Foundations of Artificial Intelligence

42. Board Games: Alpha-Beta Search

Thomas Keller and Florian Pommerening

University of Basel

May 22, 2023

Board Games: Overview

chapter overview:

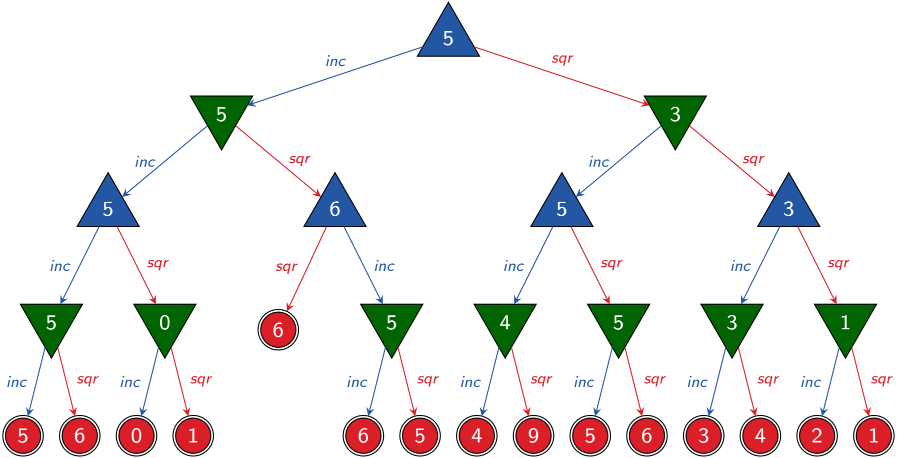
Limitations of Minimax

What if the size of the game tree is too big for minimax?

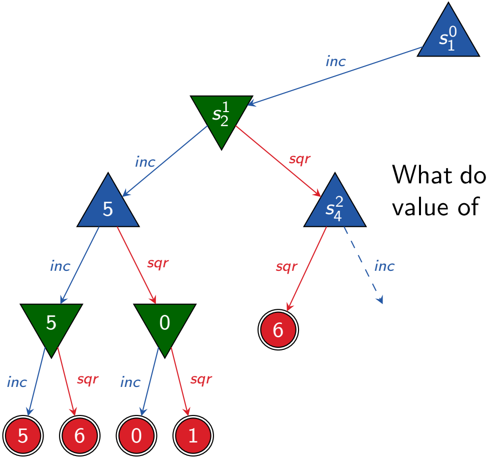
$\leadsto$ Heuristic Alpha-Beta Search

Alpha-Beta Search

Can We Save Search Effort?



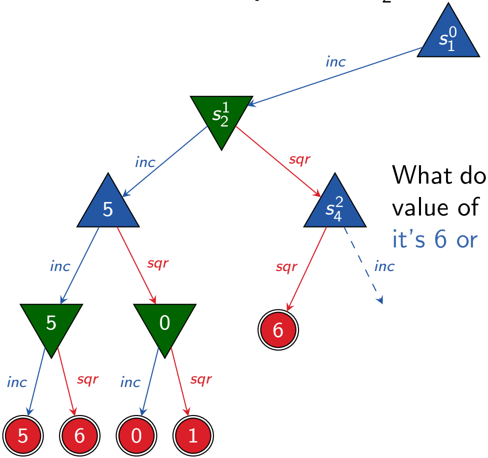
Can We Save Search Effort?



What do we know about the utility value of s_4^2 in this situation?

Can We Save Search Effort?

And about the utility value of s_2^1?

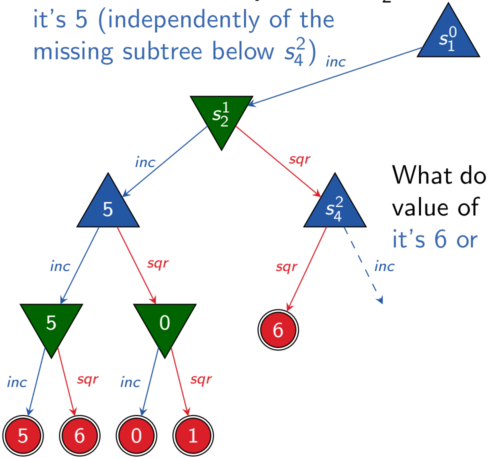


What do we know about the utility value of s_4^2 in this situation?

it's 6 or higher

Can We Save Search Effort?

And about the utility value of s_2^1?
it's 5 (independently of the missing subtree below s_4^2)

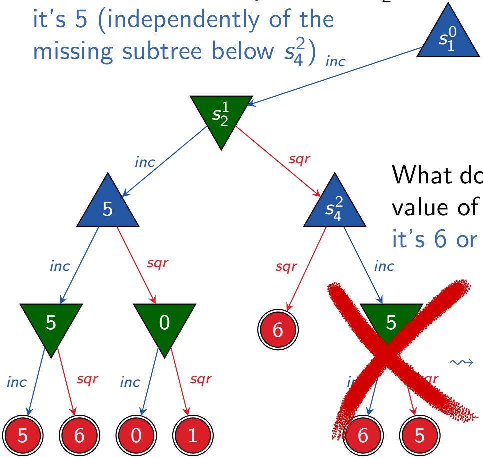


What do we know about the utility value of s_4^2 in this situation?
it's 6 or higher

Can We Save Search Effort?

And about the utility value of s_2^1?

it's 5 (independently of the missing subtree below s_4^2)



What do we know about the utility value of s_4^2 in this situation?

it's 6 or higher

$\rightsquigarrow$ we don't have to look at this

Idea

idea: use two values α and β during minimax search such that

- α is known lower bound on utility of *max* and
- β is known upper bound on utility of *min*

Idea

idea: use two values α and β during minimax search such that

- α is known lower bound on utility of *max* and
- β is known upper bound on utility of *min*

it holds for every recursive call that a subtree

- is not interesting if its utility value is $\leq \alpha$
  because *max* will never enter it when playing optimally
- is not interesting if its utility value is $\geq \beta$
  because *min* will never enter it when playing optimally
- rooted at a *max* node is pruned if utility $\geq \beta$
- rooted at a *min* node is pruned if utility $\leq \alpha$

Alpha-Beta Search: Pseudo Code

- algorithm skeleton the same as minimax
- function signature extended by two variables α and β

function alpha-beta-main(p)

$\langle v, move \rangle :=$ alpha-beta($p, -\infty, +\infty$)

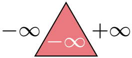
return *move*

Alpha-Beta Search: Pseudo-Code

function alpha-beta(p, α, β)

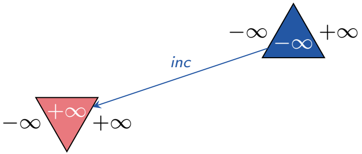
if p is terminal position:
    return $\langle \mathit{utility}(p), \textbf{none} \rangle$
initialize v and *best_move* [as in minimax]
for each $\langle \mathit{move}, p' \rangle \in \mathrm{succ}(p)$:
    $\langle v', \mathit{best\_move}' \rangle := \mathit{alpha\text{-}beta}(p', \alpha, \beta)$
    update v and *best_move* [as in minimax]
    if $\mathit{player}(p) = \mathit{max}$:
        if $v \geq \beta$:
            return $\langle v, \textbf{none} \rangle$
        $\alpha := \max\{\alpha, v\}$
    if $\mathit{player}(p) = \mathit{min}$:
        if $v \leq \alpha$:
            return $\langle v, \textbf{none} \rangle$
        $\beta := \min\{\beta, v\}$
return $\langle v, \mathit{best\_move} \rangle$

Example



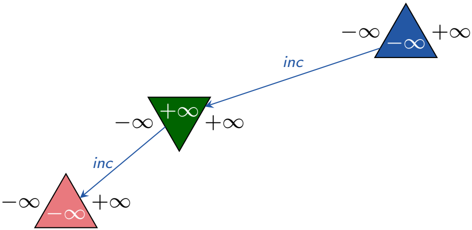
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



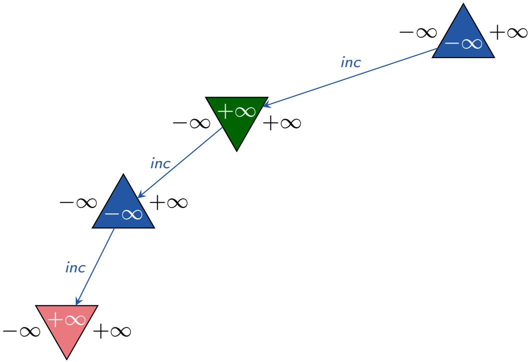
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



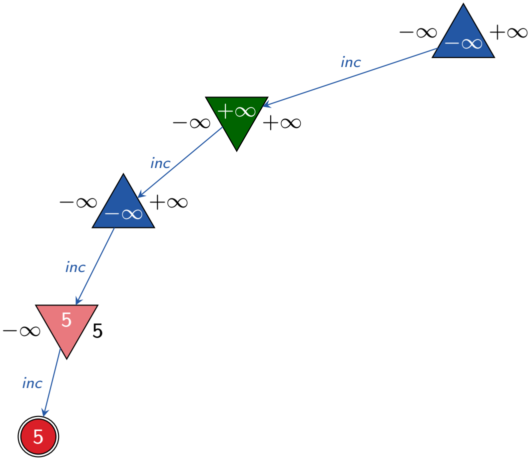
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



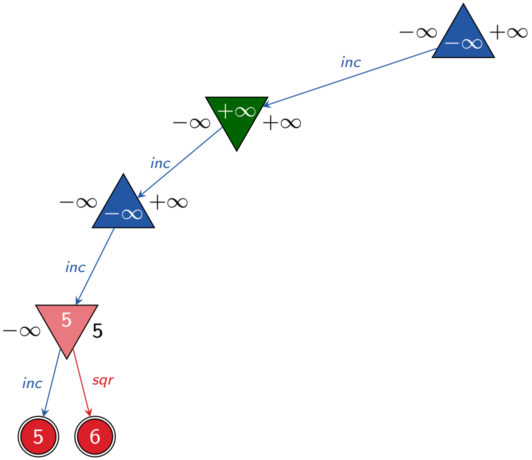
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



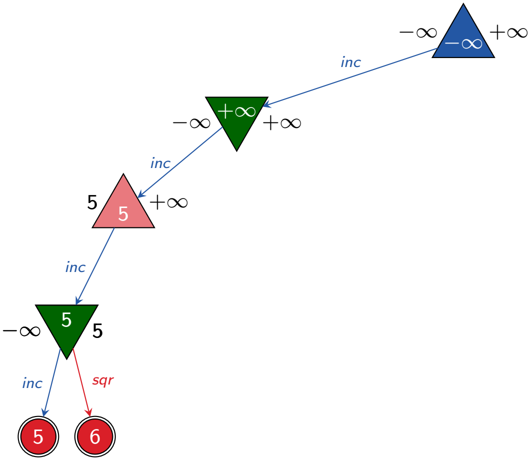
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



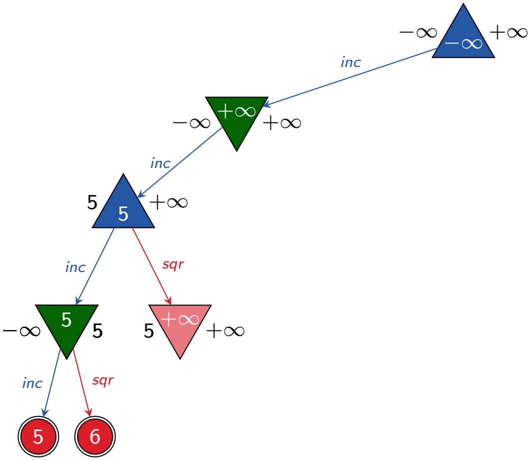
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



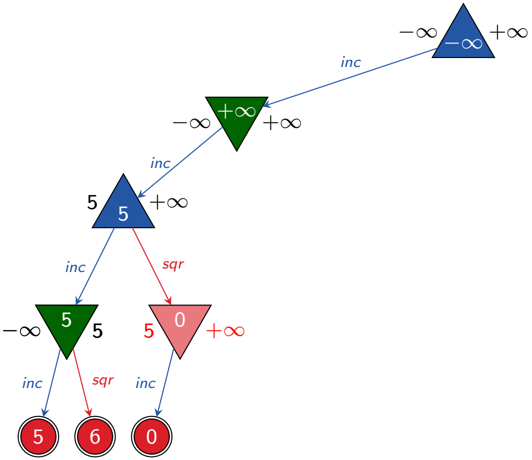
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



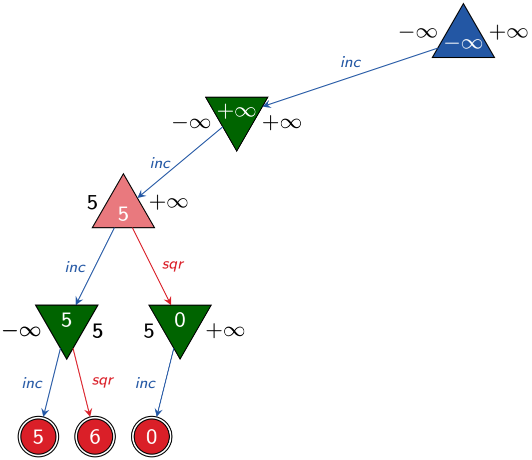
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



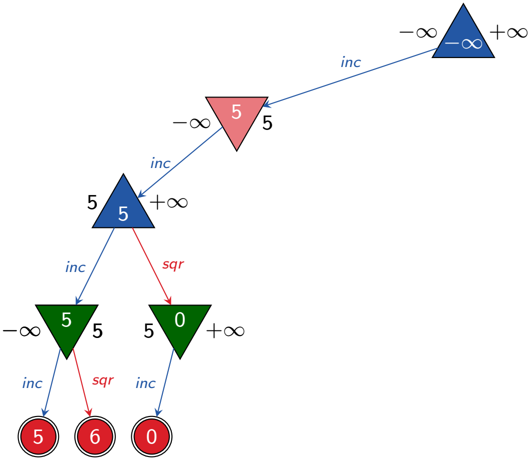
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



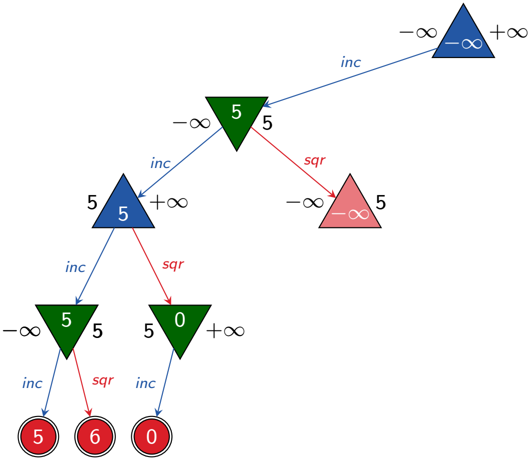
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



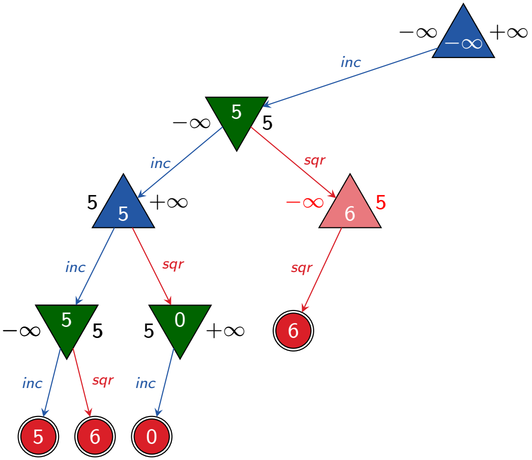
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



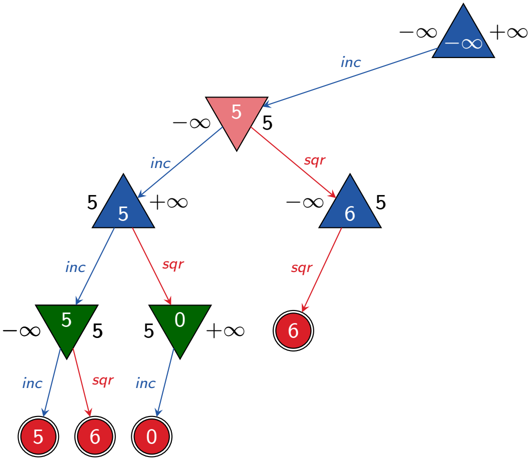
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



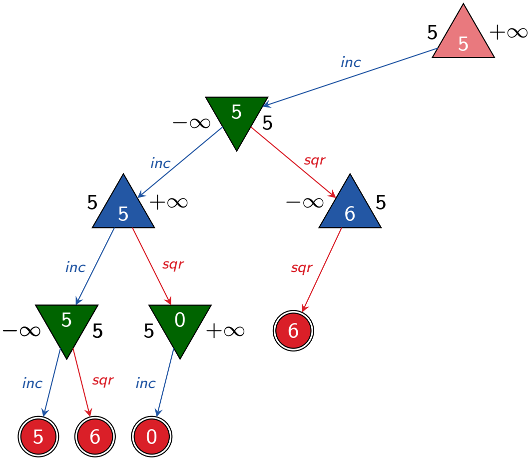
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



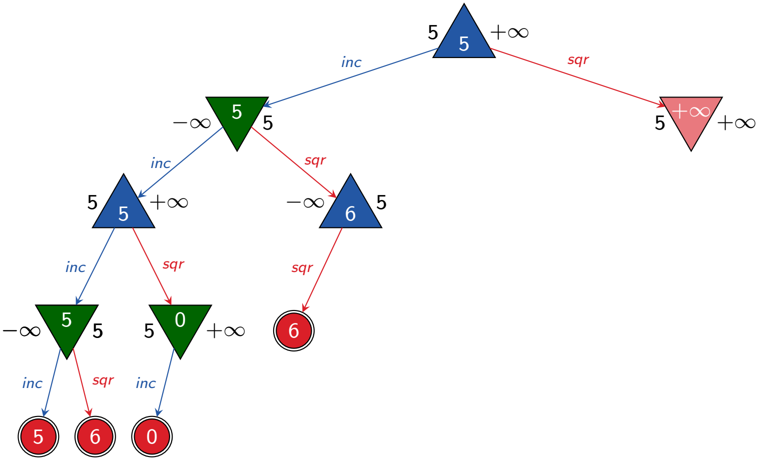
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



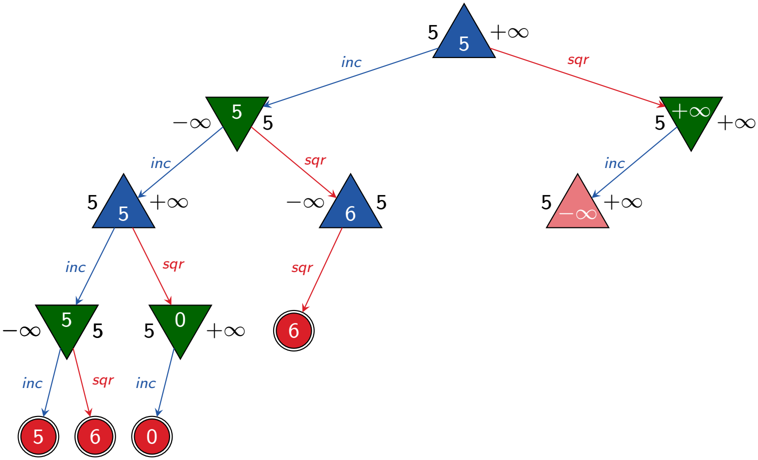
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



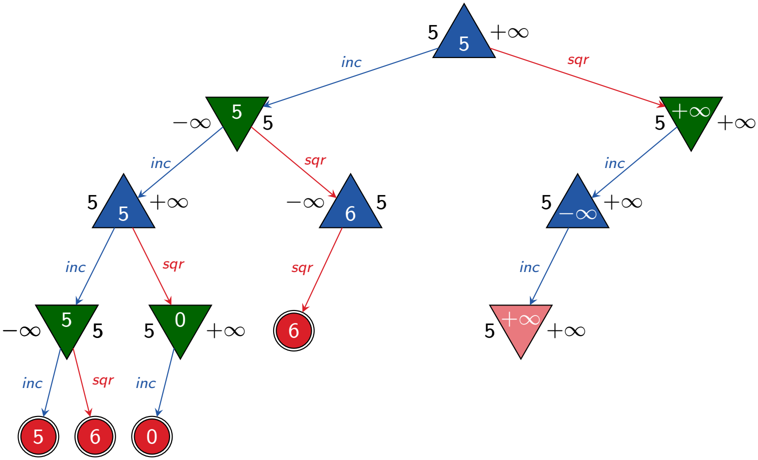
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



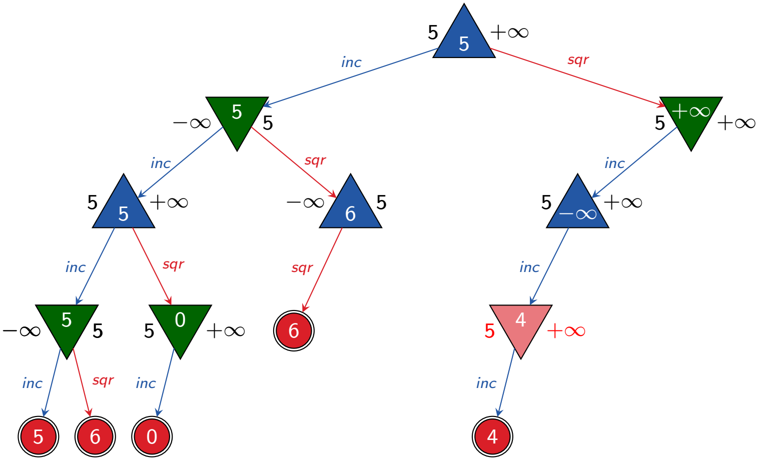
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



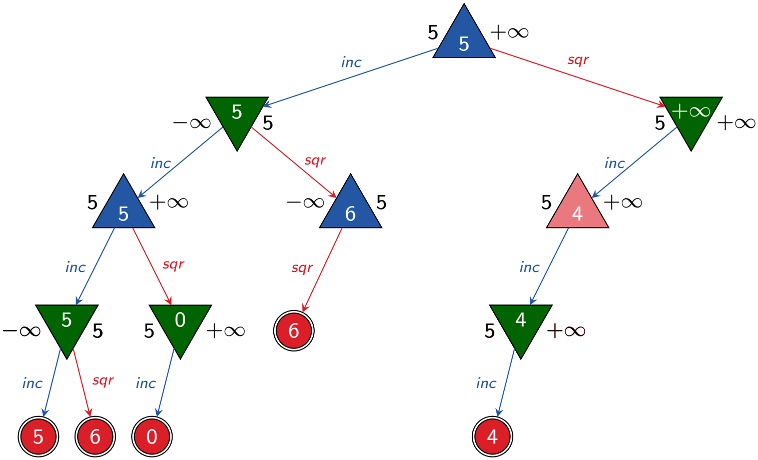
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



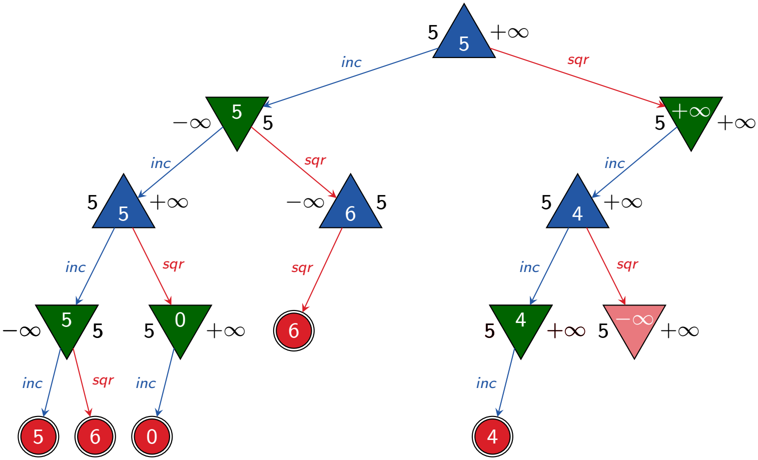
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



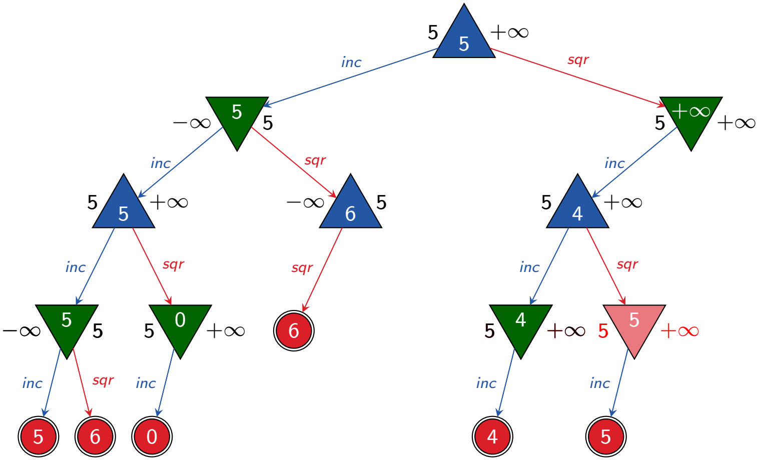
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



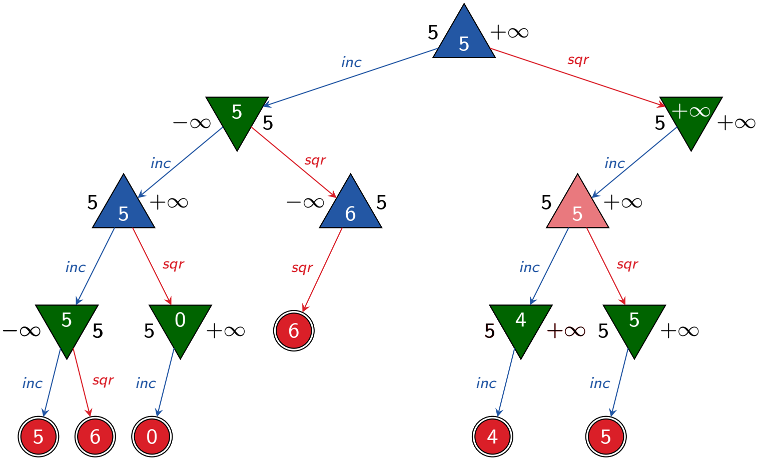
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



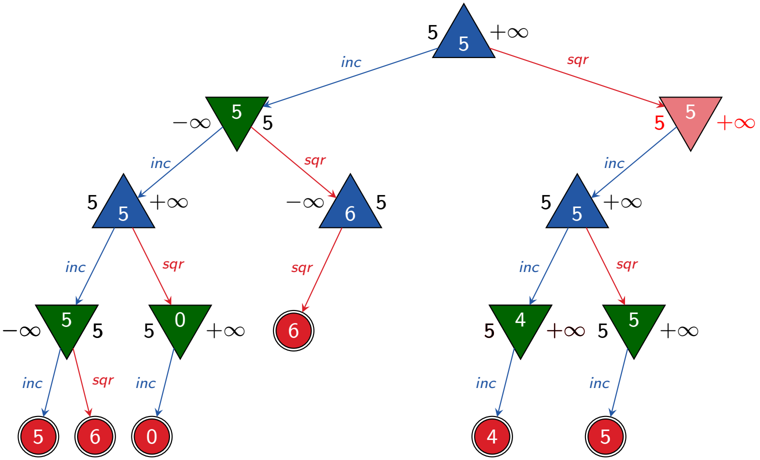
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



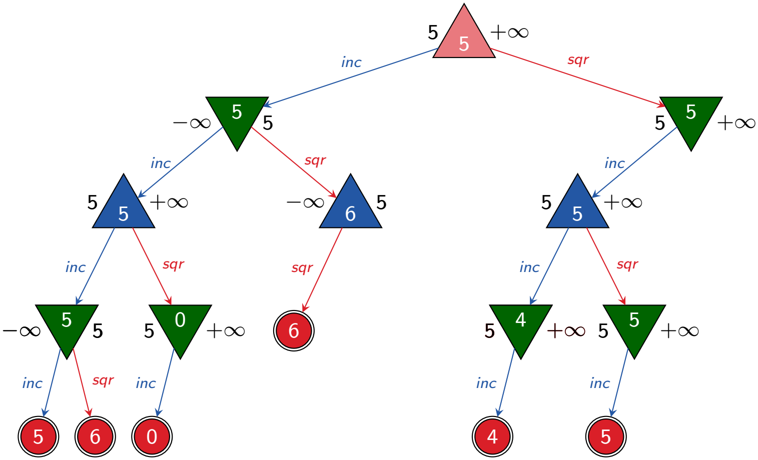
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Example



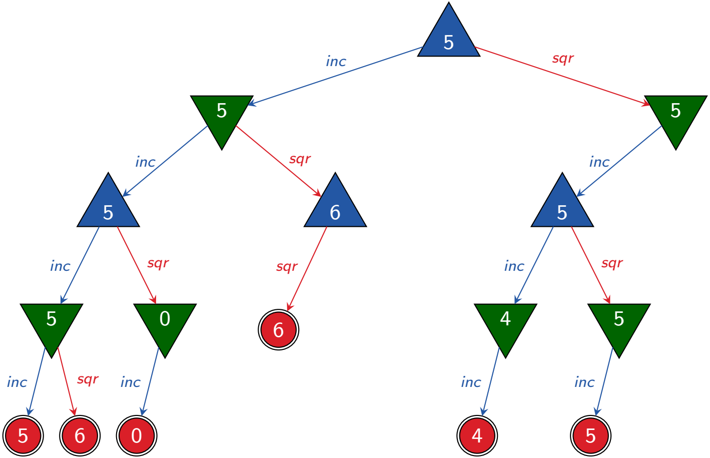
- α is lower bound on utility of *max*
- β is upper bound on utility of *min*
- a *max* subtree is pruned if utility $\geq \beta$
- a *min* subtree is pruned if utility $\leq \alpha$

Example



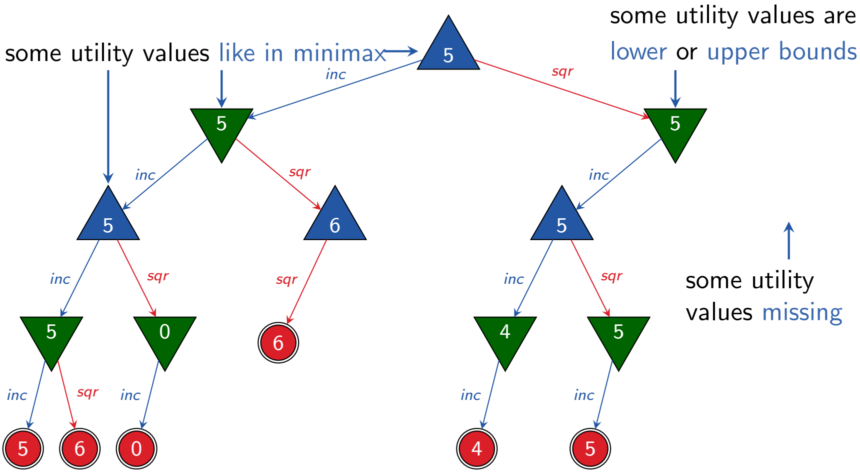
- α is lower bound on utility of *max*
- a *max* subtree is pruned if utility $\geq \beta$
- β is upper bound on utility of *min*
- a *min* subtree is pruned if utility $\leq \alpha$

Discussion



What do the utility values express?

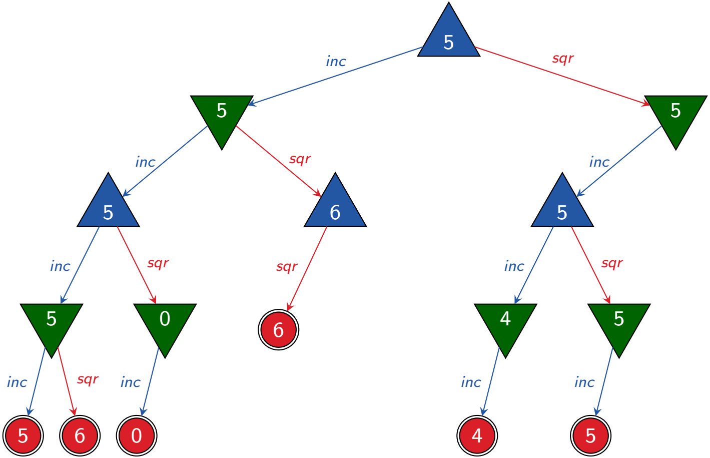
Discussion



What do the utility values express?

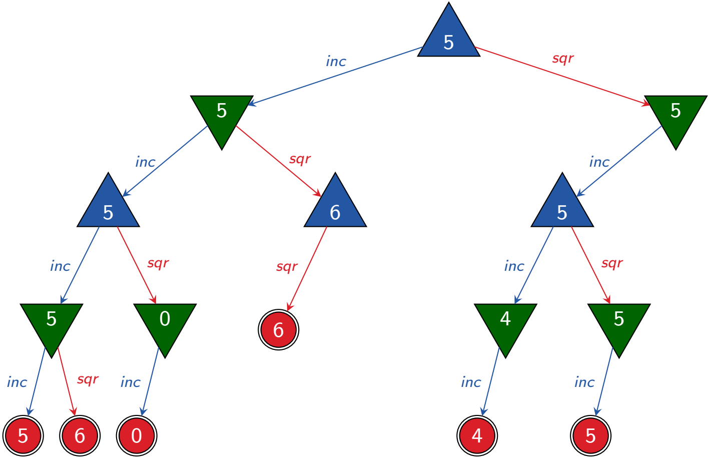
Discussion

What does this mean for the computed policy?



Discussion

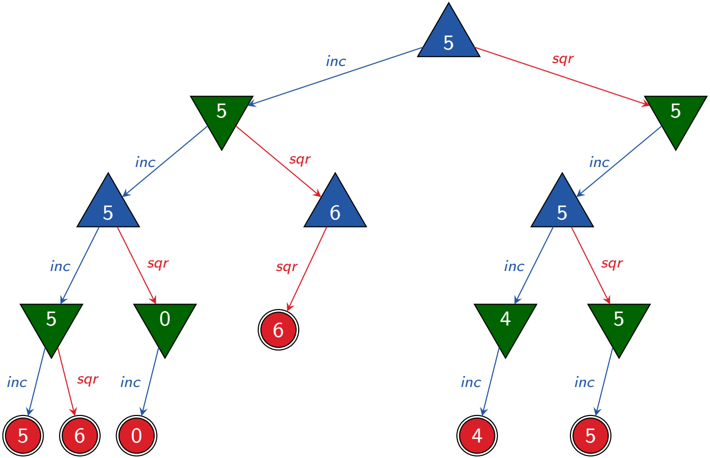
What does this mean for the computed policy?



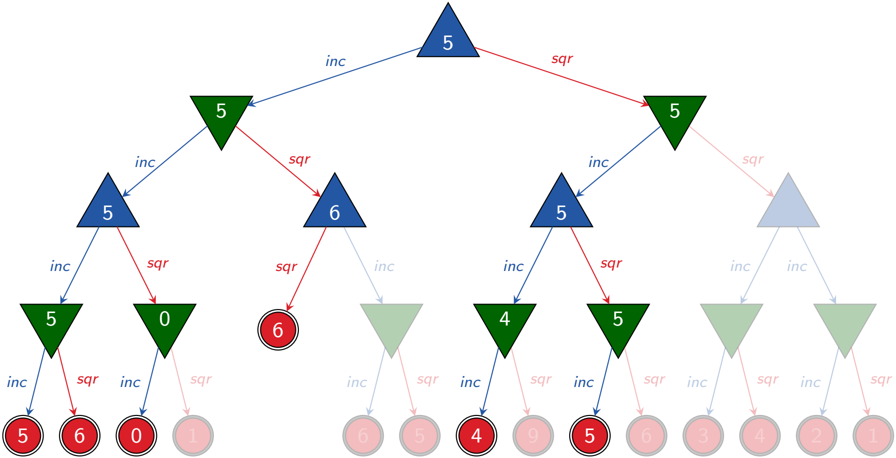
- only partial
- optimal in positions reachable under optimal play
- need to take earliest move in case of ties

Move Ordering

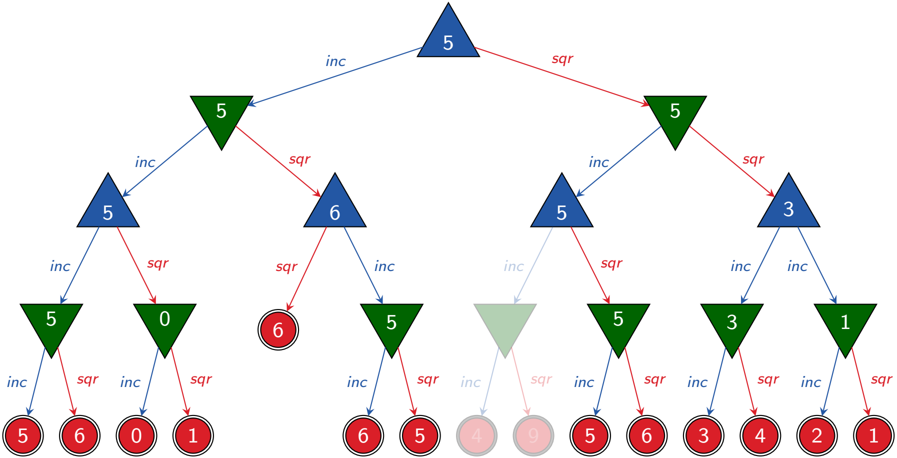
How Much Effort Did We Save?



How Much Effort Did We Save?



Have We Been Lucky?



if successors are considered in reverse order, we prune only a few positions

Move Ordering

idea: first consider the successors that are likely to be best

- domain-specific ordering function
  e.g. chess: captures $<$ threats $<$ forward moves $<$ backward moves
- dynamic move-ordering
  - first try moves that have been good in the past
  - e.g., in iterative deepening search:
    best moves from previous iteration

How Much Do We Gain with Alpha-Beta Pruning?

assumption: uniform game tree, depth d, branching factor $b \geq 2$; *max* and *min* positions alternating

- perfect move ordering
  - best move in every position is considered first (this cannot be done in practice)
  - effort reduced from $O(b^d)$ (minimax) to $O(b^{d/2})$
  - doubles the search depth that can be achieved in same time
- random move ordering
  - effort still reduced to $O(b^{3d/4})$ (for moderate b)

in practice, it is often possible to get close to the optimum

Heuristic Alpha-Beta Search

- combines evaluation function and alpha-beta search
- often uses additional techniques, e.g.
  - quiescence search
  - transposition tables
  - forward pruning
  - specialised sub-procedure for critical parts of the game (e.g., endgame database in chess)
  - ...

$\rightsquigarrow$ reaches expert level of play in chess

Summary

Summary

alpha-beta search

- stores which utility both players can force somewhere else in the game tree
- exploits this information to avoid unnecessary computations
- can have significantly lower search effort than minimax
- best case: search twice as deep in the same time