

Foundations of Artificial Intelligence

G3. Board Games: Alpha-Beta Search

Malte Helmert

University of Basel

May 19, 2025

Foundations of Artificial Intelligence

May 19, 2025 — G3. Board Games: Alpha-Beta Search

G3.1 Alpha-Beta Search

G3.2 Move Ordering

G3.3 Summary

Board Games: Overview

chapter overview:

- ▶ G1. Introduction and State of the Art
- ▶ G2. Minimax Search and Evaluation Functions
- ▶ G3. Alpha-Beta Search
- ▶ G4. Stochastic Games
- ▶ G5. Monte-Carlo Tree Search Framework
- ▶ G6. Monte-Carlo Tree Search Configurations

Limitations of Minimax



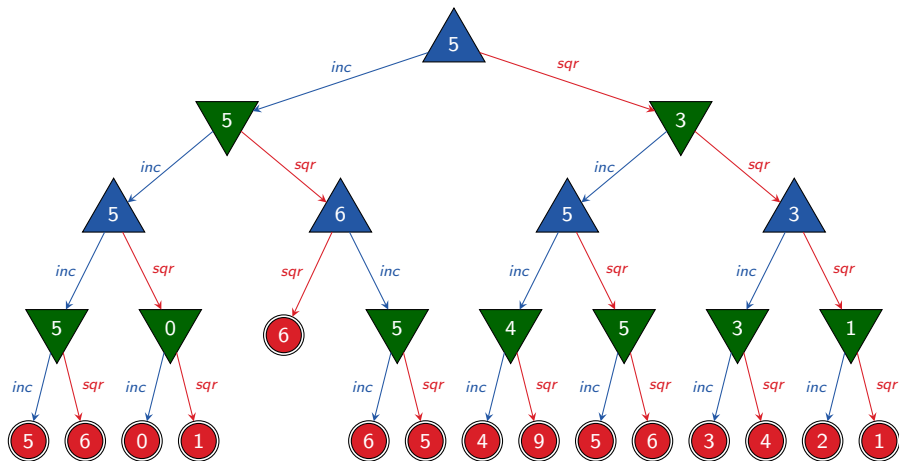
What if the size of the game tree is **too big for minimax**?

~> **heuristic alpha-beta search**

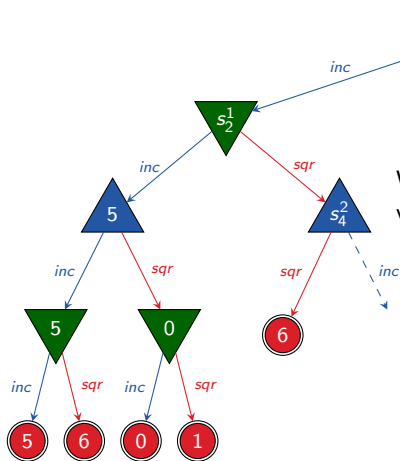
- ▶ heuristics (evaluation functions): [previous chapter](#)
- ▶ alpha-beta search: [this chapter](#)

G3.1 Alpha-Beta Search

Can We Save Search Effort?



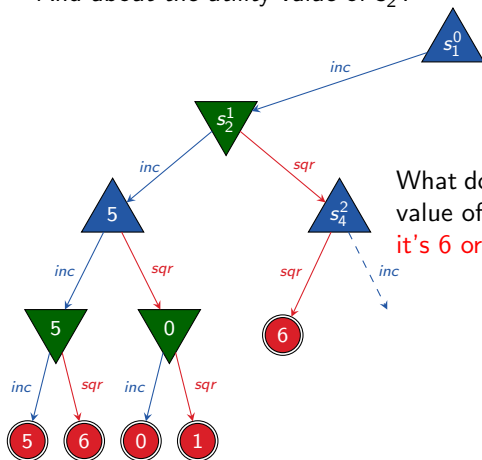
Can We Save Search Effort?



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 ?



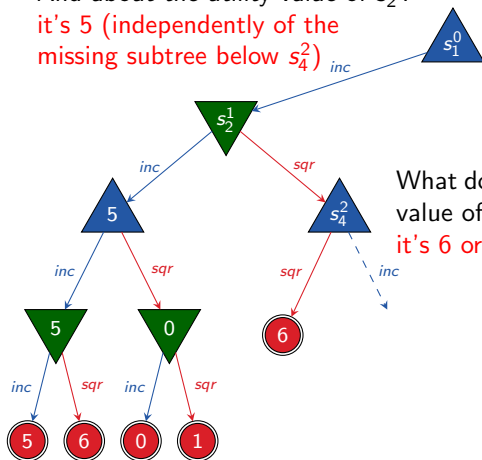
What do we know about the utility value of s_4^2 in this situation?

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)



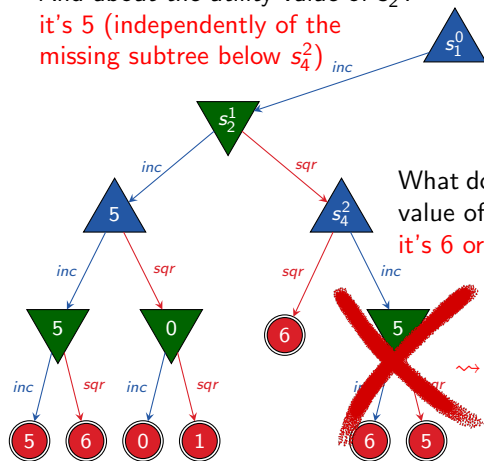
What do we know about the utility value of s_4^2 in this situation?

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

we don't have to look at this

Idea

idea: for every search node, use two values α and β such that we know that the subtree rooted at the node

- ▶ **is irrelevant** if its utility is $\leq \alpha$
because MAX will prevent entering it when playing optimally
- ▶ **is irrelevant** if its utility is $\geq \beta$
because MIN will prevent entering it when playing optimally

We can **prune** every node with $\alpha \geq \beta$
because it must be irrelevant (no matter what its utility is).

Alpha-Beta Search: Pseudo Code

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

```
function alpha-beta-main( $p$ )  
   $\langle v, move \rangle := \text{alpha-beta}(p, -\infty, +\infty)$   
  return  $move$ 
```

Alpha-Beta Search: Pseudo-Code

function $\text{alpha-beta}(p, \alpha, \beta)$

if p is terminal position:

return $\langle \text{utility}(p), \text{none} \rangle$

initialize v and best_move

[as in minimax]

for each $\langle \text{move}, p' \rangle \in \text{succ}(p)$:

$\langle v', \text{best_move}' \rangle := \text{alpha-beta}(p', \alpha, \beta)$

 update v and best_move

[as in minimax]

if $\text{player}(p) = \text{MAX}$:

if $v \geq \beta$:

return $\langle v, \text{none} \rangle$

$\alpha := \max\{\alpha, v\}$

if $\text{player}(p) = \text{MIN}$:

if $v \leq \alpha$:

return $\langle v, \text{none} \rangle$

$\beta := \min\{\beta, v\}$

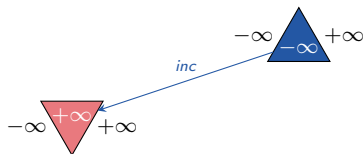
return $\langle v, \text{best_move} \rangle$

Example



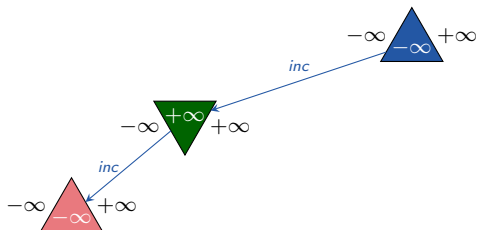
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



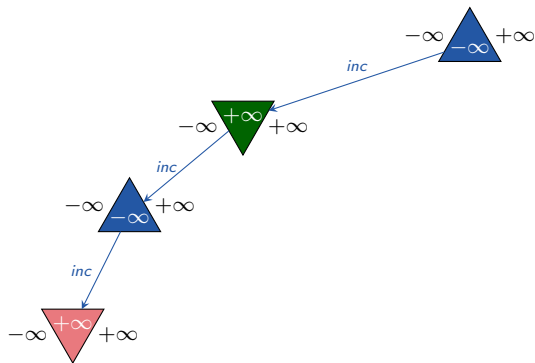
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



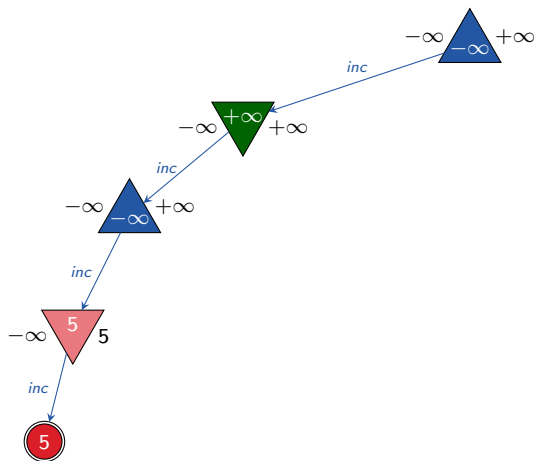
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



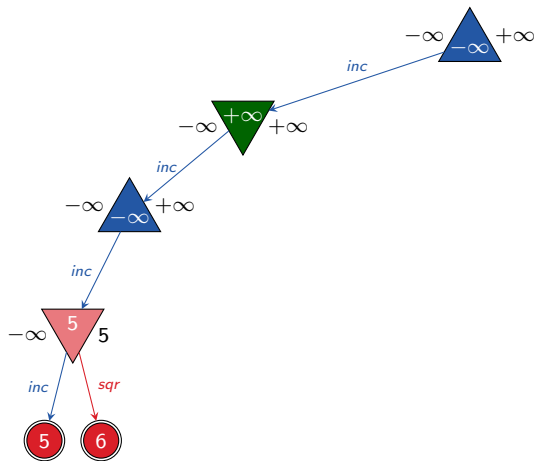
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



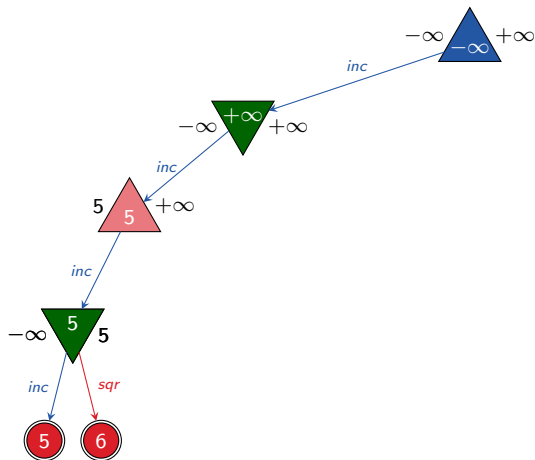
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



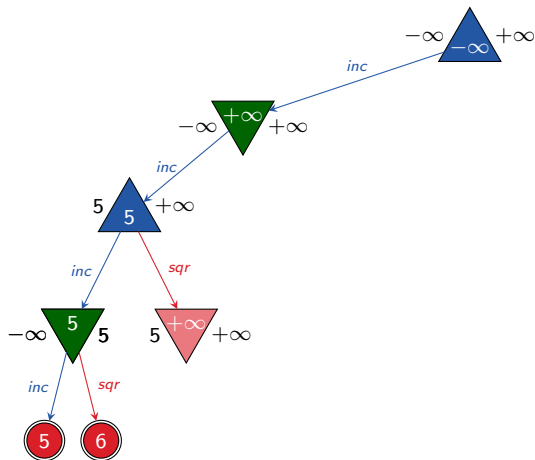
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



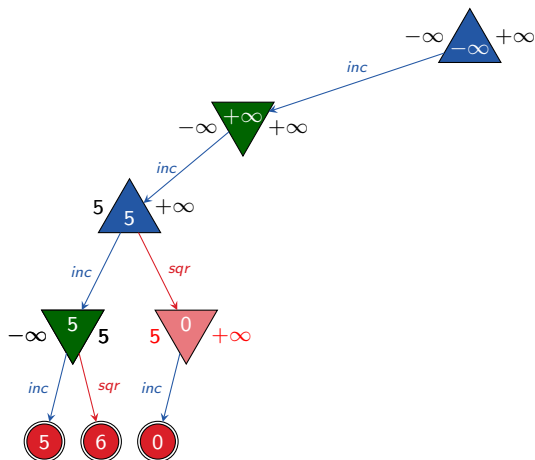
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



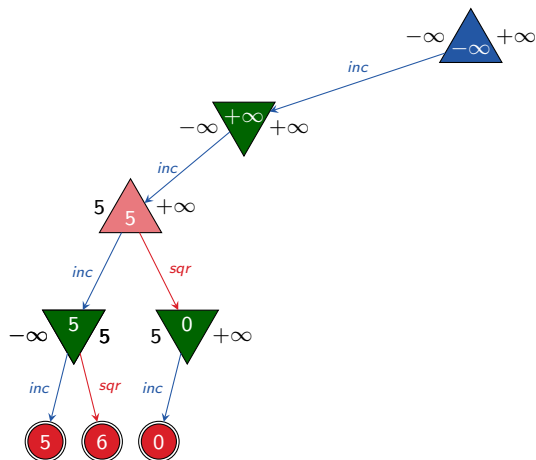
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



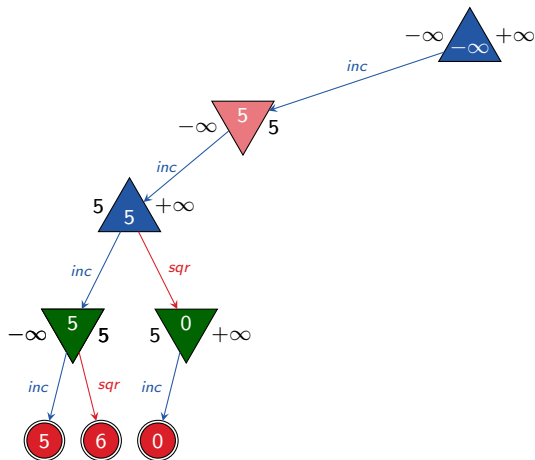
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



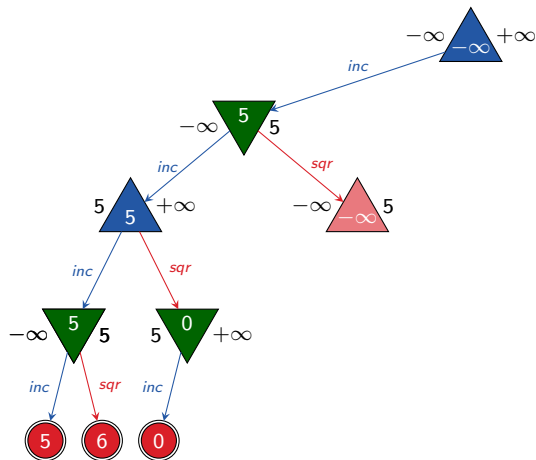
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



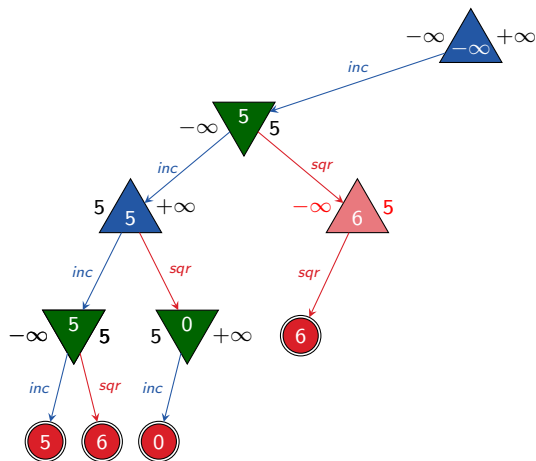
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



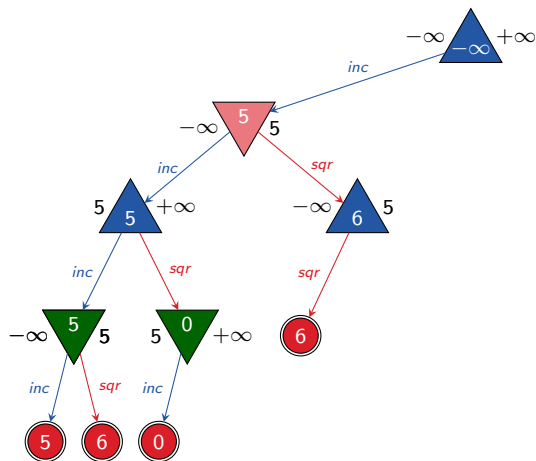
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



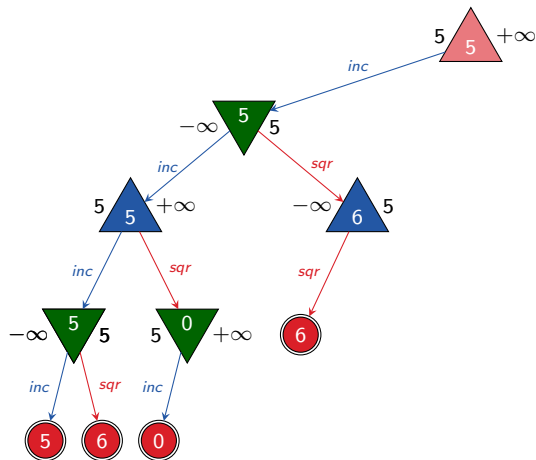
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



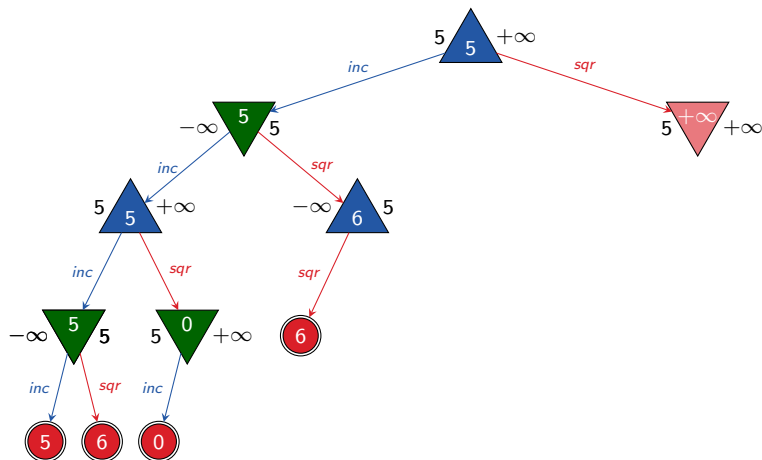
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



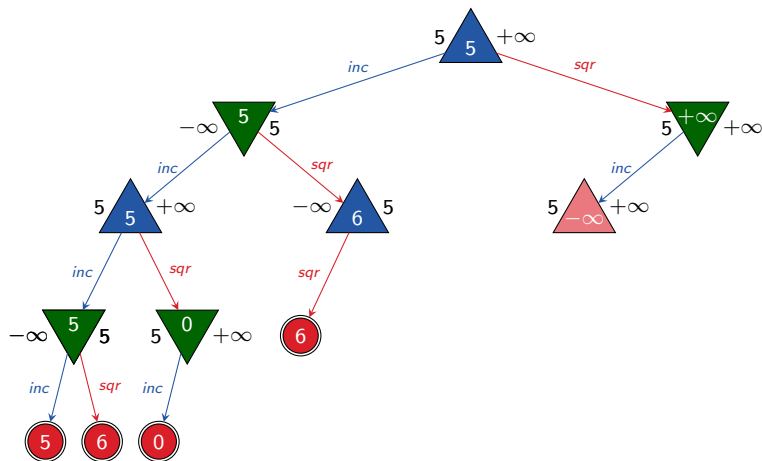
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



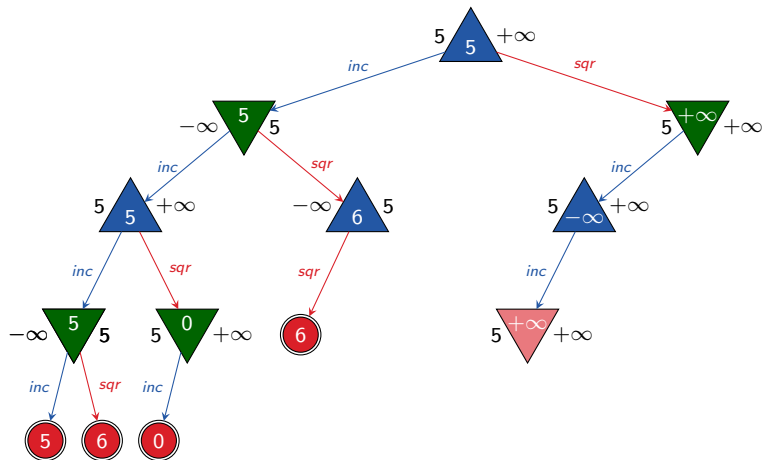
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



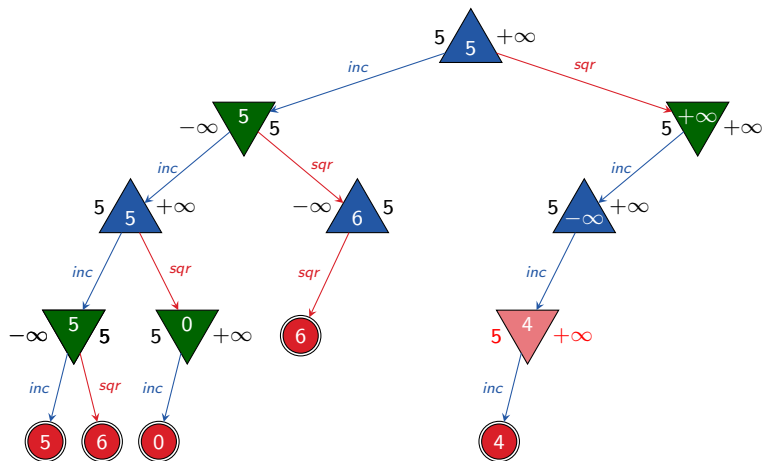
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



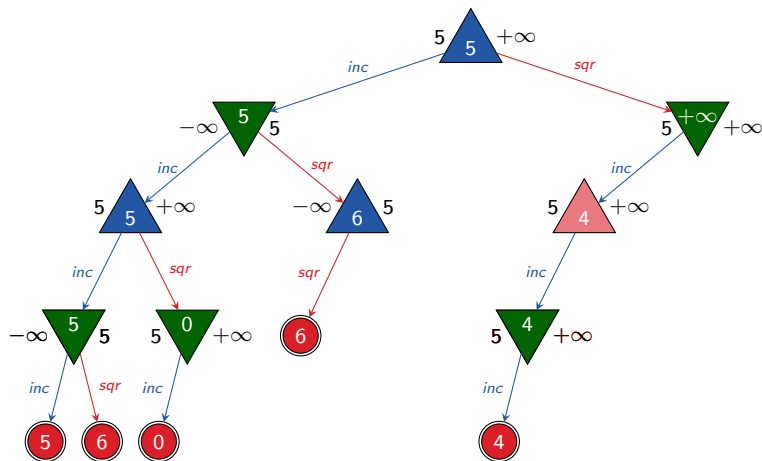
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



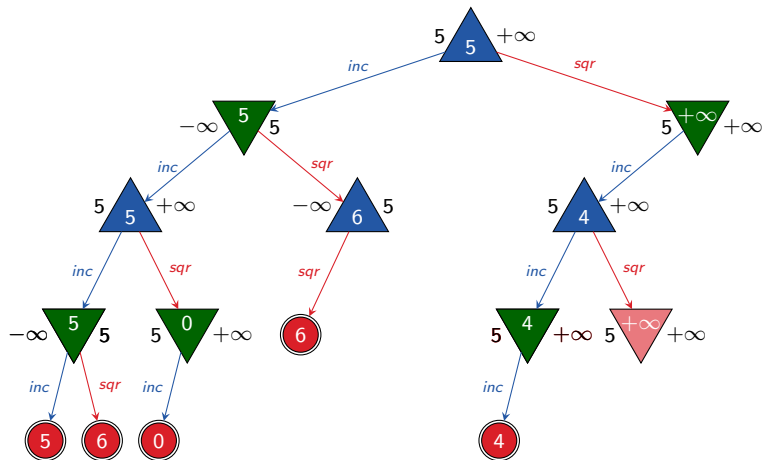
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



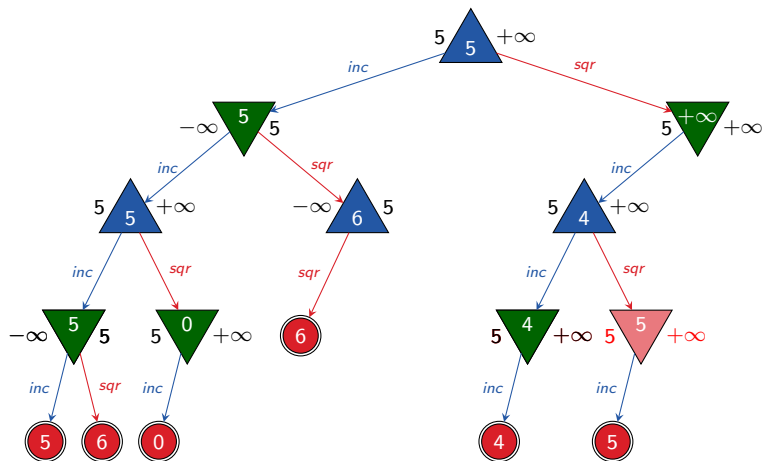
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



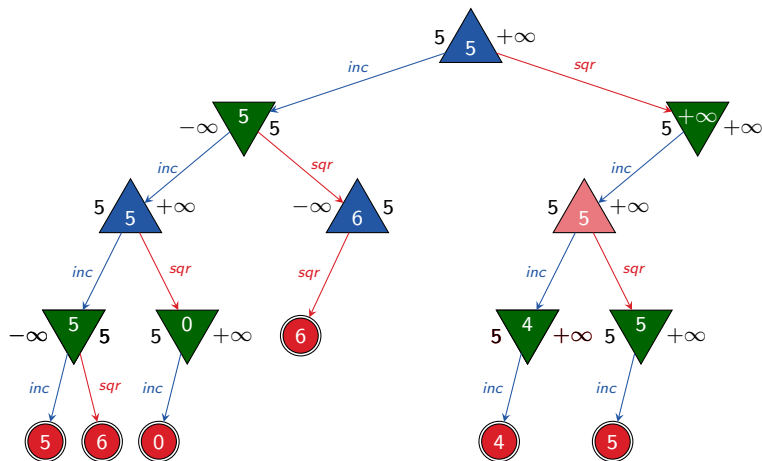
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



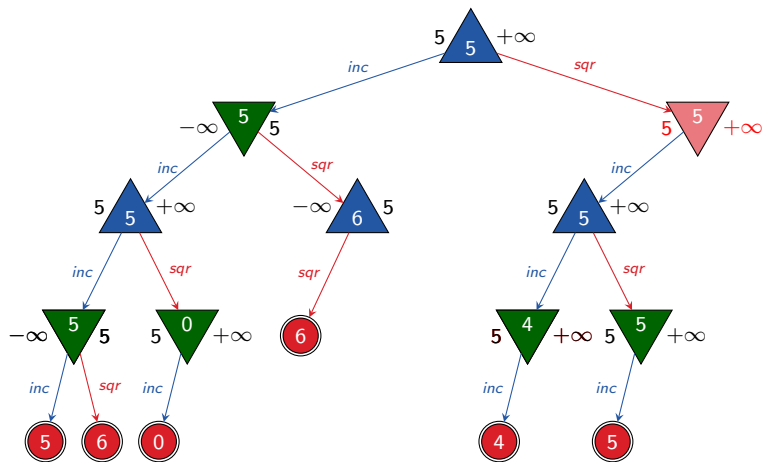
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



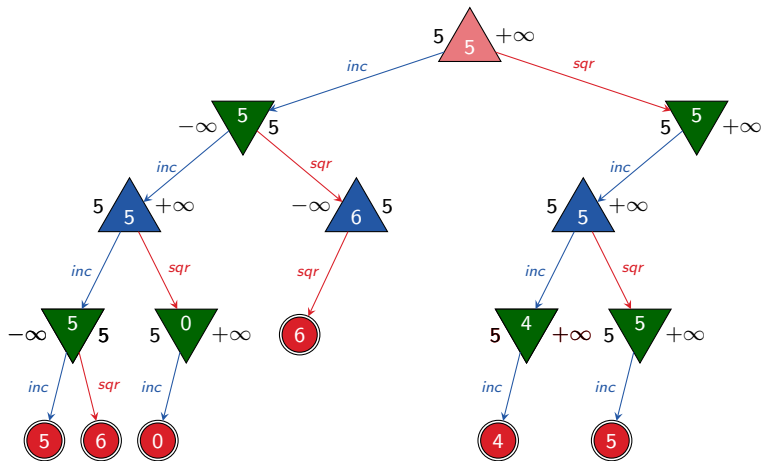
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



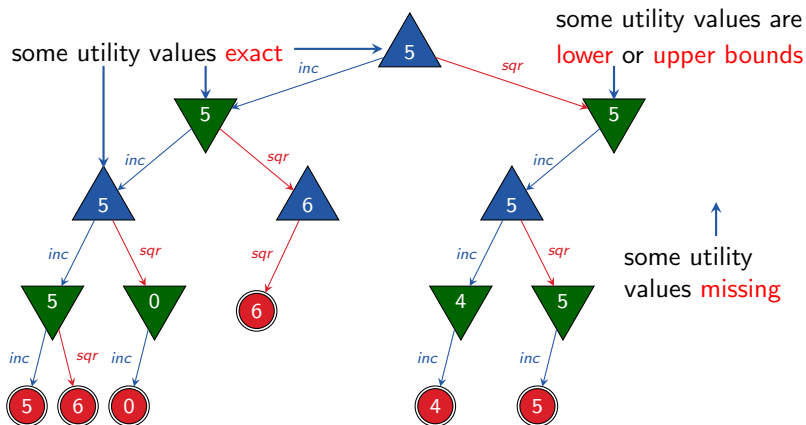
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Example



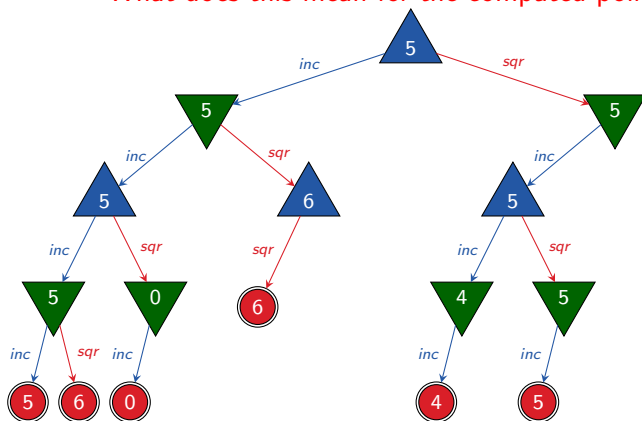
- ▶ α : lower bound of relevant utility
- ▶ β : upper bound of relevant utility
- ▶ a MAX subtree is pruned if utility $\geq \beta$
- ▶ a MIN subtree is pruned if utility $\leq \alpha$

Discussion



Discussion

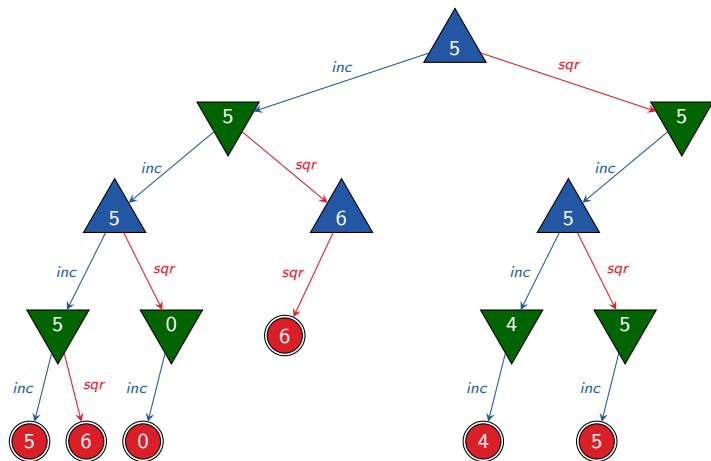
What does this mean for the computed policy?



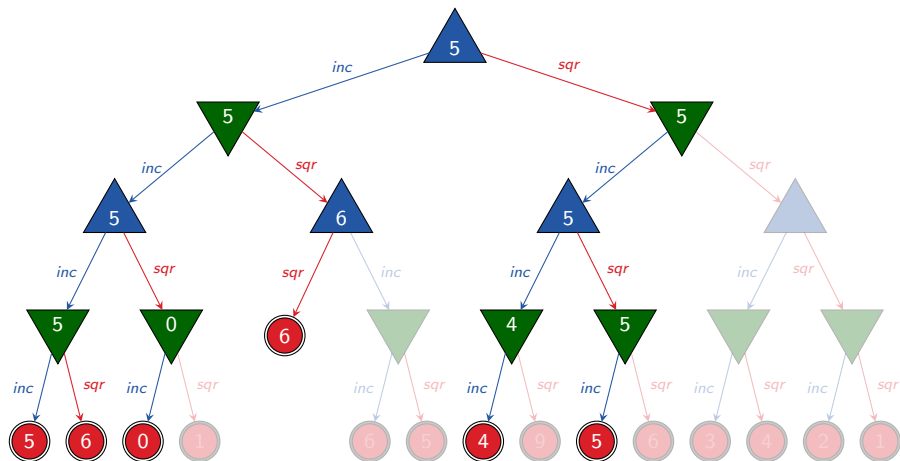
- ▶ **only partial** ▶ **optimal** in positions reachable under optimal play
- ▶ need to take **earliest move** in case of ties

G3.2 Move Ordering

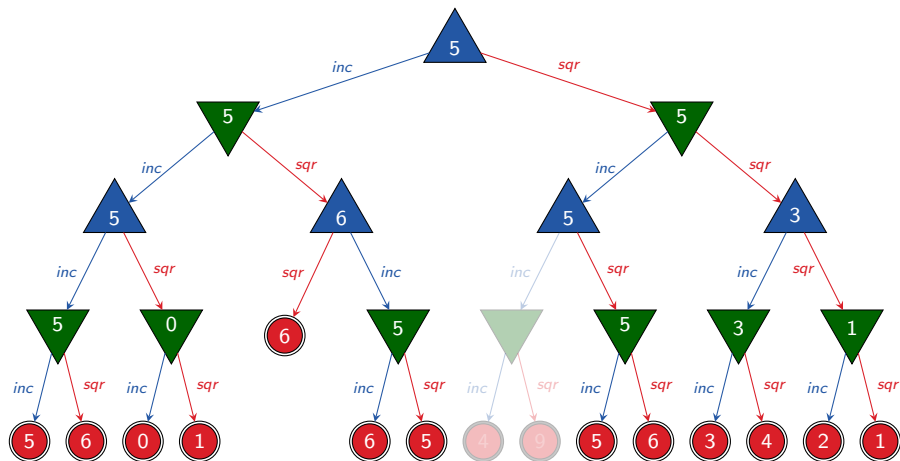
How Much Effort Do We Save?



How Much Effort Do We Save?



Were We 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 were 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 alternate

- ▶ **perfect move ordering**
 - ▶ best move at every position is considered first
 - ▶ maximizing move for MAX, minimizing move for MIN
 - ▶ 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 $O(b^{3d/4})$

In practice, we can often get close to the perfect move ordering.

Heuristic Alpha-Beta Search

- ▶ combines **evaluation function** and **alpha-beta search**
- ▶ often uses additional techniques, e.g.
 - ▶ quiescence search
 - ▶ transposition tables
 - ▶ forward pruning
 - ▶ specialized subprocedures for certain parts of the game (e.g., opening libraries and endgame databases)
 - ▶ ...

G3.3 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