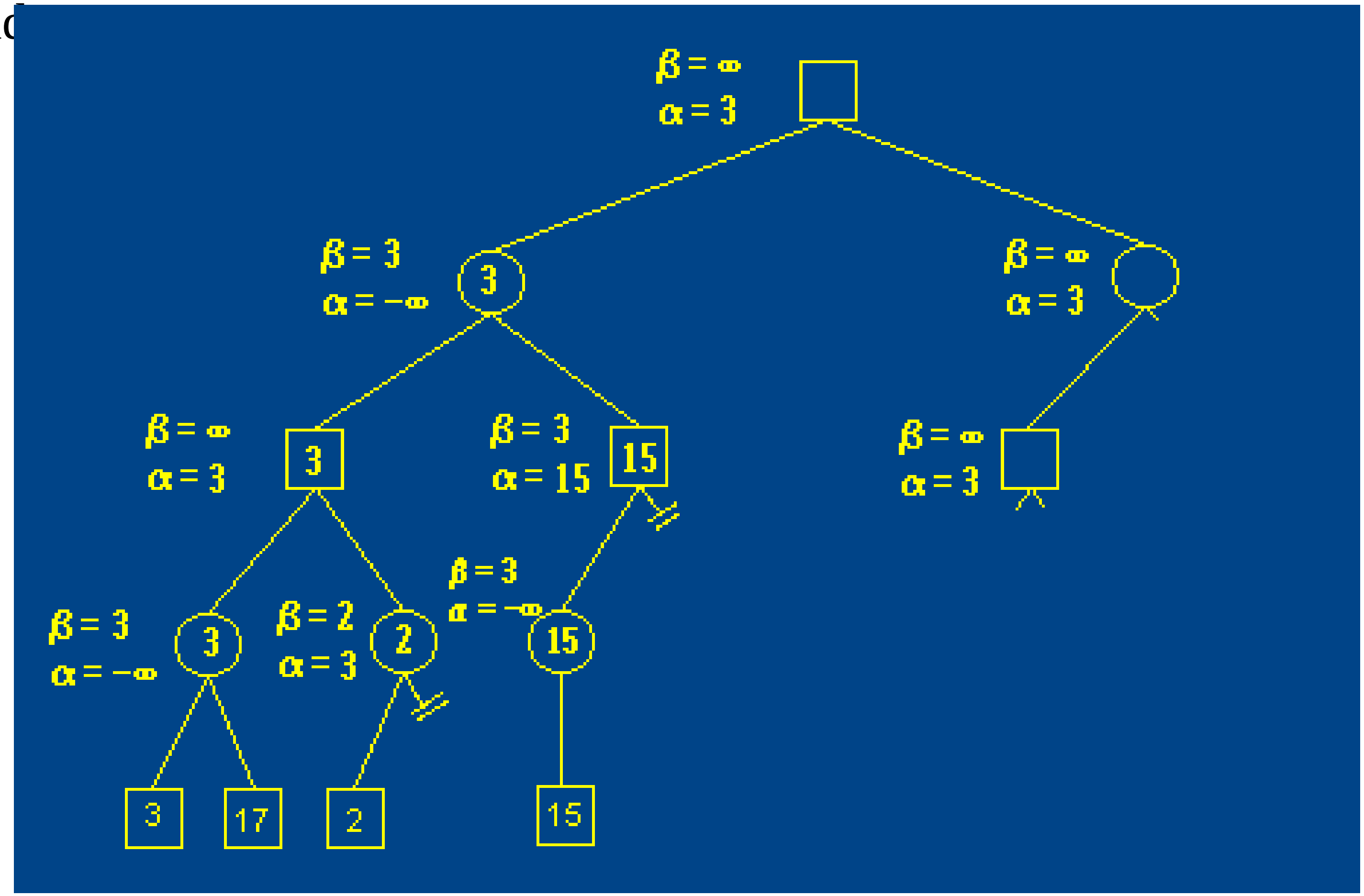
Example

- Passing the alpha and beta values along as we go, we generate the second child of the root node...

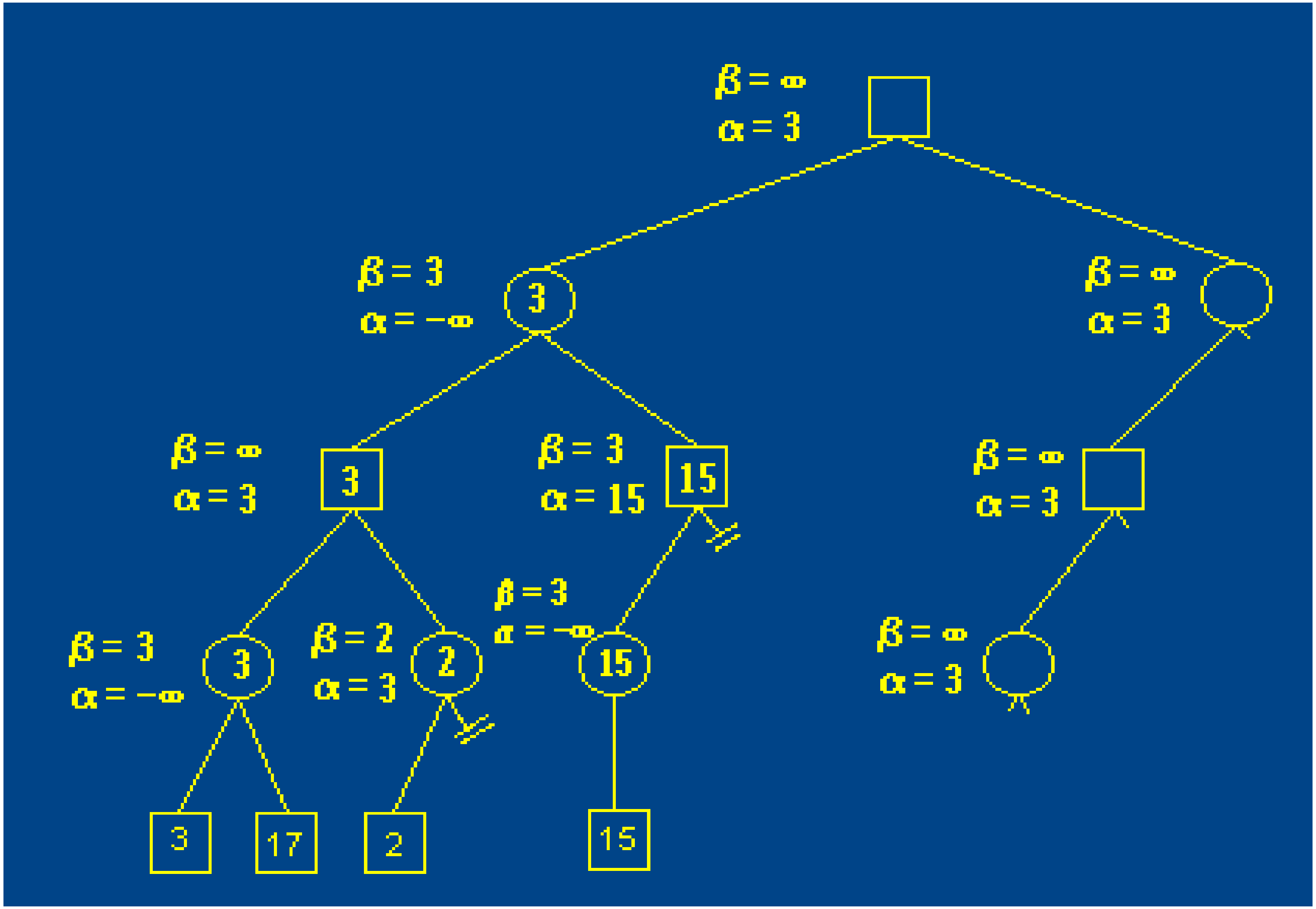


Example

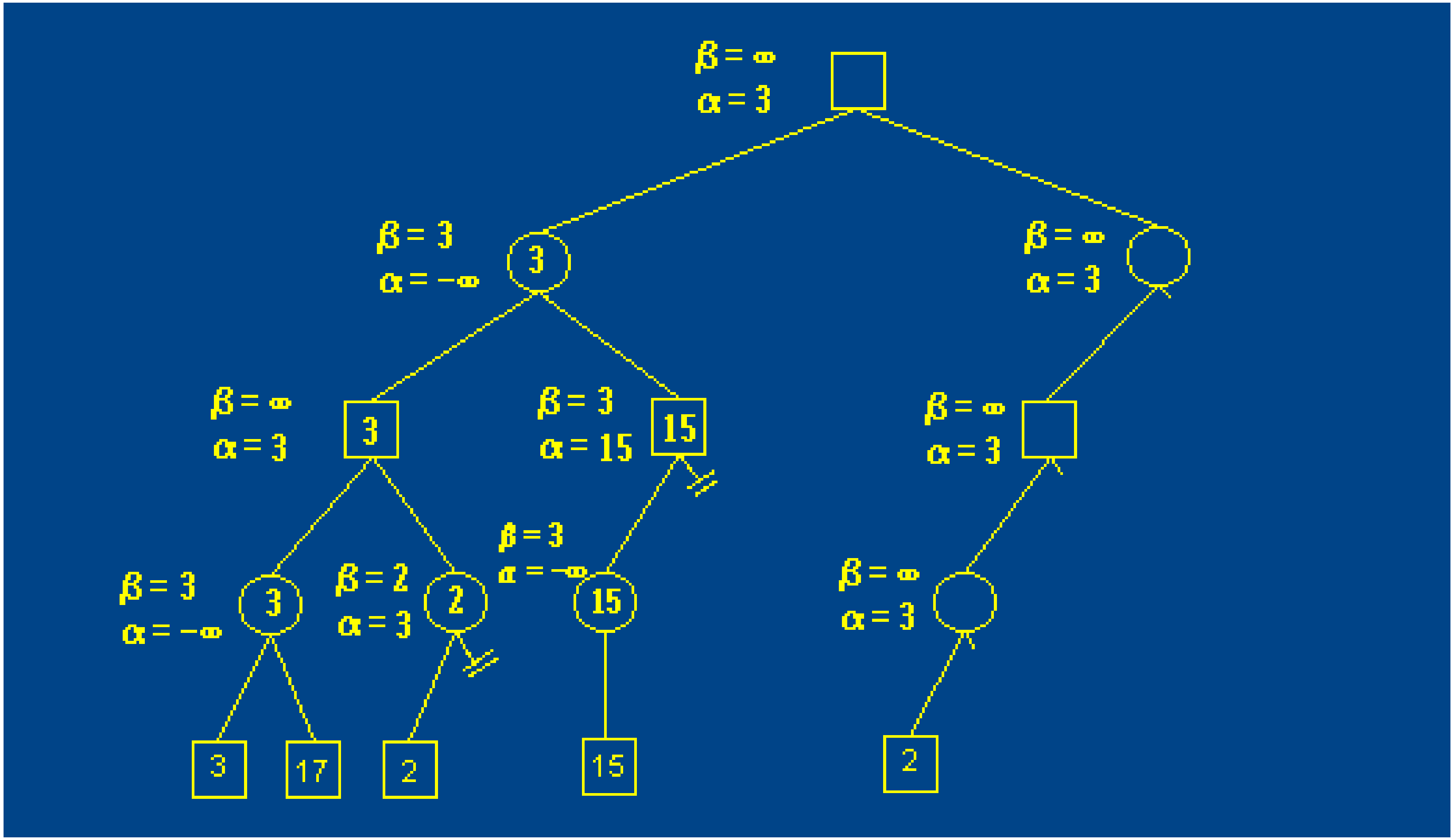
- ... and its first child



Example

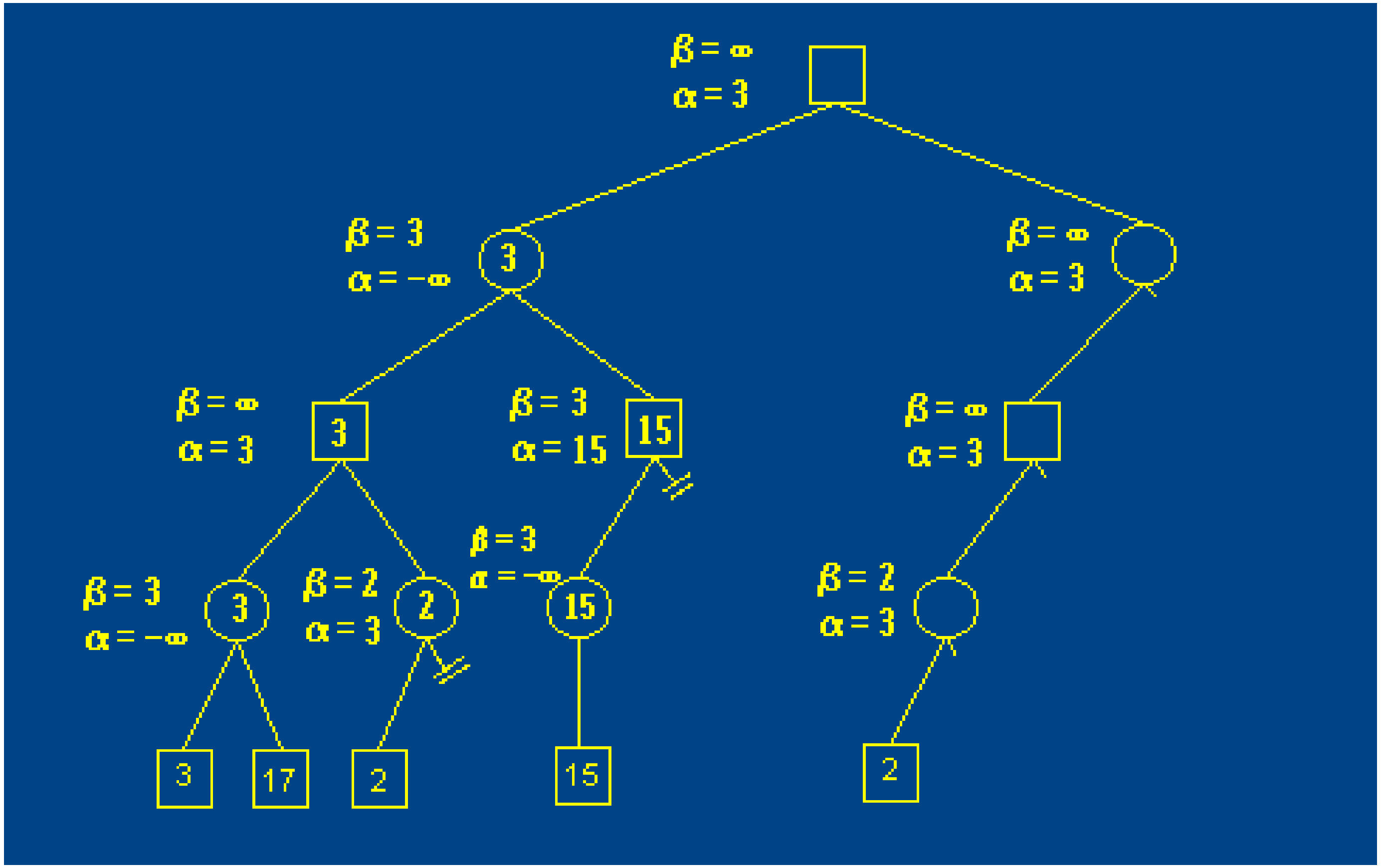


Example



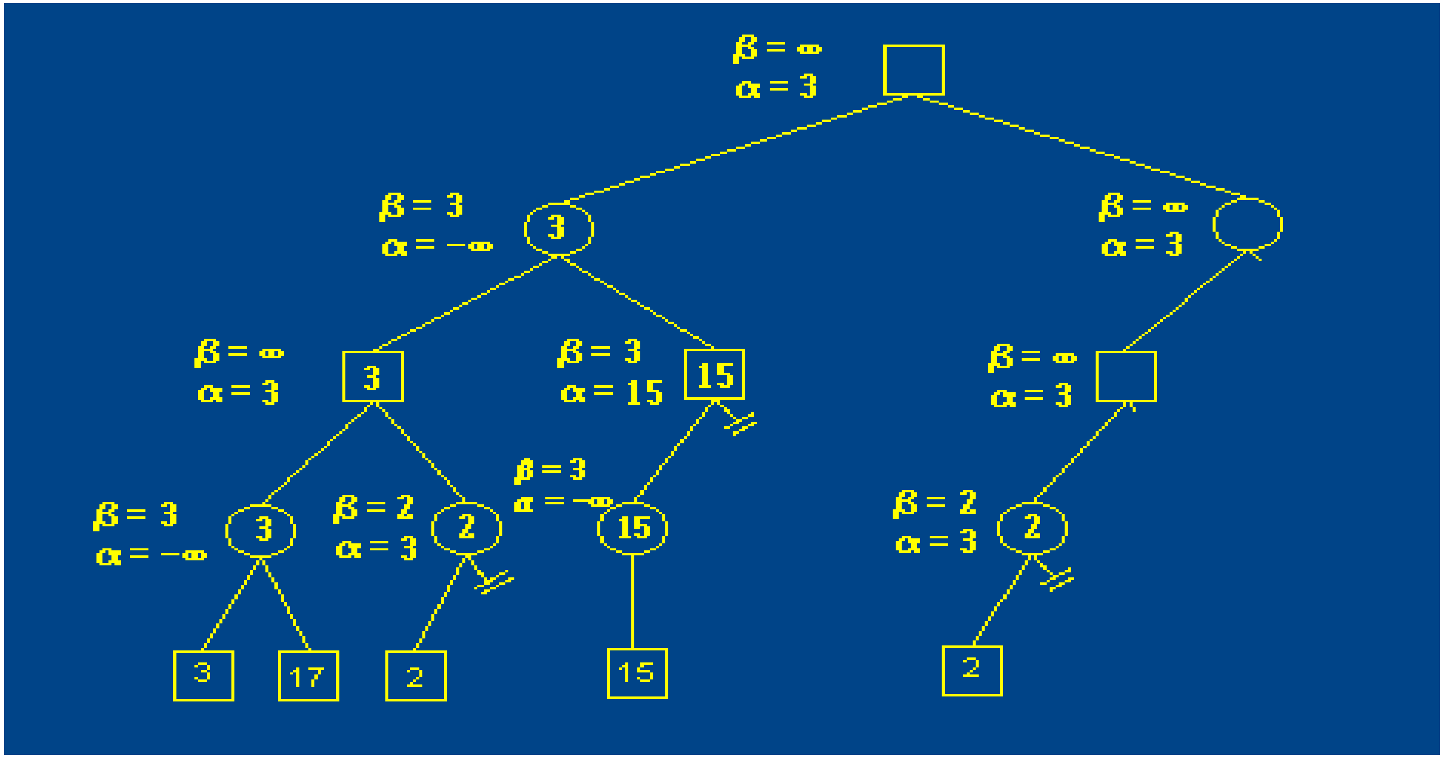
Example

- The min node parent uses this value to set it's beta value to 2:



Example

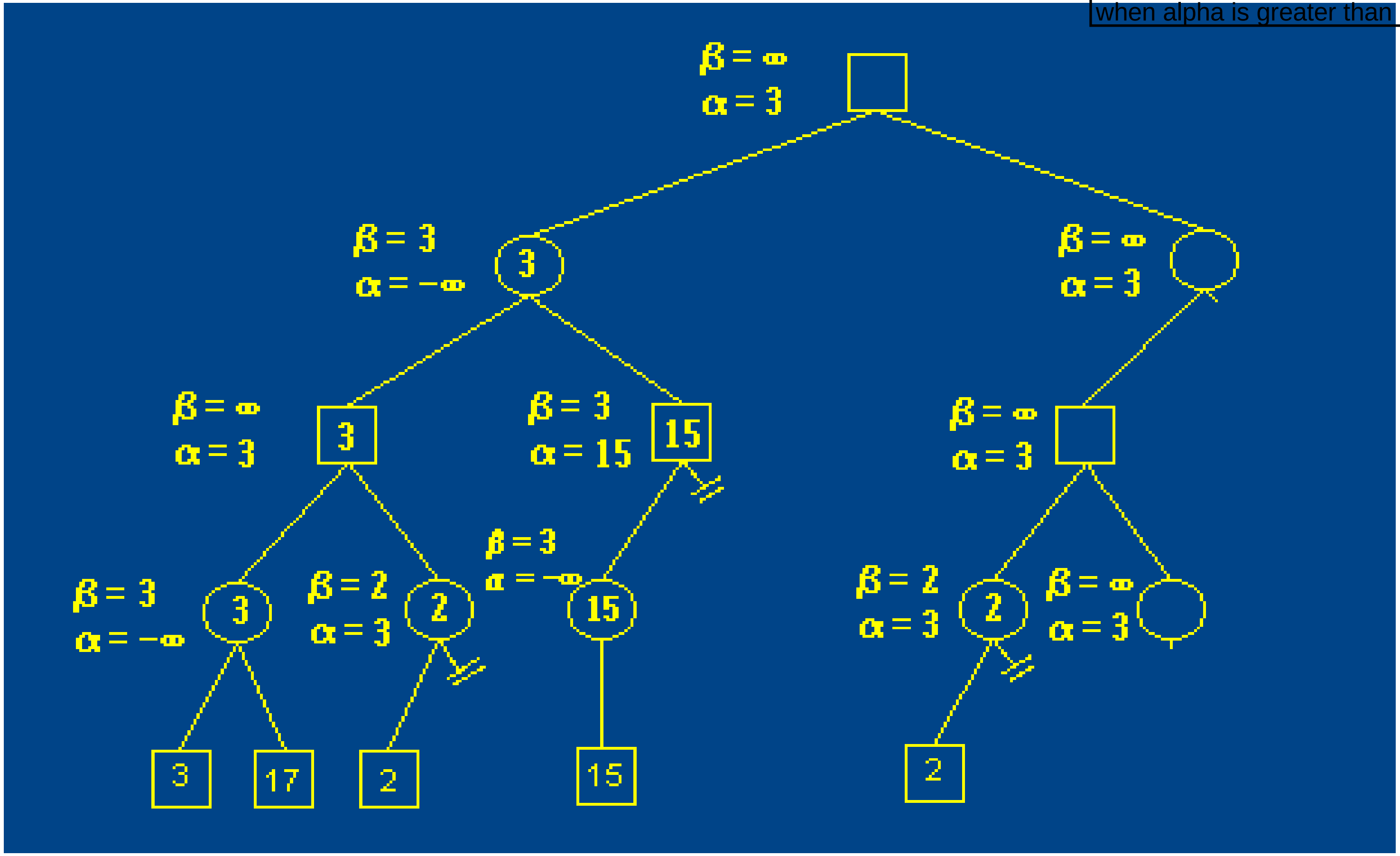
- Once again we are able to prune the other children of this node and return the value that exceeded the bound. Since this value isn't greater than the alpha bound of the parent max node, we don't change the bounds.



Example

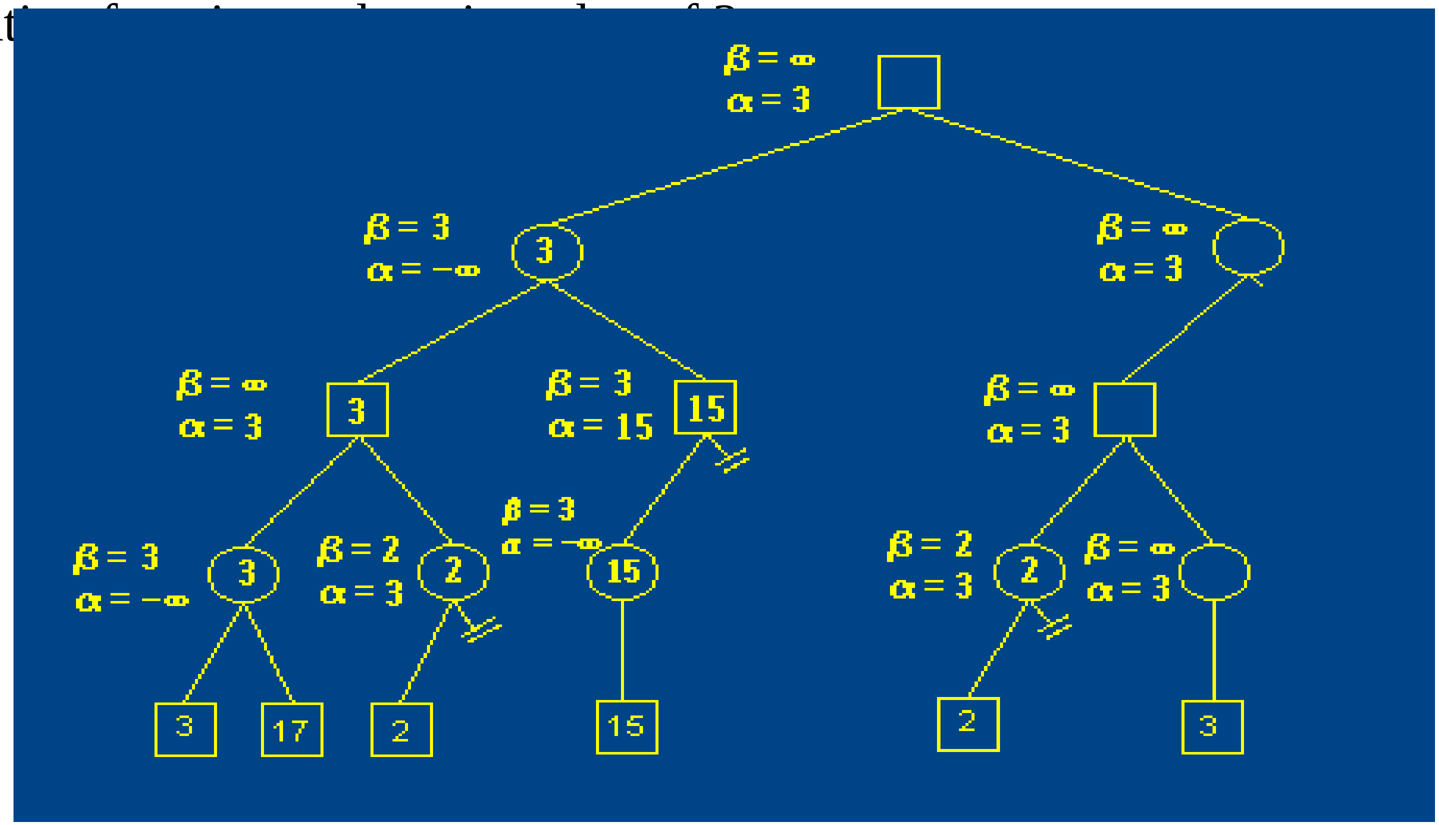
- From here, we generate the next child of the max node:

lenovo
2021-04-16 11:49:08
----
when alpha is greater than beta, prune



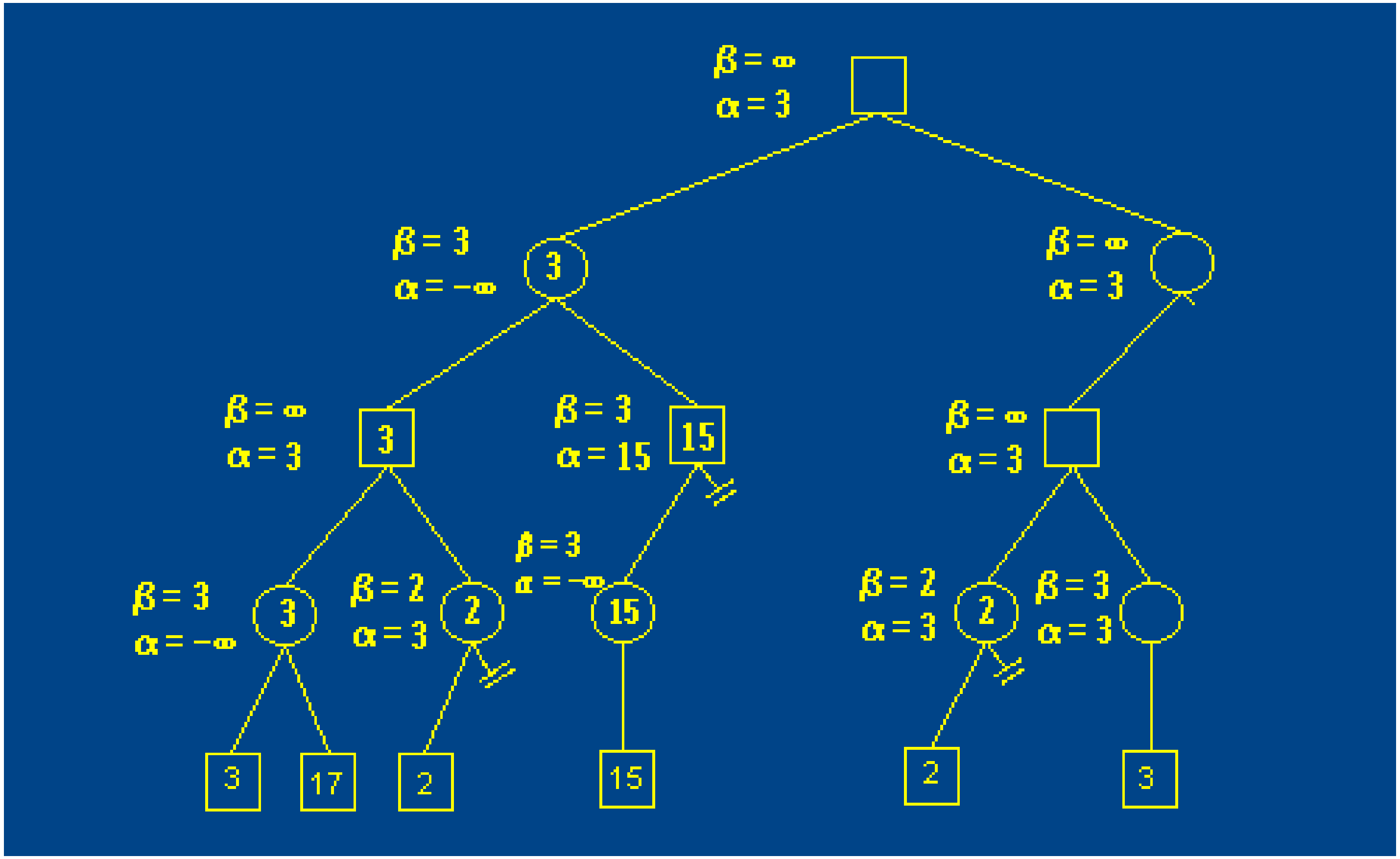
Example

- Then we generate its child, which is at the target depth. We call the evaluat

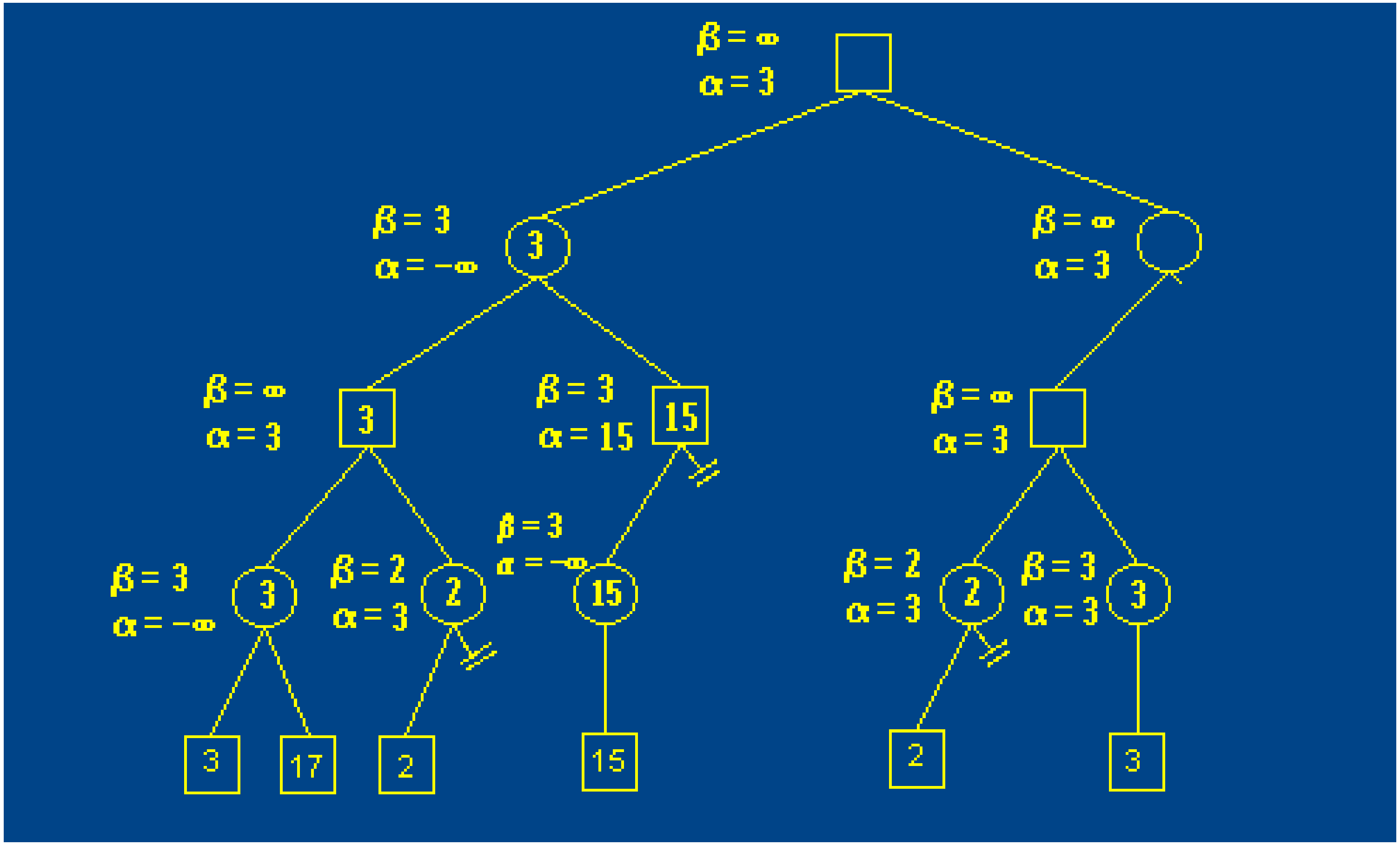


Example

- The parent min node uses this value to set its upper bound (beta) to 3:



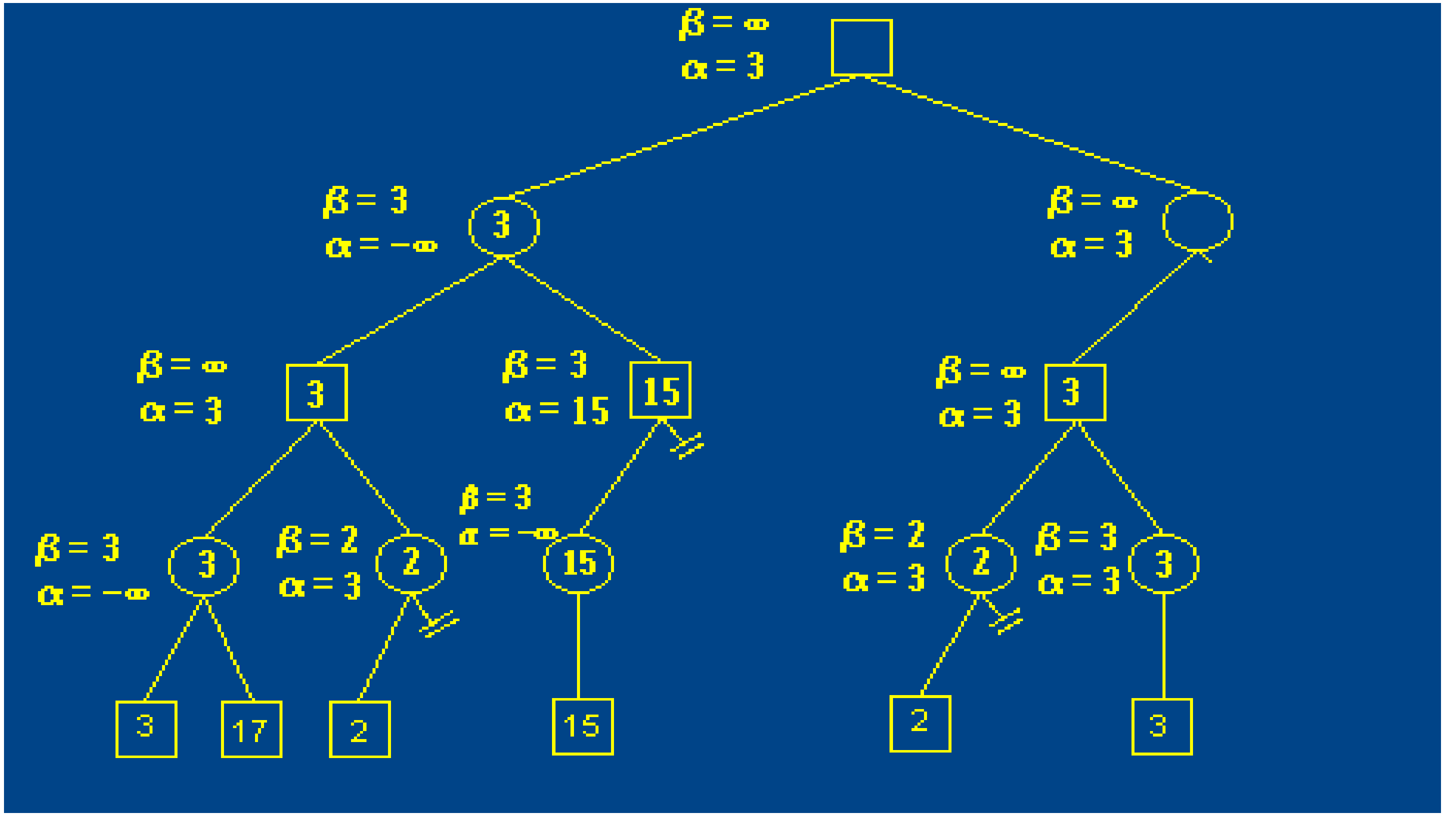
Example



β = ∞
α = 3
β = 3
α = −∞
3
β = ∞
α = 3
β = ∞
α = 3
3
β = 3
α = 15
15
β = ∞
α = 3
β = 3
α = −∞
3
β = 2
α = 3
2
β = 3
α = −∞
15
β = 2
α = 3
2
β = 3
α = 3
3
3
17
2
15
2
3

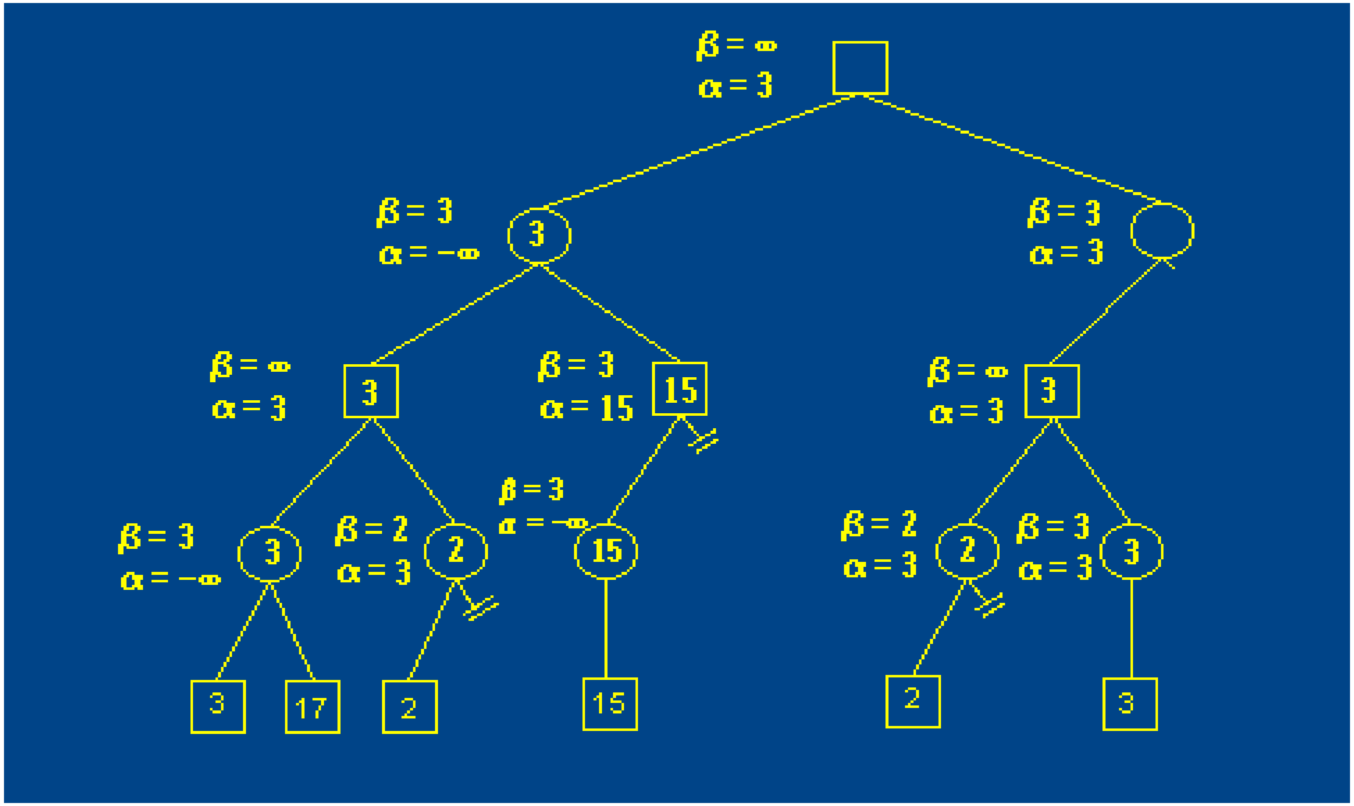
Example

- The max node above has now seen all of its children, so it returns the maximum value of those it has seen, which is 3.



Example

- This value is returned to its parent min node, which then has a new upper bound of 3, so it sets beta to 3:

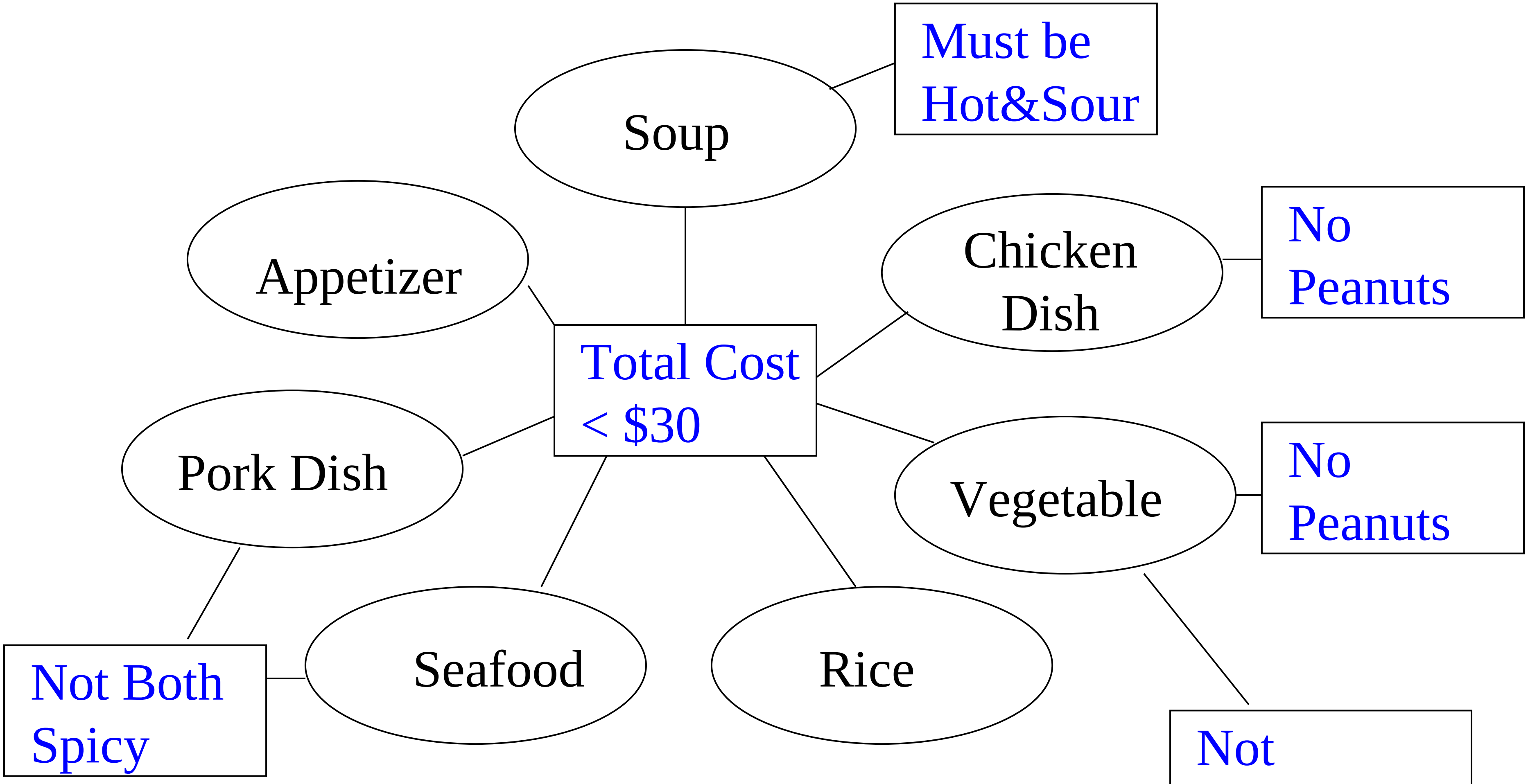


- Once again, we're at a point where alpha and beta are tied, so we prune. Note that a real solution doesn't just indicate a number, but what move led to that number.
- If you were to run minimax on the list version presented at the start of the example, your minimax would return a value of 3 and 6 terminal nodes would have been examined

Conclusion

- Pruning does not affect final results.
- Entire subtrees can be pruned, not just leaves.
- Good move *ordering* improves effectiveness of pruning.
- With *perfect ordering*, time complexity is $O(b^{m/2})$.
  - Effective branching factor of sqrt(b)
  - Consequence: alpha-beta pruning can look twice as deep as minimax in the same amount of time.

Constraint Satisfaction Problems



Constraint Network

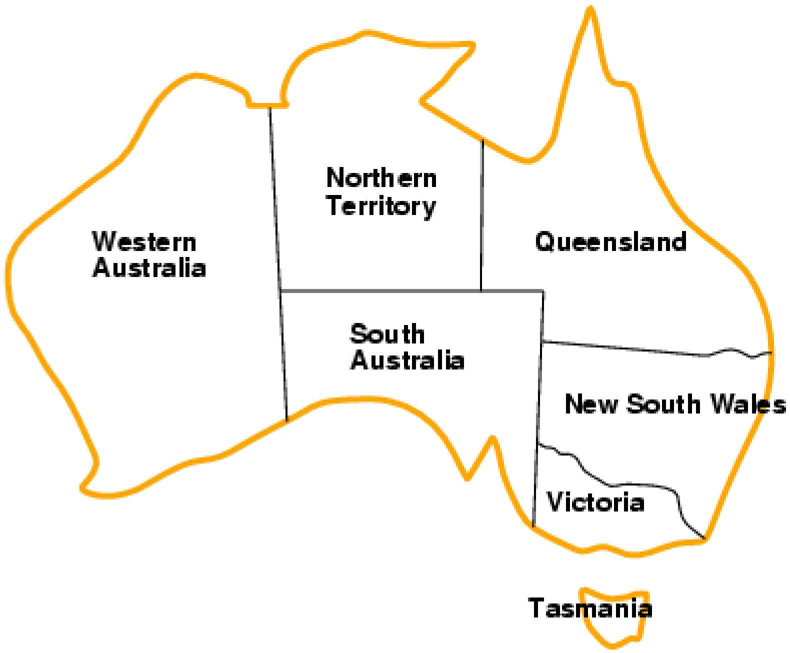
Formal Definition of CSP

- A constraint satisfaction problem (CSP) is a triple (V, D, C) where
  - V is a set of variables $X_1, \ldots, X_n$.
  - D is the union of a set of domain sets $D_1,\ldots,D_n$, where D_i is the domain of possible values for variable X_i.
  - C is a set of constraints on the values of the variables, which can be pairwise (simplest and most common) or k at a time.

CSPs vs. Standard Search Problems

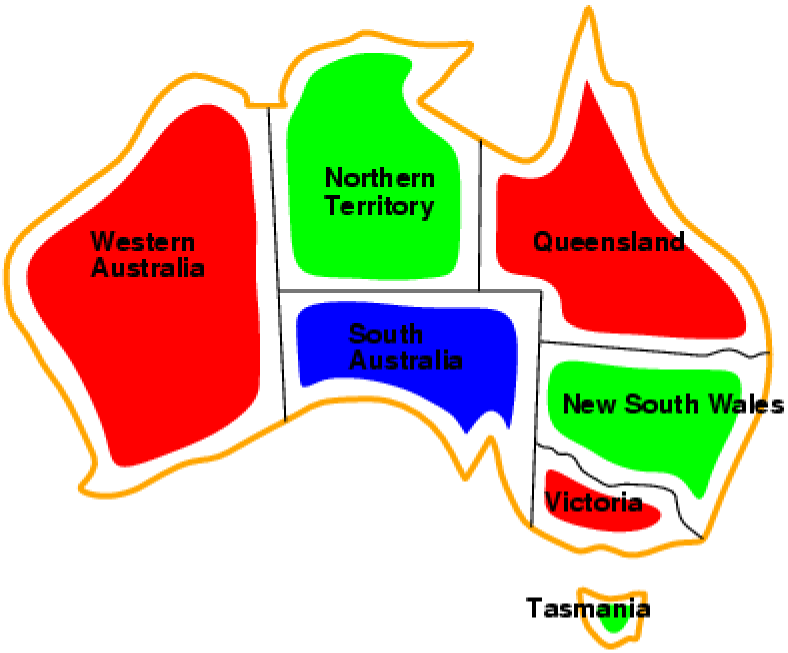
- Standard search problem:
  - state is a "black box“ – any data structure that supports successor function, heuristic function, and goal tes
- CSP:
  - state is defined by variables X_i with values from domain D_i
  - goal test is a set of constraints specifying allowable combinations of values for subsets of variables

Example: Map-Coloring



- Variables *WA*, *NT*, *Q*, *NSW*, *V*, *SA*, *T*
- Domains D_i = {red,green,blue}
- Constraints: adjacent regions must have different colors
- e.g., WA ≠ NT, or (WA,NT) in {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}

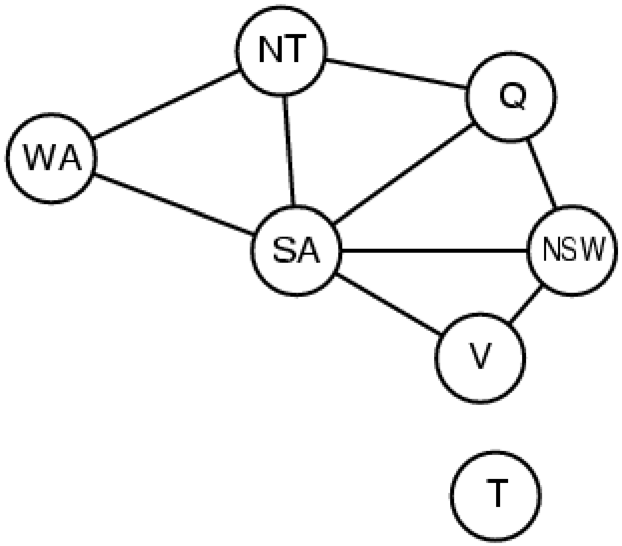
Example: Map-Coloring



- Solutions are complete and consistent assignments, e.g., WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green

Constraint graph

- Binary CSP: each constraint relates two variable
- Constraint graph: nodes are variables, arcs are constraints



Varieties of constraints

- Unary constraints involve a single variable,
  - e.g., SA ≠ green
- Binary constraints involve pairs of variables,
  - e.g., value(SA) ≠ value(WA)
- Higher-order constraints involve 3 or more variables,
  - e.g., cryptarithmetic column constraints
  -

Real-world CSPs

- Assignment problems
  - e.g., who teaches what class
- Timetabling problems
  - e.g., which class is offered when and where?
- Transportation scheduling
- Factory scheduling

  Notice that many real-world problems involve real-valued variables

-

The Consistent Labeling Problem

- Let P = (V,D,C) be a constraint satisfaction problem.
- An assignment is a partial function f : V -> D that assigns a value (from the appropriate domain) to each variable
- A consistent assignment or consistent labeling is an assignment f that satisfies all the constraints.
- A complete consistent labeling is a consistent labeling in which every variable has a value.

Constraint Satisfaction

- Constraint Satisfaction is a two step process:
  - First constraints are discovered and propagated as far as possible throughout the system.
  - Then if there still not a solution, search begins. A guess about something is made and added as a new constraint.

Algorithm: Constraint Satisfaction

1. Propagate available constraints. To do this first set OPEN to set of all objects that must have values assigned to them in a complete solution. Then do until an inconsistency is detected or until OPEN is empty:
   a. Select an object OB from OPEN. Strengthen as much as possible the set of constraints that apply to OB.
   b. If this set is different from the set that was assigned the last time OB was examined or if this is the first time OB has been examined, then add to OPEN all objects that share any constraints with OB.
   c. Remove OB from OPEN.

2. If the union of the constraints discovered above defines a solution, then quit and report the solution.

3. If the union of the constraints discovered above defines a contradiction, then return the failure.

4. If neither of the above occurs, then it is necessary to make a guess at something in order to proceed. To do this loop until a solution is found or all possible solutions have been eliminated:

   a. Select an object whose value is not yet determined and select a way of strengthening the constraints on that object.

   b. Recursively invoke constraint satisfaction with the current set of constraints augmented by strengthening constraint just selected.

Crypt Arithmetic Problems

Cryptarithmetic Problem

"It is an arithmetic problem which is represented in letters. It involves the decoding of digit represented by a character. It is in the form of some arithmetic equation where digits are distinctly represented by some characters. The problem requires finding of the digit represented by each character. Assign a decimal digit to each of the letters in such a way that the answer to the problem is correct. If the same letter occurs more than once, it must be assigned the same digit each time. No two different letters may be assigned the same digit".

Procedure

- Cryptarithmatic problem is an interesting constraint satisfaction problem for which different algorithms have been developed. Cryptarithm is a mathematical puzzle in which digits are replaced by letters of the alphabet or other symbols. Cryptarithmatic is the science and art of creating and solving cryptarithms.
- The different constraints of defining a cryptarithmatic problem are as follows.
  1) Each letter or symbol represented only one and a unique digit throughout the problem.
  2) When the digits replace letters or symbols, the resultant arithmetical operation must be correct.
- The above two constraints lead to some other restrictions in the problem.

Example

- Consider that, the base of the number is 10. Then there must be at most 10 unique symbols or letters in the problem. Otherwise, it would not possible to assign a unique digit to unique letter or symbol in the problem. To be semantically meaningful, a number must not begin with a 0. So, the letters at the beginning of each number should not correspond to 0. Also one can solve the problem by a simple blind search. But a rule based searching technique can provide the solution in minimum time.

```
 . EAT
+ THAT
---------------
 A PP LE
```

Step 1

- In the above problem, M must be 1. You can visualize that, this is an addition problem. The sum of two four digit numbers cannot be more than 10,000. Also M cannot be zero according to the rules, since it is the first letter.

```
  S END
+ MORE
-------------
 M ONEY
```

→

```
 SEN D
+1 0 R E
--------------
 1 0 N Y
```

Step 2

- Now in the column s10, s+1 $\geq$ 10. S must be 8 because there is a 1 carried over from the column EON or 9. O must be 0 (if s=8 and there is a 1 carried or s = 9 and there is no 1 carried) or 1 (if s=9 and there is a 1 carried). But 1 is already taken, so O must be 0.

8 9

```
  S END
+ 1 0 RE
-----------
1 0 NEY
```

0 1

Step 3

- There cannot be carry from column EON because any digit $+0 < 10$, unless there is a carry from the column NRE, and $E=9$; But this cannot be the case because then N would be 0 and 0 is already taken. So $E < 9$ and there is no carry from this column. Therefore $S=9$ because $9+1=10$.

Step 4

- In the column EON, E cannot be equal to N. So there must be carry from the column NRE; E+1=N. We now look at the column NRE, we know that E+1=N. Since we know that carry from this column, N+R=1E (if there is no carry from the column DEY) or N+R+1=1E (if there is a carry from the column DEY).

```
     1
  9 E N D
+ 1 O R E
---------
1 O N E Y
```

- Let us see both the cases:
  - No carry: N +R =10 +(N- 1)= N + 9
  - R = 9
- But 9 is already taken, so this will not work
  - Carry: N +R + 1= 9
  - R = 9 – 1= 8 This must be solution for R

```
  9 E N D
+ 1 0 8 E
---------
1 O N E Y
```

Step 5

- Now just think what are the digits we have left? They are 7, 6, 5, 4, 3 and 2. We know there must be a carry from the column DEY. So D+E>10.N = E+1, So E cannot be 7 because then N would be 8 which is already taken. D is almost 7, so E cannot be 2 because then D +E <10 and E cannot be 3 because then D+E=10 and Y=0, but 0 is already taken. Also E cannot be 4 because if D >6,D+E<10 and if D=6 or D=7 then Y=0 or Y=1, which are both taken. So E is 5 or 6.
- If E=6, then D=7 and Y=3. So this part will work but look the column N8E. Point that there is a carry from the column D5Y.N+8+1=16(As there is a carry from this column). But then N=7 and 7 is taken by D therefore E=5.

```
   9 5 N D
 + 1 0 8 5
-------------------
 1 0 N 5 Y
```

Step 6

- Now we have gotten this important digit, it gets much simpler from here

$$N+8+1=15,\ N=6$$

```
   9 5 6 D
 + 1 0 8 5
-----------
 1 0 6 5 Y
```

Step 7

- The digits left are 7, 4, 3 and 2. We know there is carry from the column D5Y, so the only pair that works is D=7 and Y= 2.

```
   9567
 + 1085
---------
 1 0652
```

Which is final solution of this problem.

11

Transposition Table

- A transposition is the re-occurrence of a position in a search process.
- For example, in Chess the position after 1. e4 e5 2. Nf3 is the same as after 1. Nf3 e5 2. e4.

Storing the information in a table can give huge gains.E.g., in an alpha-beta search process, store the:

- Value
- Best move/action
- Search depthFlag (real value, upper bound or lower bound)
- Identification (hash key, see further)

Normally the number of possible positions largely exceeds the available memory for a transposition table. E.g., Chess has some 1050 possible positions.

Solution: hashing.

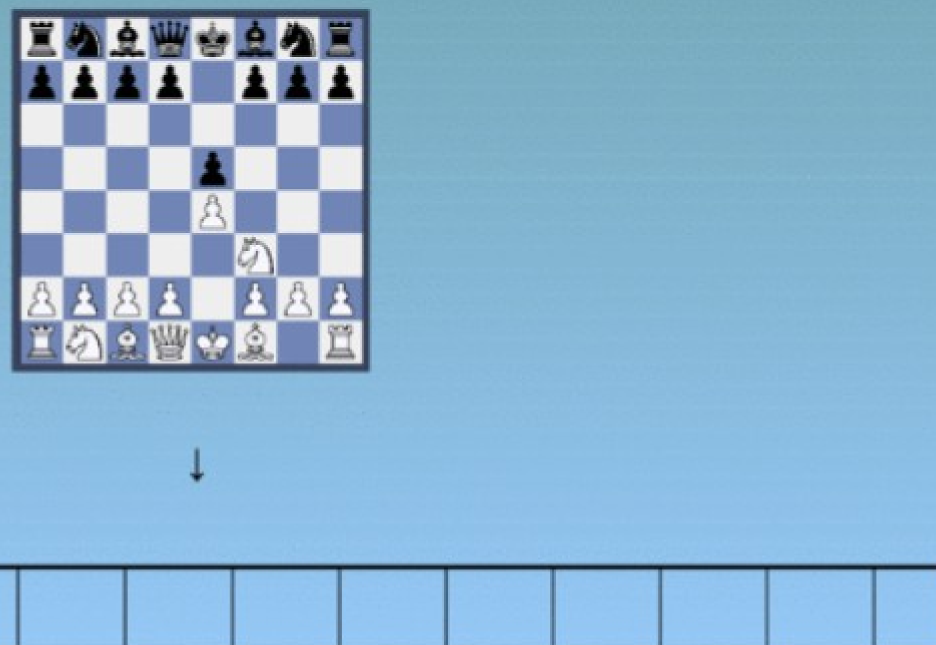
Requirements:

Unique mapping from position to table

Quick calculation of table entry

Uniform distribution of positions over the table

- The hash value is used to map a position to a table:



- we only use part of the hash value (say, k bits) as a the entry. This is called the primary hash code. Therefore, transposition tables typically have 2k entries.
- Another hash value (or typically the remaining bits) are used for identifications purposes (secondary hash code or hash key).E.g., for 64-bits random numbers 20 bits are used as primary hash code for the mapping on a 220 entry transposition table, and the remaining 44 bits are used as hash key.

- Transposition tables can be of great importance, with huge savings.Importance depends on type of game and type of position

Any Queries?

- MAX MIN algorithm is used two player gaming problem.
- Number of moves to be searched can be reduces by using alpha beta pruning.
- Solution of problem to satisfy all constraints so Constraint Search Algorithm .