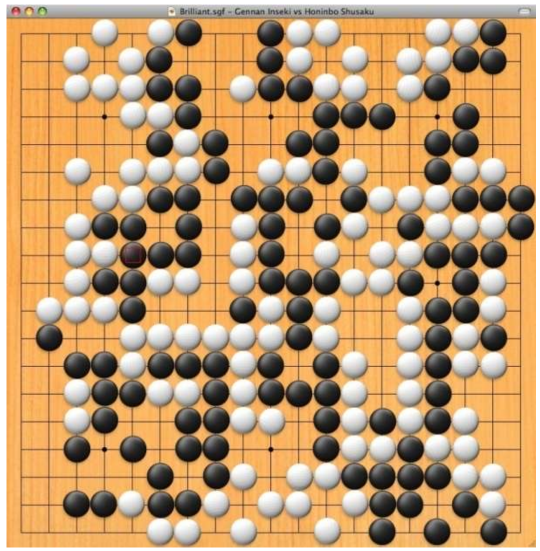
Artificial Intelligence: State Space Search for Game Playing

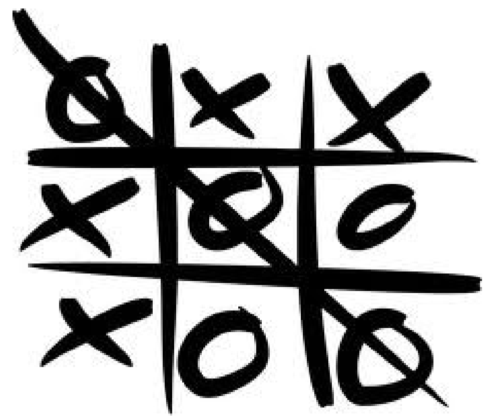
- Russell & Norvig: Sections 5.1 & 5.4

Motivation



GO

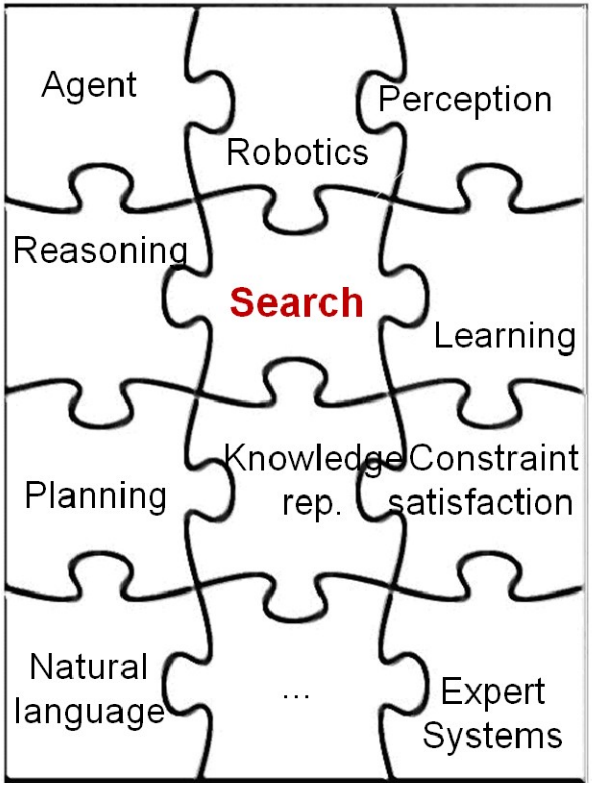
chess



tic-tac-toe

Today

- State Space Search for Game Playing
  - MiniMax
  - Alpha-beta pruning
- Where we are today



State Space Search for Game Playing

- Classical application for heuristic search
  - simple games: exhaustibly searchable
  - complex games: only partial search possible
  - additional problem: playing against opponent
- Type of game:
  - 2 person adversarial games
    - win, lose or tie
  - Perfect information
    - both players know the state of the game and all possible moves
  - No chance involved
    - outcome of the game is only dependent on player's moves
  - zero-sum game
    - If the total gains of one player are added up, and the total losses are subtracted, they will sum to zero.
    - a gain by one player must be matched by a loss by the other player
  - ex. chess, GO, tic-tac-toe, …

Today

- State Spa rch for Game Playing
  - MiniMax
  - Alpha-beta pruning
- Where we are today

MiniMax Search

- Game between two opponents, MIN and MAX
  - MAX tries to win, and
  - MIN tries to minimize MAX's score
- Existing heuristic search methods do not work
  - would require a helpful opponent
  - Need to incorporate "hostile" moves into search strategy

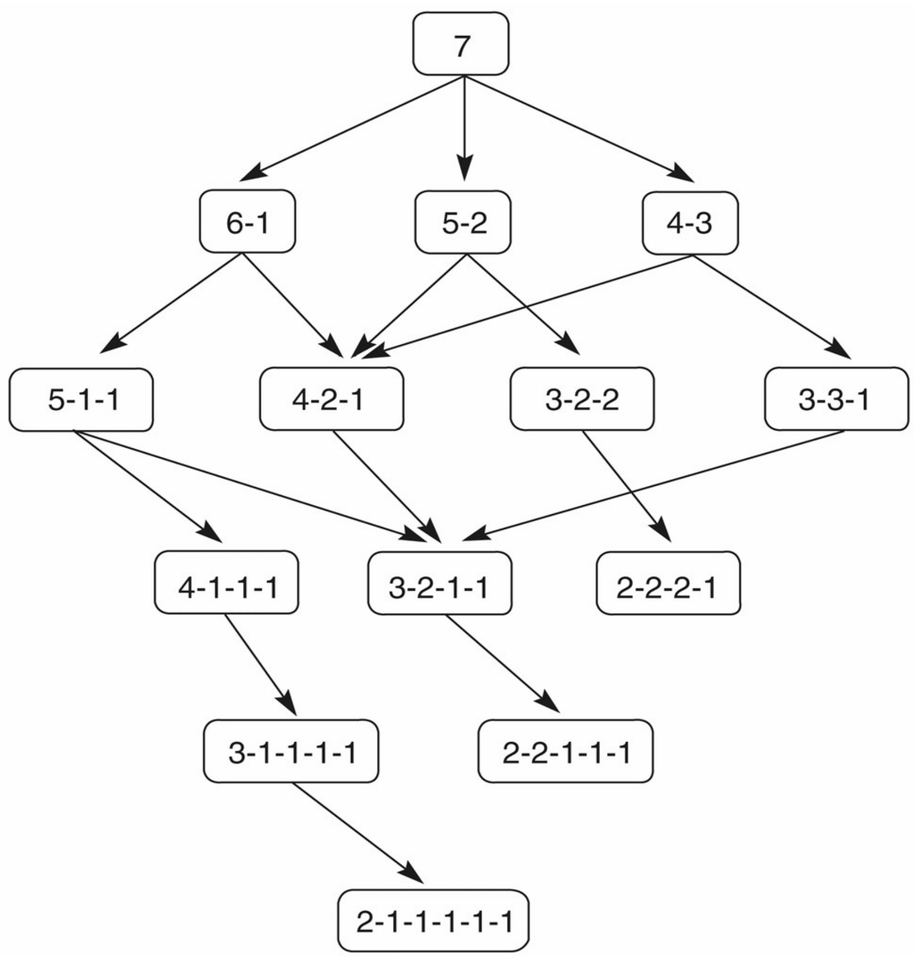
Exhaustive MiniMax Search

- For small games where exhaustive search is feasible
- Procedure:
  1. build complete game tree
  2. label each level according to player's turn (MAX or MIN)
  3. label leaves with a utility function to determine the outcome of the game
     - e.g., (0, 1) or (-1, 0, 1)
  4. propagate this value up:
     - if parent=MAX, give it max value of children
     - if parent=MIN, give it min value of children
  5. Select best next move for player at root as the move leading to the child with the highest value (for MAX) or lowest values (for MIN)

Example: Game of Nim

- Rules
  - 2 players start with a pile of tokens
  - move: split (any) existing pile into two non-empty differently-sized piles
  - game ends when no pile can be unevenly split
  - player who cannot make his move loses

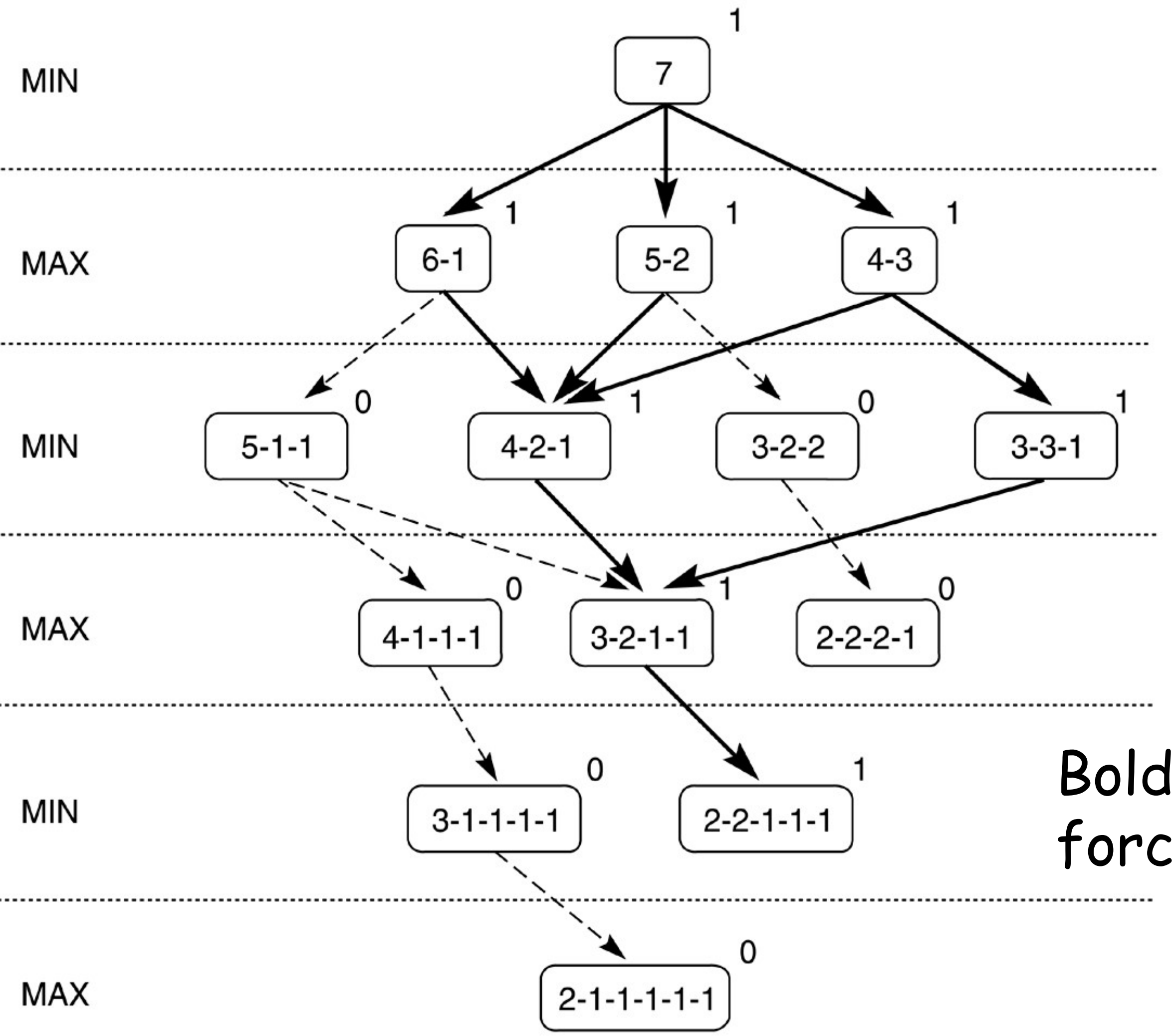
State Space of Game Nim



- start with one pile of tokens
- each step has to divide one pile of tokens into 2 non-empty piles of different size
- player without a move left loses game

source: *G. Luger (2005)*

Exhaustive MiniMax for Nim



Bold lines indicate forced win for MAX

source: G. Luger (2005)

n-ply MiniMax with Heuristic

- Exhaustive search for interesting games is rarely feasible
- Search only to predefined level
  - called n-ply look-ahead
  - n is number of levels
- No exhaustive search
  - nodes evaluated with heuristics and not win/loss
  - indicates best state that can be reached
  - horizon effect
- Games with opponent
  - simple strategy: try to maximize difference between players using a heuristic function *e(n)*

Heuristic Function for 2-player games

- simple strategy:
  - try to maximize difference between MAX's game and MIN's game
- typically called *e(n)*
- *e(n)* is a heuristic that estimates how favorable a node n is for MAX
  - *e(n)* > 0 --> n is favorable to MAX
  - *e(n)* < 0 --> n is favorable to MIN
  - *e(n)* = 0 --> n is neutral

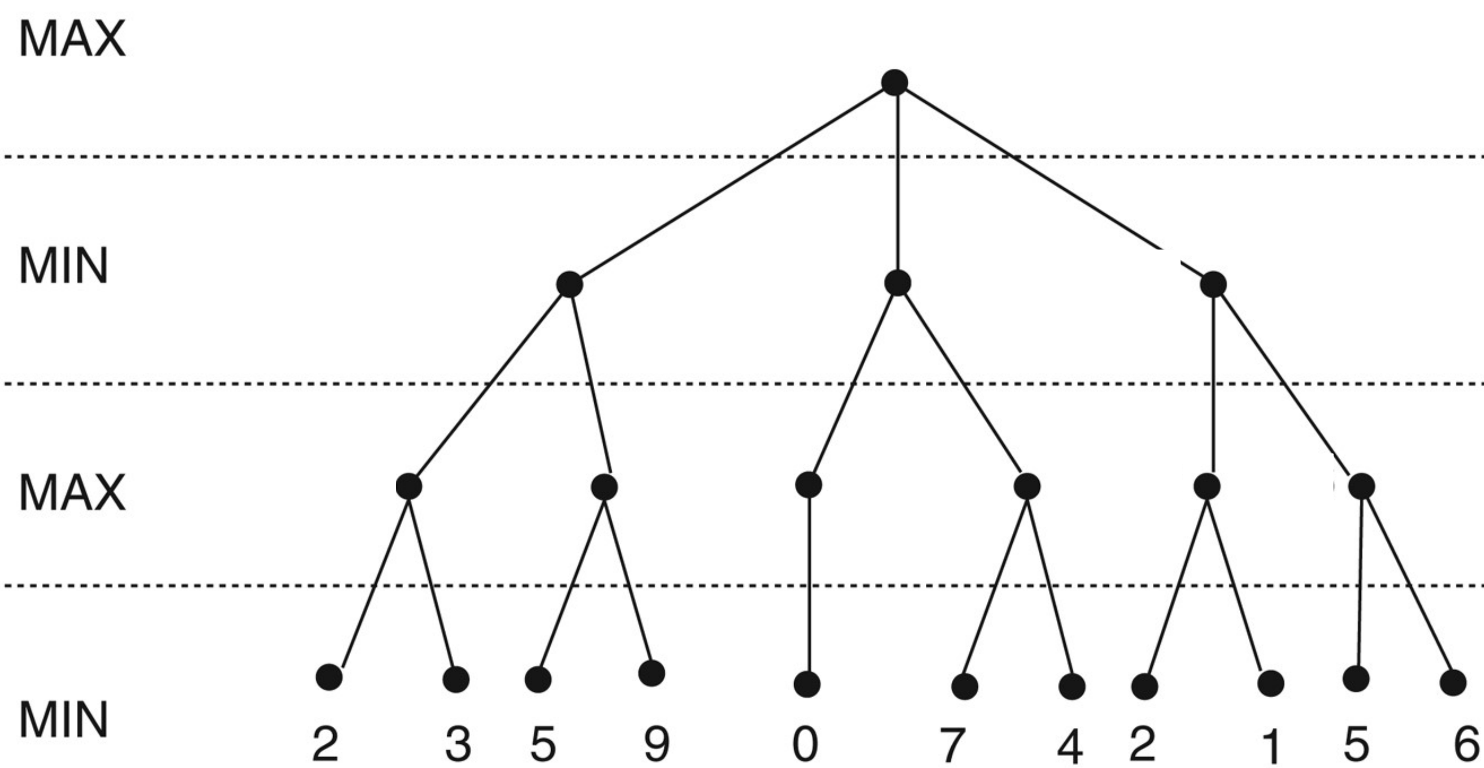
Choosing a Heuristic Function e(n)

- Usually e(n) is a weighted sum of various features:

$$e(n) = \sum w_i \, f_i(n)$$

- E.g. of features:
  - f_1 = number of pieces left on the game for MAX
  - f_2 = number of possible moves left for MAX
  - f_3 = -(number of pieces left on the game for MIN)
  - f_4 = -(number of possible moves left for MIN)
- E.g. of weights:
  - w_1 = 0.5 // f_1 is a very important feature
  - w_2 = 0.2 // f_2 is not very important
  - w_3 = 0.2 // f_3 is not very important
  - w_4 = 0.1 // f_4 is really not important

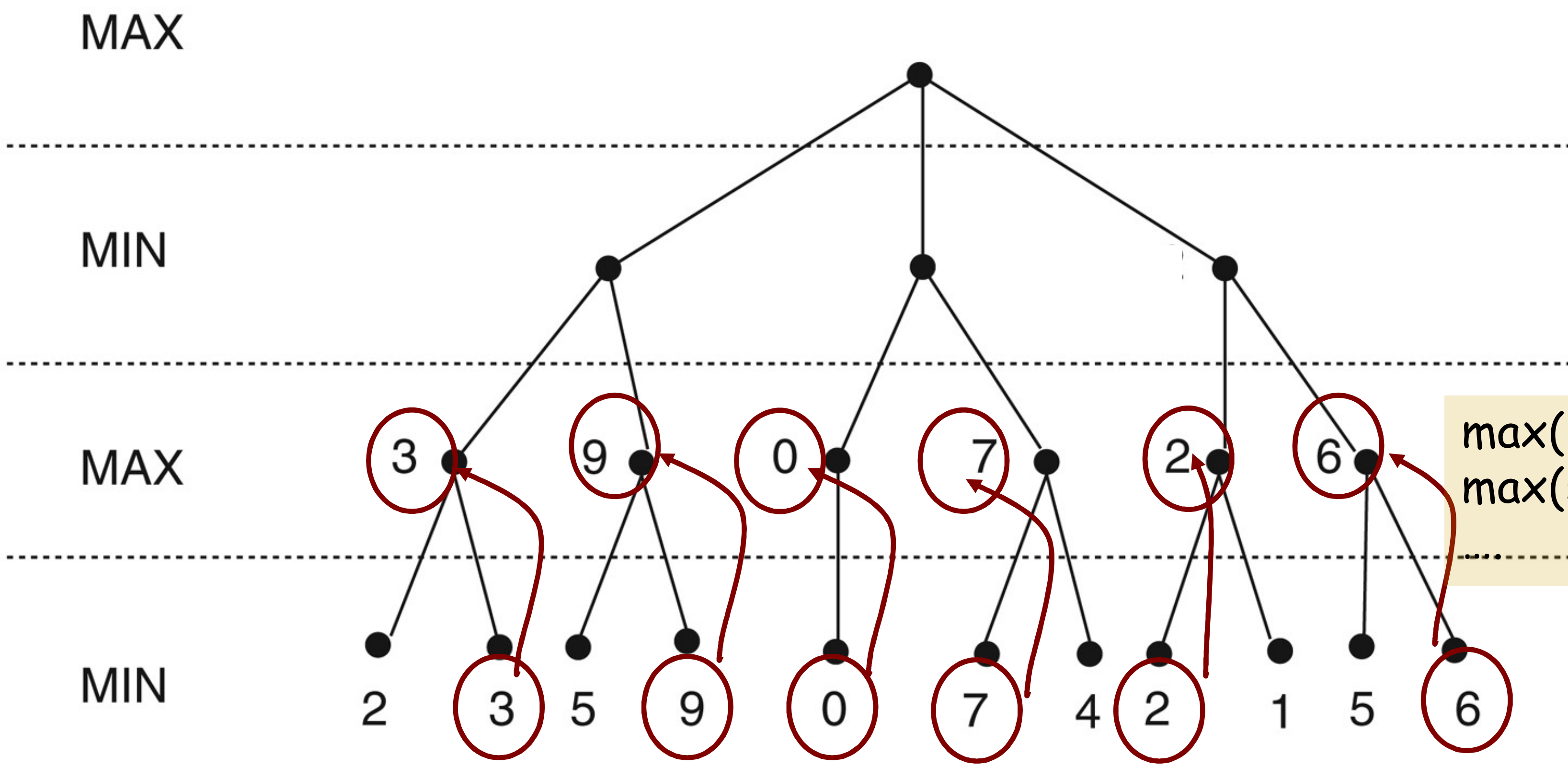
MiniMax with Fixed Ply Depth



Leaf nodes show the actual heuristic value *e(n)*

source: G. Luger (2005)

MiniMax with Fixed Ply Depth

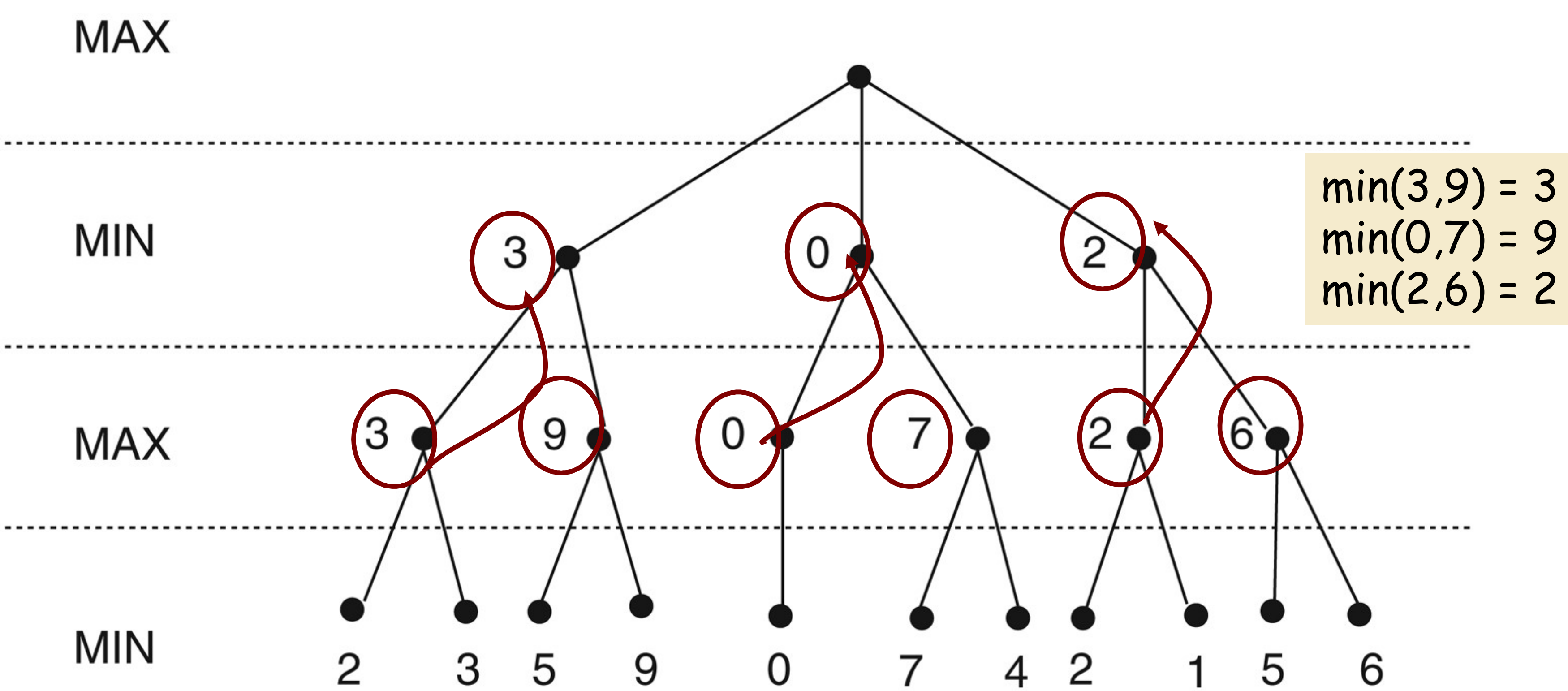


max(2,3) = 3
max(5,9) = 9
....

Leaf nodes show the actual heuristic value *e(n)*
Internal nodes show back-up heuristic value

source: G. Luger (2005)

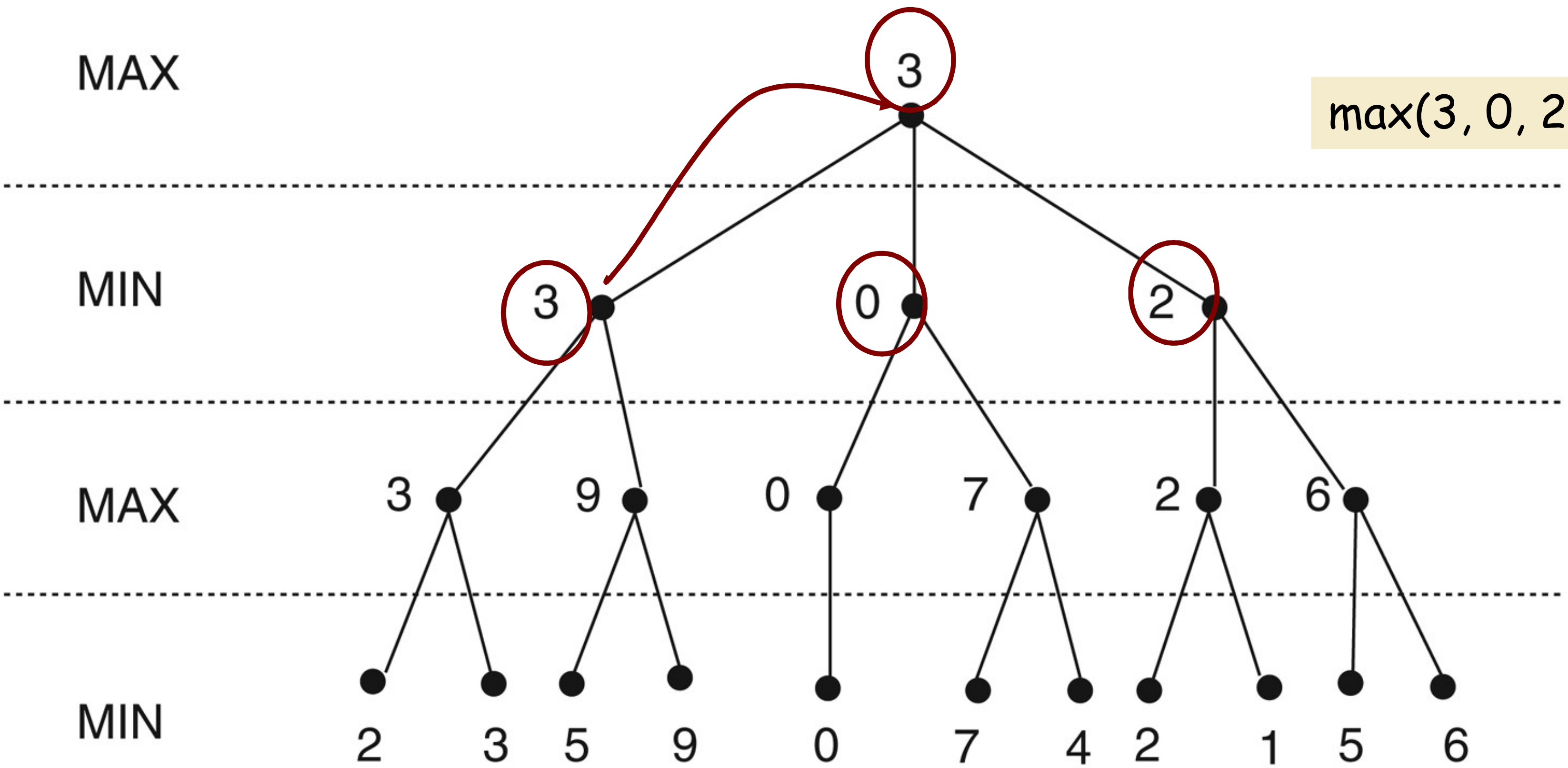
MiniMax with Fixed Ply Depth



Leaf nodes show the actual heuristic value *e(n)*
Internal nodes show back-up heuristic value

source: G. Luger (2005)

MiniMax with Fixed Ply Depth

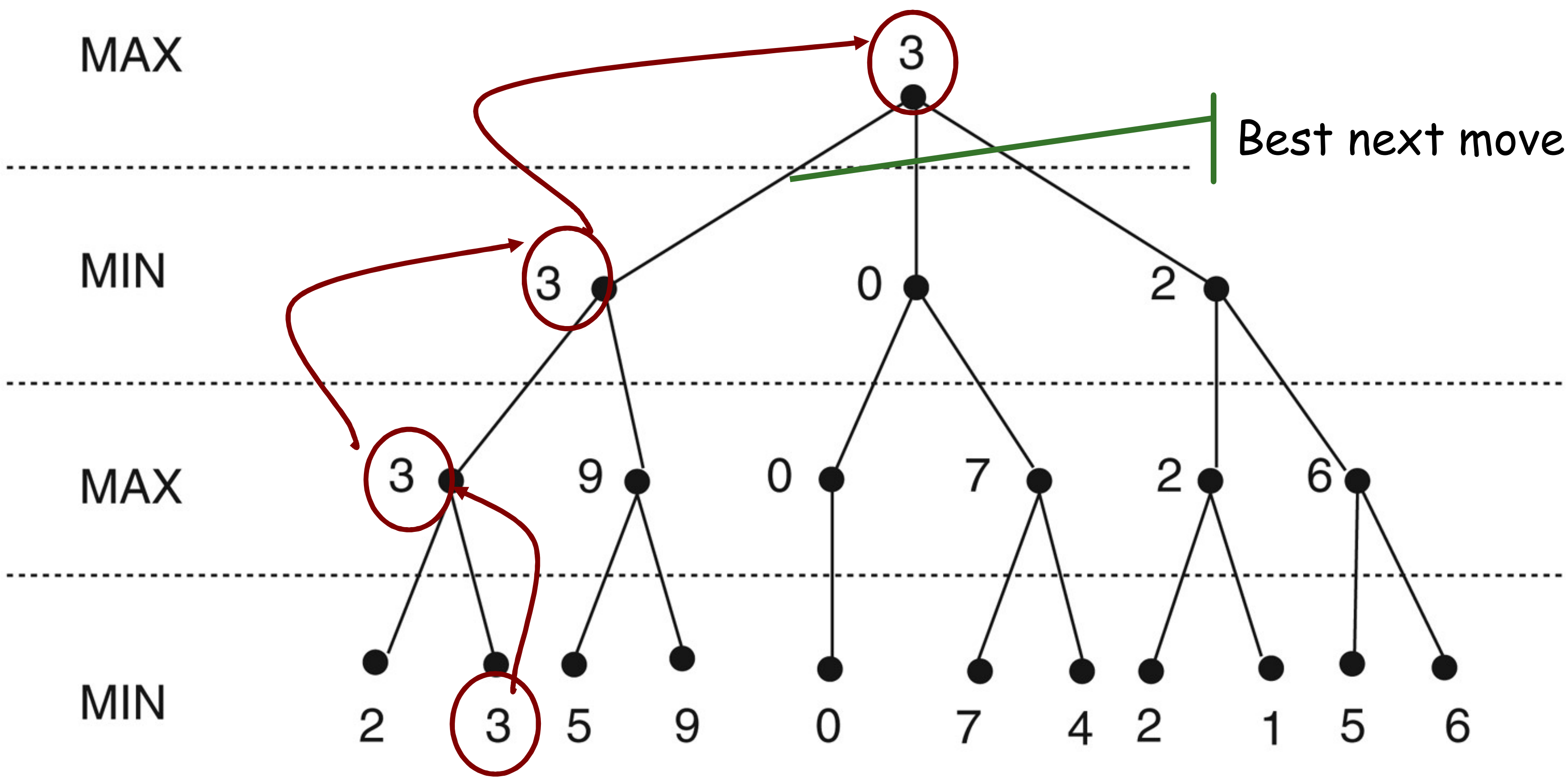


max(3, 0, 2) = 3

Leaf nodes show the actual heuristic value e(n)
Internal nodes show back-up heuristic value

source: G. Luger (2005)

MiniMax with Fixed Ply Depth



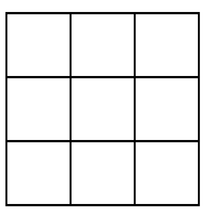
Leaf nodes show the actual heuristic value *e(n)*
Internal nodes show back-up heuristic value

source: G. Luger (2005)

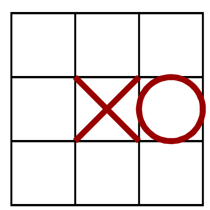
Example: $e(n)$ for Tic-Tac-Toe

- Possible $e(n)$

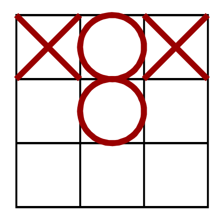
$$e(n) = \begin{cases} \text{number of rows, columns, and diagonals open for MAX} \\ \quad\quad \text{- number of rows, columns, and diagonals open for MIN} \\ +\infty\text{, if n is a forced win for MAX} \\ -\infty\text{, if n is a forced win for MIN} \end{cases}$$



$e(n) = 8–8 = 0$

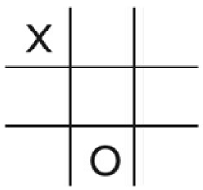


$e(n) = 6–4 = 2$



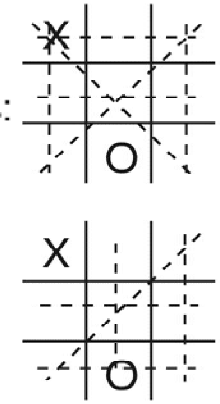
$e(n) = 3–3 = 0$

More examples…

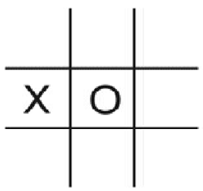


X has 6 possible win paths:

O has 5 possible wins:

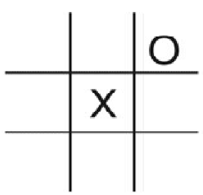


$E(n) = 6 - 5 = 1$



X has 4 possible win paths;
O has 6 possible wins

$E(n) = 4 - 6 = -2$



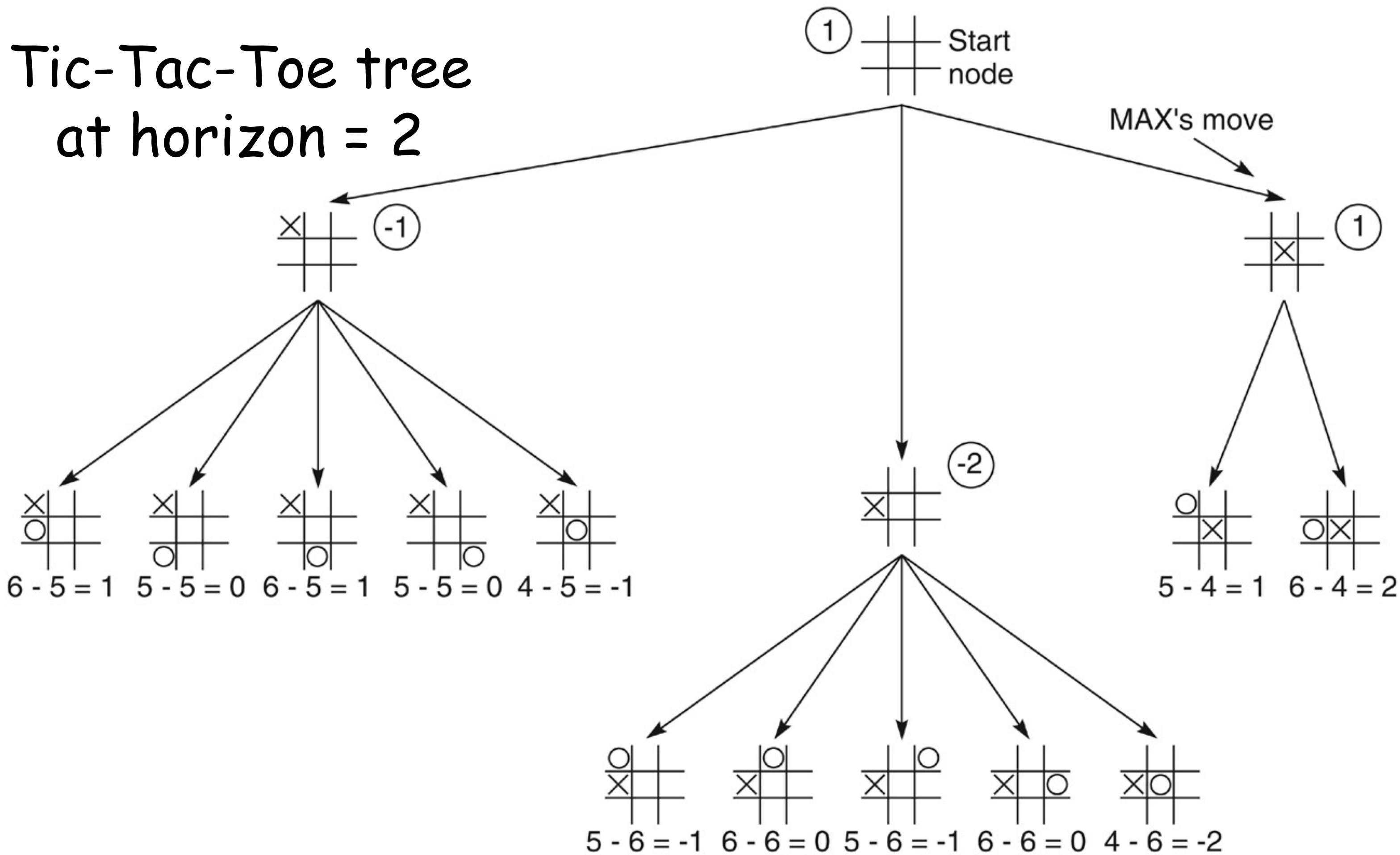
X has 5 possible win paths;
O has 4 possible wins

$E(n) = 5 - 4 = 1$

source: G. Luger (2005)

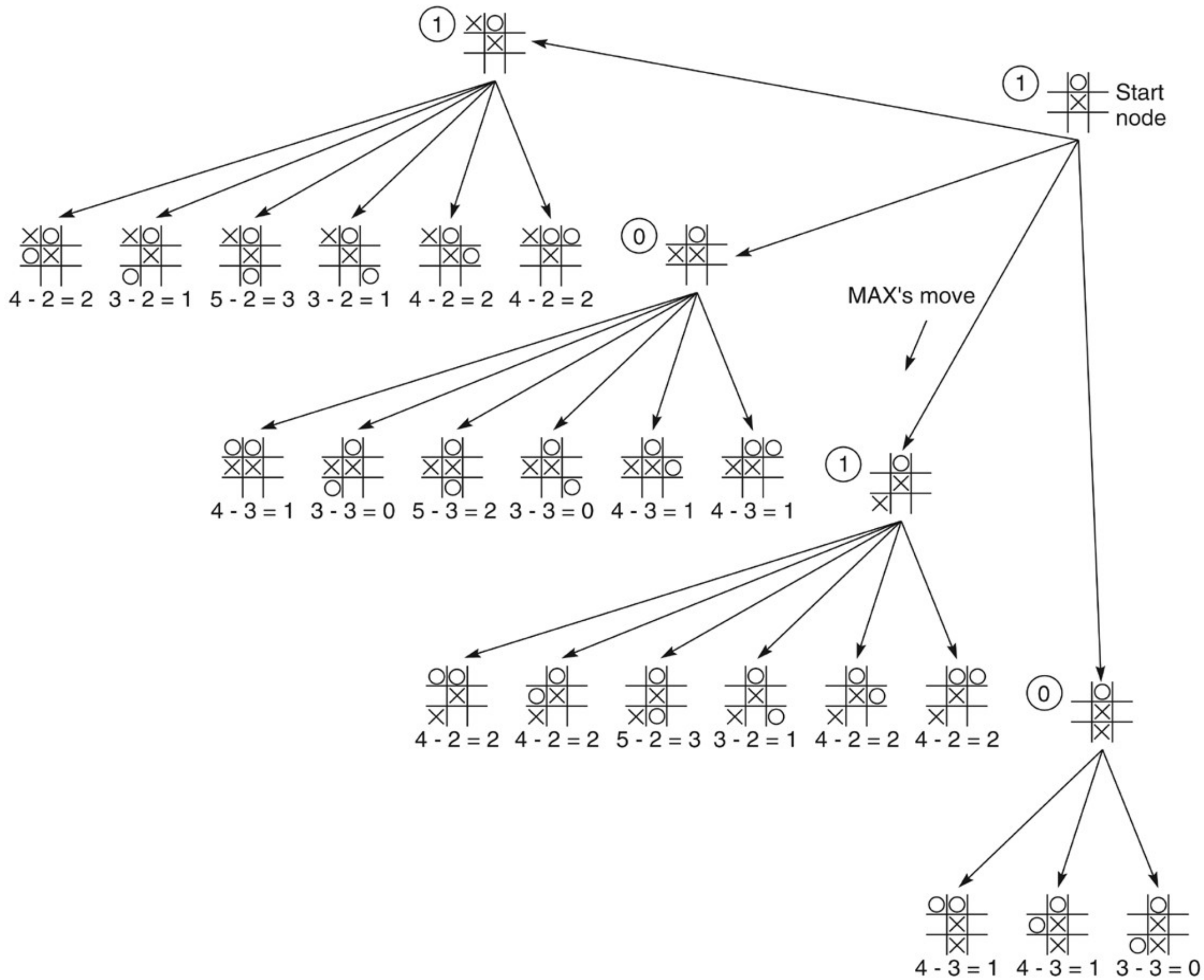
Two-ply MiniMax for Opening Move

Tic-Tac-Toe tree
at horizon = 2



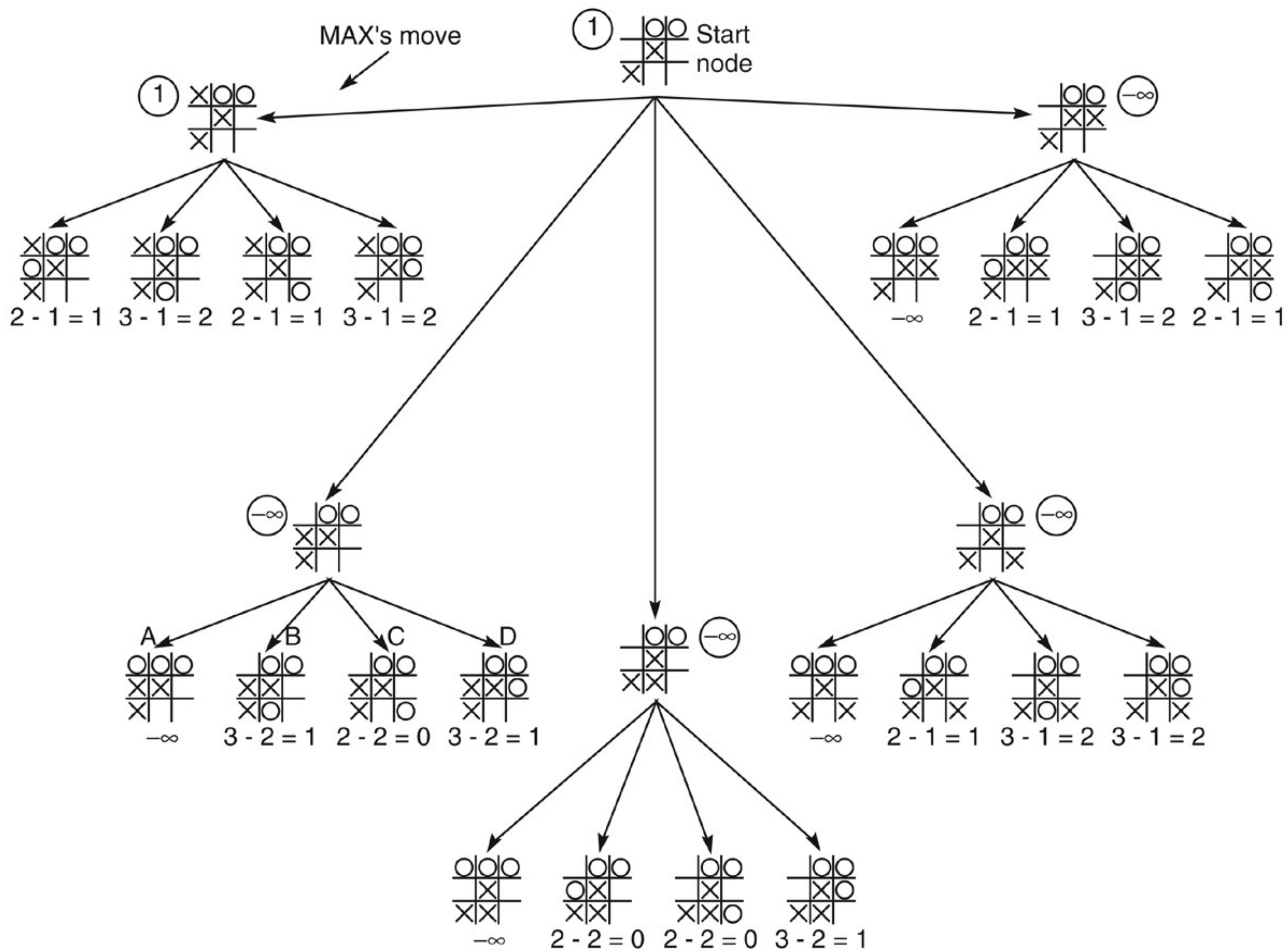
source: G. Luger (2005)

Two-ply MiniMax: MAX's possible 2nd moves



source: *G. Luger (2005)*

Two-ply minimax: MAX's move at end



source: G. Luger (2005)

Today

- State Space Search for Game Playing
  - MiniMax
  - Alpha-beta pruning 

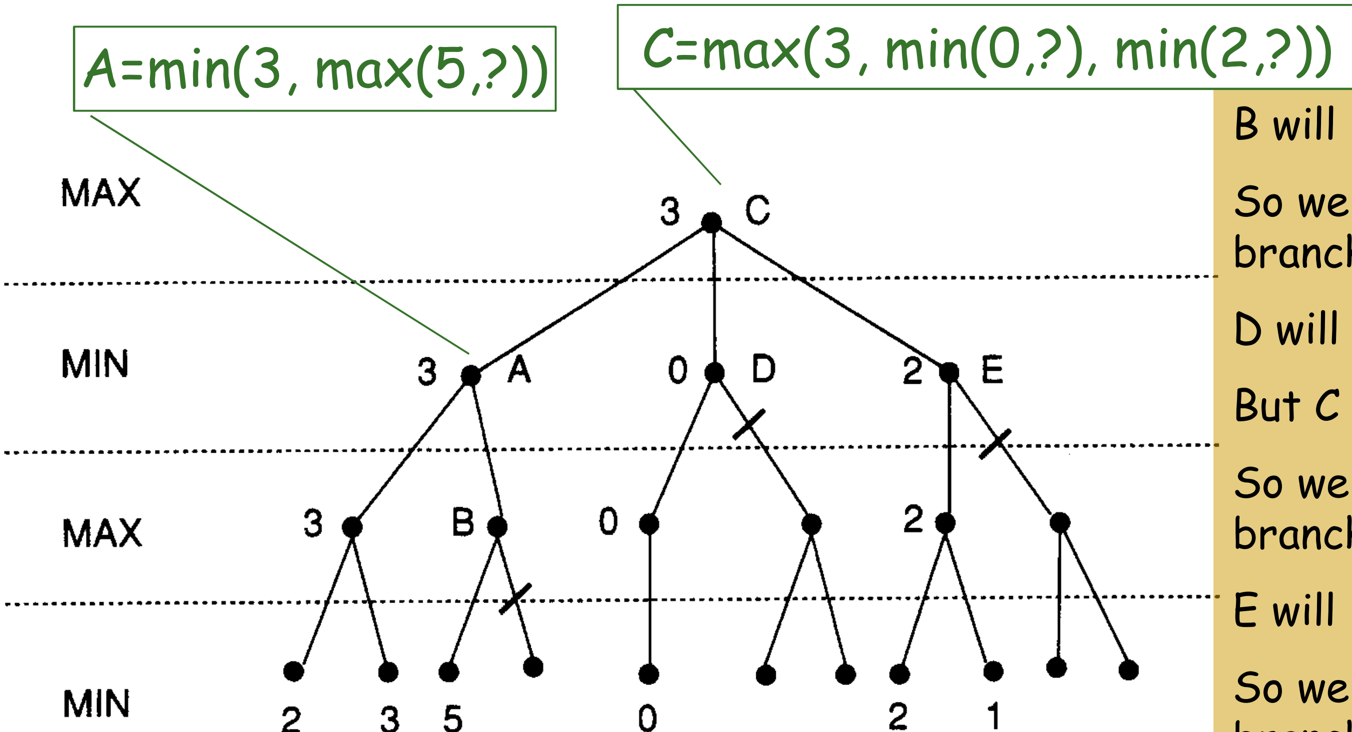
- Where we are today

Alpha-Beta Pruning

- Optimization over minimax, that:
  - ignores (cuts off, prunes) branches of the tree that cannot possibly lead to a better solution
  - reduces branching factor
  - allows deeper search with same effort

Alpha-Beta Pruning: Example 1

- With minimax, we look at all possible nodes at the n-ply depth
- With α-β pruning, we ignore branches that could not possibly contribute to the final decision



B will be >= 5

So we can ignore B's right branch, because A must be 3

D will be <= 0

But C will be >= 3

So we can ignore D's right branch

E will be <= 2.

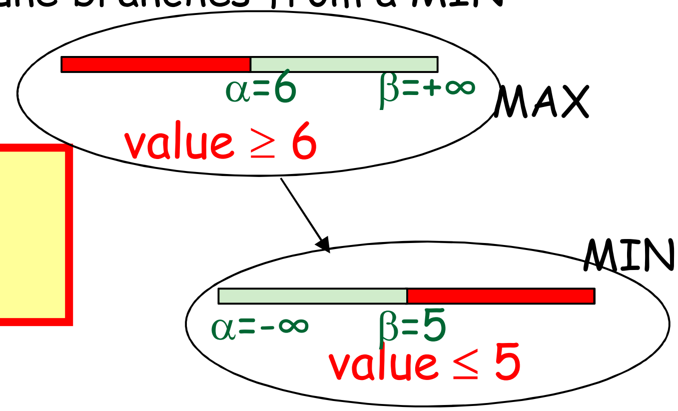
So we can ignore E's right branch

Because C will be 3.

source: G. Luger (2005)

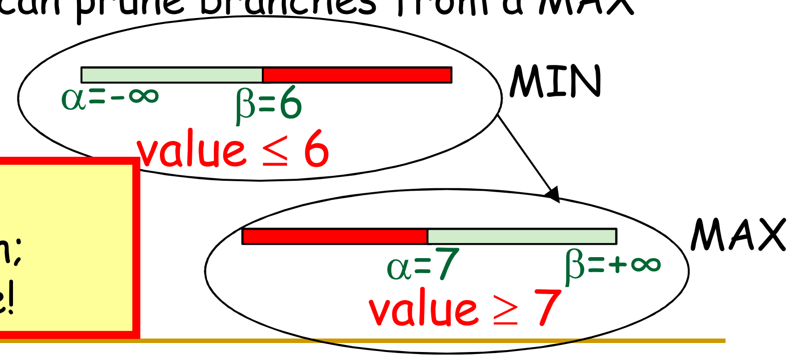
Alpha-Beta Pruning Algorithm

- α : lower bound on the final backed-up value.
- β : upper bound on the final backed-up value.
- Alpha pruning:
  - eg. if MAX node's α = 6, then the search can prune branches from a MIN descendant that has a β <= 6.
  - if child β <= ancestor α → prune



incompatible…
so stop searching the right branch;
the value cannot come from there!

- Beta pruning:
  - eg. if a MIN node's β = 6, then the search can prune branches from a MAX descendant that has an α >= 6.
  - if ancestor β <= child α → prune



incompatible…
so stop searching the right branch;
the value cannot come from there!

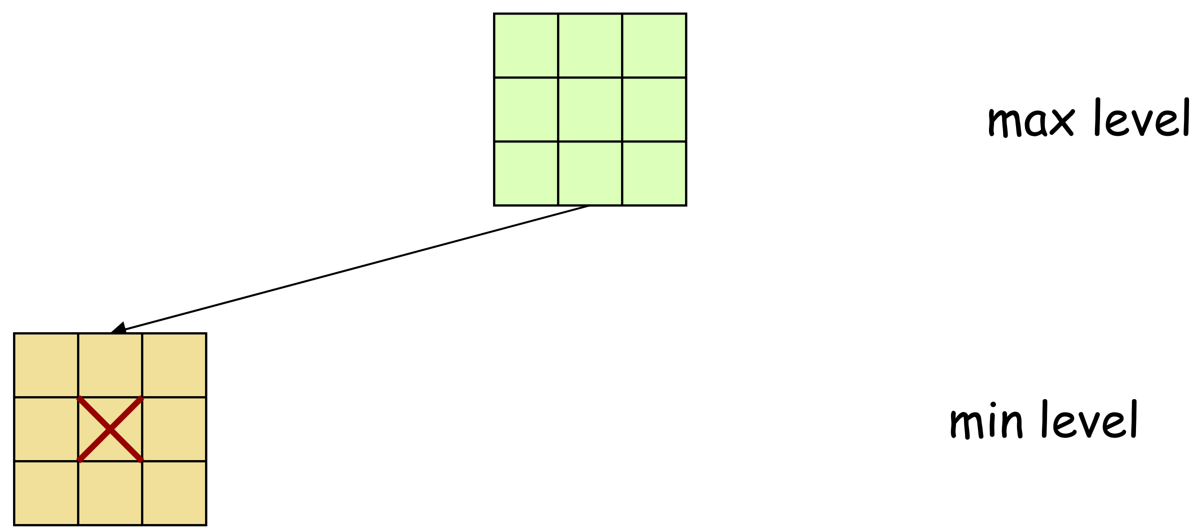
Alpha-Beta Pruning Algorithm

```
function alphabeta(node, depth, α, β, maximizingPlayer)
     if depth = 0 or node is a terminal node
         return the heuristic value of node
     if maximizingPlayer
         v := -∞
         for each child of node
             v := max(v, alphabeta(child, depth - 1, α, β, FALSE))
             α := max(α, v)
             if β ≤ α
                 break (* β cut-off *)
         return v
     else
         v := ∞
         for each child of node
             v := min(v, alphabeta(child, depth - 1, α, β, TRUE))
             β := min(β, v)
             if β ≤ α
                 break (* α cut-off *)
         return v
```

Initial call:
`alphabeta(origin, depth, -∞, +∞, TRUE)`

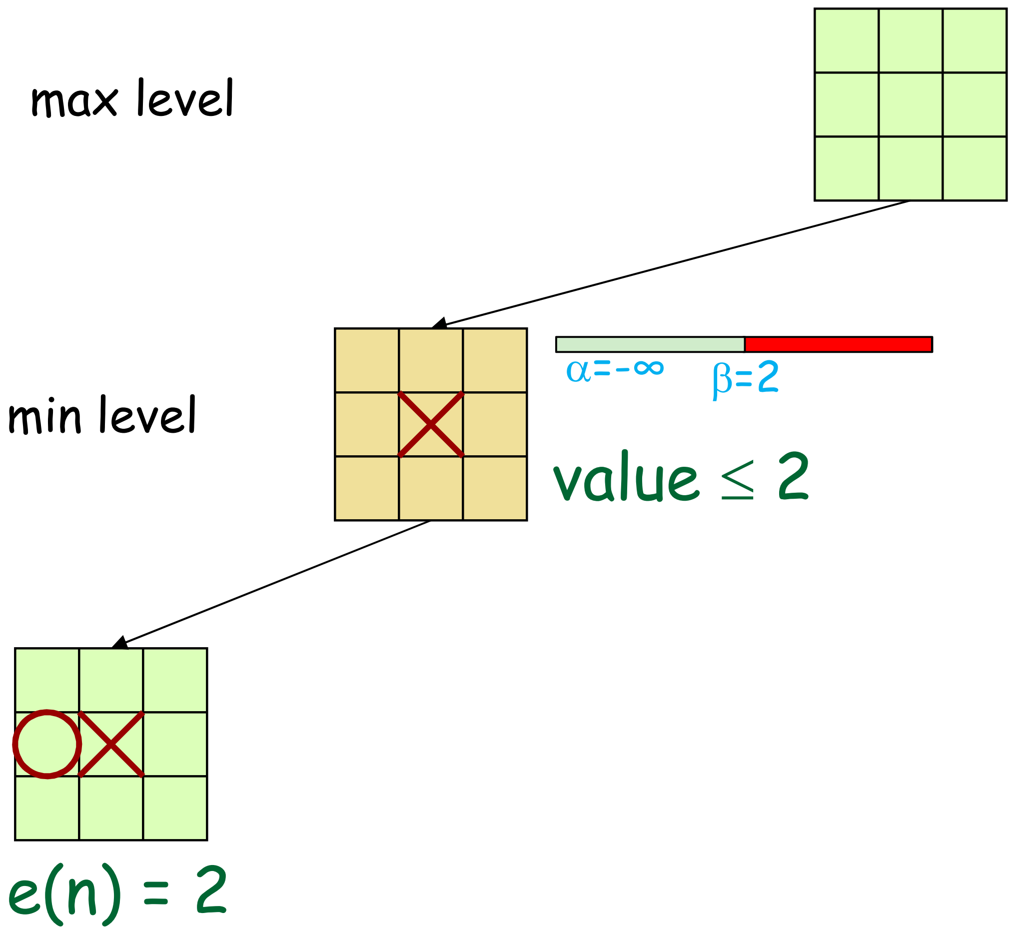
source: http://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

Example with tic-tac-toe



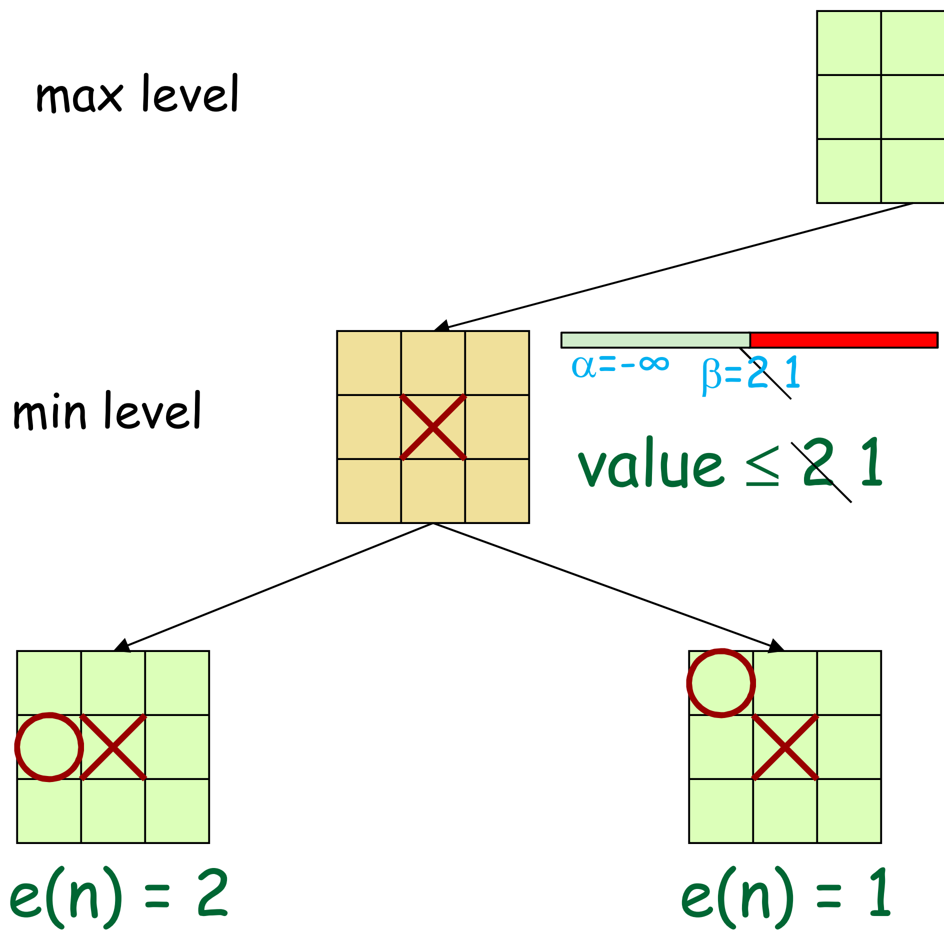
source: robotics.stanford.edu/~latombe/cs121/2003/home.htm

Example with tic-tac-toe



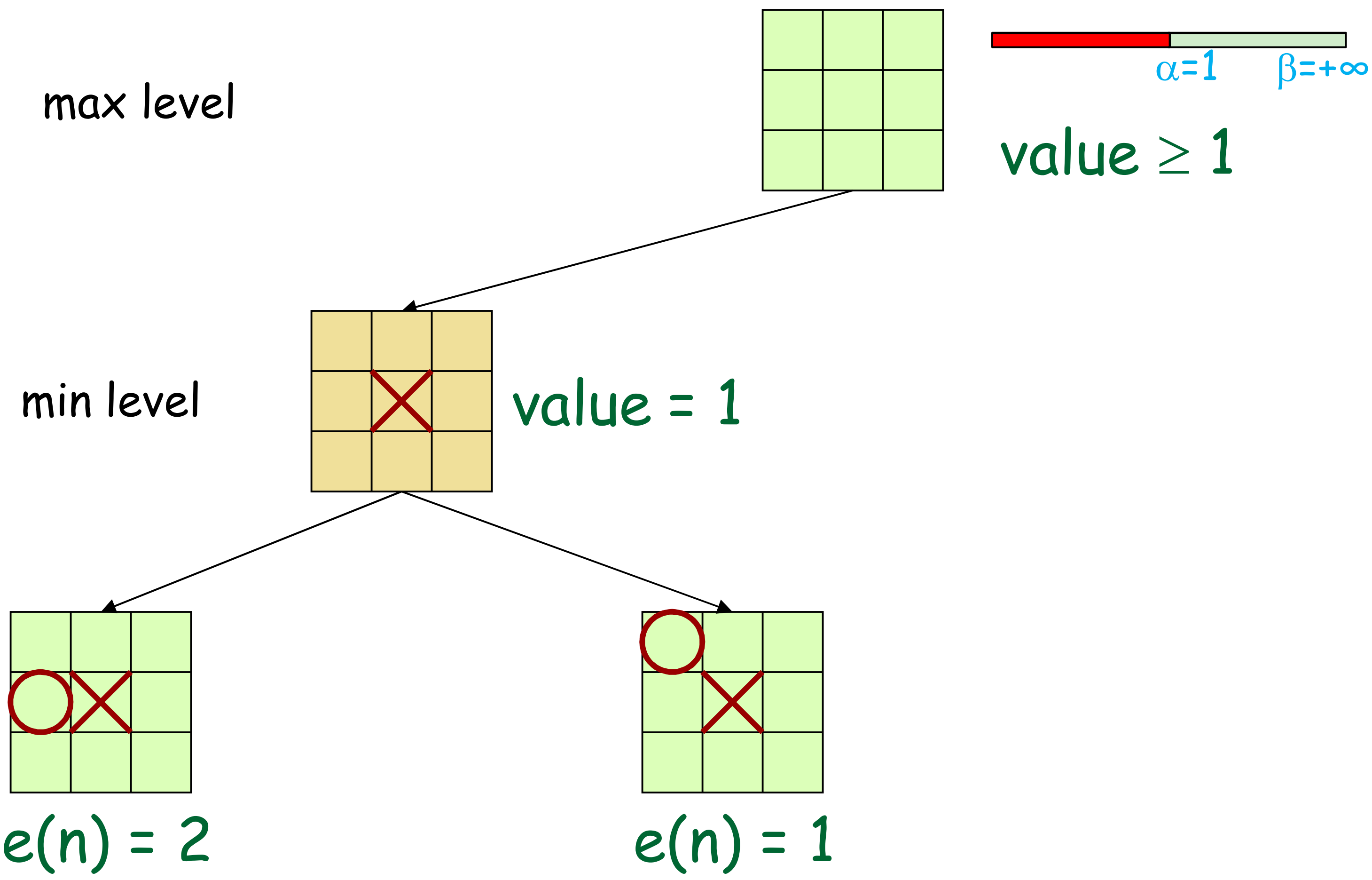
source: robotics.stanford.edu/~latombe/cs121/2003/home.htm

Example with tic-tac-toe



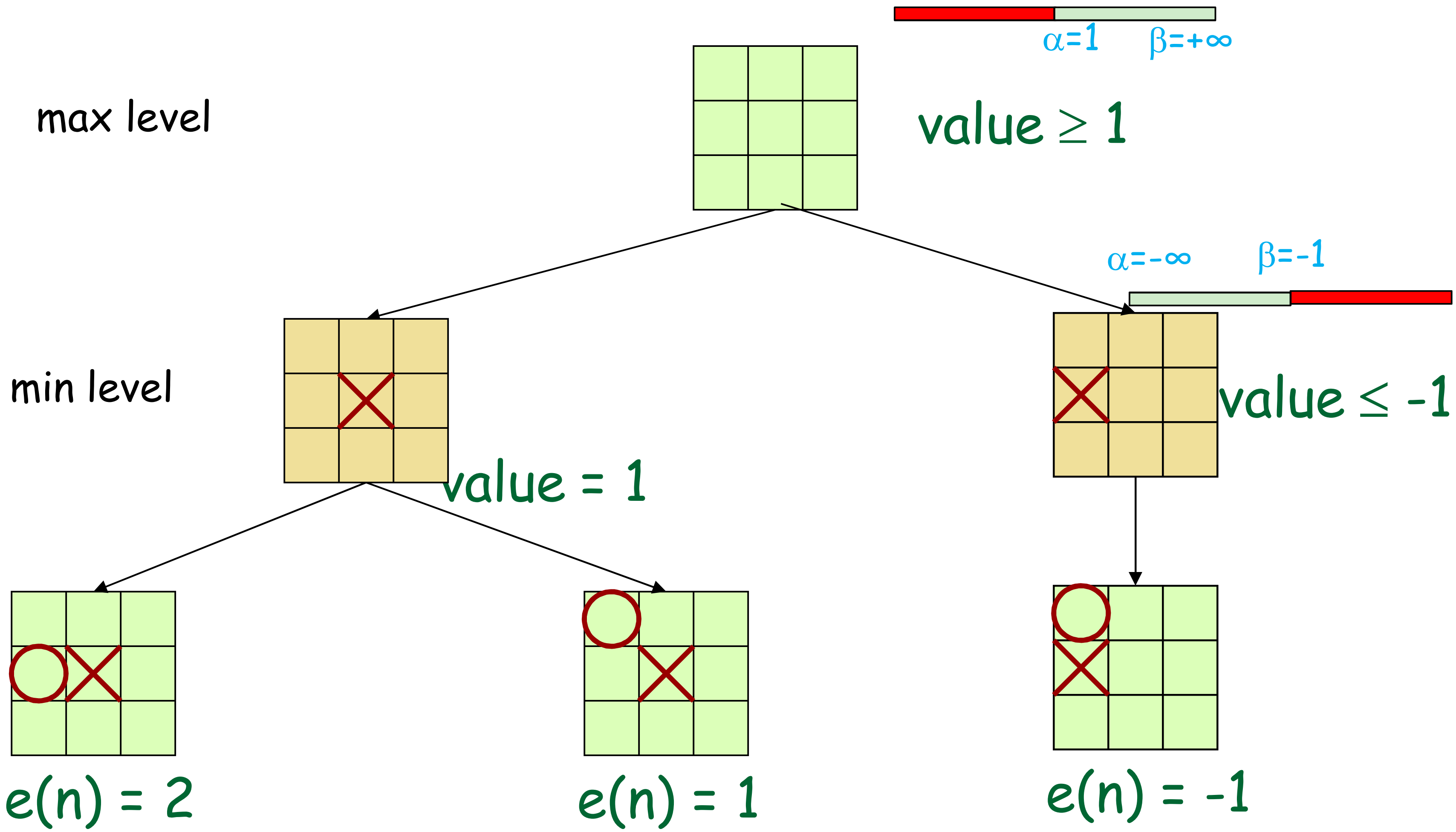
source: robotics.stanford.edu/~latombe/cs121/2003/home.htm

Example with tic-tac-toe



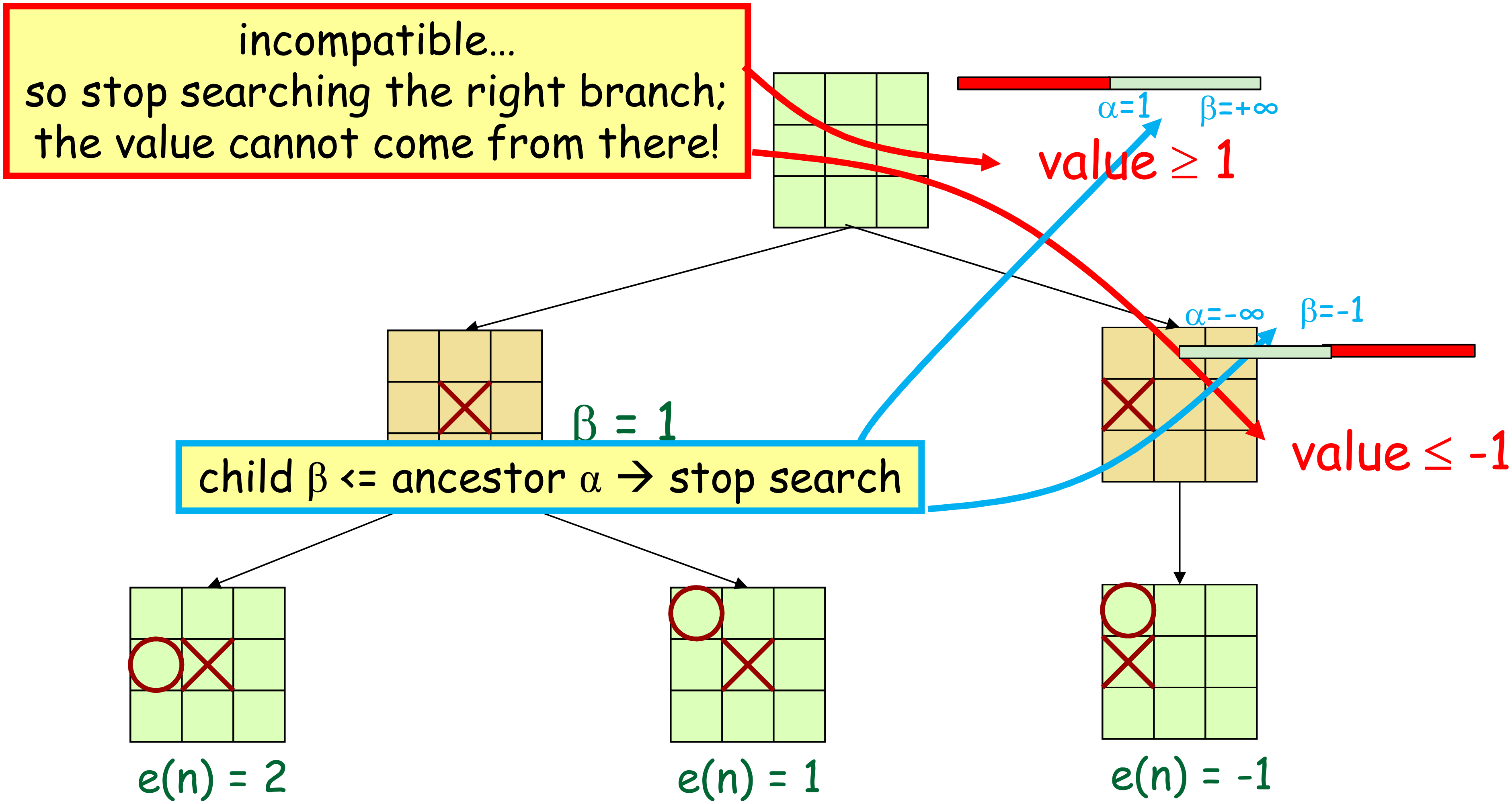
source: robotics.stanford.edu/~latombe/cs121/2003/home.htm

Example with tic-tac-toe



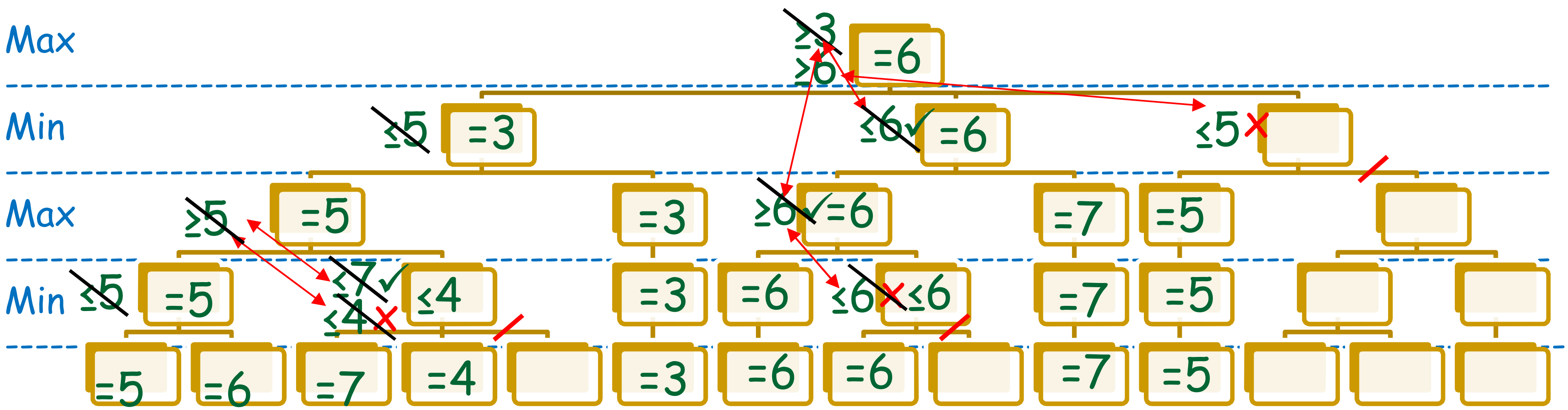
source: robotics.stanford.edu/~latombe/cs121/2003/home.htm

Example with tic-tac-toe



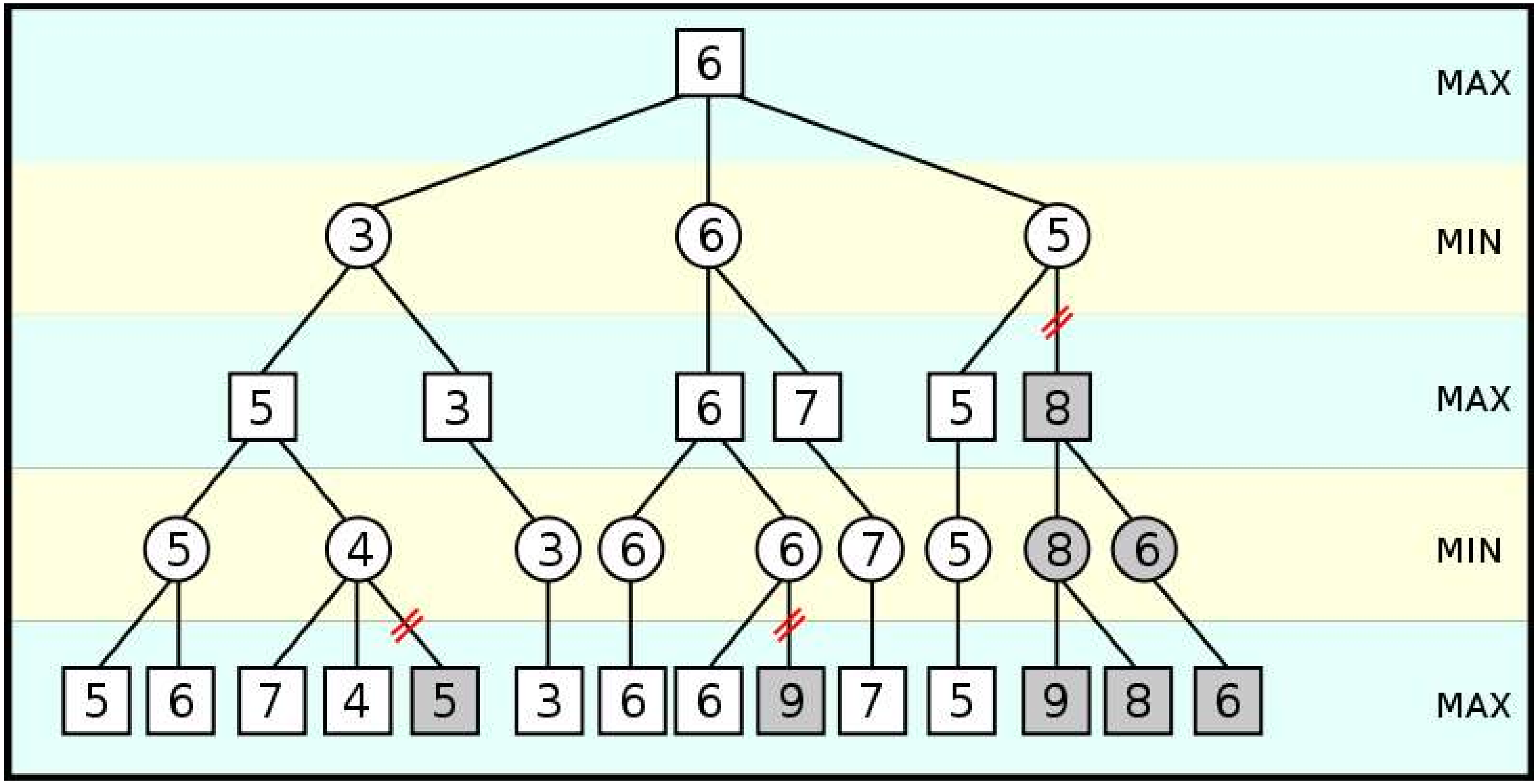
source: robotics.stanford.edu/~latombe/cs121/2003/home.htm

Alpha-Beta Pruning: Example 2



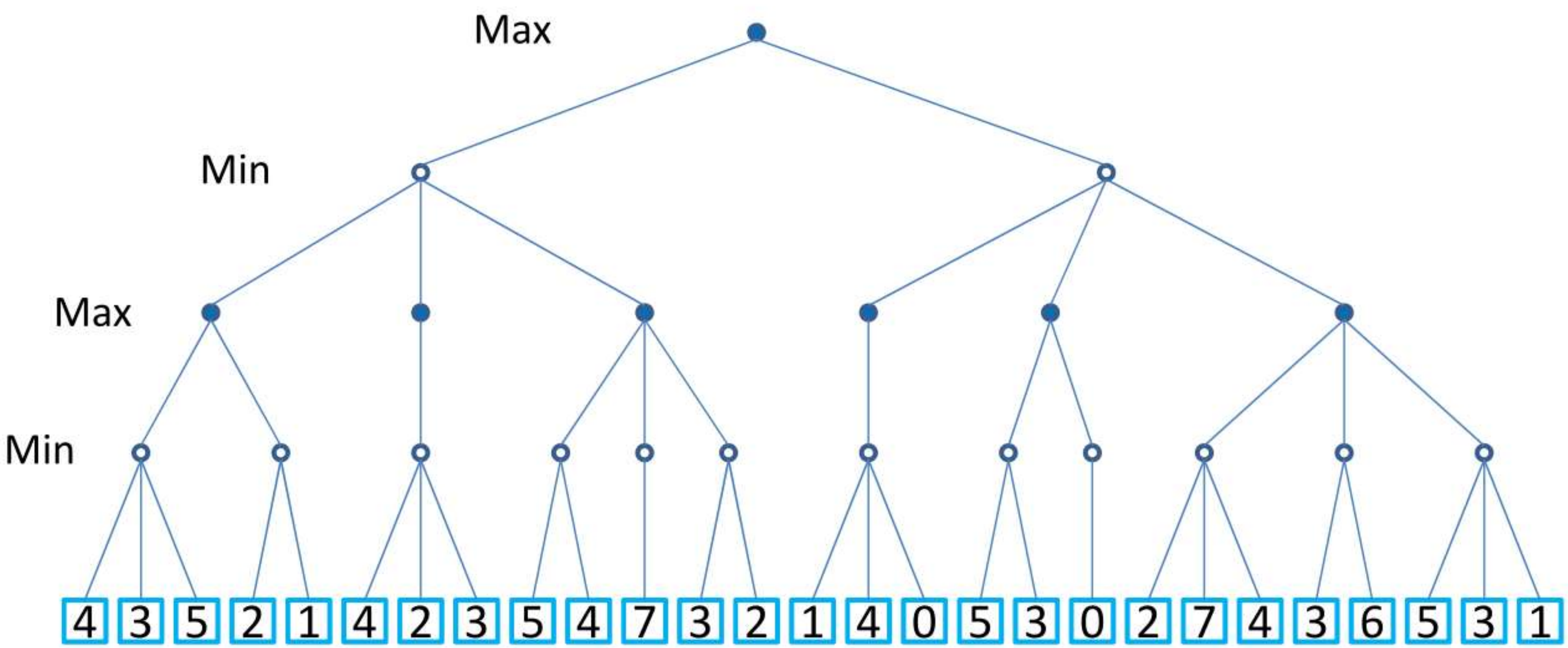
source: http://en.wikipedia.org/wiki/File:AB_pruning.svg

Alpha-Beta Pruning: Example 2

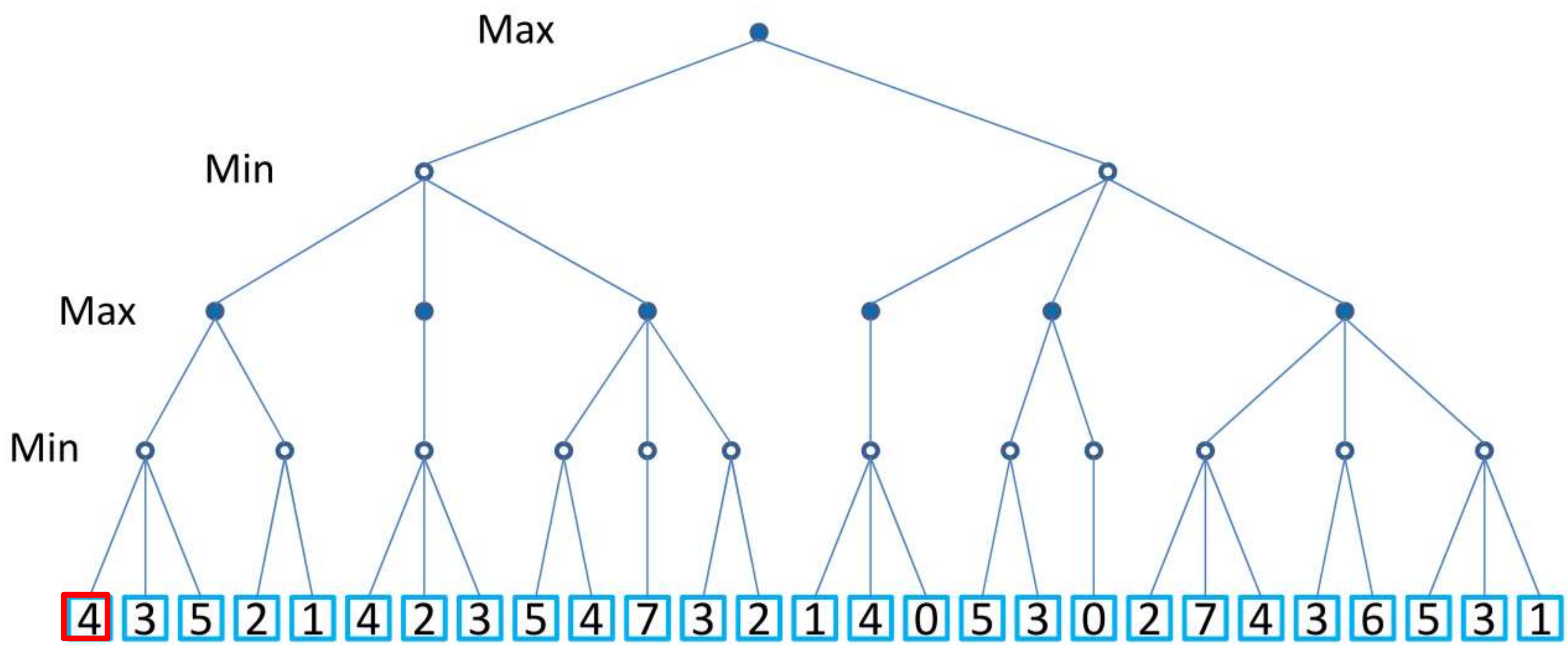


source: http://en.wikipedia.org/wiki/File:AB_pruning.svg

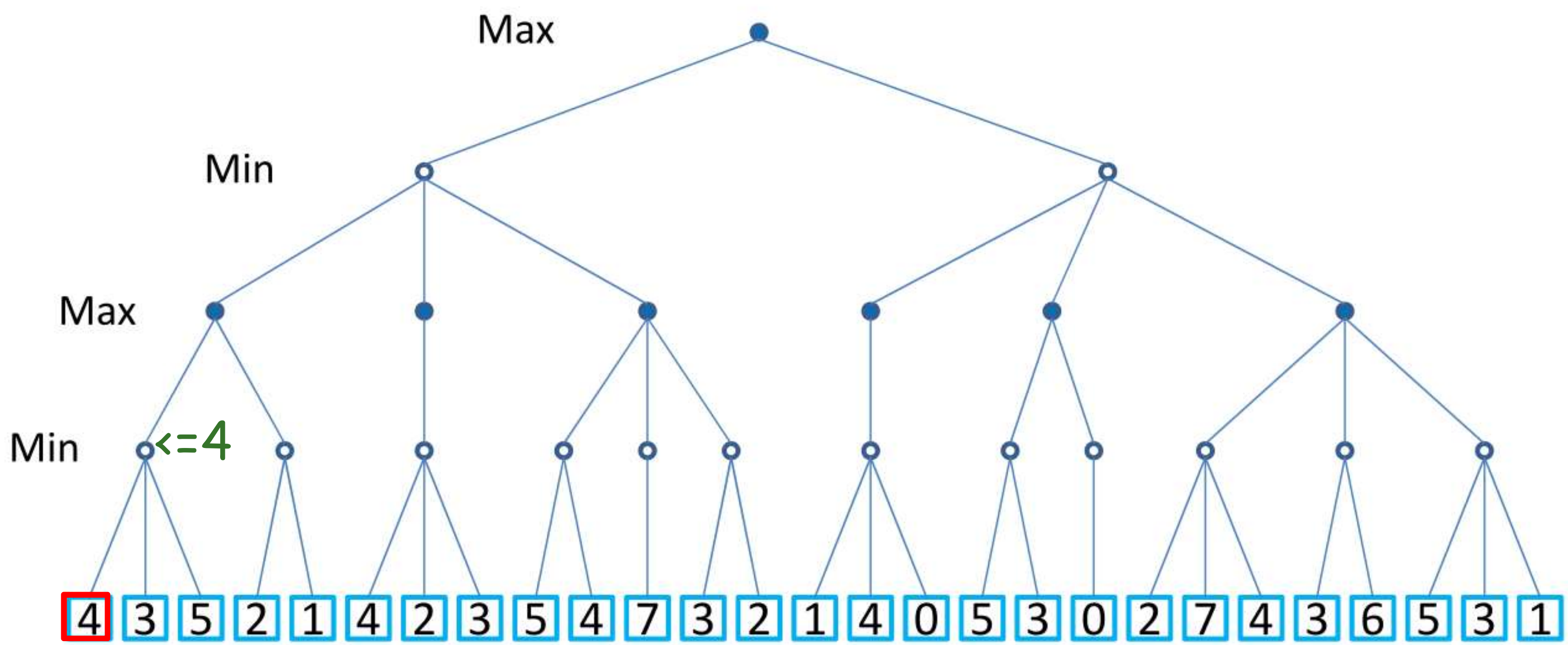
Alpha-Beta Pruning: Example 3



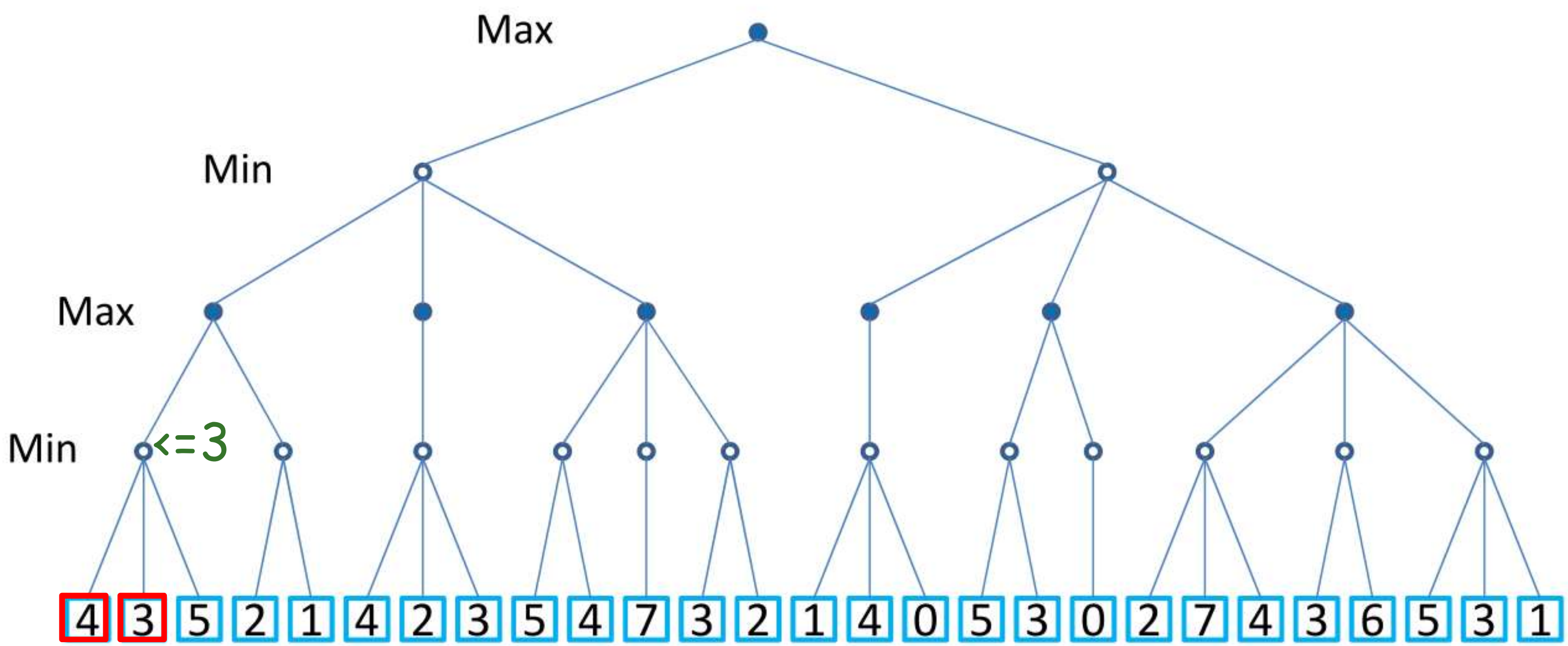
Alpha-Beta Pruning: Example 3



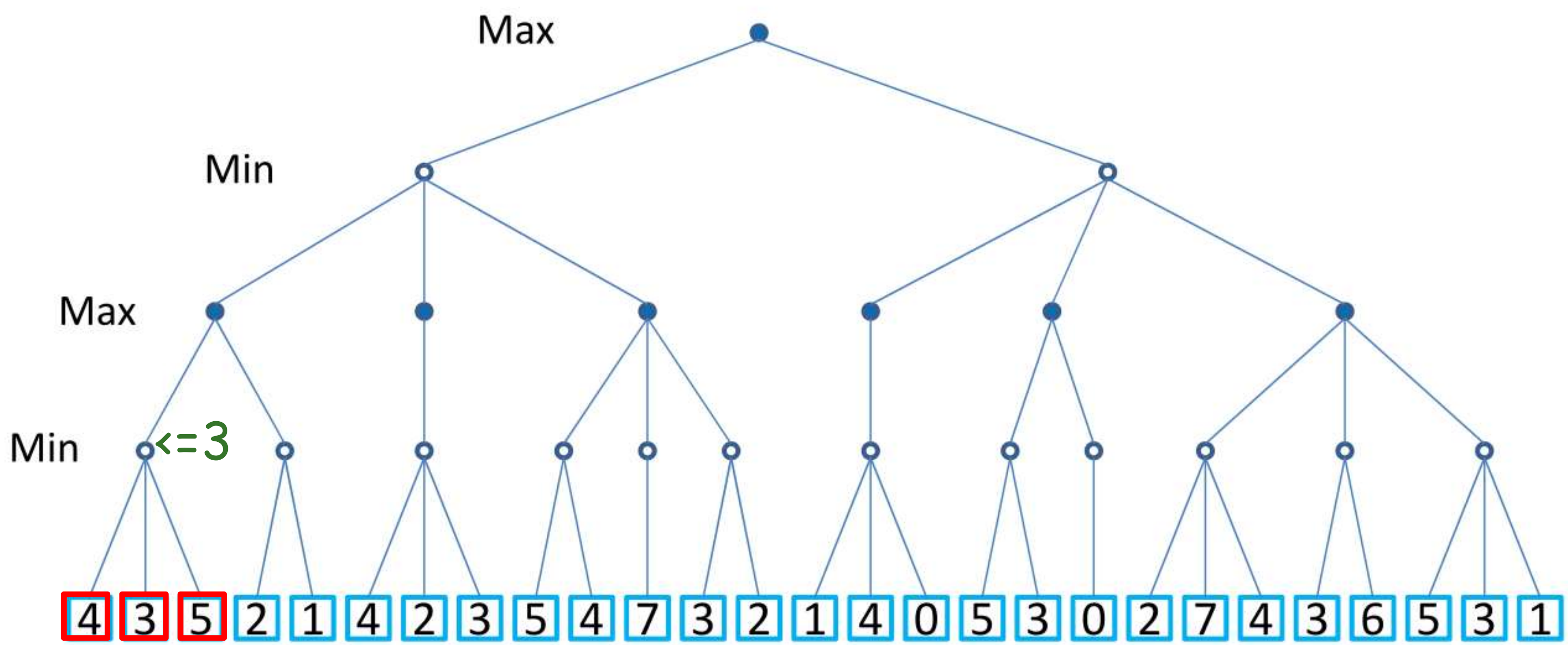
Alpha-Beta Pruning: Example 3



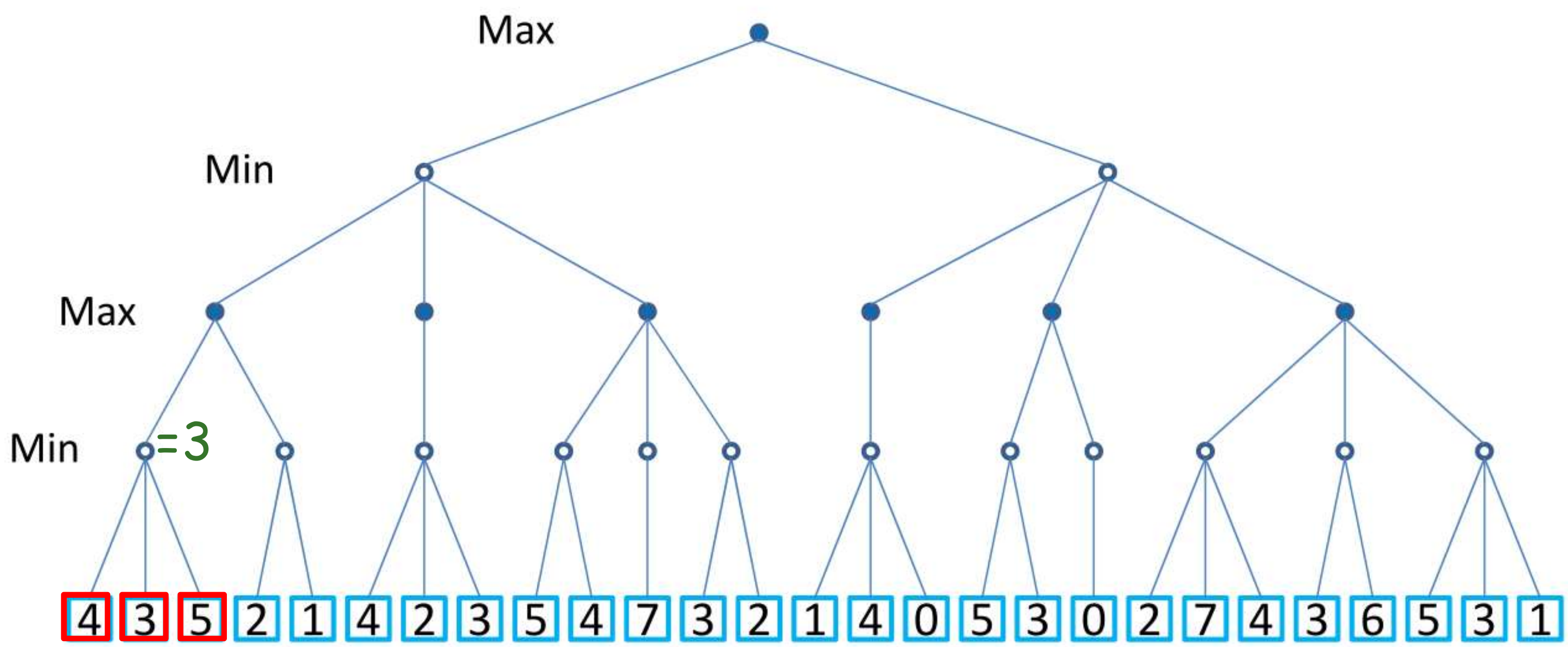
Alpha-Beta Pruning: Example 3



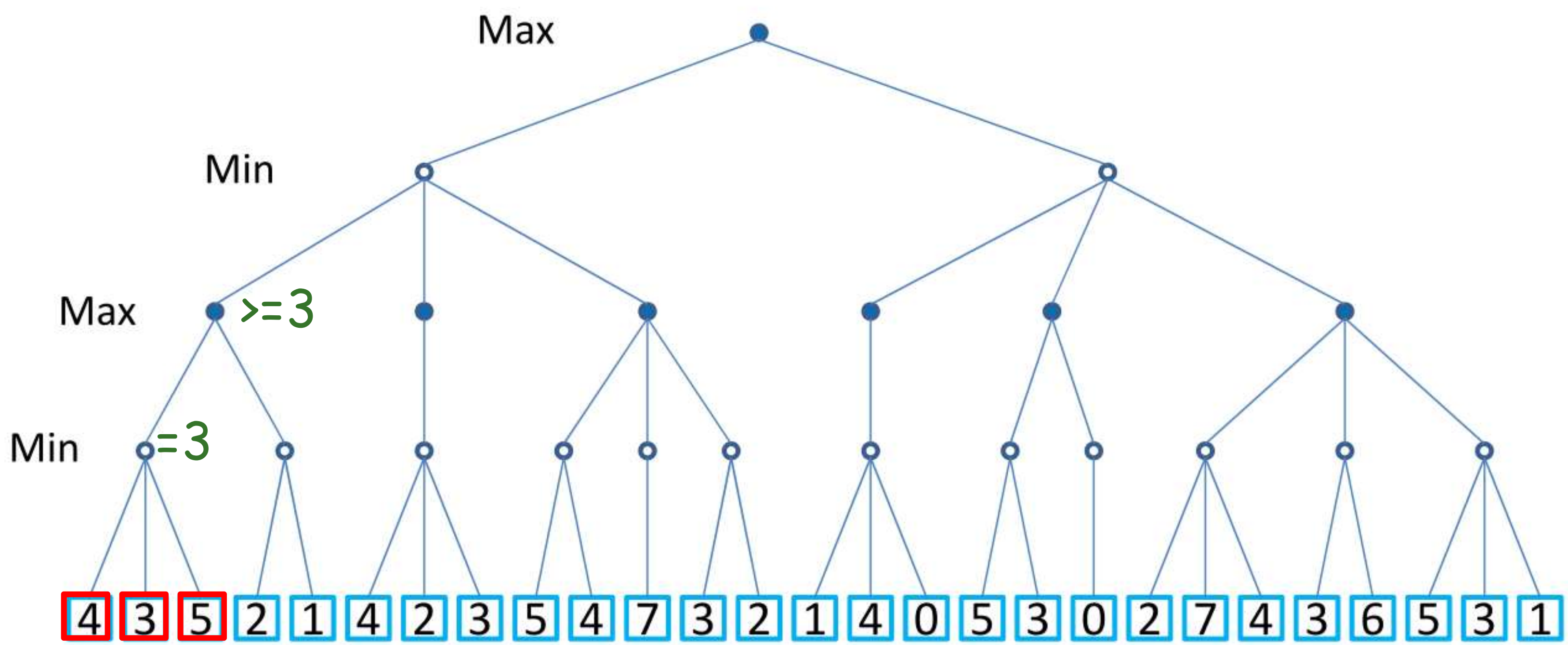
Alpha-Beta Pruning: Example 3



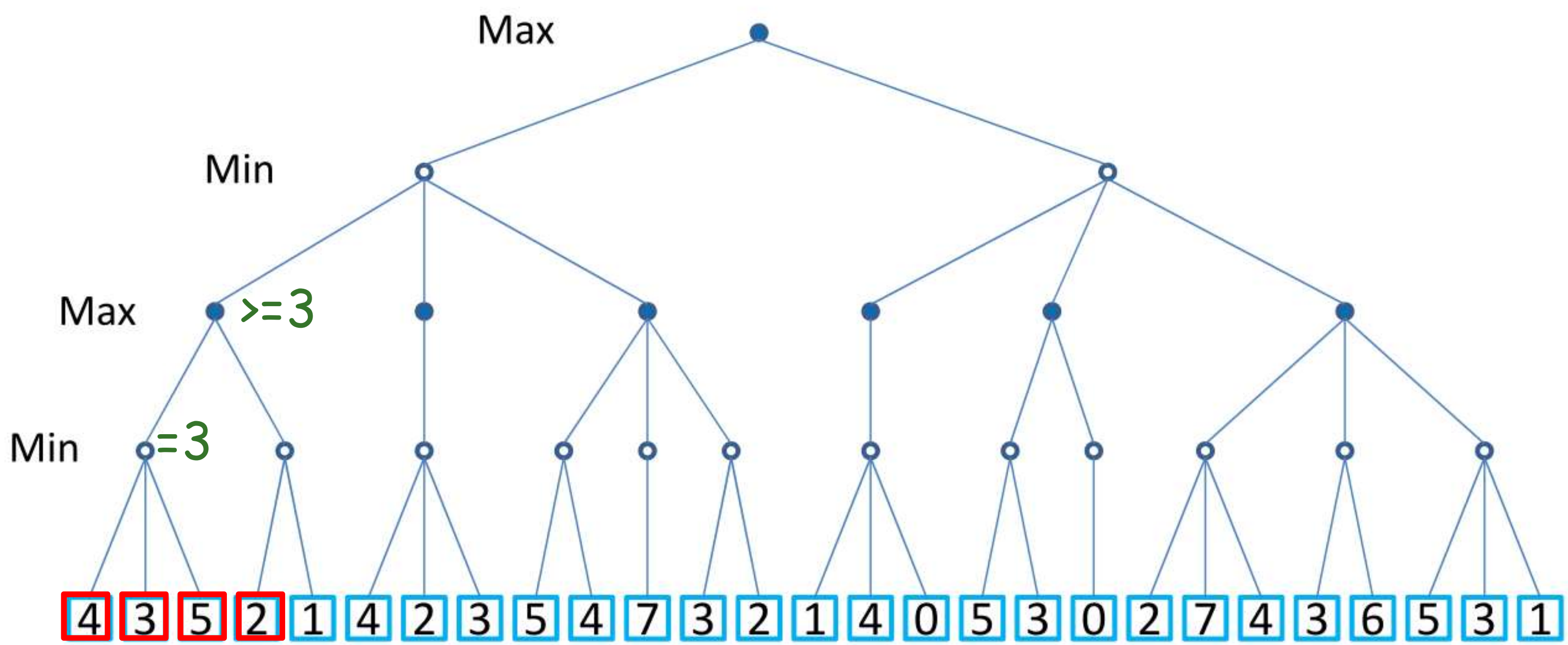
Alpha-Beta Pruning: Example 3



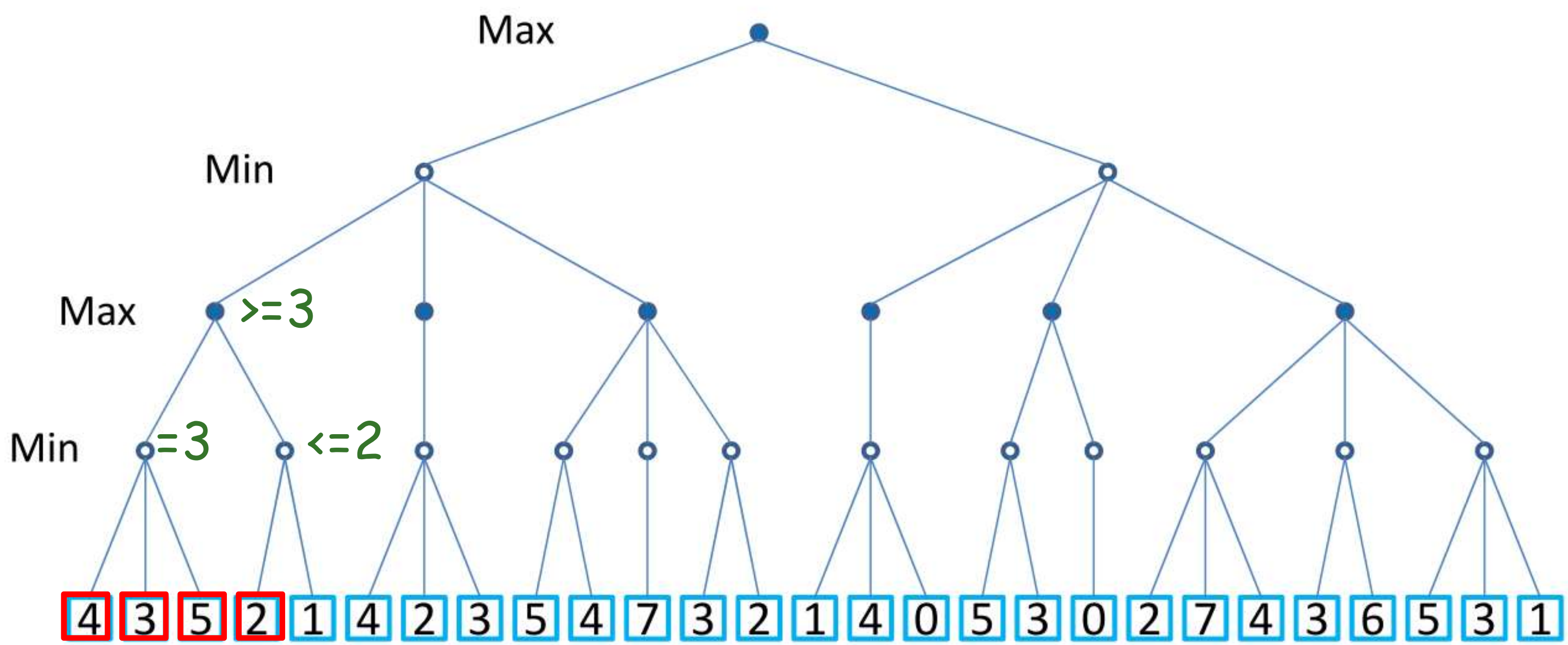
Alpha-Beta Pruning: Example 3



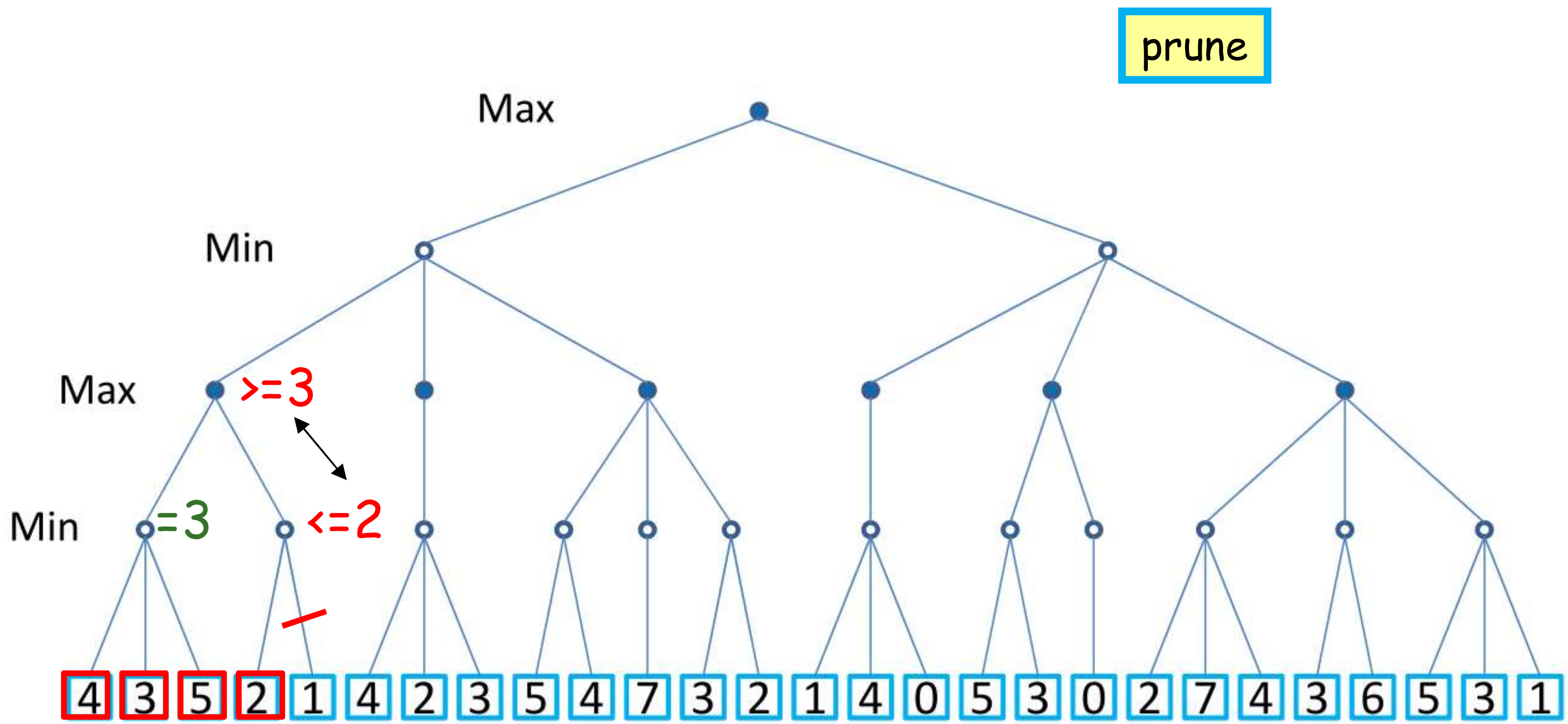
Alpha-Beta Pruning: Example 3



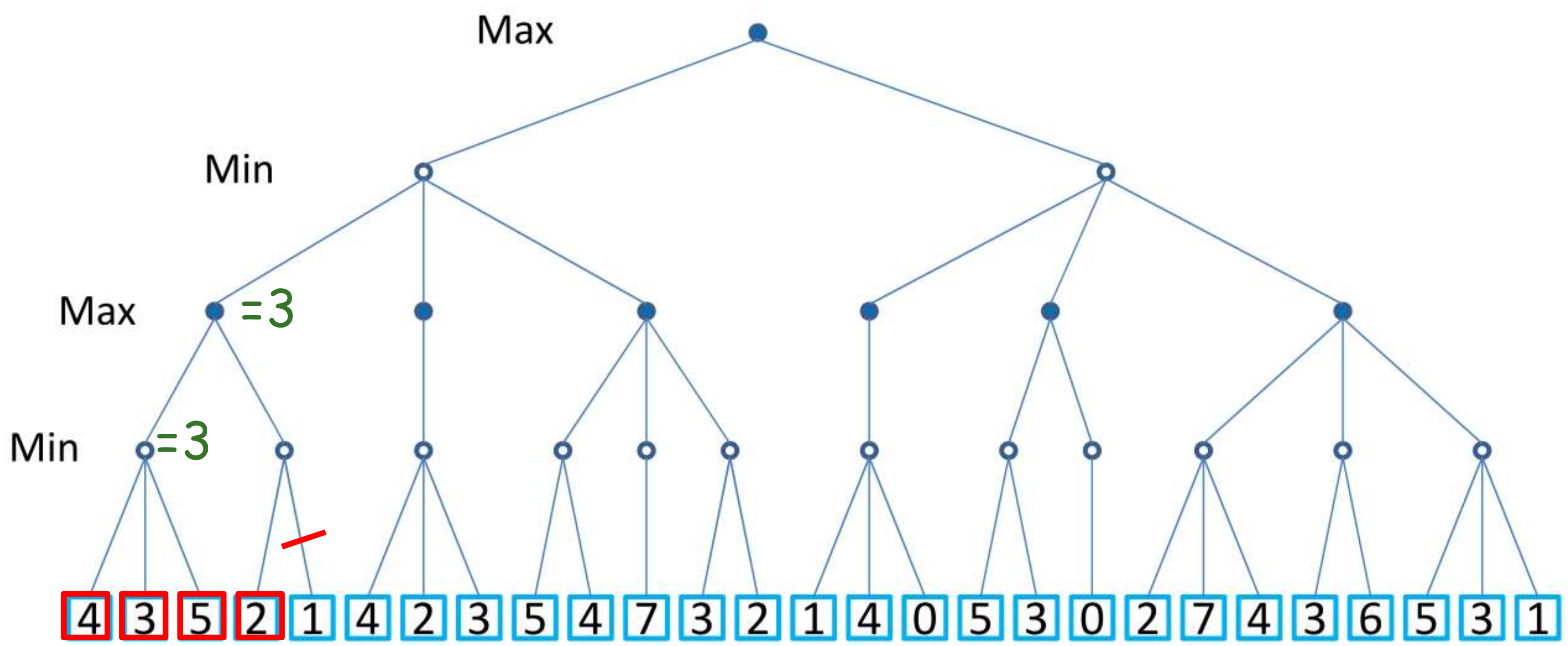
Alpha-Beta Pruning: Example 3



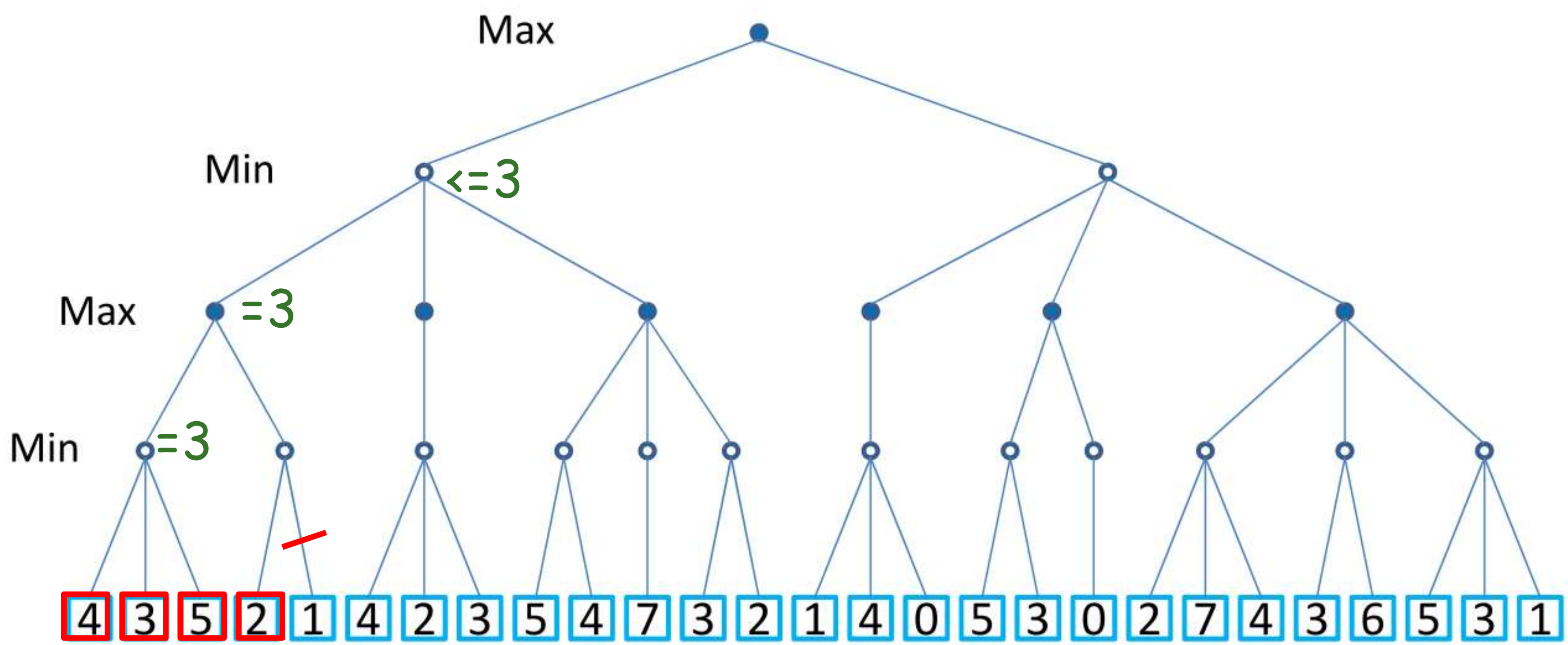
Alpha-Beta Pruning: Example 3



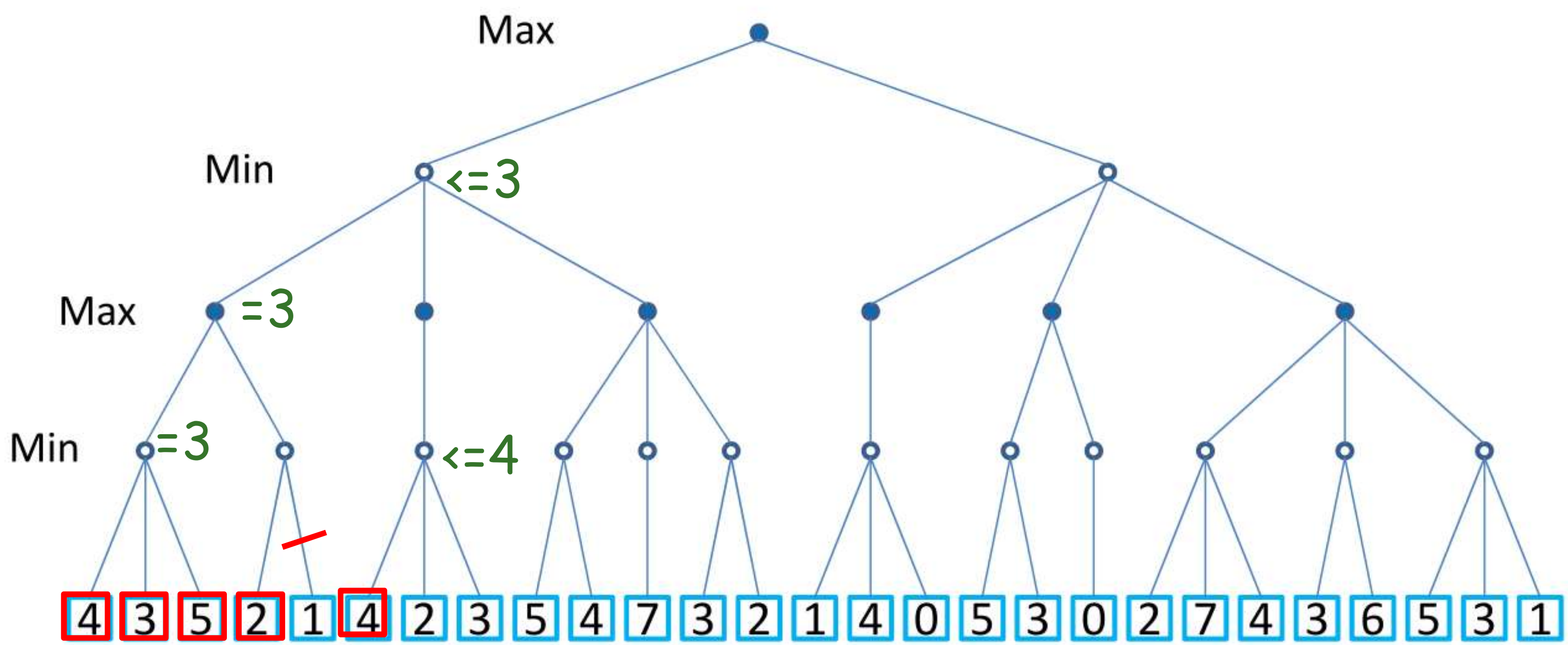
Alpha-Beta Pruning: Example 3



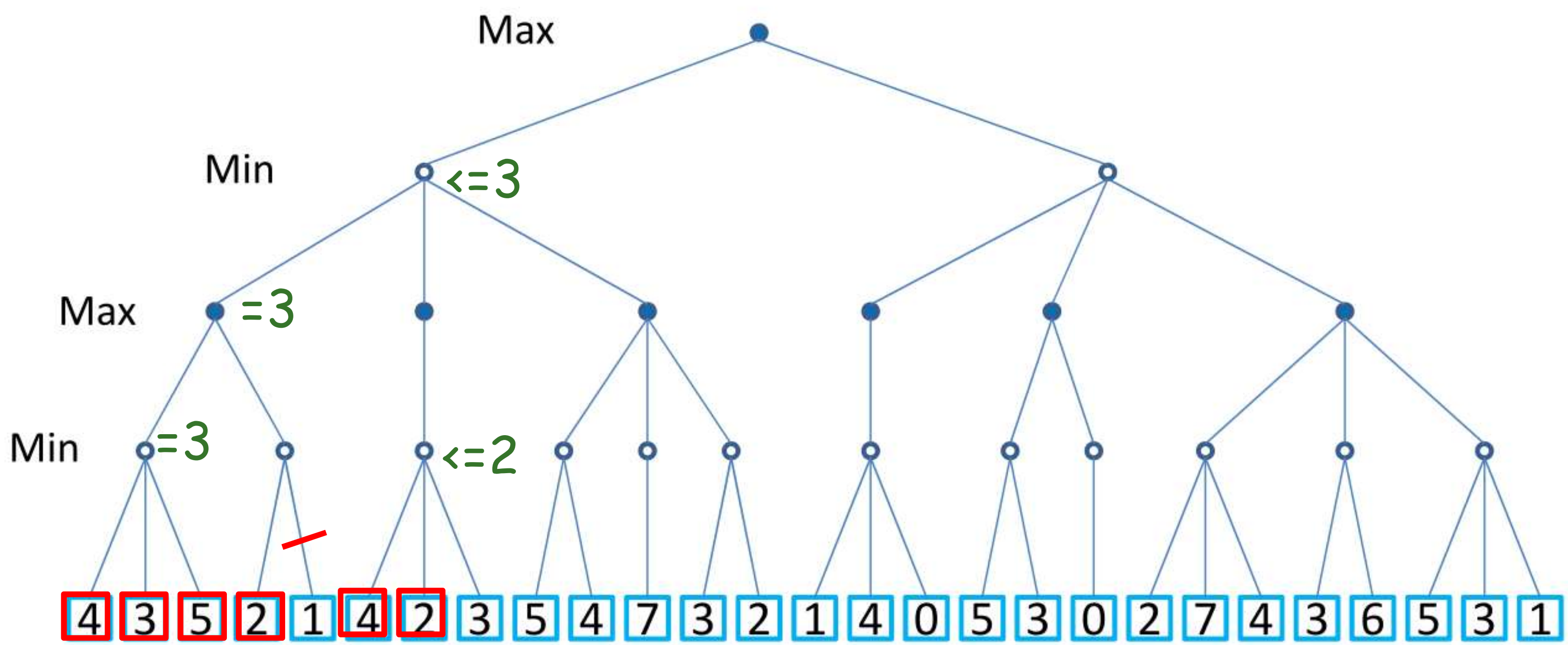
Alpha-Beta Pruning: Example 3



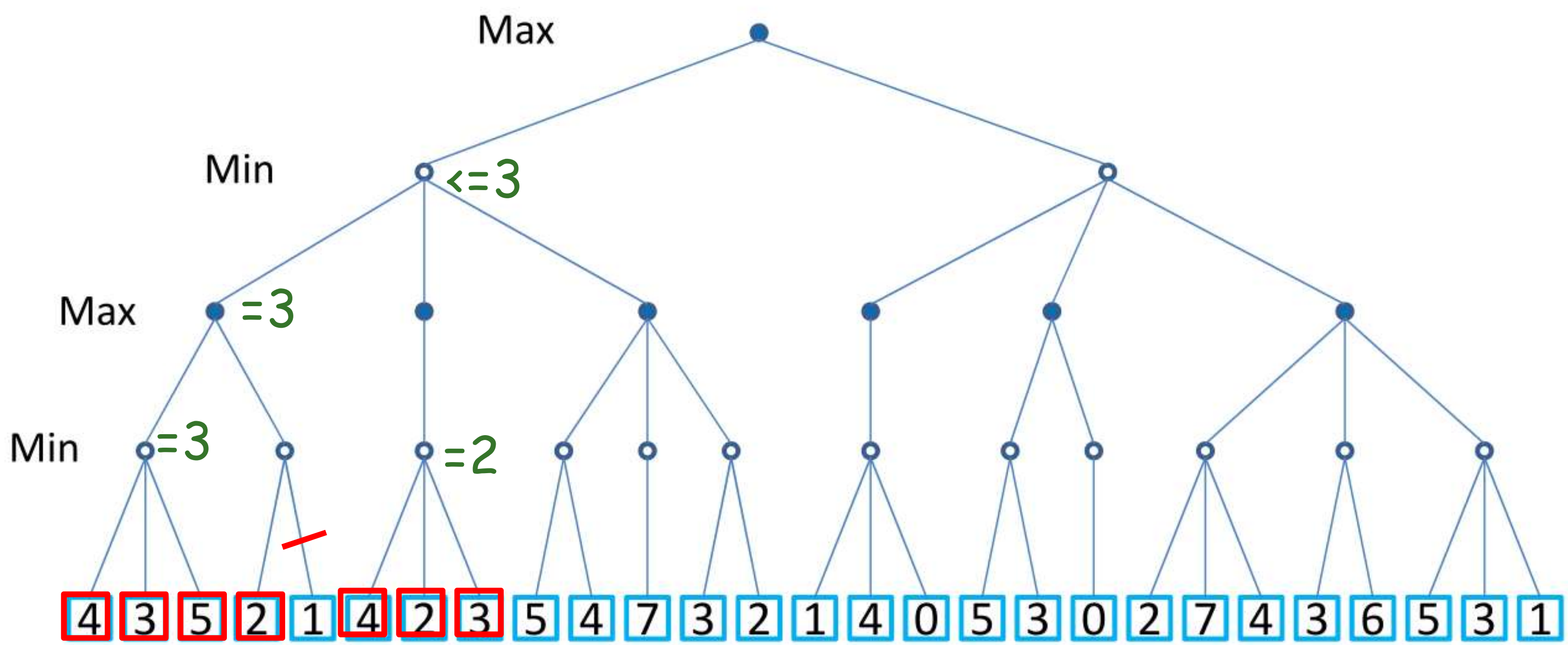
Alpha-Beta Pruning: Example 3



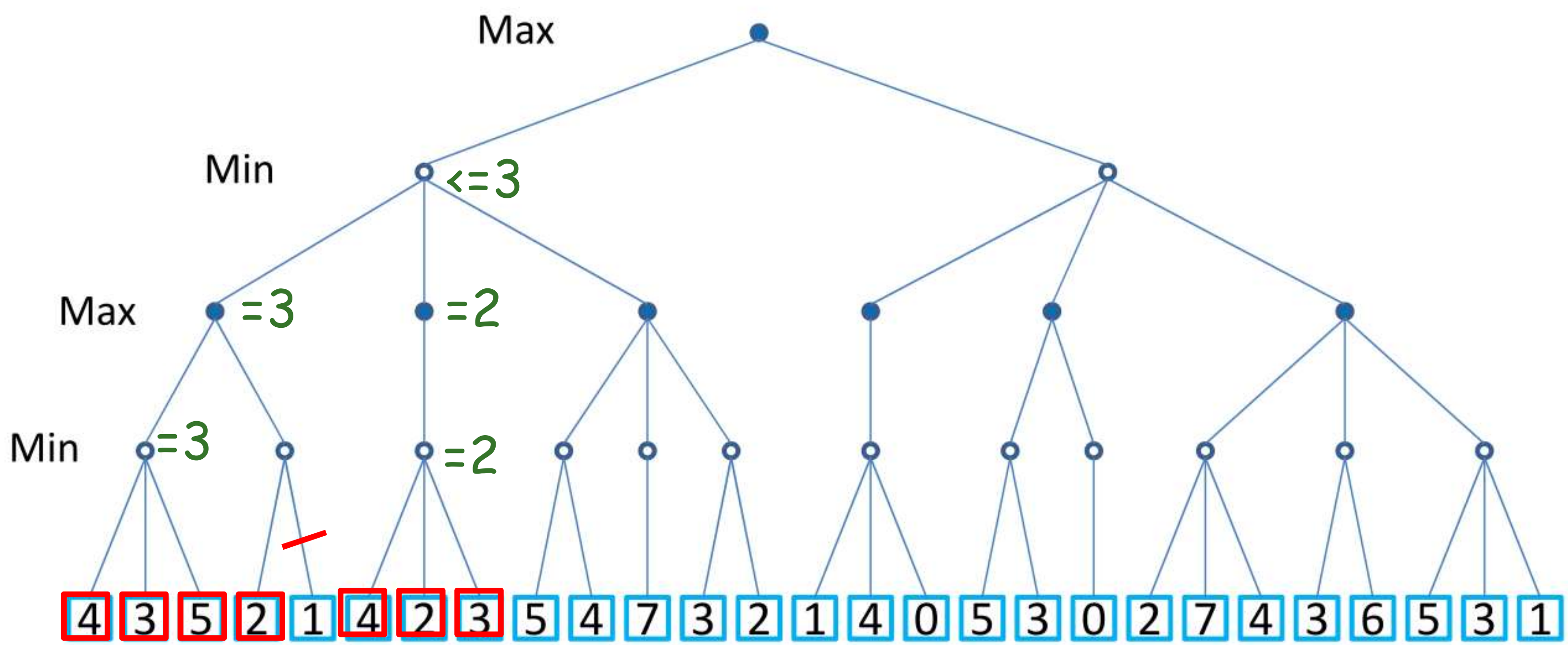
Alpha-Beta Pruning: Example 3



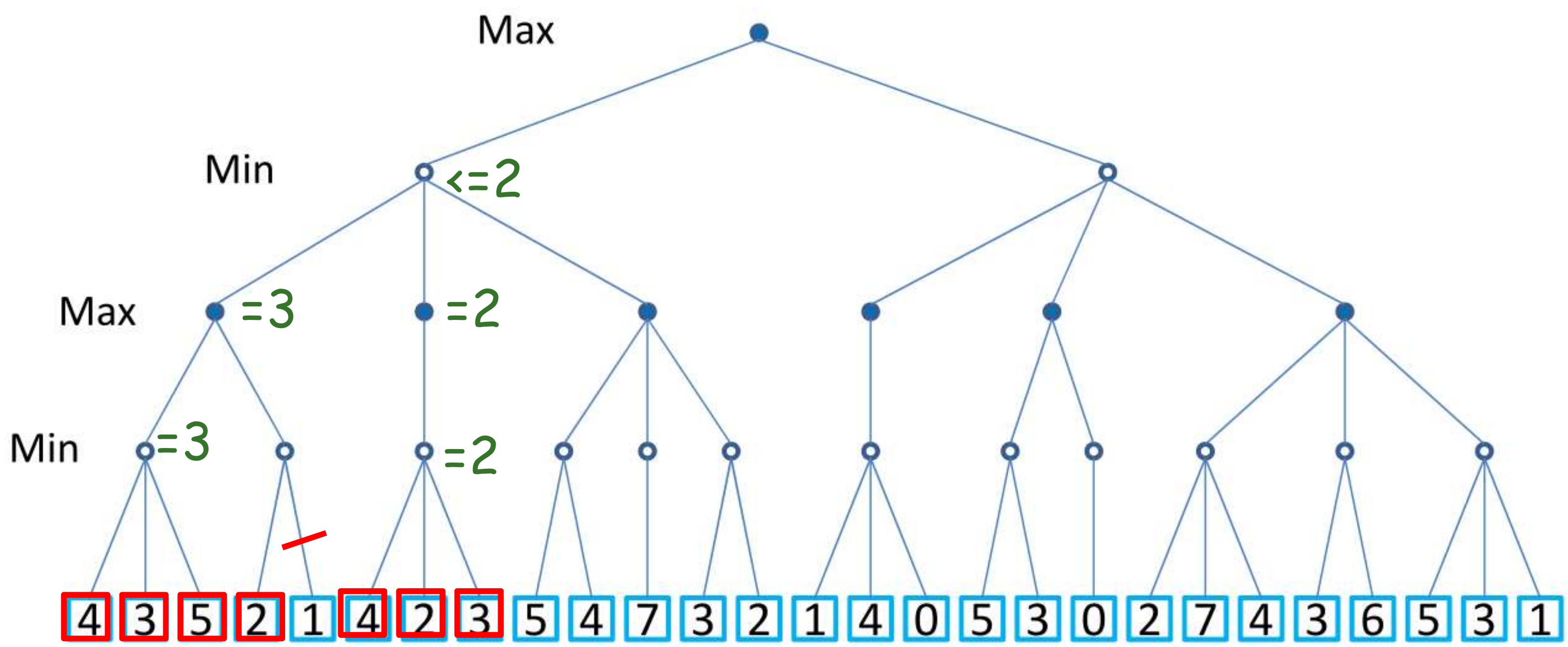
Alpha-Beta Pruning: Example 3



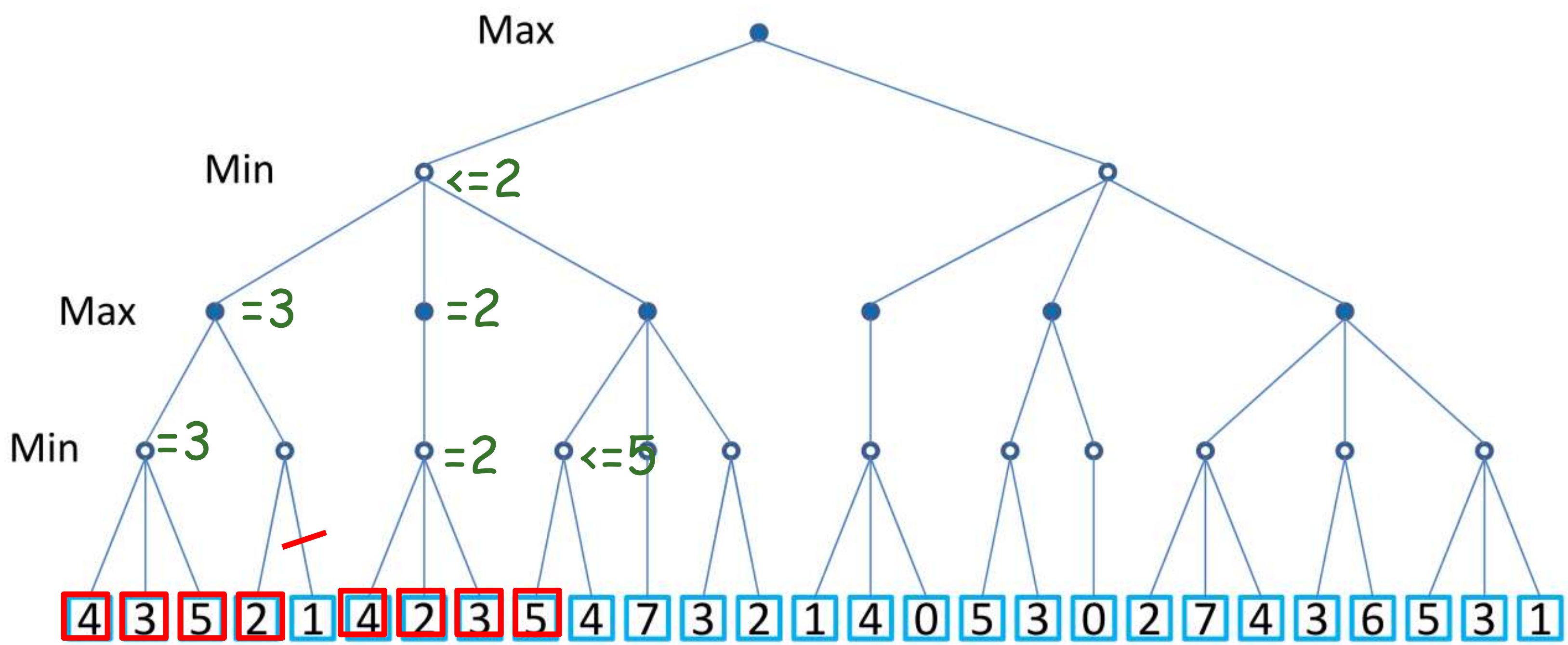
Alpha-Beta Pruning: Example 3



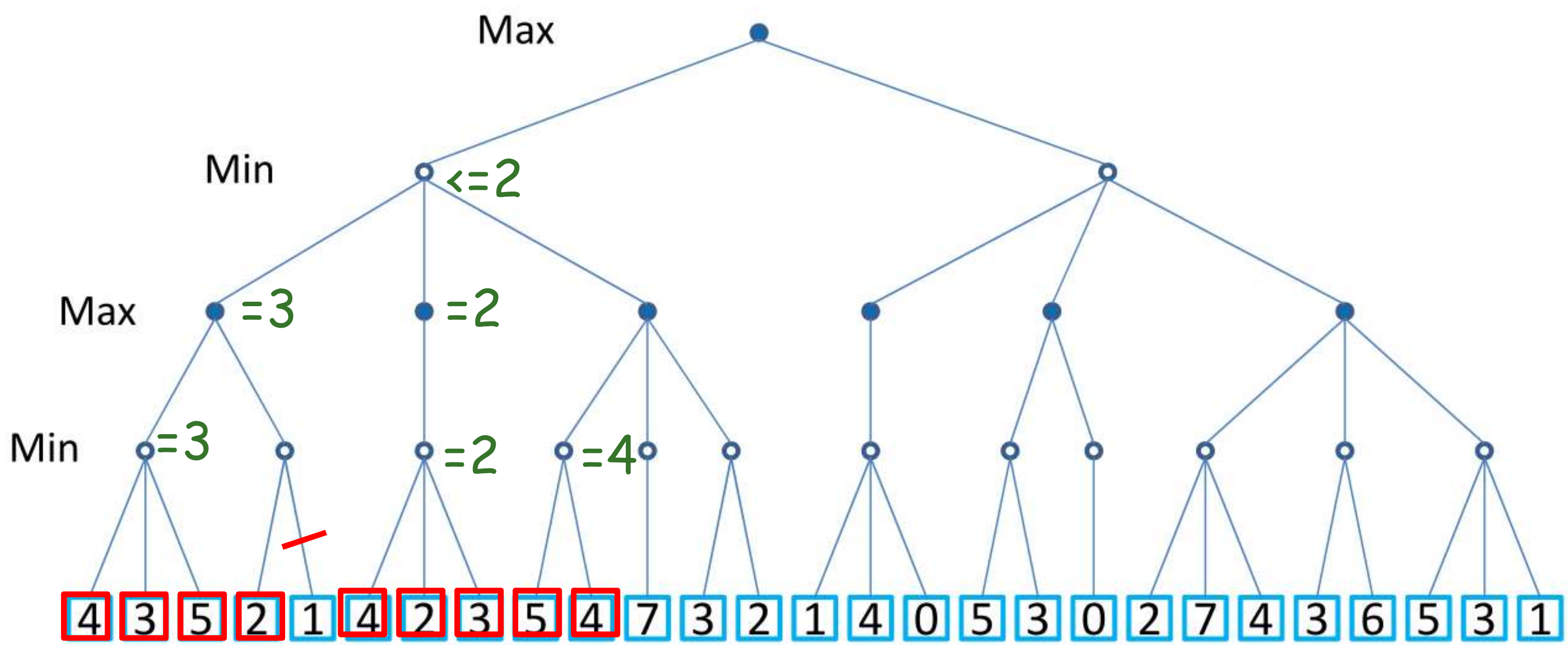
Alpha-Beta Pruning: Example 3



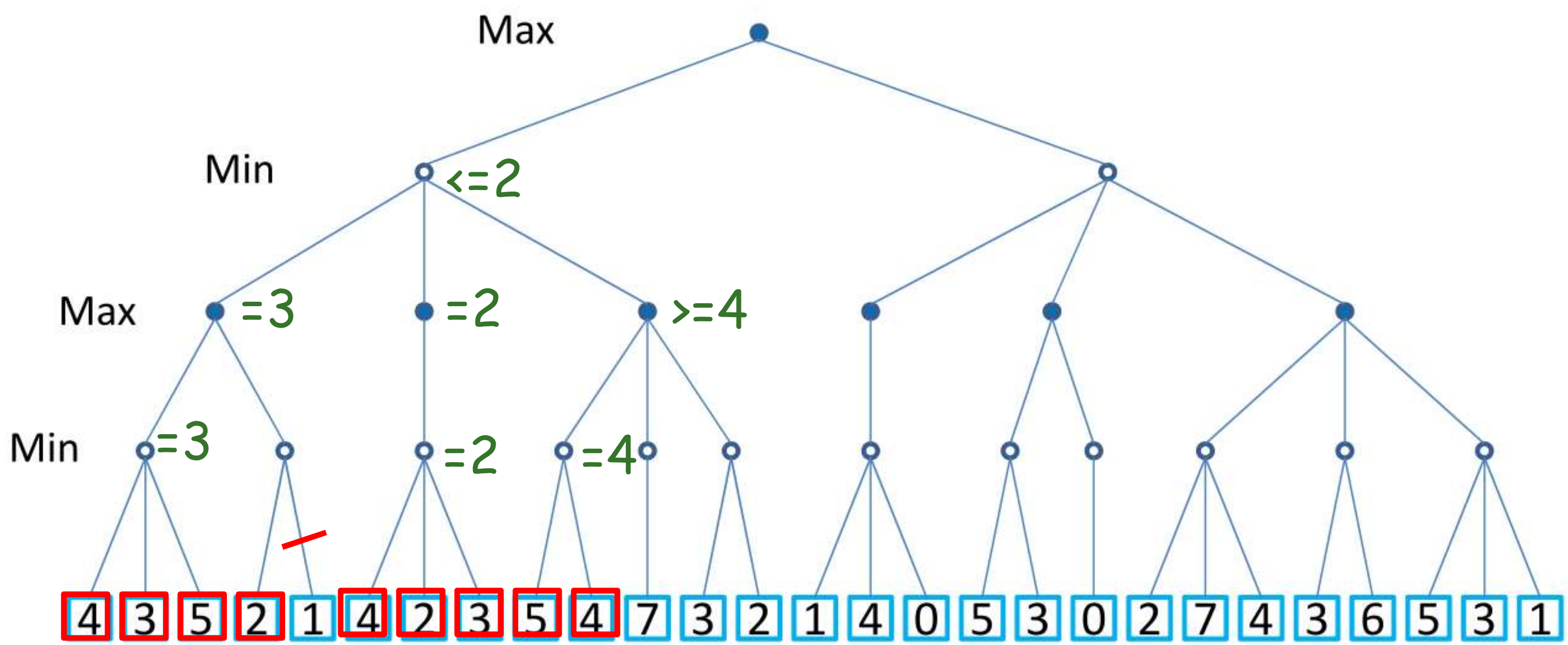
Alpha-Beta Pruning: Example 3



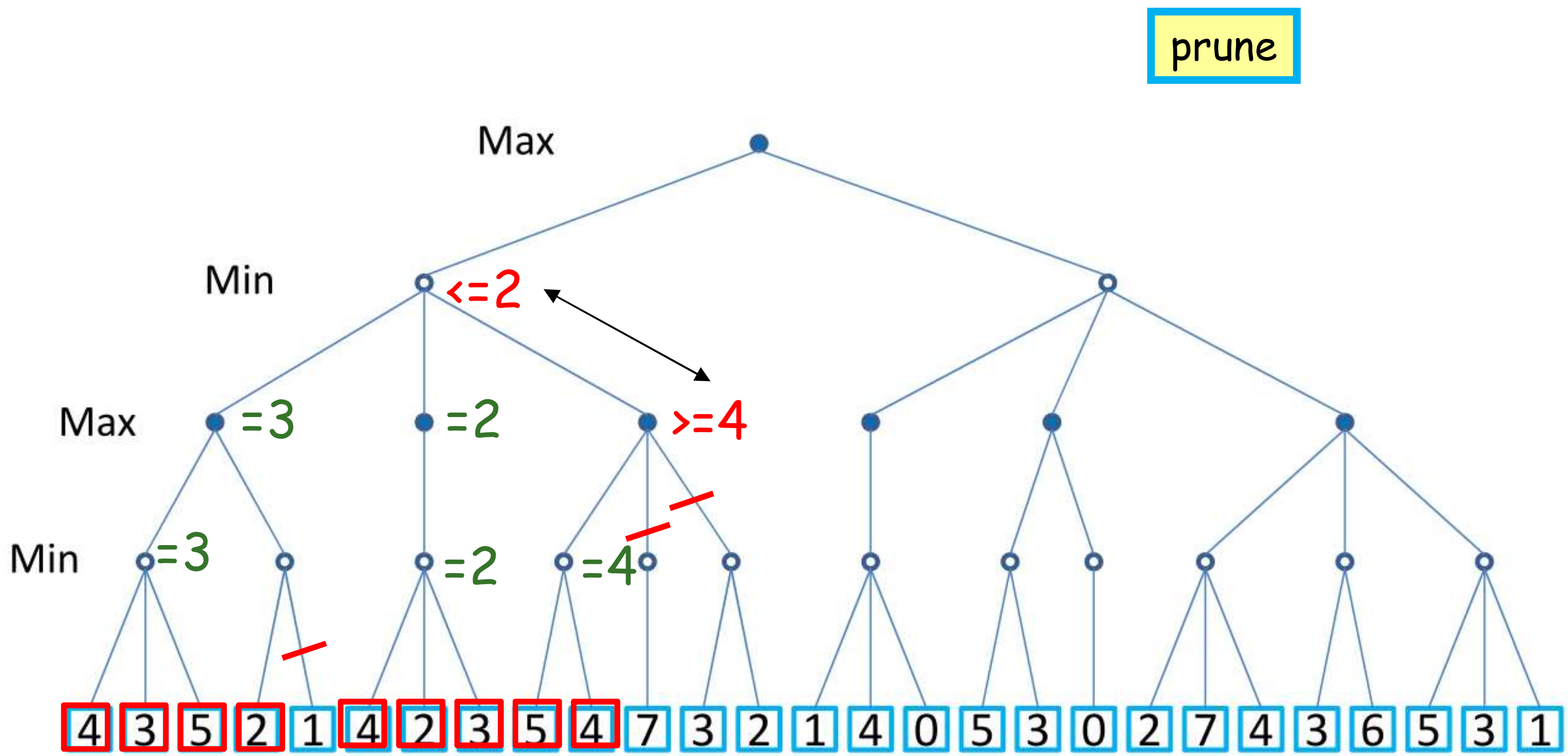
Alpha-Beta Pruning: Example 3



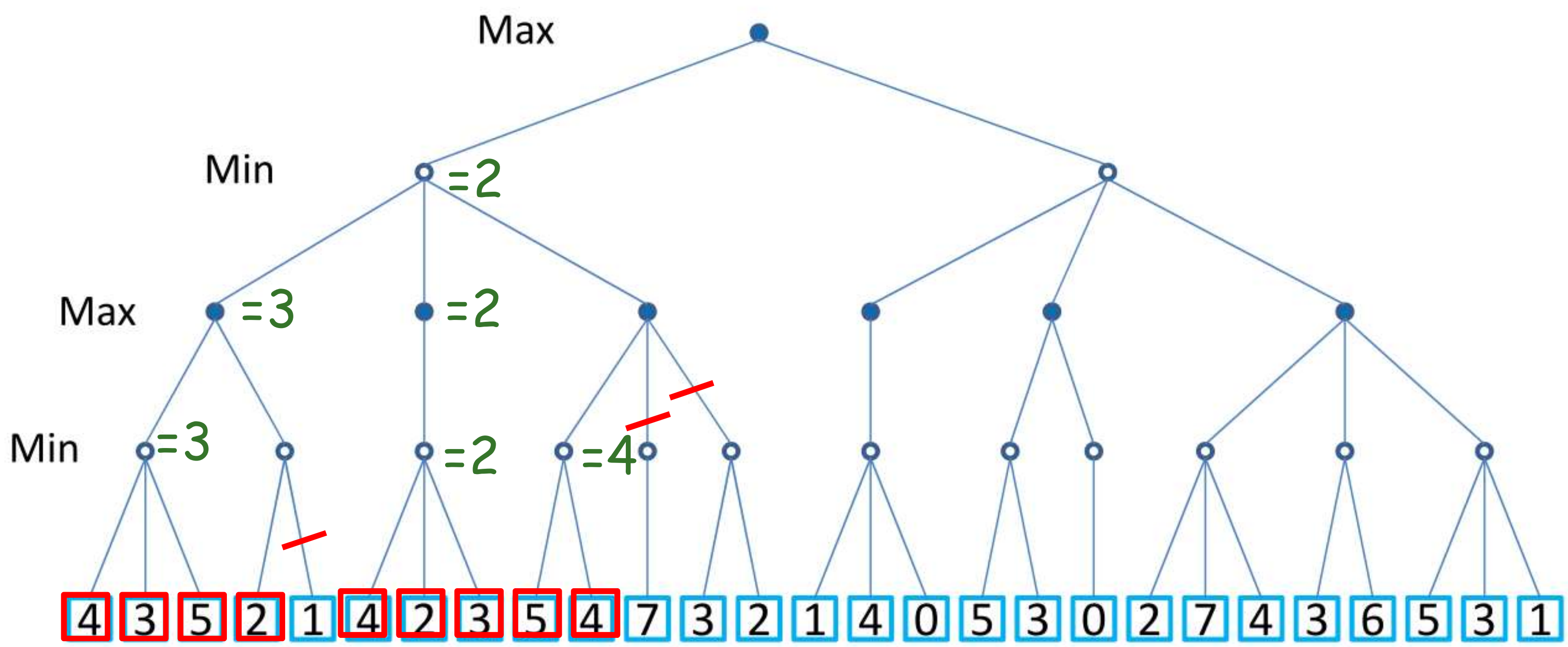
Alpha-Beta Pruning: Example 3



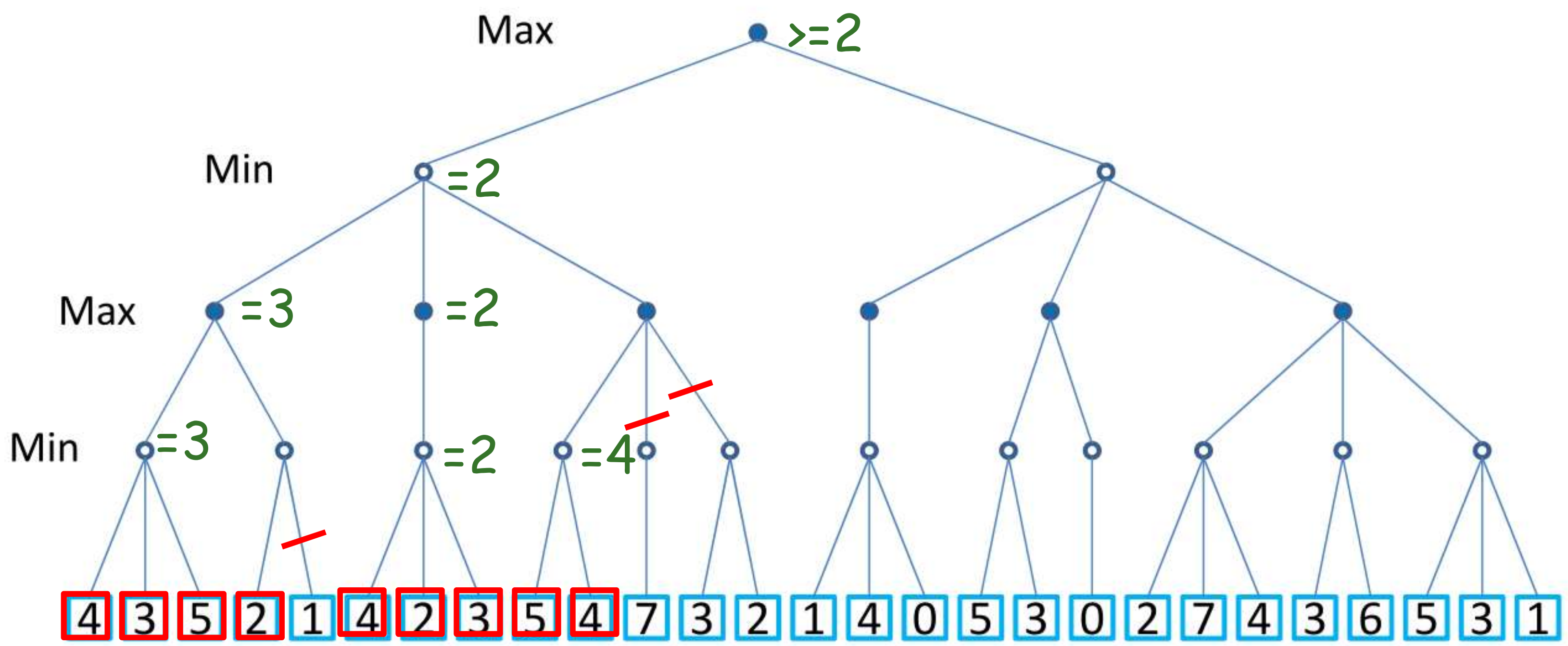
Alpha-Beta Pruning: Example 3



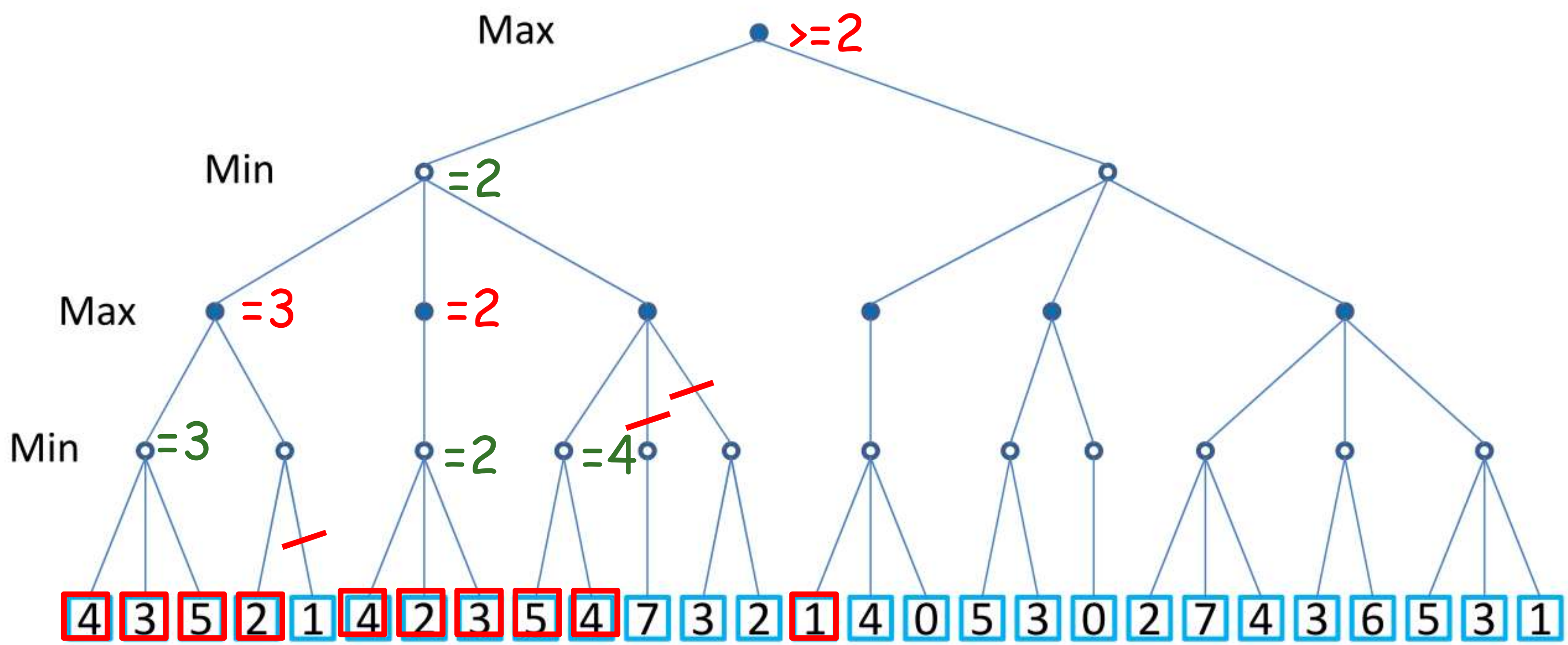
Alpha-Beta Pruning: Example 3



Alpha-Beta Pruning: Example 3



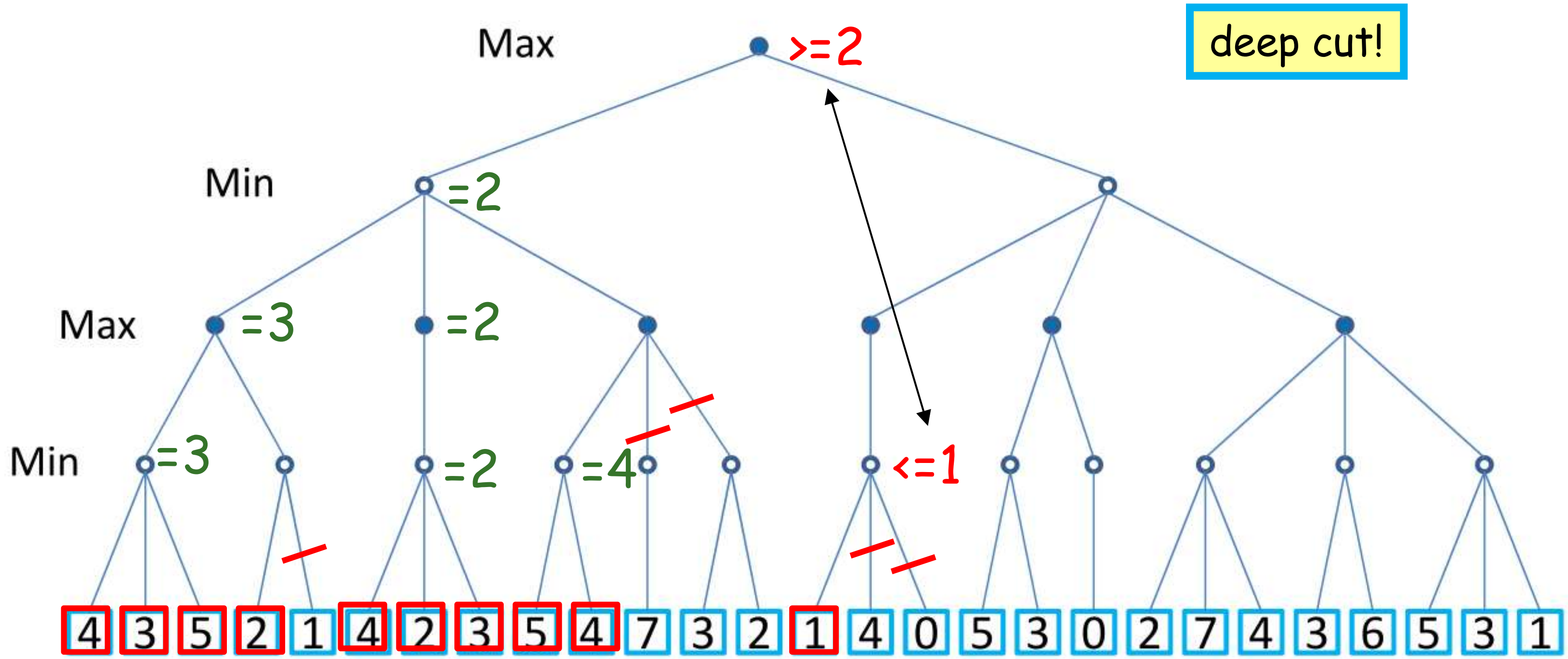
Alpha-Beta Pruning: Example 3



Alpha-Beta Pruning: Example 3

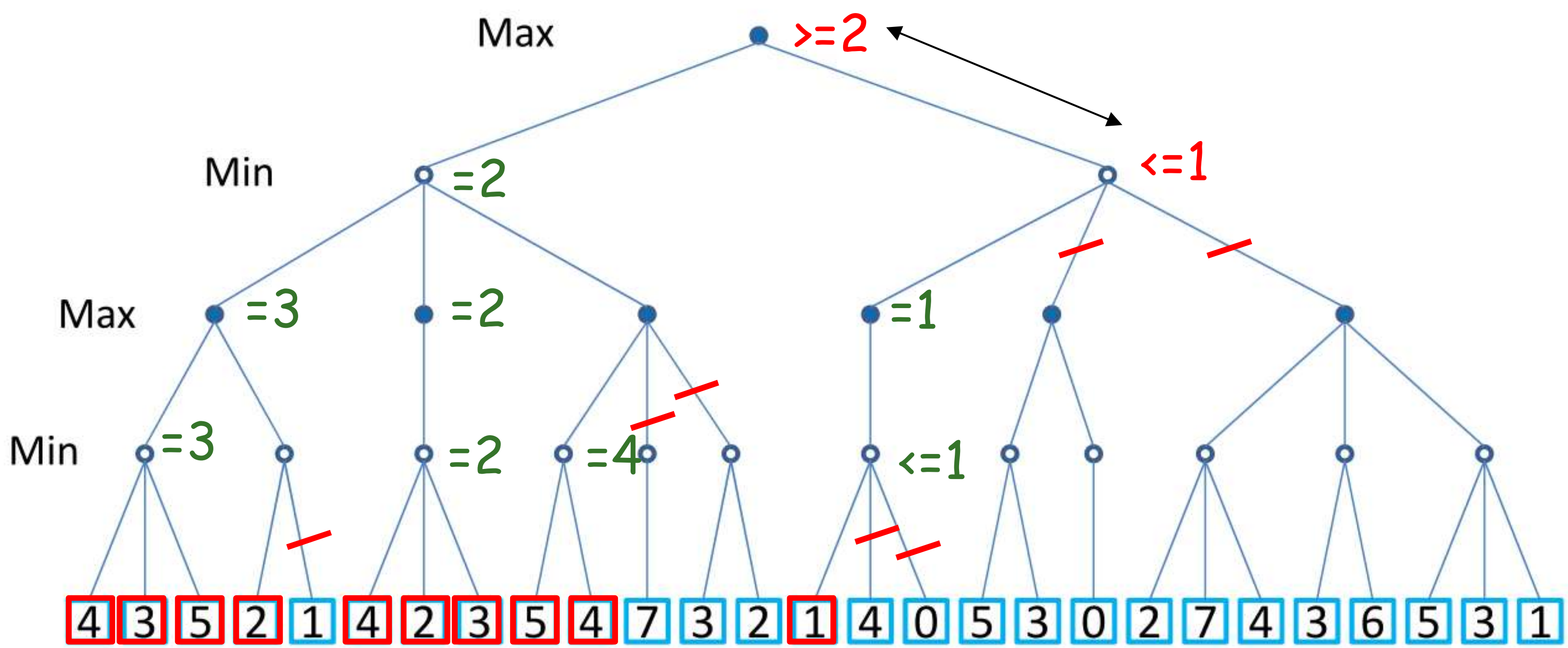
prune

deep cut!



Alpha-Beta Pruning: Example 3

prune



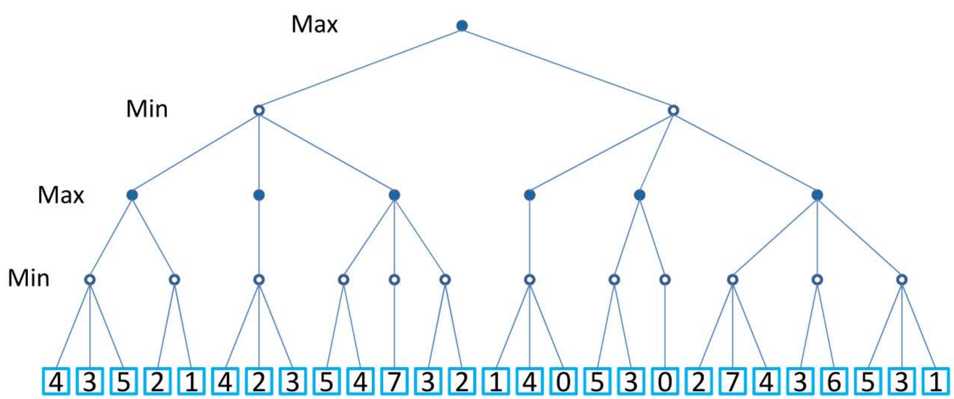
10 nodes explored out of 27

Efficiency of Alpha-Beta Pruning

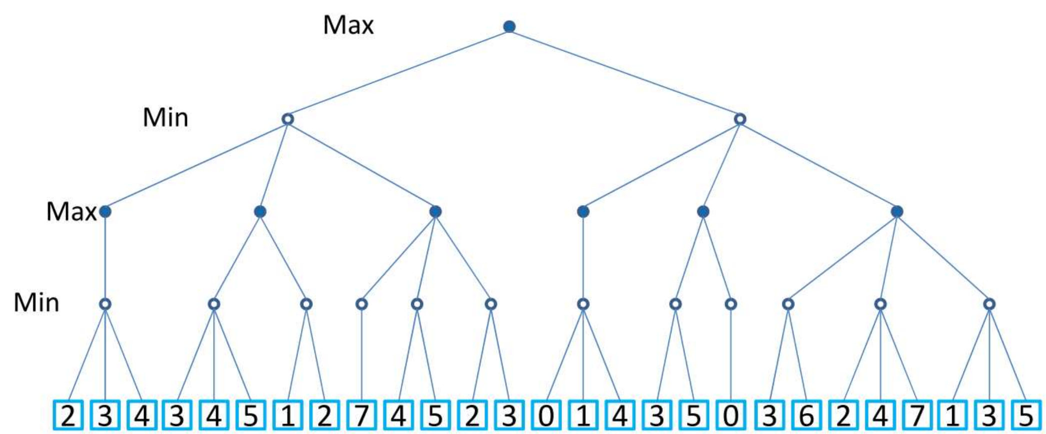
- Depends on the order the siblings

- In worst case:
  - alpha-beta provides no pruning
- In best case:
  - branching factor is reduced to its square root

Alpha-Beta: Best ordering



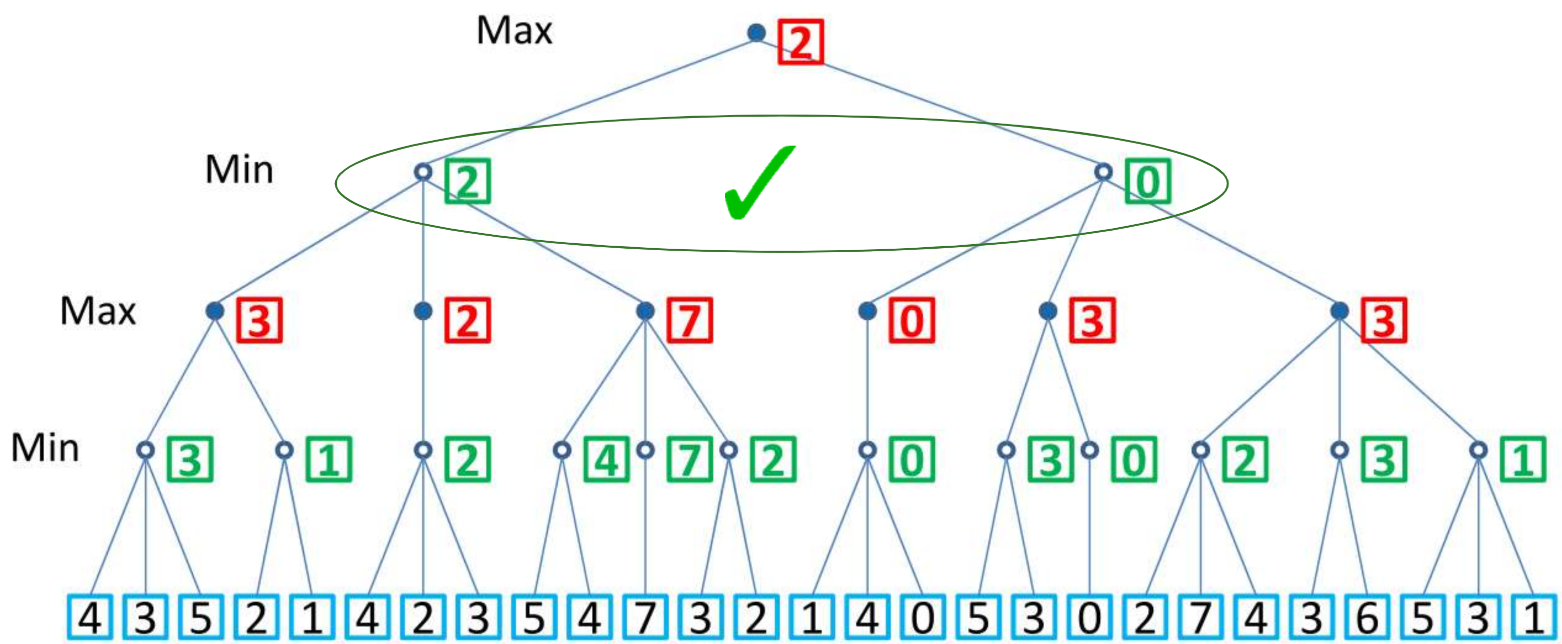
Original (arbitrary) game tree



Best ordering for alpha-beta

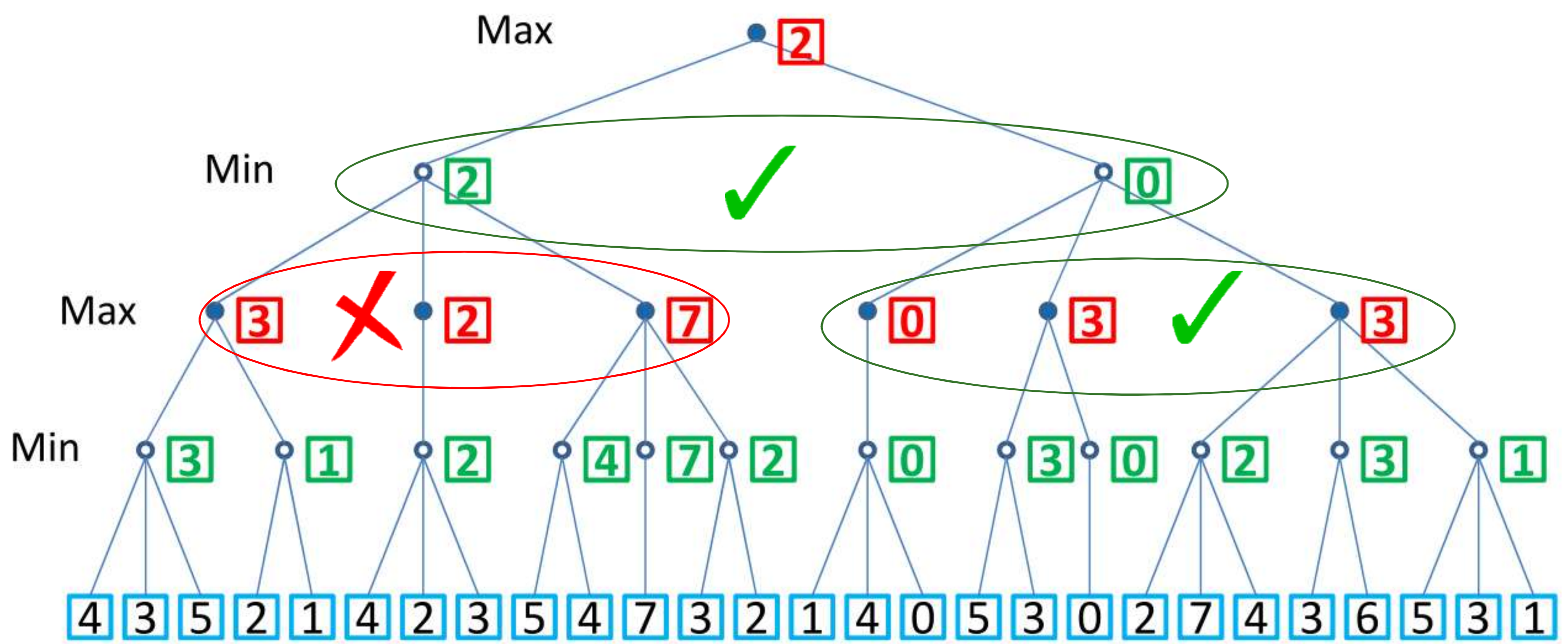
Alpha-Beta: Best ordering

- best ordering:
  1. children of MIN : smallest node first
  2. children of MAX: largest node first



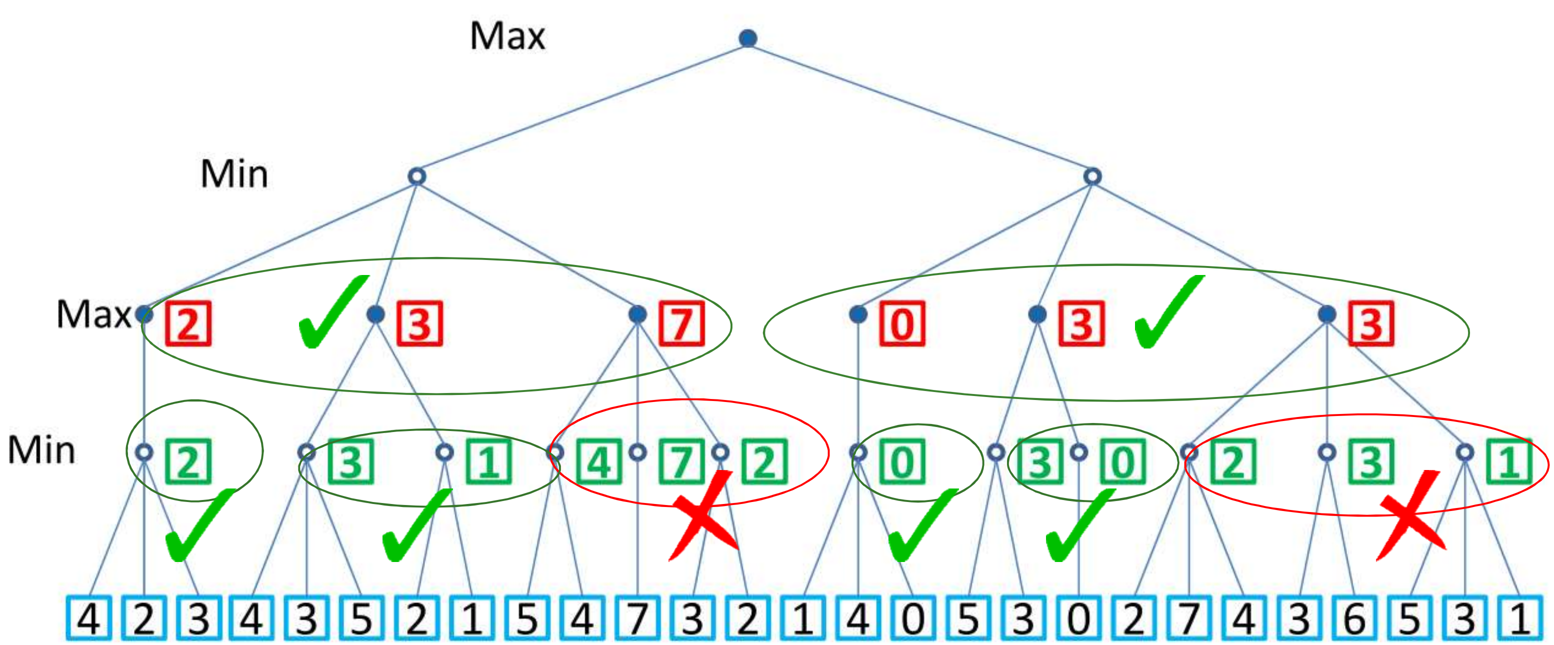
Alpha-Beta: Best ordering

- best ordering:
  1. children of MIN : smallest node first
  2. children of MAX: largest node first

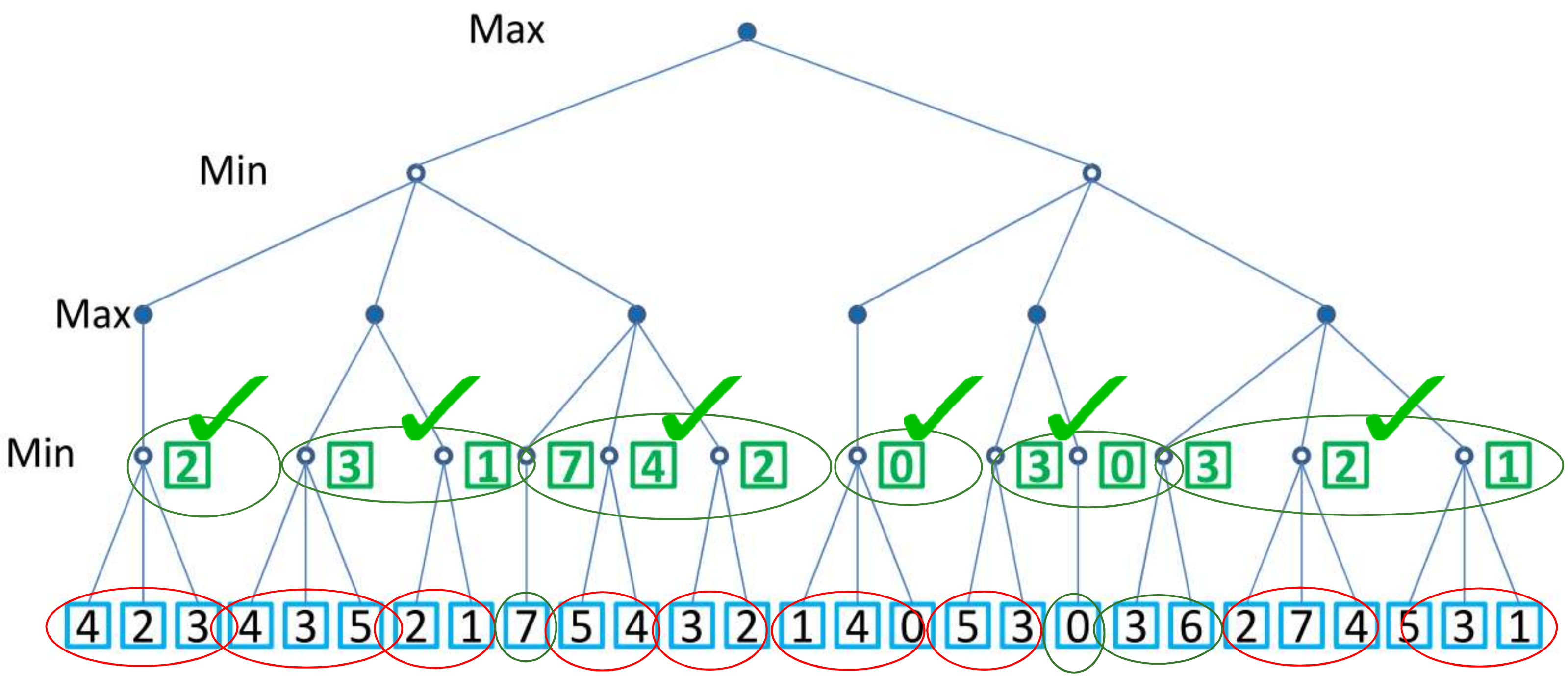


Alpha-Beta: Best ordering

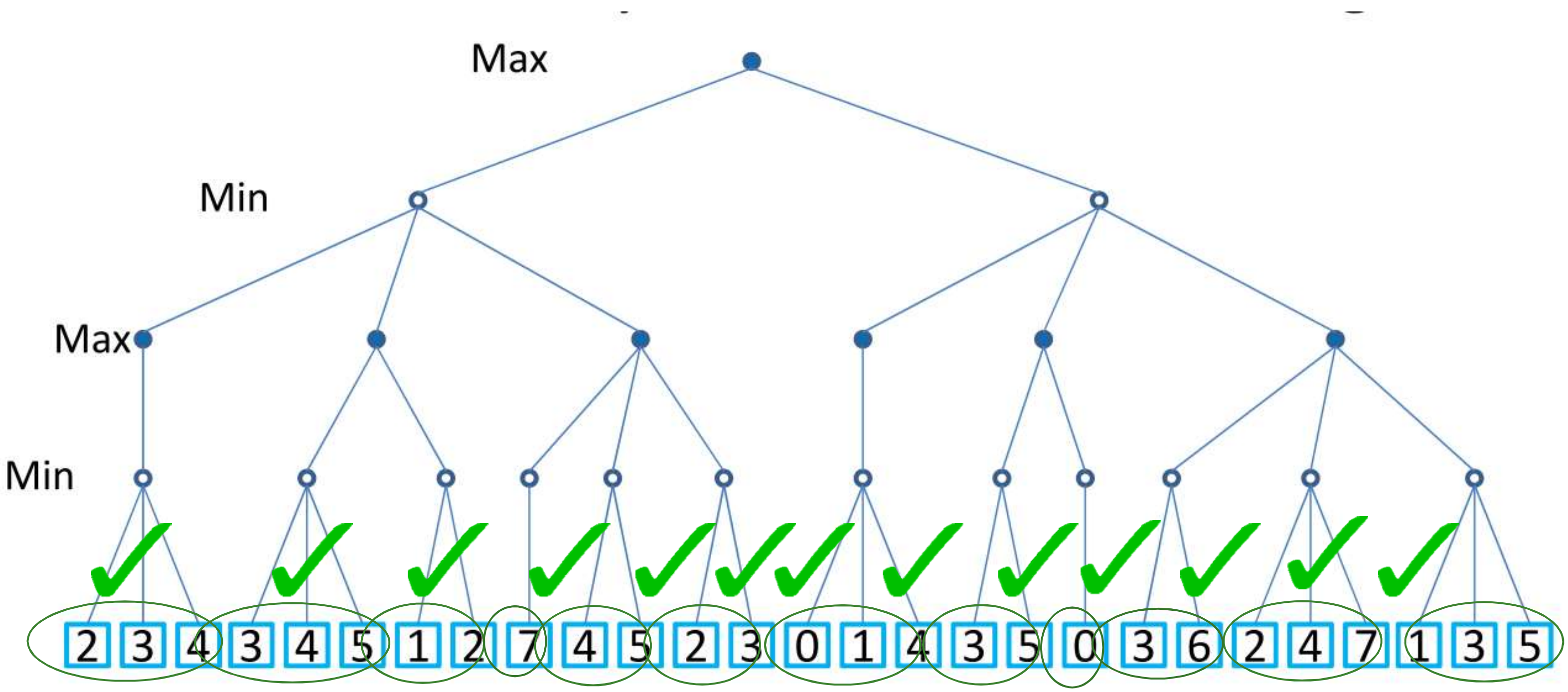
- best ordering:
  1. children of MIN : smallest node first
  2. children of MAX: largest node first



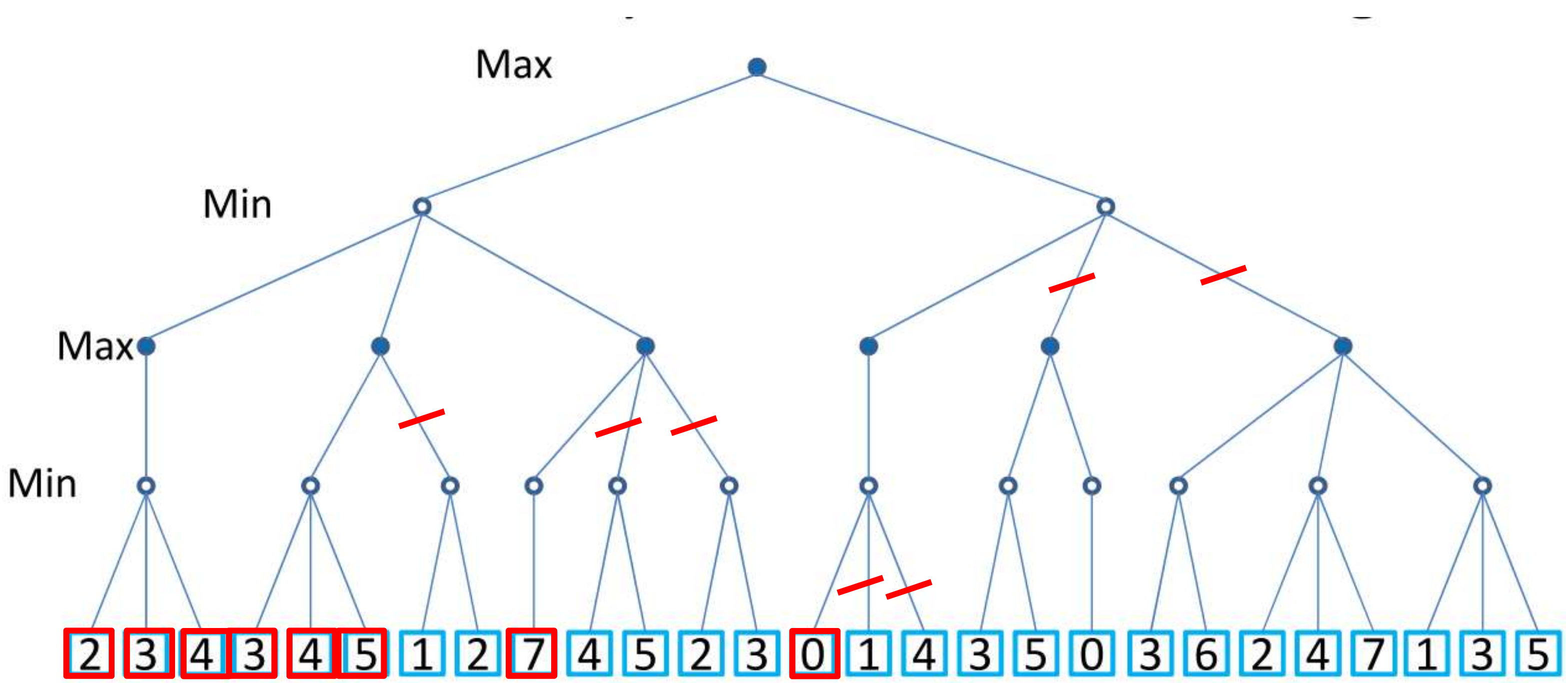
Alpha-Beta: Best ordering



Alpha-Beta: Best ordering



Alpha-Beta: Best ordering

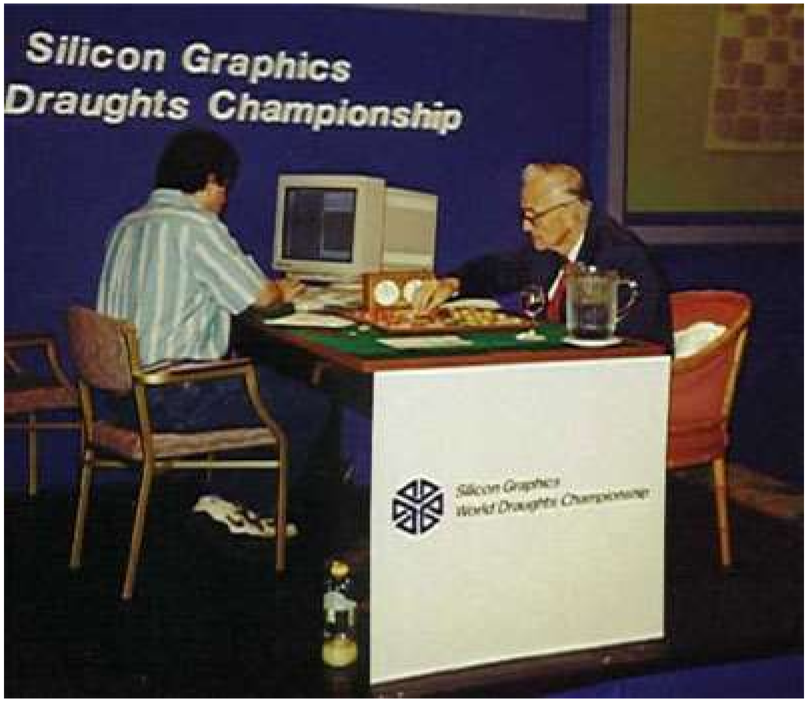


8 nodes explored out of 27

Today

- State Space Search for Game Playing
  - MiniMax
  - Alpha-beta pruning
- Where we are today

1992-1994 - Checkers: Tinsley vs. Chinook



Marion Tinsley
World champion
for over 40 years

VS

Chinook
Developed by
Jonathan Schaeffer,
professor at the U. of Alberta

1992: Tinsley beat Chinook in 4 games to 2, with 33 draws.
1994: 6 draws

In 2007, Schaeffer announced that checkers was solved,
and anyone playing against Chinook would only be able to draw, never win.

Play against Chinook: http://games.cs.ualberta.ca/cgi-bin/player.cgi?nodemo

1997 - Othello: Murakami vs. Logistello

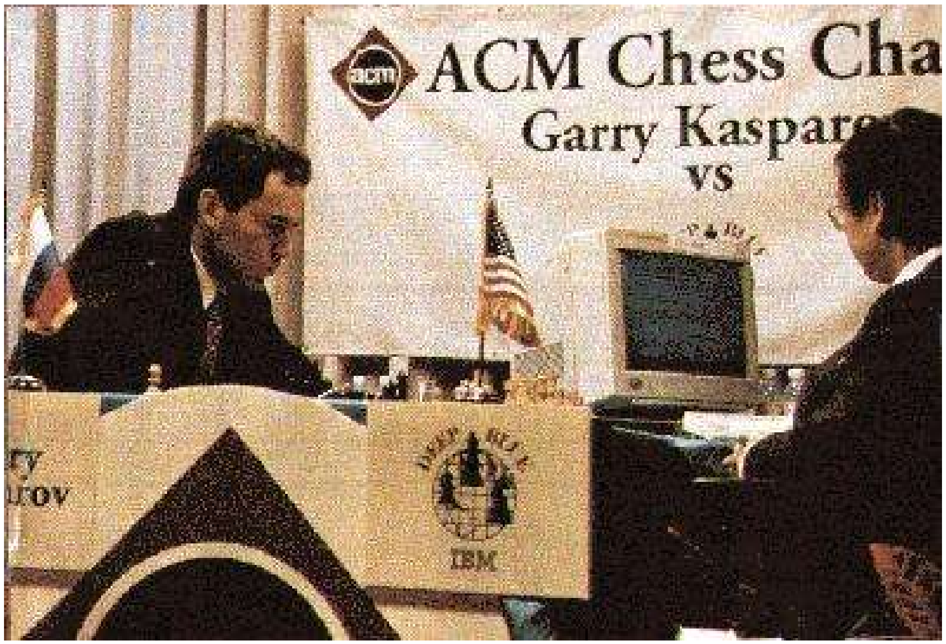
Takeshi Murakami
World Othello champion

VS

Logistello
developed by Michael Buro
runs on a standard PC
https://skatgame.net/mburo/log.html
(including source code)

Logistello beat Murakami by 6 games to 0

1997- Chess: Kasparov vs. Deep Blue



Garry Kasparov
50 billion neurons
2 positions/sec
VS
Deep Blue
32 RISC processors
+ 256 VLSI chess engines
200,000,000 pos/sec

Deep Blue wins by 3 wins, 1 loss, and 2 draws

2003 - Chess: Kasparov vs. Deep Junior

Garry Kasparov
still 50 billion neurons
still 2 positions/sec

VS

Deep Junior
8 CPU, 8 GB RAM, Win 2000
2,000,000 pos/sec
Available at $100

Match ends in a 3/3 tie!

2016 – Go: AlphaGo vs Lee Se-dol

- GO was always considered a much harder game to automate than chess because of its very o high a branching factor (35 for chess vs 250 for Go!)
- In 2016, AlphaGo beat Lee Sedol in a five-game match of GO.
- In 2017 AlphaGo beat Ke Jie, the world No.1 ranked player at the time

- uses a Monte Carlo tree search algorithm to find its moves based on knowledge previously "learned" by deep learning

https://www.theverge.com/2016/3/15/11213518/alphago-deepmind-go-match-5-result

Today

- State Space Search for Game Playing
  - MiniMax
  - Alpha-beta pruning
- Where we are today