Alpha-beta pruning and additional refinement

Dr. S. Selot

The minimax algorithm: problems

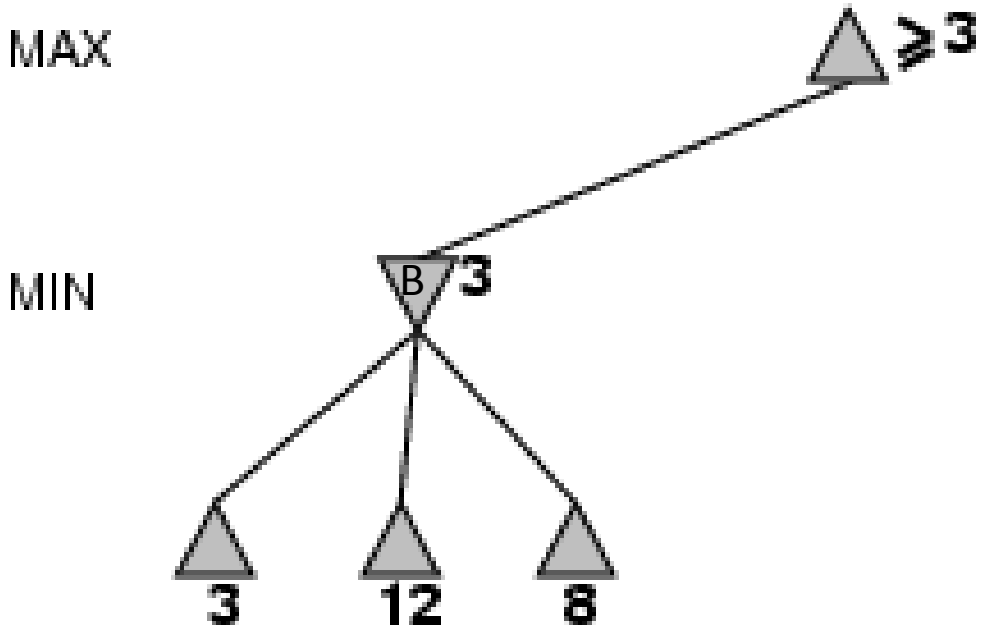
- For real games the time cost of minimax is totally impractical, but this algorithm serves as the basis:
  - for the mathematical analysis of games and
  - for more practical algorithms
- Problem with minimax search:
  - The number of game states it has to examine is exponential in the number of moves.
- Unfortunately, the exponent can't be eliminated, but it can be cut in half.

Alpha-beta pruning

- It is possible to compute the correct minimax decision without looking at every node in the game tree.

- Alpha-beta pruning allows to eliminate large parts of the tree from consideration, without influencing the final decision.

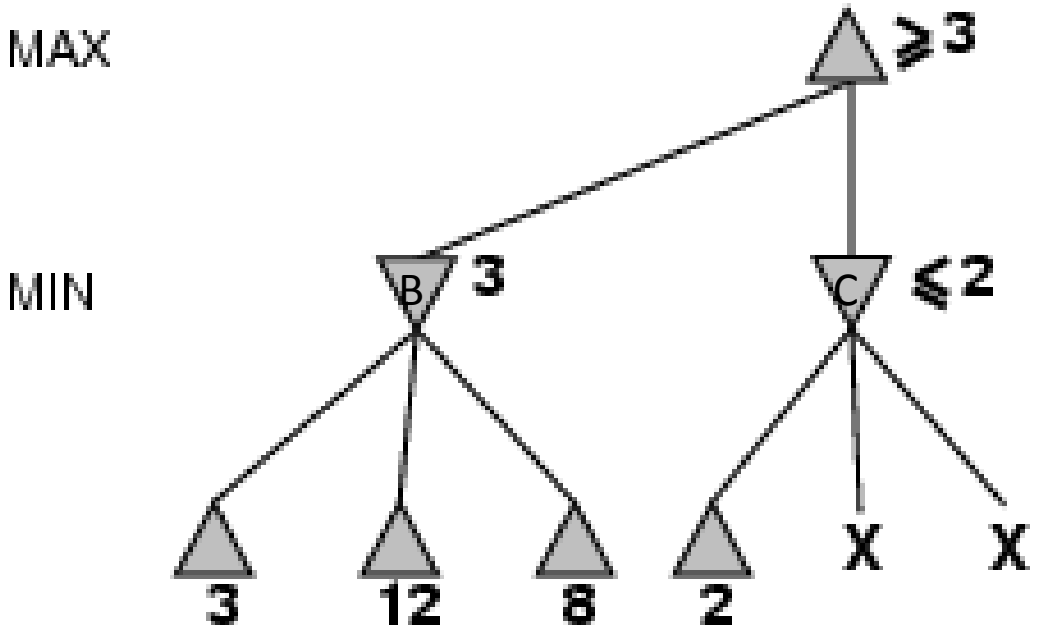
Alpha-beta pruning



,

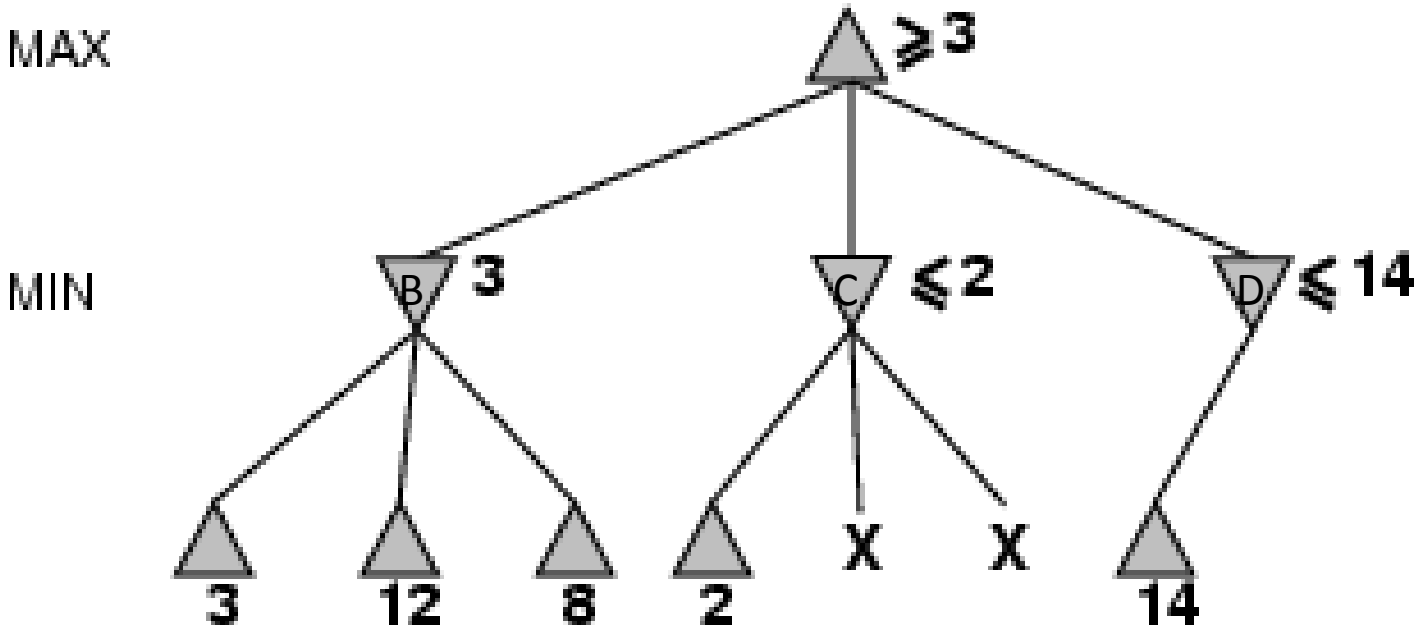
- It can be inferred that the value at the root is *at least* 3, because MAX has a choice worth 3.

Alpha-beta pruning



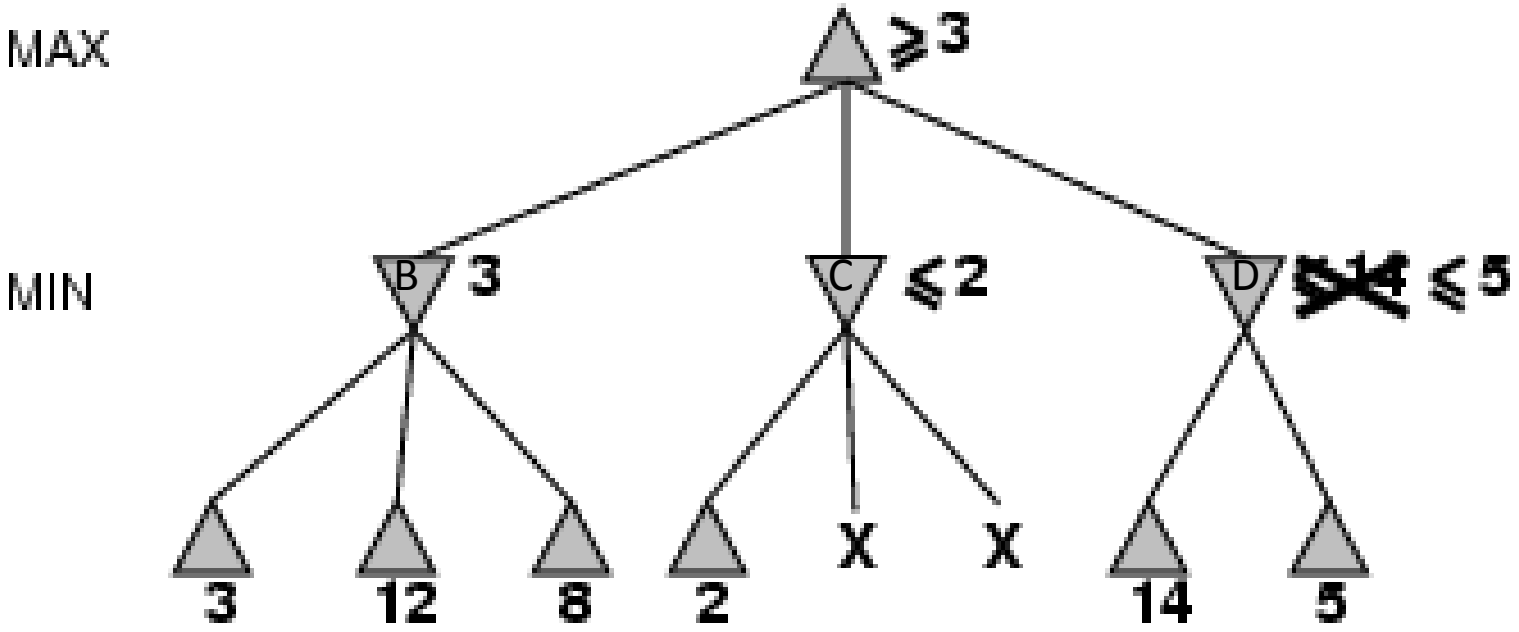
- Therefore, there is no point in looking at the other successors of *C*.

Alpha-beta pruning



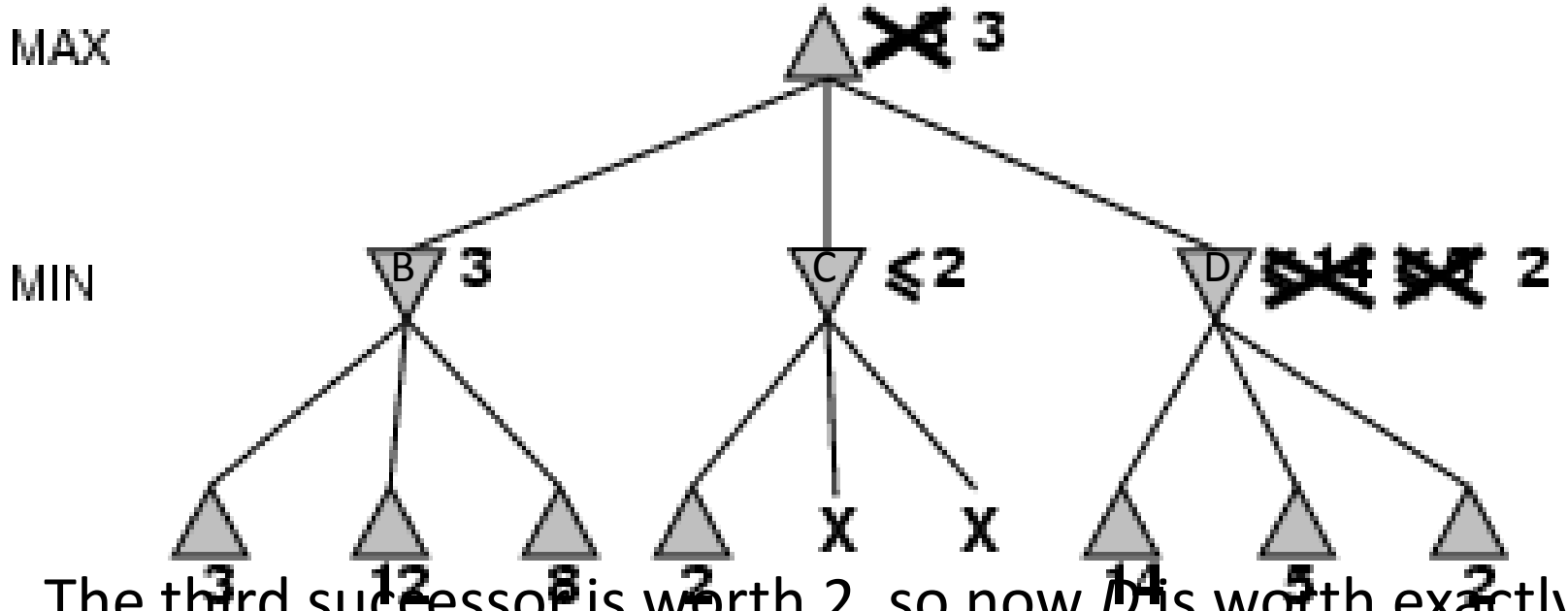
- This is still higher than MAX's best alternative (i.e., 3), so *D*'s other successors are explored.

Alpha-beta pruning



- The second successor of *D* is worth 5, so the exploration continues.

Alpha-beta pruning



- The third successor is worth 2, so now *D* is worth exactly 2.
- MAX's decision at the root is to move to *B*, giving a value of 3

Alpha-beta pruning

- Alpha-beta pruning require two thresholds
  - **Lower bound α** on the value that maximizing node may assigned
    - the value of the best (i.e., highest value) choice found so far along the path for MAX
    - Each MAX node has a α value that never decreases
  - **Upper bound β** on the value that minimizing node may be assigned
    - the value of the best (i.e., lowest value) choice found so far along the path for MIN
    - Each MIN node has β value that never increses

Alpha-beta pruning

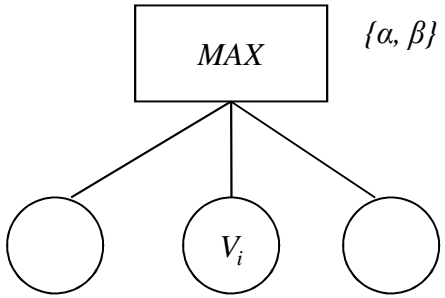
- Alpha-beta values are set and updated when the value of the successor node is obtained

- Search is depth first and stops
  - At any MIN node whose beta value is smaller that alpha value of its parent
  - At any MAX node whose alpha value is greater than beta value of its parent

Alpha-beta bounds

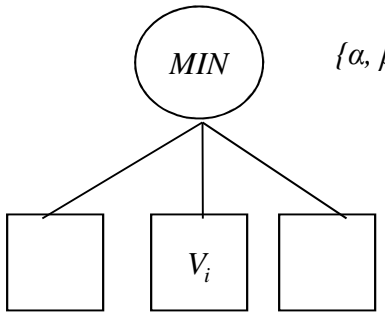
- Alpha bound of j
  - Maximum current value of all ancestor of j
  - Exploration of a min node is stopped when its value equals or falls below alpha-----alpha pruning
  - In a MIN node we update beta and use alpha for cutting
- Beta bound of j
  - The minimum current of all ancestor of j
  - Exploration of a max node is stopped when its value equals or exceeds beta -----beta pruning
  - In MAX node we update alpha and use beta for cutting

Alpha-beta pruning



If $V_i > \alpha$, modify α
If $Vi \geq \beta$, β pruning

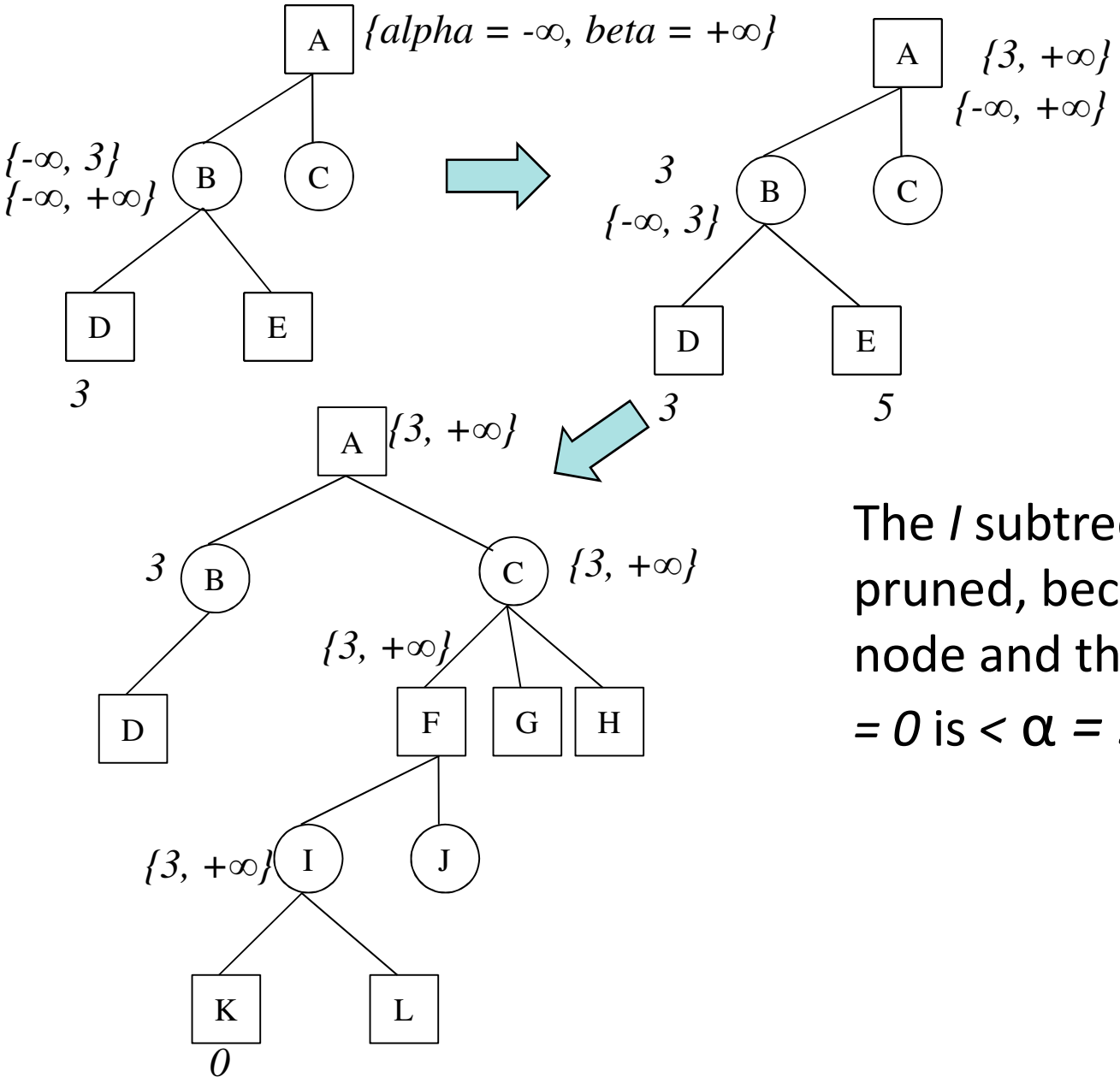
Return α



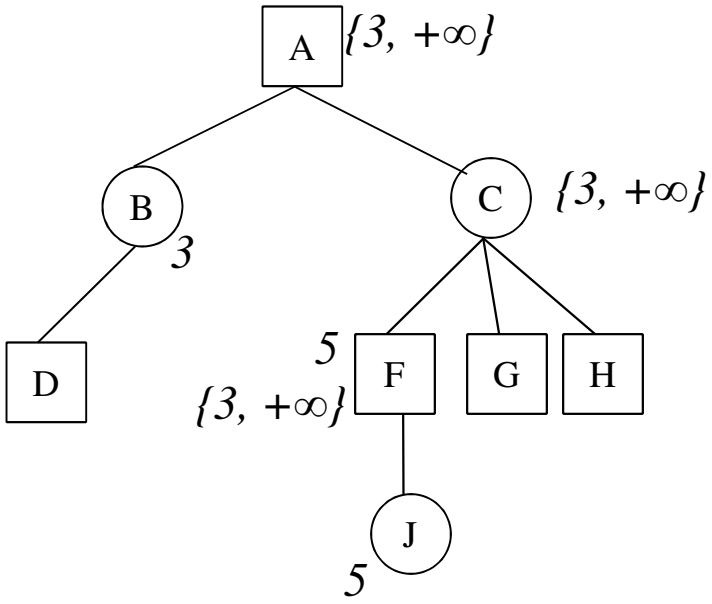
If $V_i < \beta$, modify β
If $Vi \leq \alpha$, α pruning

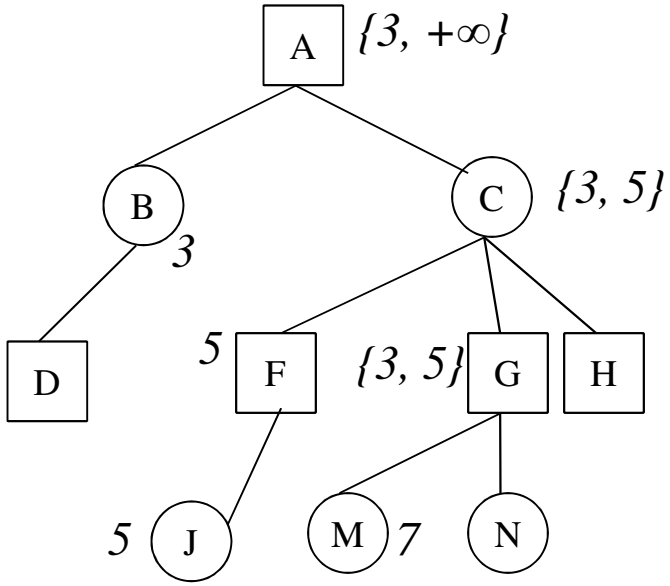
Return β

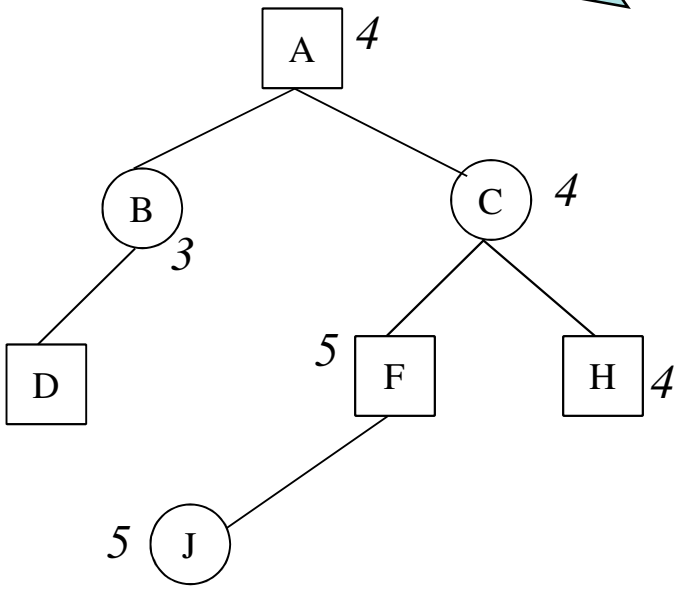
α and β bounds are transmitted from parent to child in the order of node visit. The effectiveness of pruning highly depends on the order in which successors are examined.



The *I* subtree can be pruned, because *I* is a *min* node and the value of *v(K) = 0* is < α = *3*







The *G* subtree can be pruned, because *G* is a *max* node and the value of *v(M) = 7* is > β *= 5*

Minmax_alpha_beta alg

- At maximizing level we use beta to determine whether search is cut-off
- At minimizing level we use alpha to prune the tree
- Value of alpha and beta must be known at maximizing or minimizing level so that they can be passed to next level
- Each level should posses both the value
  - One to be used for cutting *use_thresh (UT)*
  - and other to pass to nexrt level--- *pas_thresh (PT)*

Minmax _A_B(position, depth, player ,UT,PT)

{

Step1:If DEEP-ENOUGH(*position,player*) then return structure

VALUE=STATIC(*position , player)*

PATH=nil

Step2: Else

{

SUCCESSORS=MOVE_GEN(*position,player)*

Step3: if (SUCCESSOR= empty) return structure // same as DEEP_ENOUGH

Step4: if (! Empty(SUCCESSORSvalue))

{ BEST_SCORE=min that STATIC can generate

For every SUCC of SUCCESSOR

4(a) { RESULT_SUCC= MINMAX_A_B(SUCC, *depth+1,*OPPOSITE(player),-PT,-UT)

4(b) NEW-VALUE= - VALUE(RESULT_SUCC)

4(c) if NEW-VALUE > PT

(i) { PT=NEW-VALUE

(ii) BEST-PATH= SUCC + PATH(RESULT_SUCC)

path from CURRENT to SUCC .

}// end if

(d) if PT >= UT then stop examining this branch and return value=PT path=BEST-PATH

}// end for

Step5: value=PT

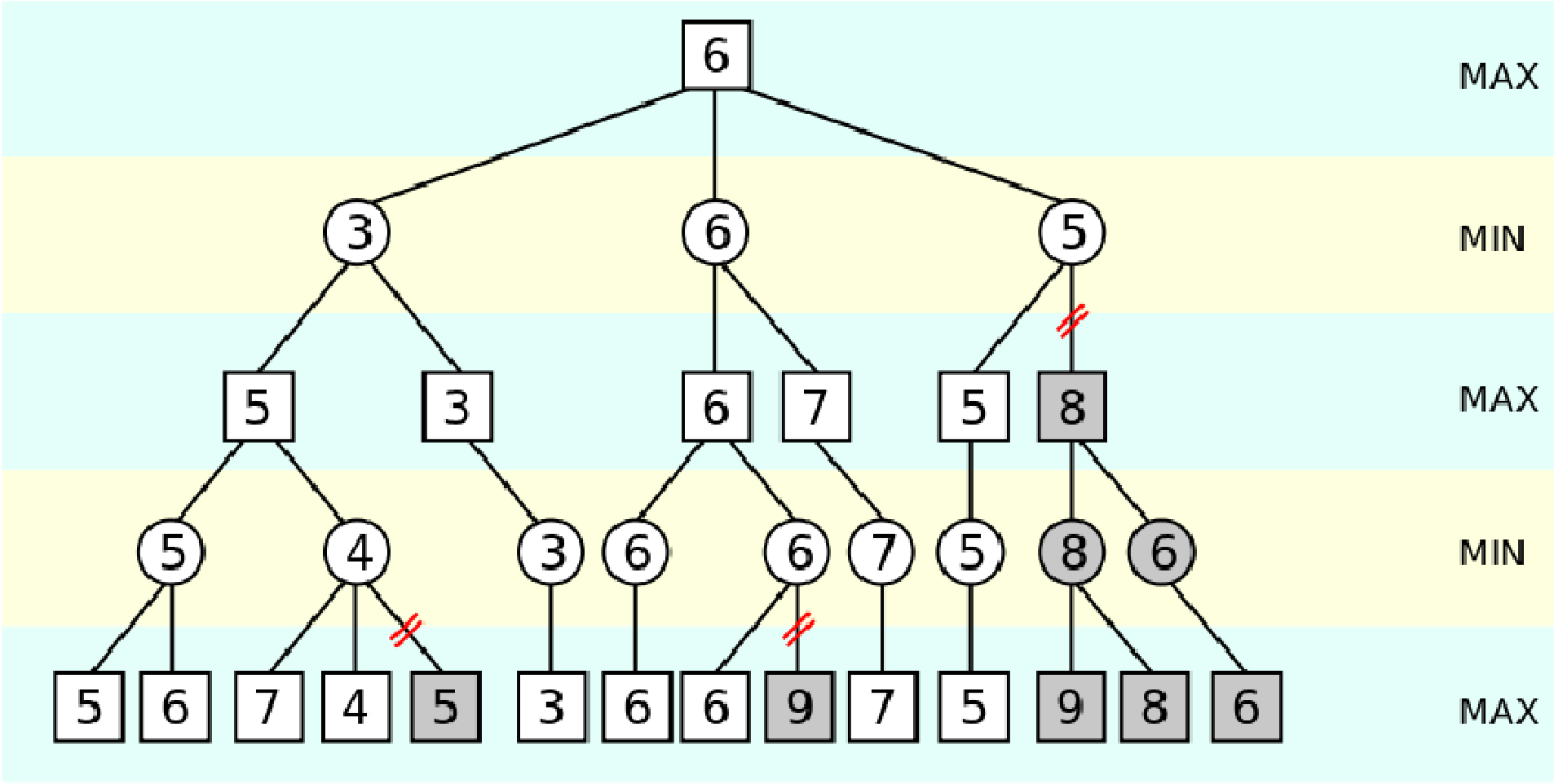
path=BEST_PATH

return structure

}// end if

}// else

Another α-β Pruning Example



Some observation in alpha beta

- Effectiveness of alpha beta depends upon the order in which the nodes are considered
- If nodes are perfectly ordered then
  - If x= number of terminal nodes in alpha beta at depth d
  - And y= number of terminal nodes without alpha beta at depth d/2
  - Then x=2*y
  - Doubling the depth only at cost of double the number of terminal nodes implies a significant gain
- This idea can be extended to cut off path that are only slightly better than current path
  - Exploration of subtree is terminated that offers little possibility of improvement , hence called *futility cut off*

Additional refinements

- Waiting for Quiescence
  - At times great difference in value is observed as we move from one level to another
  - Our decision to change path should not occur on such short term measures

- Secondary search
  - If total expansion was up to n level we can expand one node an extra level to reassure the path-singular extension
- Book moves maintaining the catalogue of all moves (list of possible moves
  - Incorporates knowledge and search together

Final comments about alpha-beta pruning

- 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.

Summary

Four steps to design AI Problem solving:

1. **Define the problem precisely.** Specify the problem space, the operators for moving within the space, and the starting and goal state.
2. **Analyze the problem** to determine where it falls with respect to seven important issues.
3. Isolate and **represent the task knowledge** required
4. **Choose problem solving technique** and apply them to problem.

Summary

- What the states in search spaces represent. Sometimes the states represent complete potential solutions. Sometimes they represent solutions that are partially specified.
- How, at each stage of the search process, a state is selected for expansion.
- How operators to be applied to that node are selected.
- Whether an optimal solution can be guaranteed.
- Whether a given state may end up being considered more than once.
- How many state descriptions must be maintained throughout the search process.
- Under what circumstances should a particular search path be abandoned.