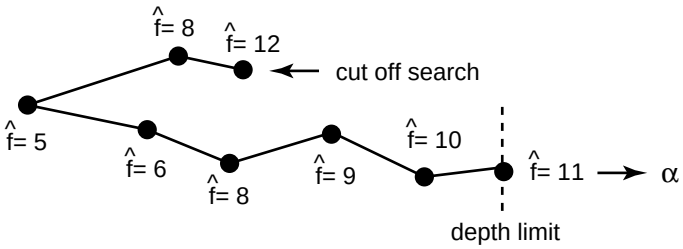
Branch and bound search & alpha cutoff

- *Branch and bound* is a class of (search) techniques from operations research
- Basic ideas: (assume optimal = minimum cost)
  - solution space is partitioned
  - lower bound established for each partition
  - partitions with bound > cost of known solution are excluded from further consideration
- A heuristic search example:
  - Depth limited search with an admissible monotonic heuristic
  - let $\hat{f}(\cdot) = \hat{g}(\cdot) + \hat{h}(\cdot)$
  - The *alpha cutoff* is the value of the best node at depth limit
  - Abort search along a path when $\hat{f}(\cdot) > \alpha$

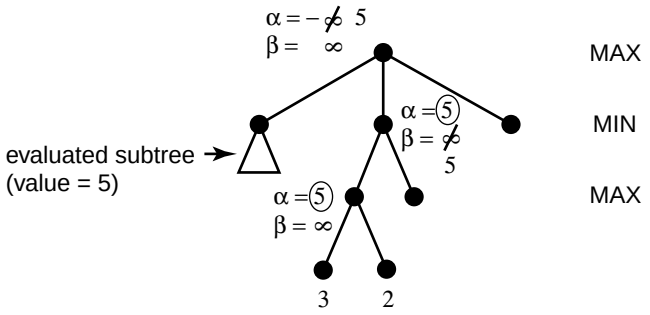


α-β pruning explained

- α-β-MINIMAX is like "branch and bound" search with an α cutoff except that there are two players.
- A "cutoff" value is maintained for each player.
- When AB:MAX-PLAYER or AB:MIN-PLAYER is called:
  - α is the highest value resulting from one of MAX's other moves at some ancestor node
  - β is the lowest value resulting from one of MIN's other moves at some ancestor node
  - $\alpha < \beta$, otherwise MAX or MIN (whichever has the first opportunity) would take the better move at a node higher in the game tree.
- If the cutoff condition ($\alpha \geq \beta$) occurs,
  - For a MIN node, we found a move that leads to a low score, but MAX would choose the (earlier) move that leads to α instead!
  - For a MIN node, we found a move that leads to a high score, but MAX would choose the (earlier) move that leads to β instead!

α-β-MINIMAX details

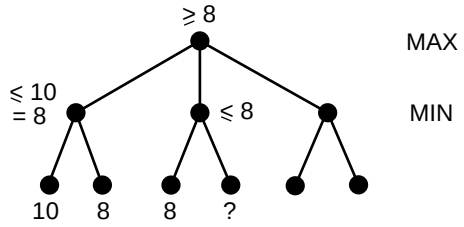
- AB:MAX-PLAYER returns:
  - normal exit: current α value (considers children of current node)
  - cutoff exit: β value from parent (MIN) node
- On cutoff exit, returning the same α or β value from the parent does not cause an update.
- On normal exit, return value may be α or β value passed to current node:



- A move leading to returned value does not necessarily exist in a subtree!
- First depth 1 child found with increased α does have an optimal game leading to that value.

Finding all optimal moves

- Finding all optimal moves requires more search:



- "Regular" α-β-MINIMAX search would stop searching middle move because left move guarantees a value of 8.
- Value of "?" node determines if middle move is same as left; it must be examined to find all optimal moves.

α-β-MINIMAX search for all optimal moves

AB:MINIMAX(n)

1. let $[v, e, \mathcal{L}]$ = AB:MAX-PLAYER(n, $-\infty$, ∞)
2. let $\mathcal{M}$ = moves leading from n to each node $c_i \in \mathcal{L}$
3. return $[v, \mathcal{M}]$

AB:MAX-PLAYER(n, α, β)

1. if game over or depth limit reached, return [eval(n), `normal`, $\emptyset$]
2. let $\alpha' = -\infty$, $\mathcal{L} = \emptyset$
3. for each child c_i of n
   - $[v_i, e_i, L_i] \leftarrow$ AB:MIN-PLAYER(c_i, $\max(\alpha, \alpha')$, β)
   - if e_i = `normal` then:
     - if $v_i > \alpha'$ then $\alpha' \leftarrow v_i$, $\mathcal{L} \leftarrow \{c_i\}$
     - if $v_i = \alpha'$ then $\mathcal{L} \leftarrow \mathcal{L} \cup \{c_i\}$
   - if $\alpha' > \beta$, return [0, `cutoff`, $\emptyset$]
4. return [α', `normal`, $\mathcal{L}$]

α-β-MINIMAX search for all optimal moves

AB:MIN-PLAYER(n, α, β)

1. if game over or depth limit reached, return [eval(n), `normal`, $\emptyset$]
2. let $\beta' = \infty$, $\mathcal{L} = \emptyset$
3. for each child c_i of n
   - $[v_i, e_i, L_i] \leftarrow$ AB:MAX-PLAYER(c_i, α, $\min(\beta, \beta')$
   - if e_i = `normal` then:
     - if $v_i < \beta'$ then $\beta' \leftarrow v_i$, $\mathcal{L} \leftarrow \{c_i\}$
     - if $v_i = \beta'$ then $\mathcal{L} \leftarrow \mathcal{L} \cup \{c_i\}$
   - if $\alpha > \beta'$, return [0, `cutoff`, $\emptyset$]
4. return [β', `normal`, $\mathcal{L}$]

Notes:

- separates value of best move(s) at ancestor (α and β) and value of best move(s) at current node (α' or β')
- keeps set of optimal child nodes $\mathcal{L}$
- AB:MAX-PLAYER and AB:MIN-PLAYER return: [value, exit-type, best-children]
- same as regular AB:MINIMAX if cutoff condition is $\alpha' \geq \beta$ or $\alpha \geq \beta'$ for AB:MAX-PLAYER and AB:MIN-PLAYER respectively

Running time of α-β-MINIMAX

- Number of nodes searched depends on the order the children are evaluated!
- α-β pruning reduces the number of nodes searched, thereby reducing the effective branching factor.
- The minimum number of leaf nodes evaluated (i.e. for perfect ordering) is:

$$N_d = \begin{cases} 2b^{d/2} - 1 & \text{for even } d \\ b^{(d+1)/2} + b^{(d-1)/2} - 1 & \text{for odd } d \end{cases}$$

- This makes the effective branching factor approximately $\sqrt{b}$ instead of b.
- (Pearl 1982) shows that average search depth is increased by approximately 4/3, corresponding to an average branching factor of $b^{3/4}$

EXPECTIMAX

- For games with a chance element, e.g. with dice.
- View search tree as the "dice move" alternating with MAX and MIN.
- When the "dice move," an expected value is computed.