Game Playing

Xiaojin Zhu

jerryzhu@cs.wisc.edu

Computer Sciences Department

University of Wisconsin, Madison

[based on slides from A. Moore http://www.cs.cmu.edu/~awm/tutorials, C. Dyer, and J. Skrentny]

Sadly, not these games…

Overview

- two-player zero-sum discrete finite deterministic game of perfect information
- Minimax search
- Alpha-beta pruning
- Large games
- two-player zero-sum discrete finite NON-deterministic game of perfect information

Two-player zero-sum discrete finite deterministic games of perfect information

Definitions:

- Zero-sum: one player's gain is the other player's loss. Does not mean *fair*.
- Discrete: states and decisions have discrete values
- Finite: finite number of states and decisions
- Deterministic: no coin flips, die rolls – no chance
- Perfect information: each player can see the complete game state. No simultaneous decisions.

Which of these are: Two-player zero-sum discrete finite deterministic games of perfect information?

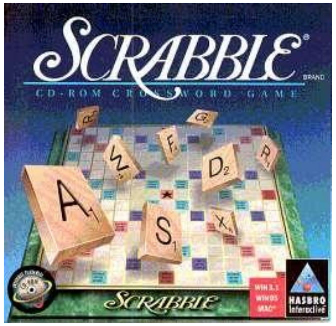
Zero-sum: one player's gain is the other player's loss. Does not mean *fair*.

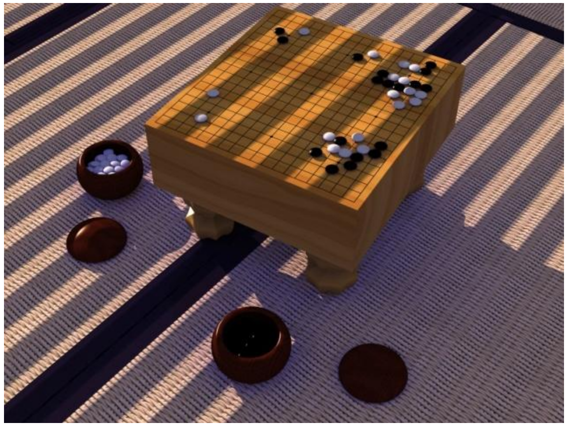
Discrete: states and decisions have discrete values

Finite: finite number of states and decisions

Deterministic: no coin flips, die rolls – no chance

Perfect information: each player can see the complete game state. No simultaneous decisions.





[Shamelessly copied from Andrew Moore]

Which of these are: Two-player zero-sum discrete finite deterministic games of perfect information?

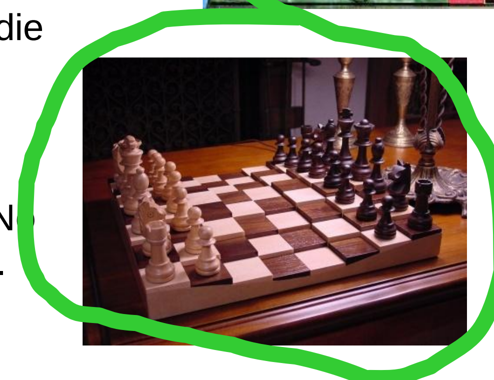
Zero-sum: one player’s gain is the other player’s loss. Does not mean *fair*.

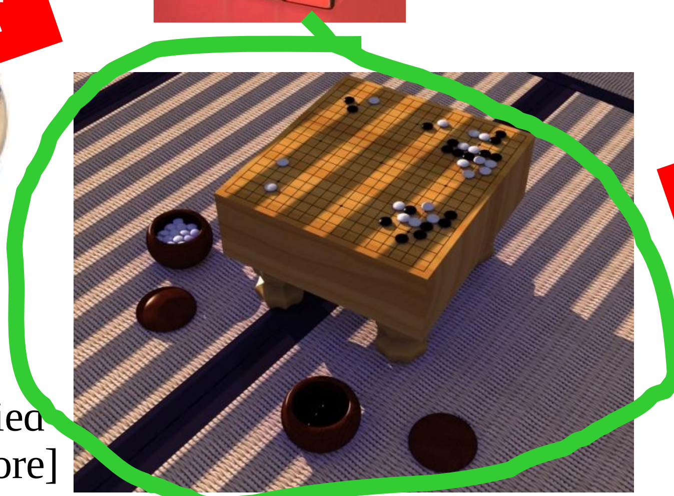
Discrete: states and decisions have discrete values

Finite: finite number of states and decisions

Deterministic: no coin flips, die rolls – no chance

Perfect information: each player can see the complete game state. No simultaneous decisions.





[Shamelessly copied from Andrew Moore]

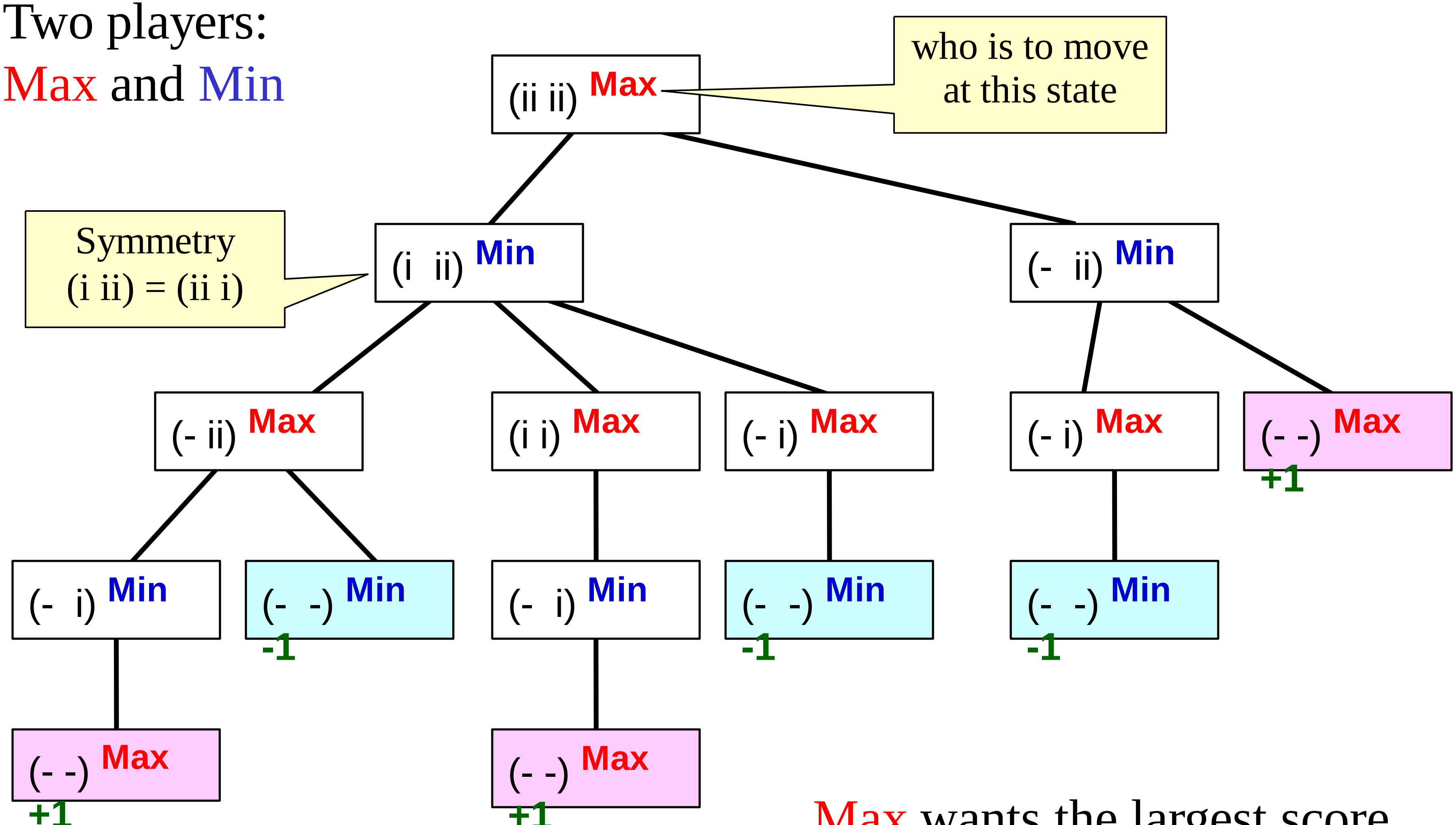
II-Nim: Max simple game

- There are 2 piles of sticks. Each pile has 2 sticks.
- Each player takes one or more sticks from one pile.
- The player who takes the last stick loses.

(ii, ii)

The game tree for II-Nim

Two players:
Max and Min

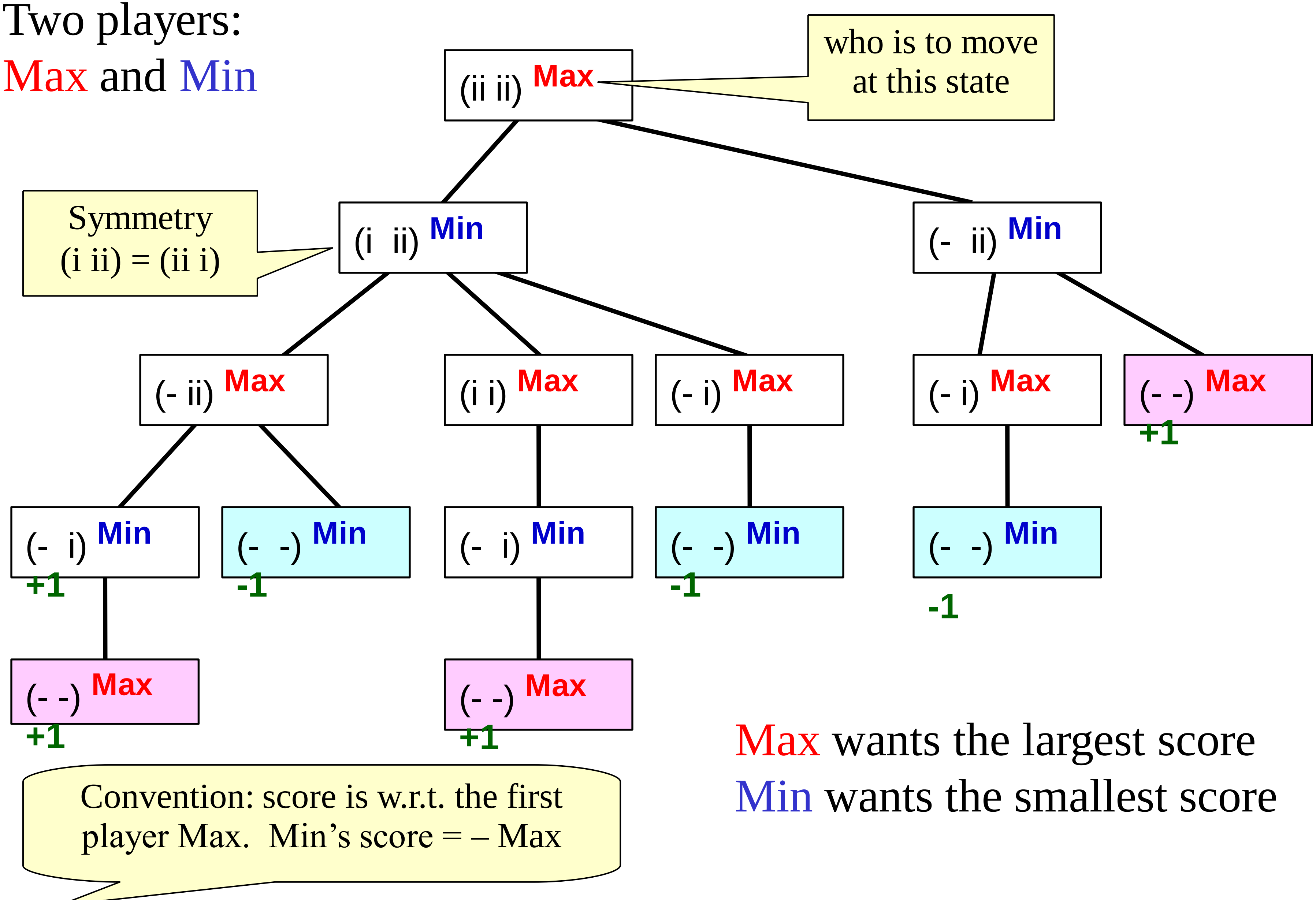


Convention: score is w.r.t. the first player Max. Min's score = – Max

Max wants the largest score
Min wants the smallest score

The game tree for II-Nim

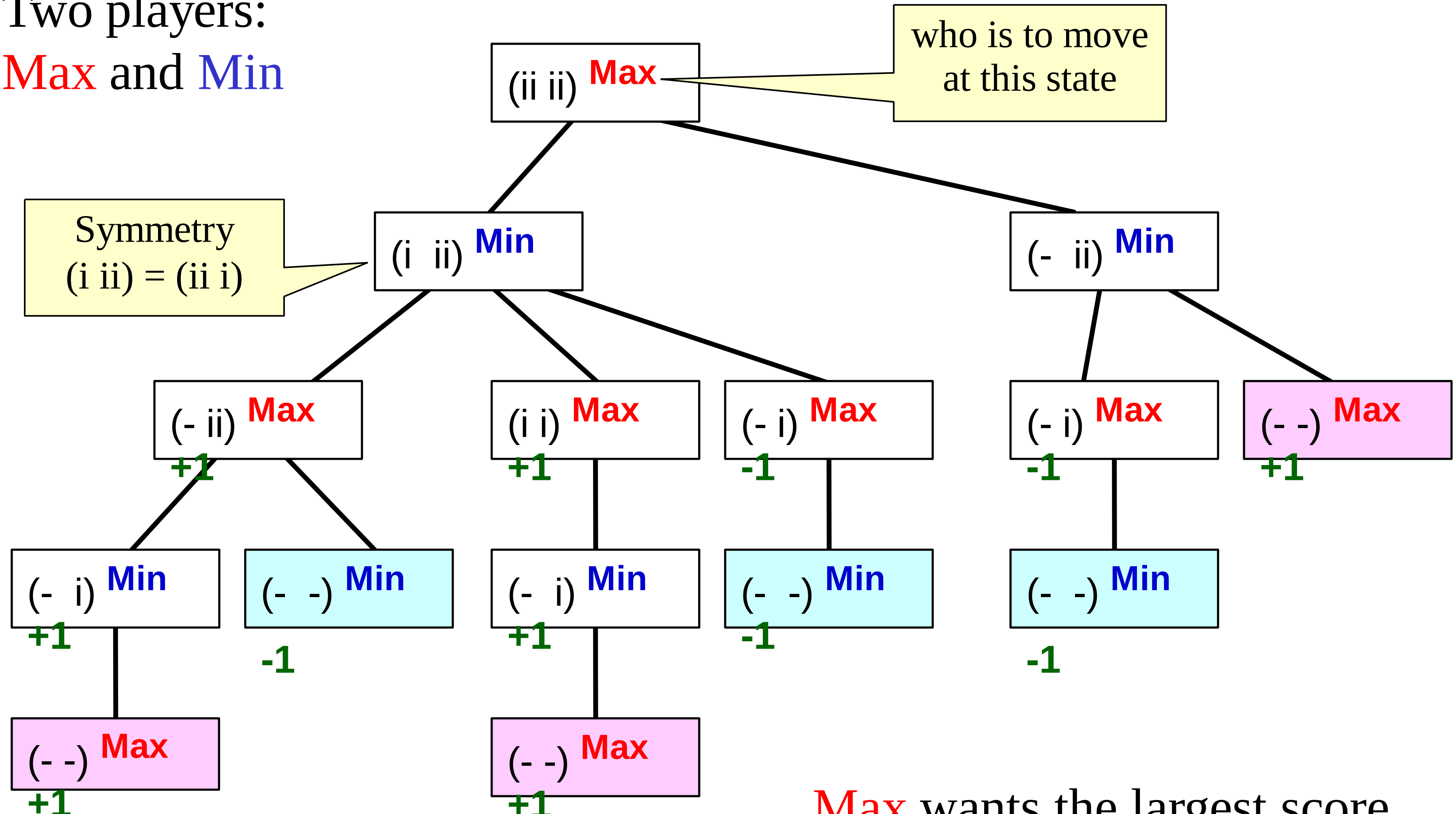
Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

The game tree for II-Nim

Two players:
Max and Min



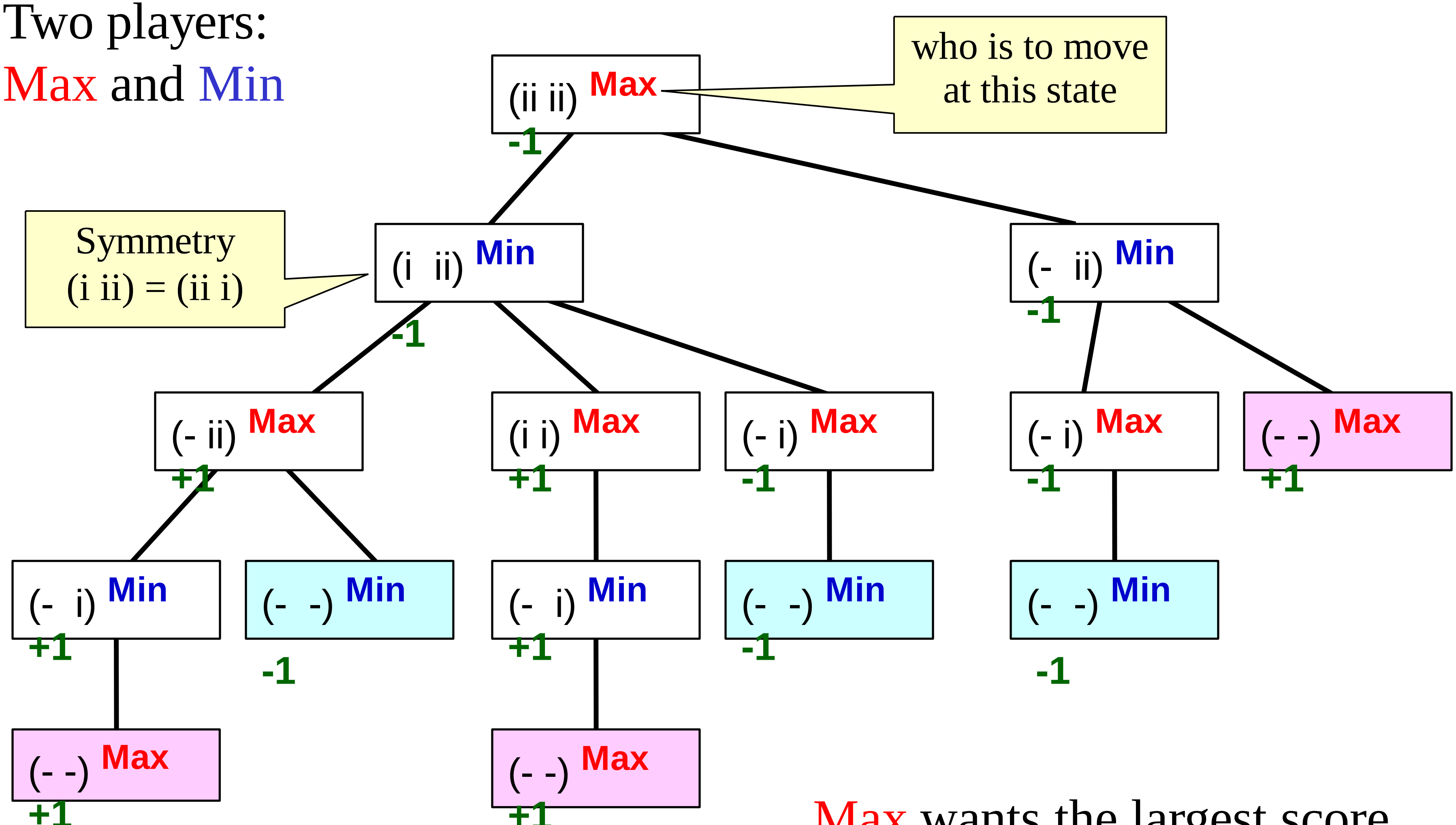
Convention: score is w.r.t. the first player Max. Min's score = – Max

Max wants the largest score
Min wants the smallest score

The game tree for II-Nim

Two players:

Max and Min



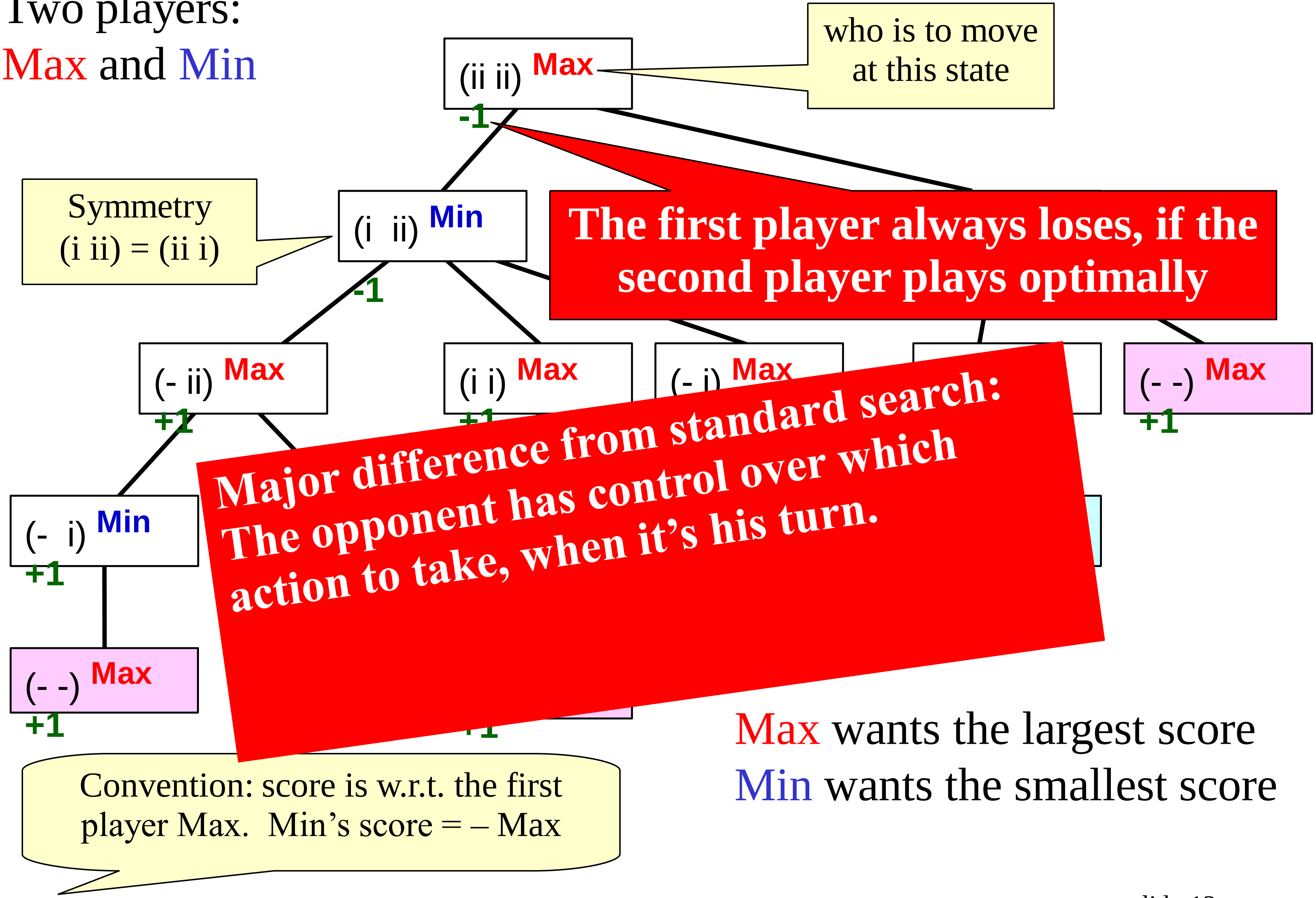
Convention: score is w.r.t. the first player Max. Min's score = – Max

Max wants the largest score

Min wants the smallest score

The game tree for II-Nim

Two players:
Max and Min



Max wants the largest score
Min wants the smallest score

Game theoretic value

- Game theoretic value (a.k.a. minimax value) of a node = the score of the terminal node that will be reached if both players play optimally.
- = The numbers we filled in.
- Computed bottom up
  - In Max's turn, take the max of the children (Max will pick that maximizing action)
  - In Min's turn, take the min of the children (Min will pick that minimizing action)
- Implemented as a modified version of DFS: minimax algorithm

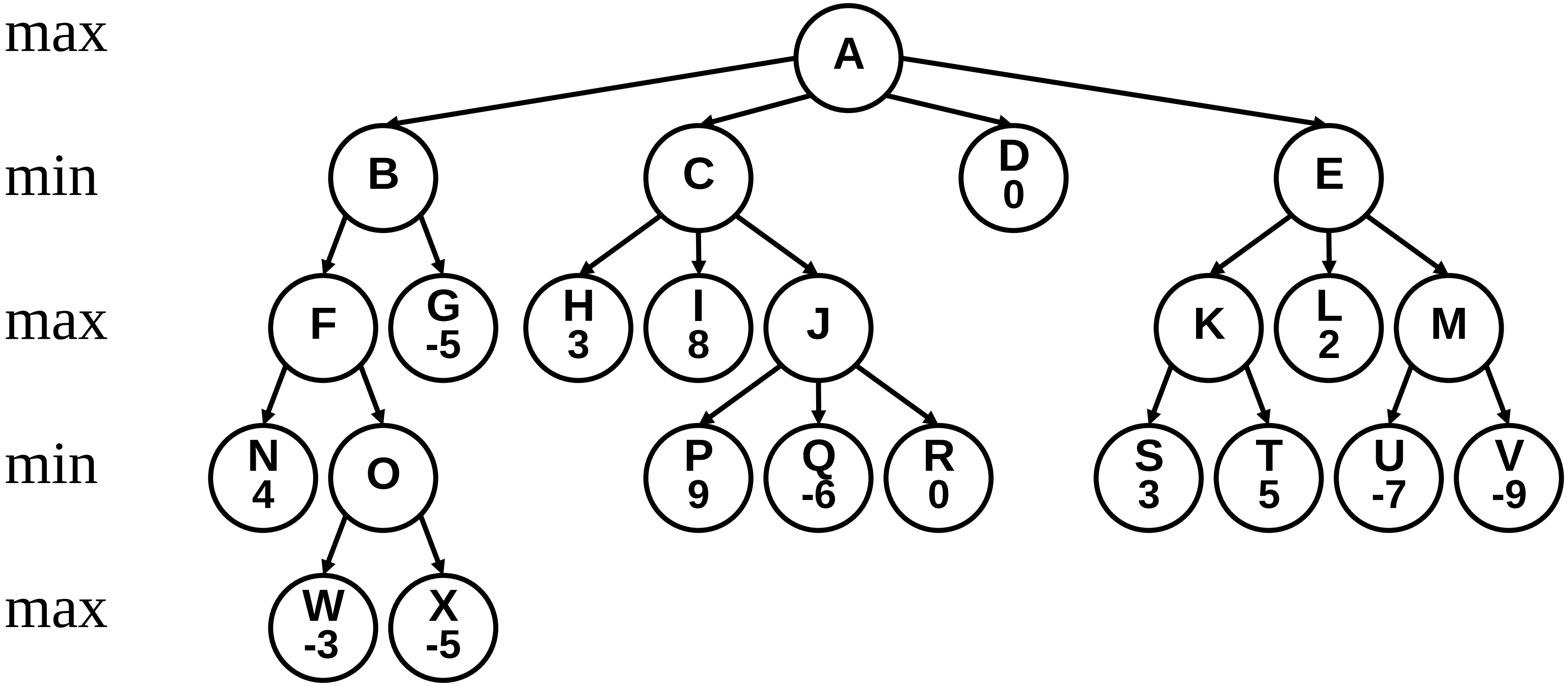
Minimax algorithm

```
function Max-Value(s)
inputs:
    s: current state in game, Max about to play
output: best-score (for Max) available from s

    if ( s is a terminal state )
    then return ( terminal value of s )
    else
            α := – ∞
            for each s' in Succ(s)
                α := max( α , Min-value(s'))
    return α
```

```
function Min-Value(s)
output: best-score (for Min) available from s

    if ( s is a terminal state )
    then return ( terminal value of s)
    else
            β := ∞
            for each s' in Succs(s)
                β := min( β , Max-value(s'))
    return β
```

- Time complexity?
- Space complexity?

Minimax example

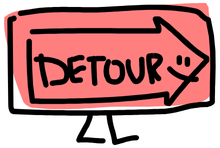


Minimax algorithm

```
function Max-Value(s)
inputs:
    s: current state in game, Max about to play
output: best-score (for Max) available from s

    if ( s is a terminal state )
    then return ( terminal value of s )
    else
            α := – ∞
            for each s' in Succ(s)
                α := max( α , Min-value(s'))
    return α
```

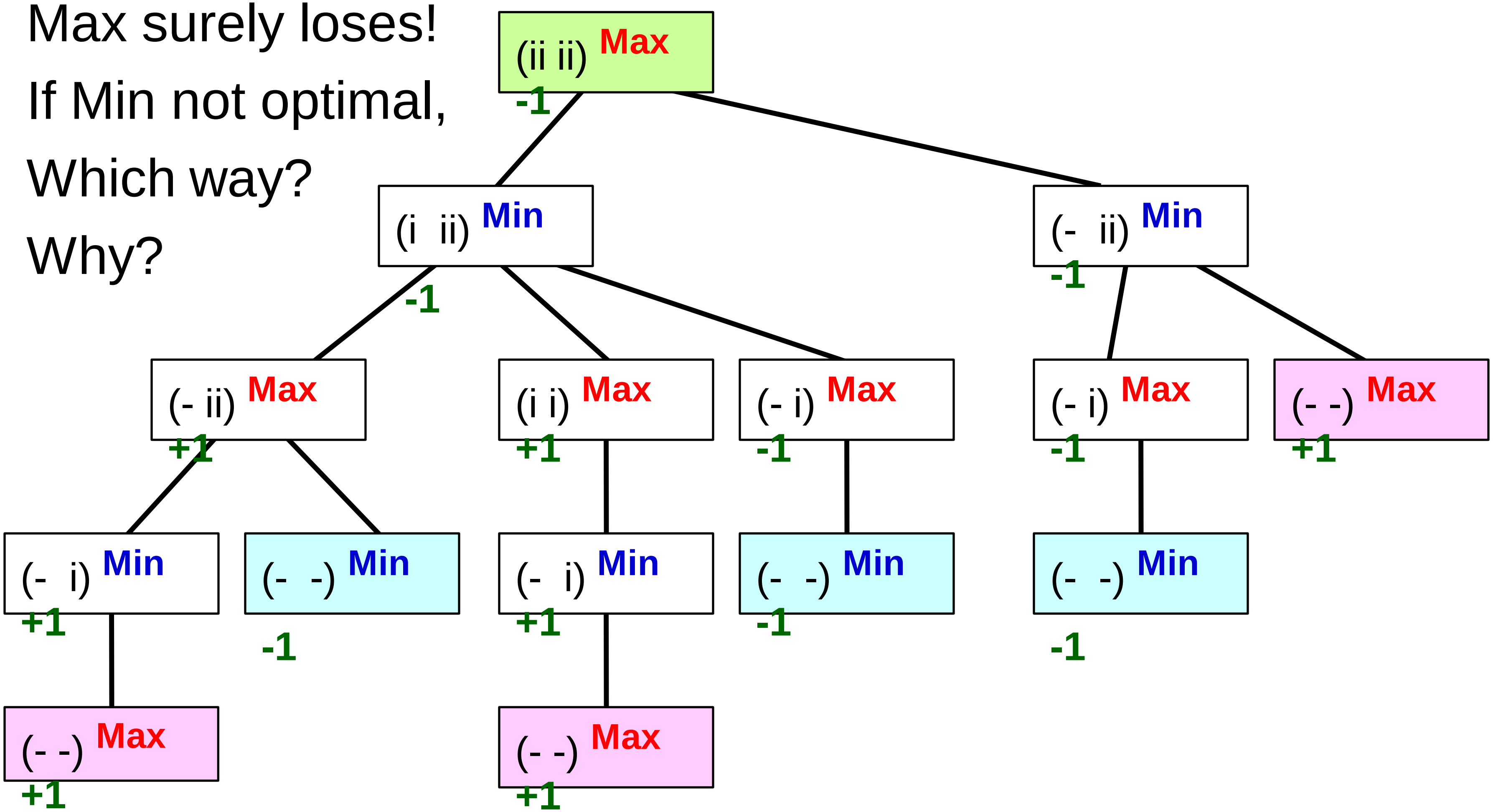
```
function Min-Value(s)
output: best-score (for Min) available from s

    if ( s is a terminal state )
    then return ( terminal value of s)
    else
            β := ∞
            for each s' in Succs(s)
                β := min( β , Max-value(s'))
    return β
```

- Time complexity?
  O(b^m) ← bad
- Space complexity?
  O(bm)



Against a dumber opponent?

- Max surely loses!
- If Min not optimal,
- Which way?
- Why?

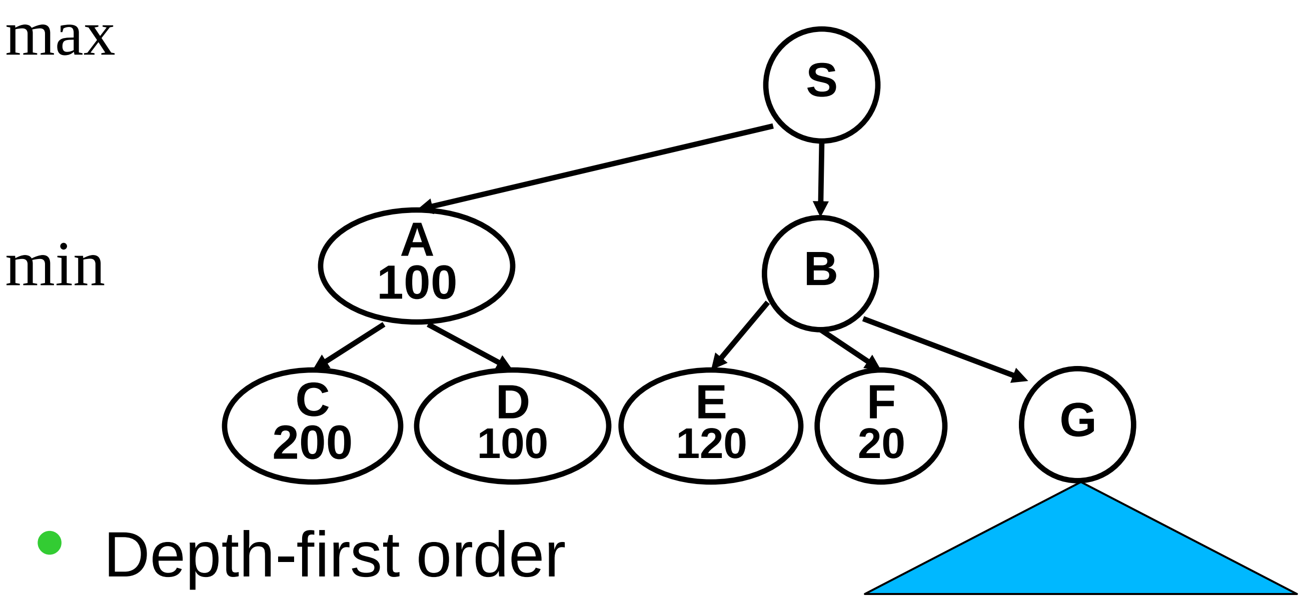


Next: alpha-beta pruning

Gives the same game theoretic values as minimax, but prunes part of the game tree.

"If you have an idea that is surely bad, don't take the time to see how truly awful it is." -- Pat Winston

Alpha-Beta Motivation



- Depth-first order
- After returning from A, Max can get at least 100 at S
- After returning from F, Max can get at most 20 at B
- At this point, Max losts interest in B
- There is no need to explore G. The subtree at G is pruned. Saves time.

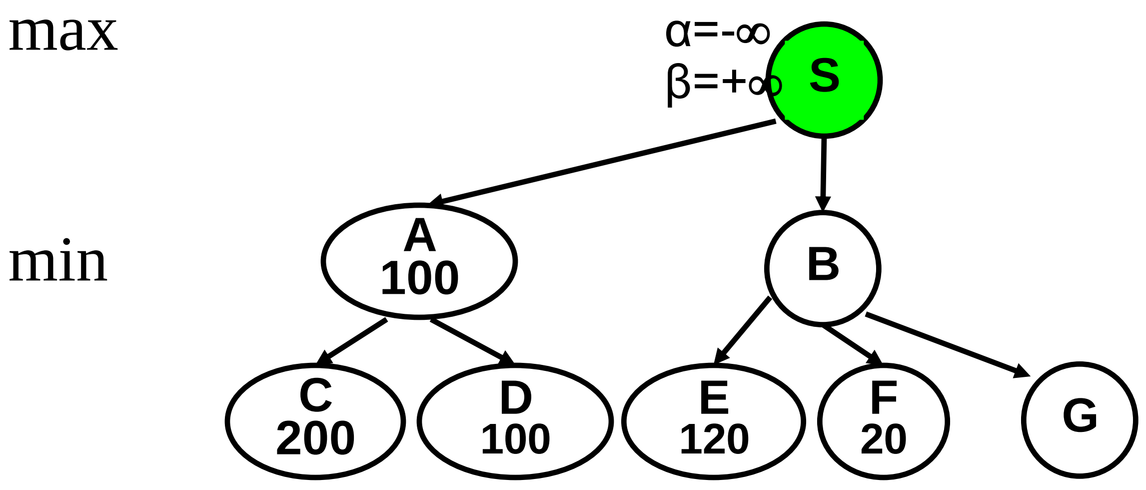
Alpha-beta pruning

```
function Max-Value (s,α,β)
inputs:
    s: current state in game, Max about to play
    α: best score (highest) for Max along path to s
    β: best score (lowest) for Min along path to s
output: min(β , best-score (for Max) available from s)

    if ( s is a terminal state )
    then return ( terminal value of s )
    else for each s’ in Succ(s)
      α := max( α , Min-value(s’,α,β))
      if ( α ≥ β ) then return β   /* alpha pruning */
    return α
```

```
function Min-Value(s,α,β)
output: max(α , best-score (for Min) available from s )

    if ( s is a terminal state )
    then return ( terminal value of s)
    else for each s’ in Succs(s)
      β := min( β , Max-value(s’,α,β))
            if (α ≥ β ) then return α   /* beta pruning */
    return β
```

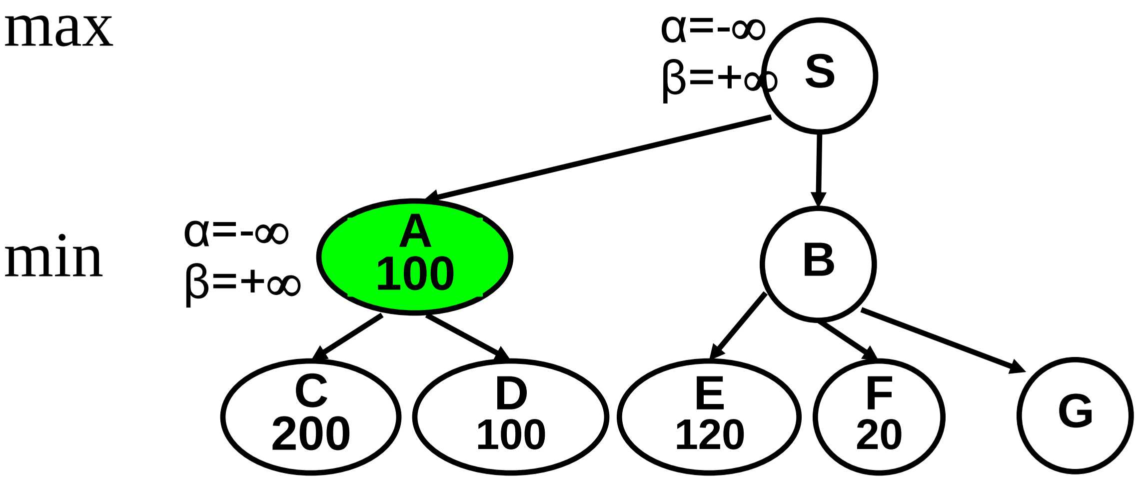
Starting from the root:
Max-Value(root, -∞, +∞)

Alpha pruning example

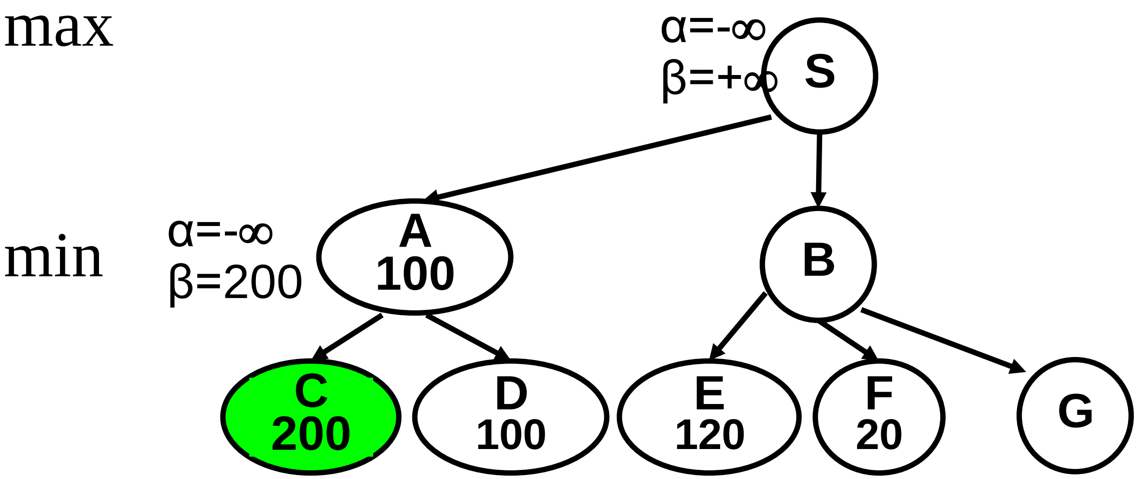


- Keep two bounds along the path
  - α: the best Max can do
  - β: the best (smallest) Min can do
- If at anytime α exceeds β, the remaining children are pruned.

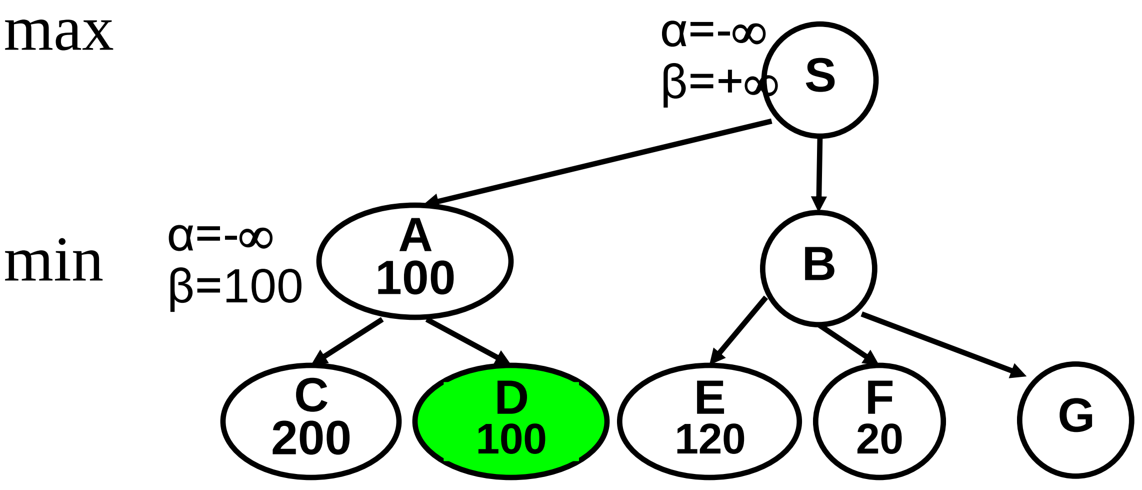
Alpha pruning example



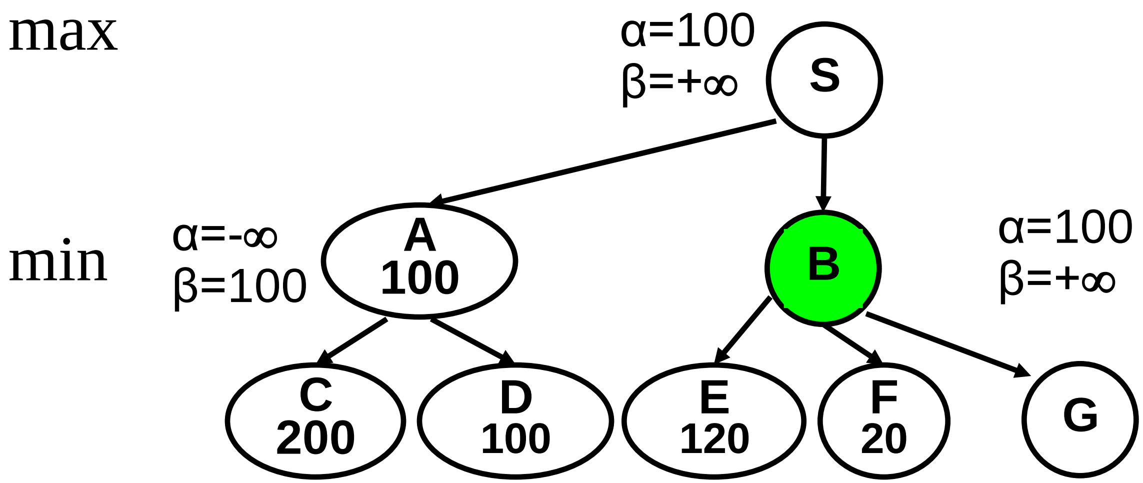
Alpha pruning example



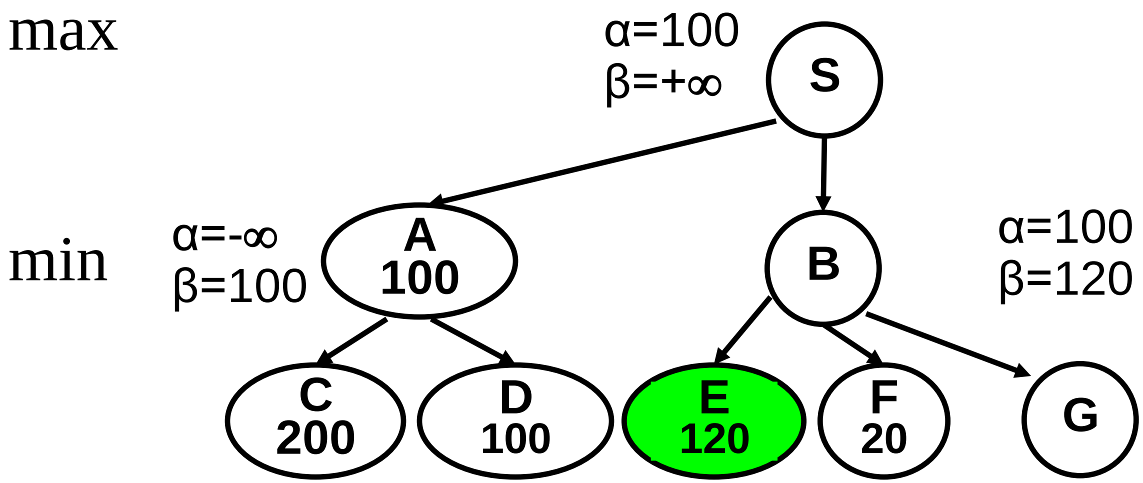
Alpha pruning example



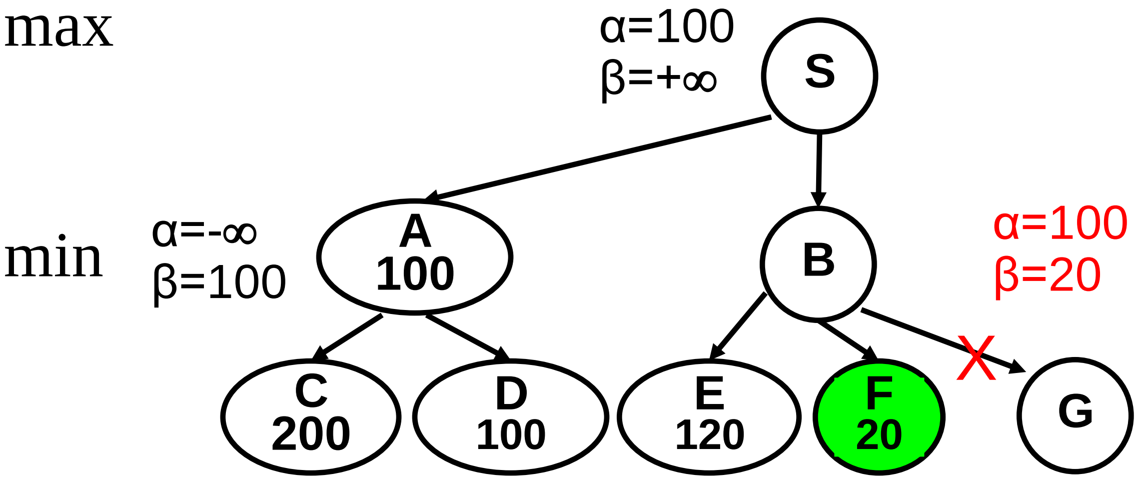
Alpha pruning example



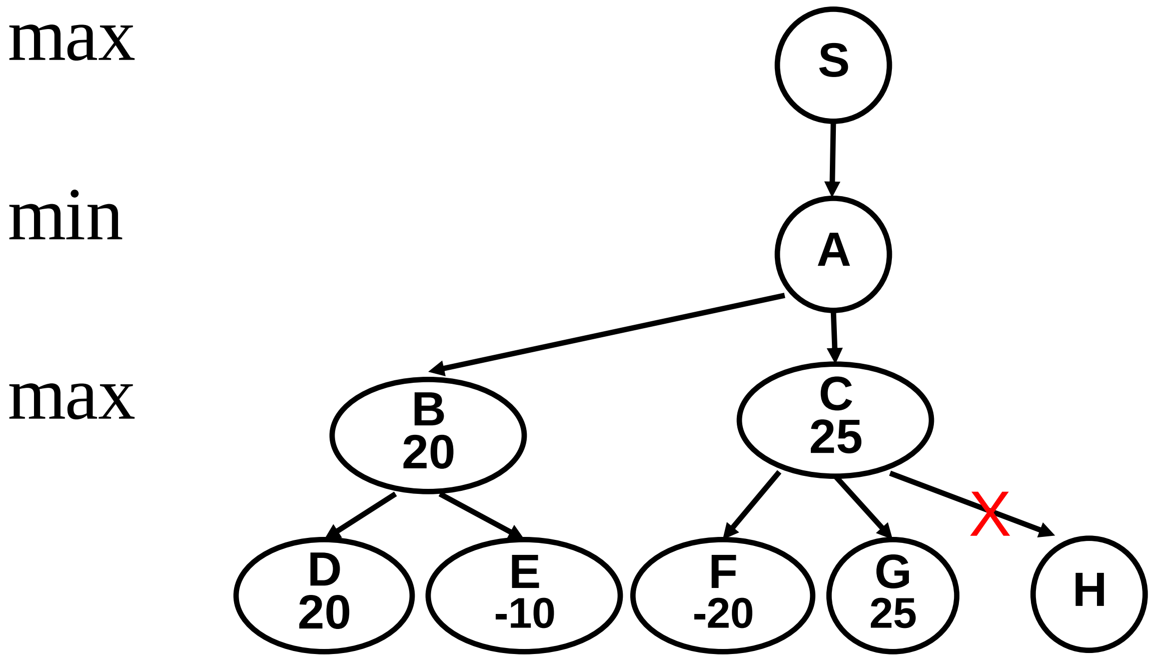
Alpha pruning example



Alpha pruning example



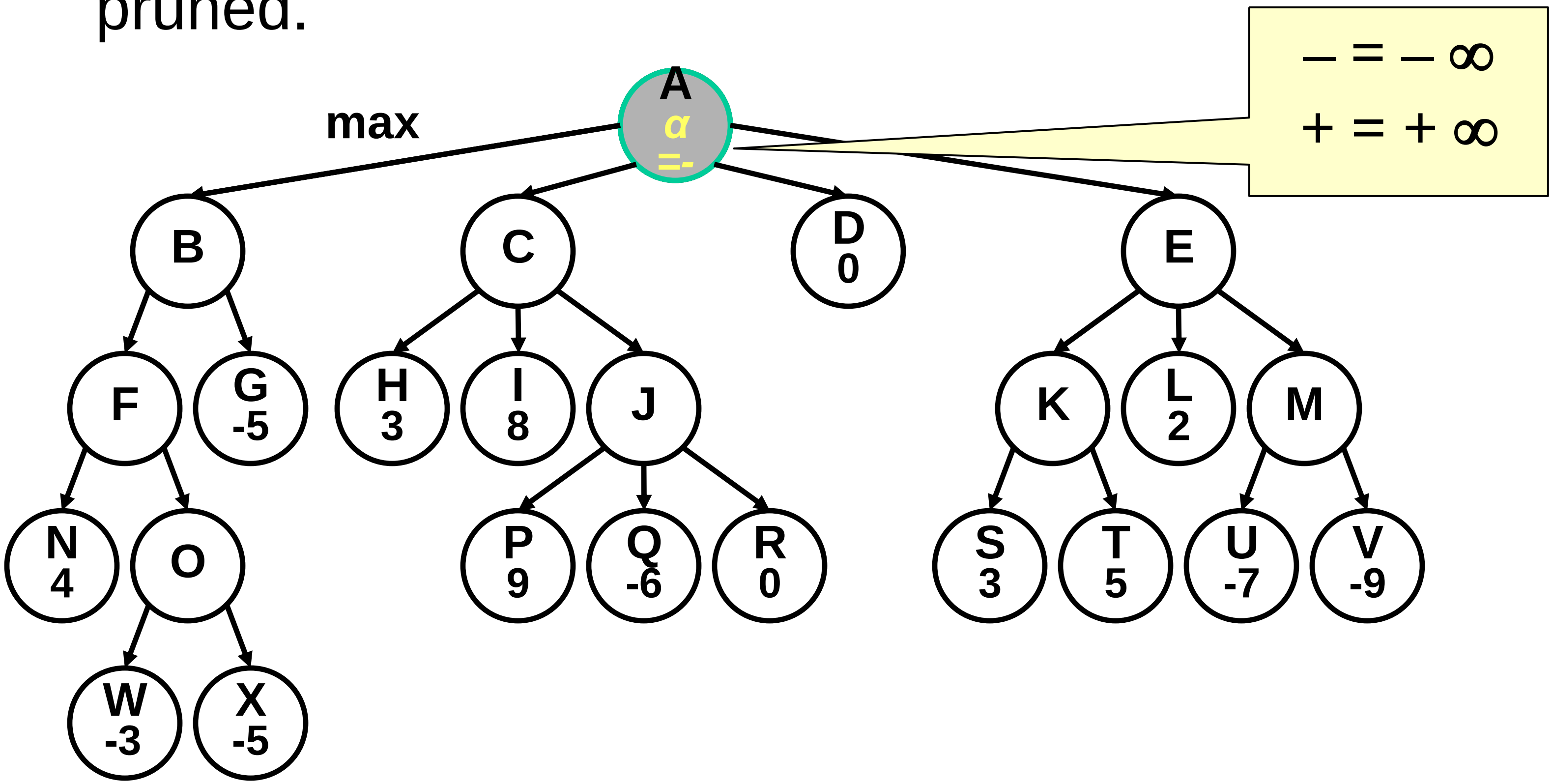
Beta pruning example



- Keep two bounds along the path
  - α: the best Max can do
  - β: the best (smallest) Min can do
- If at anytime α exceeds β, the remaining children are pruned.

Yet another alpha-beta pruning example

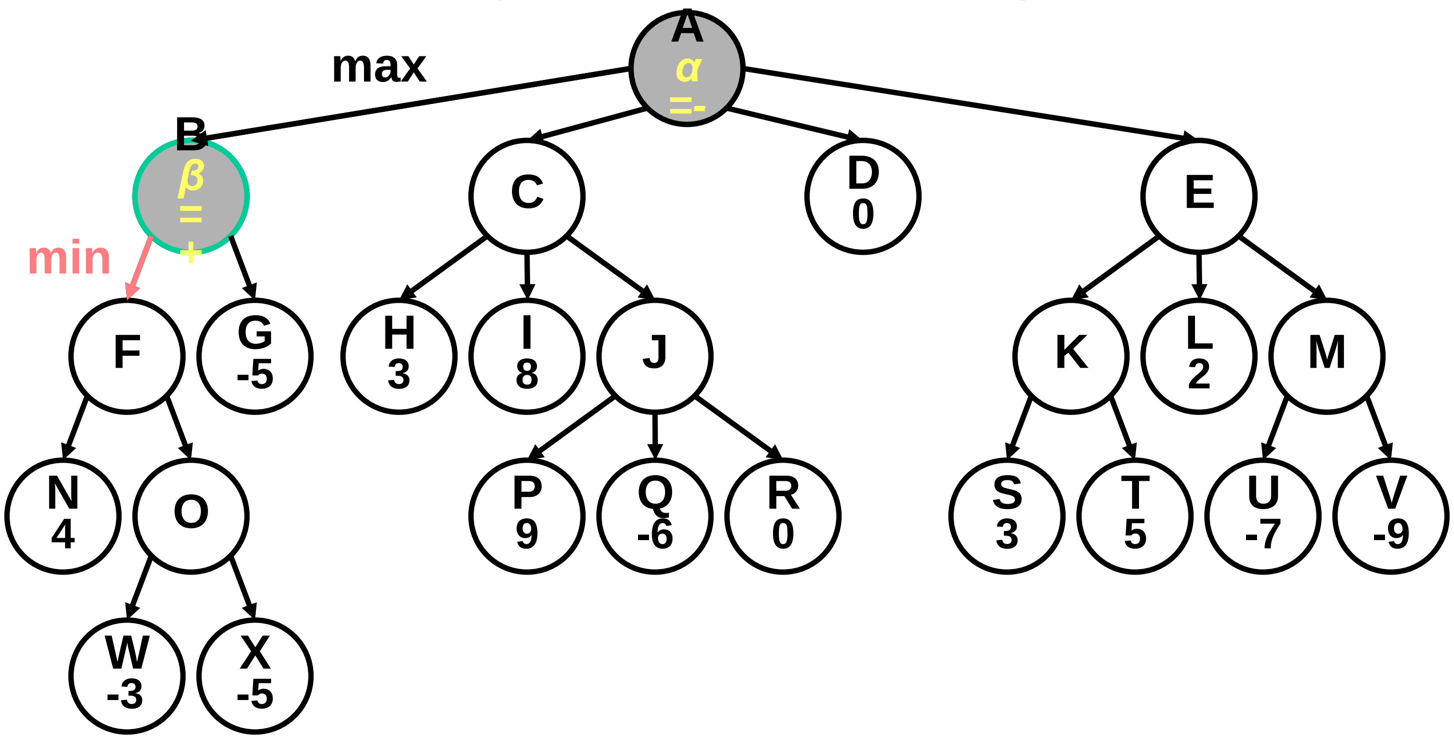
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If at anytime α exceeds β, the remaining children are pruned.



[Example from James Skrentny]

Alpha-beta pruning example

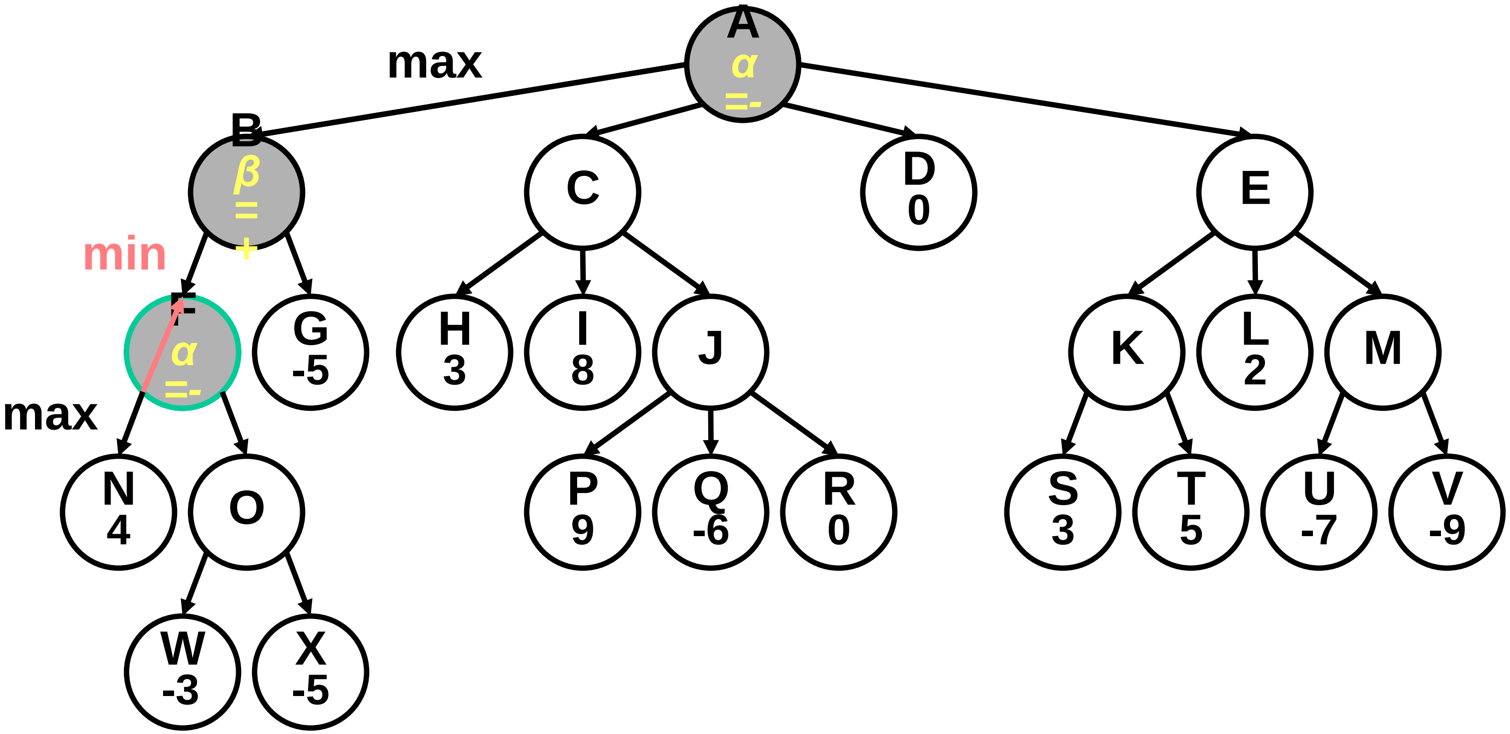
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

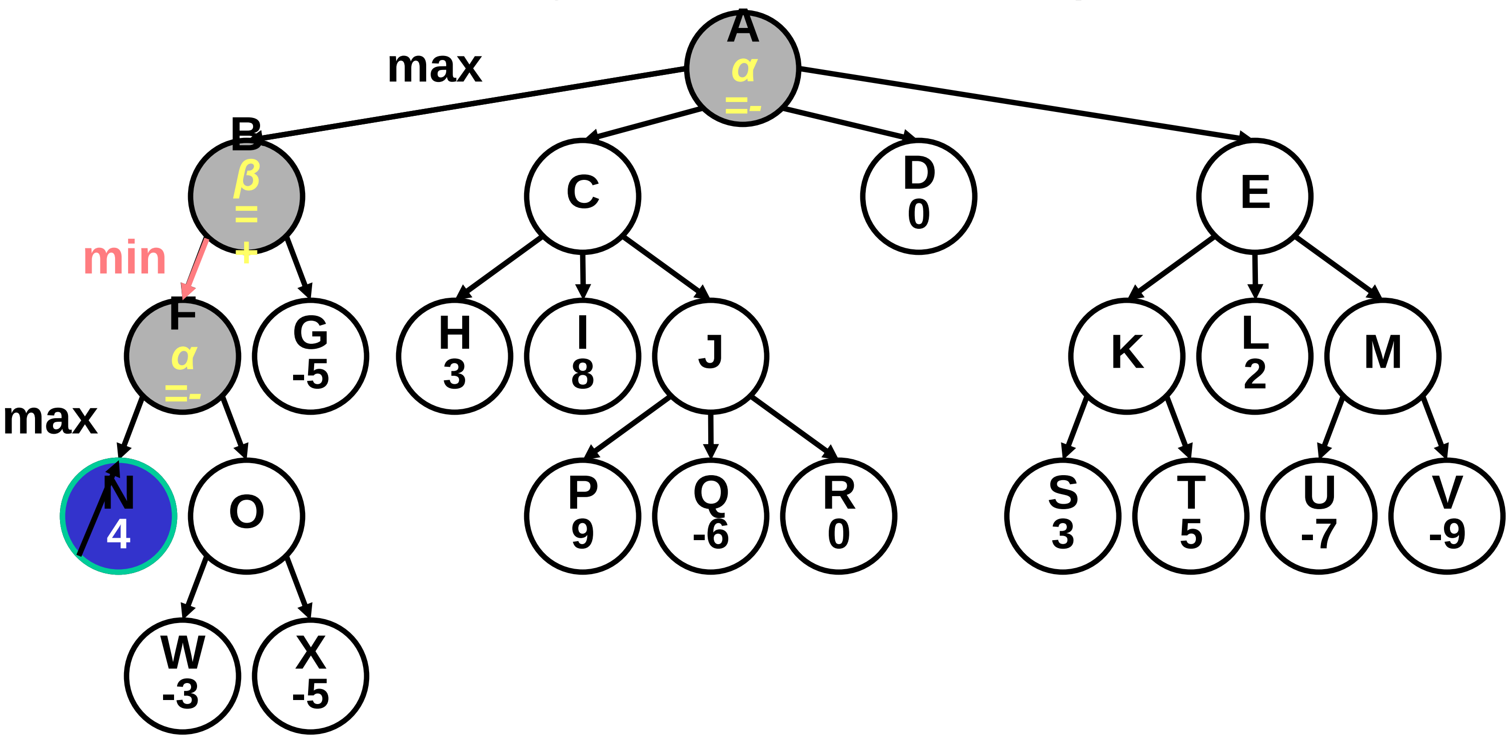
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

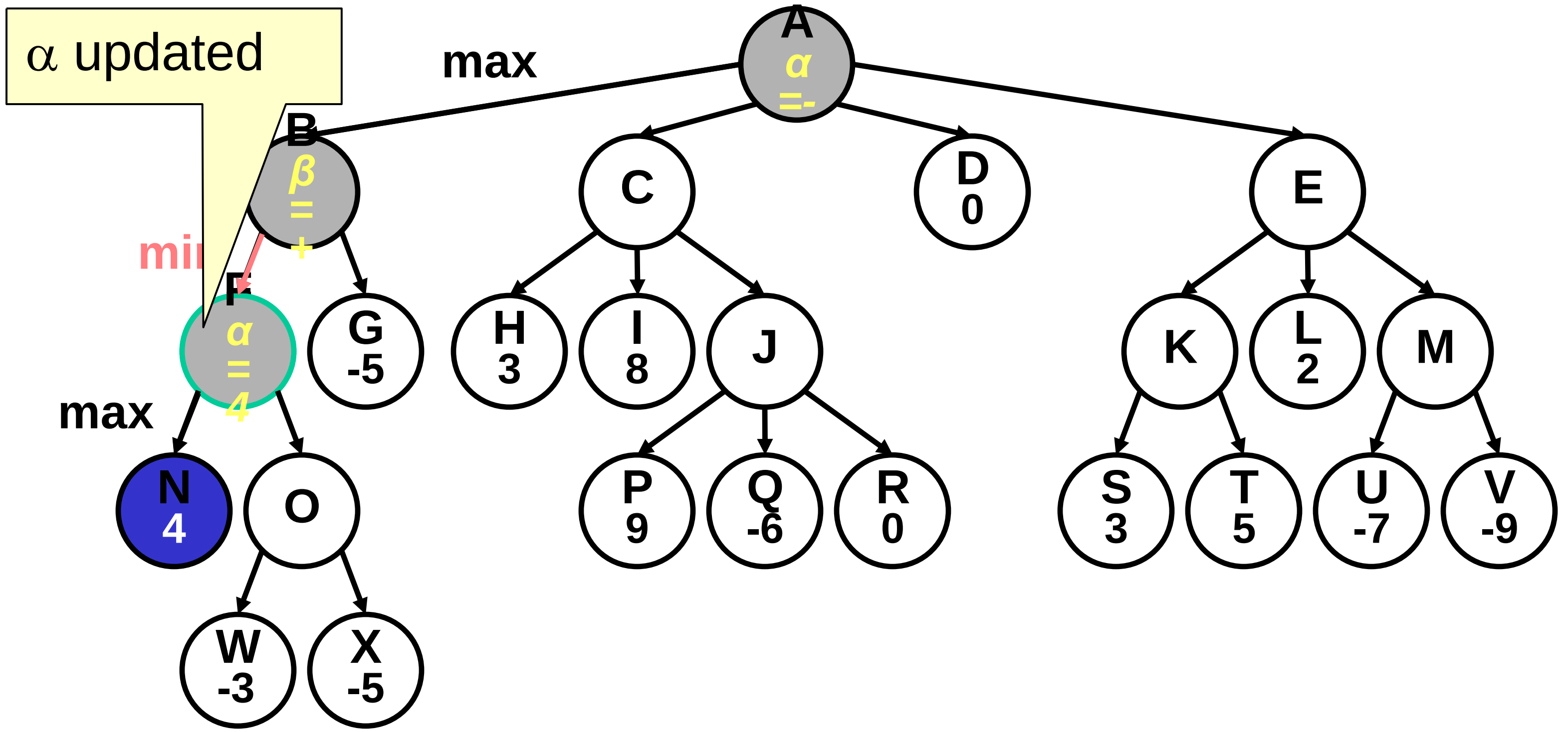
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

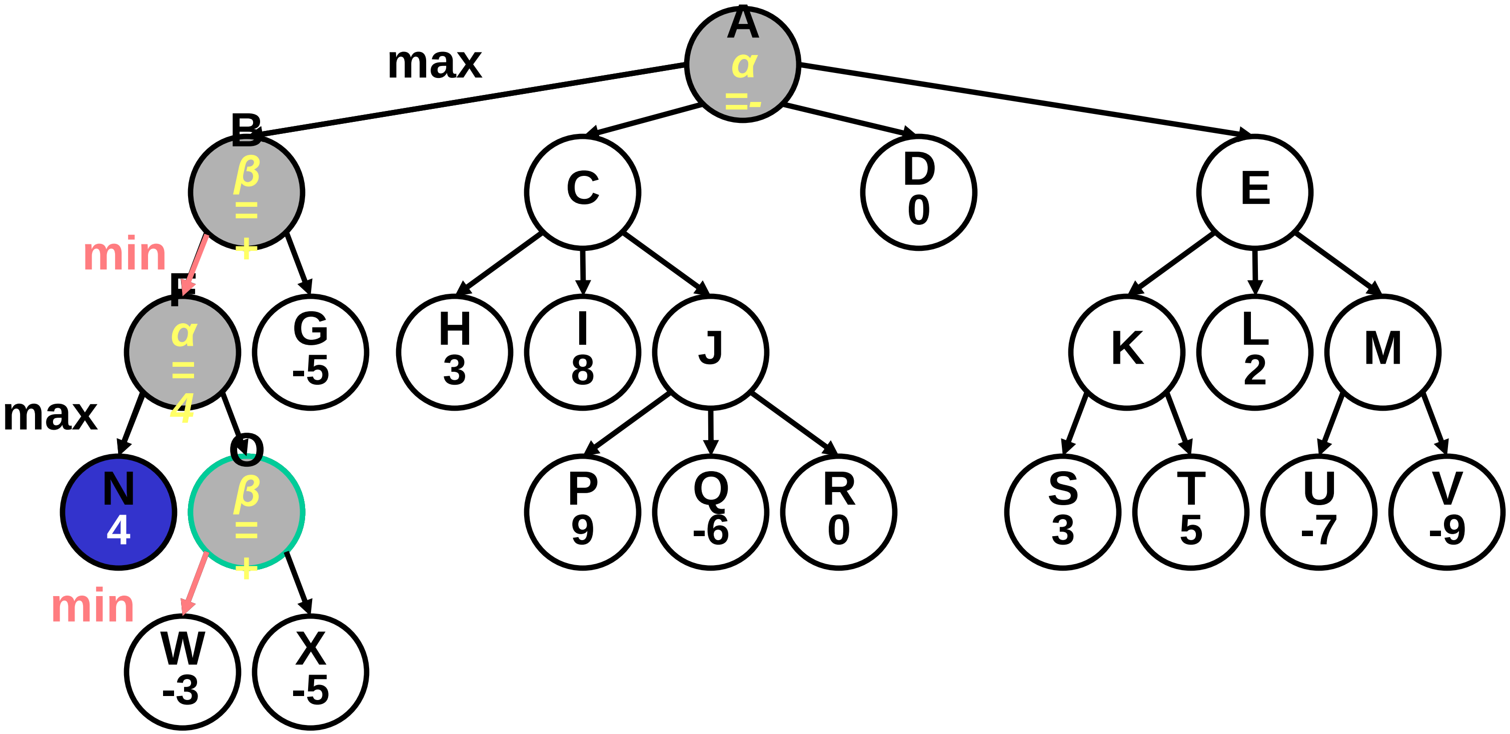
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

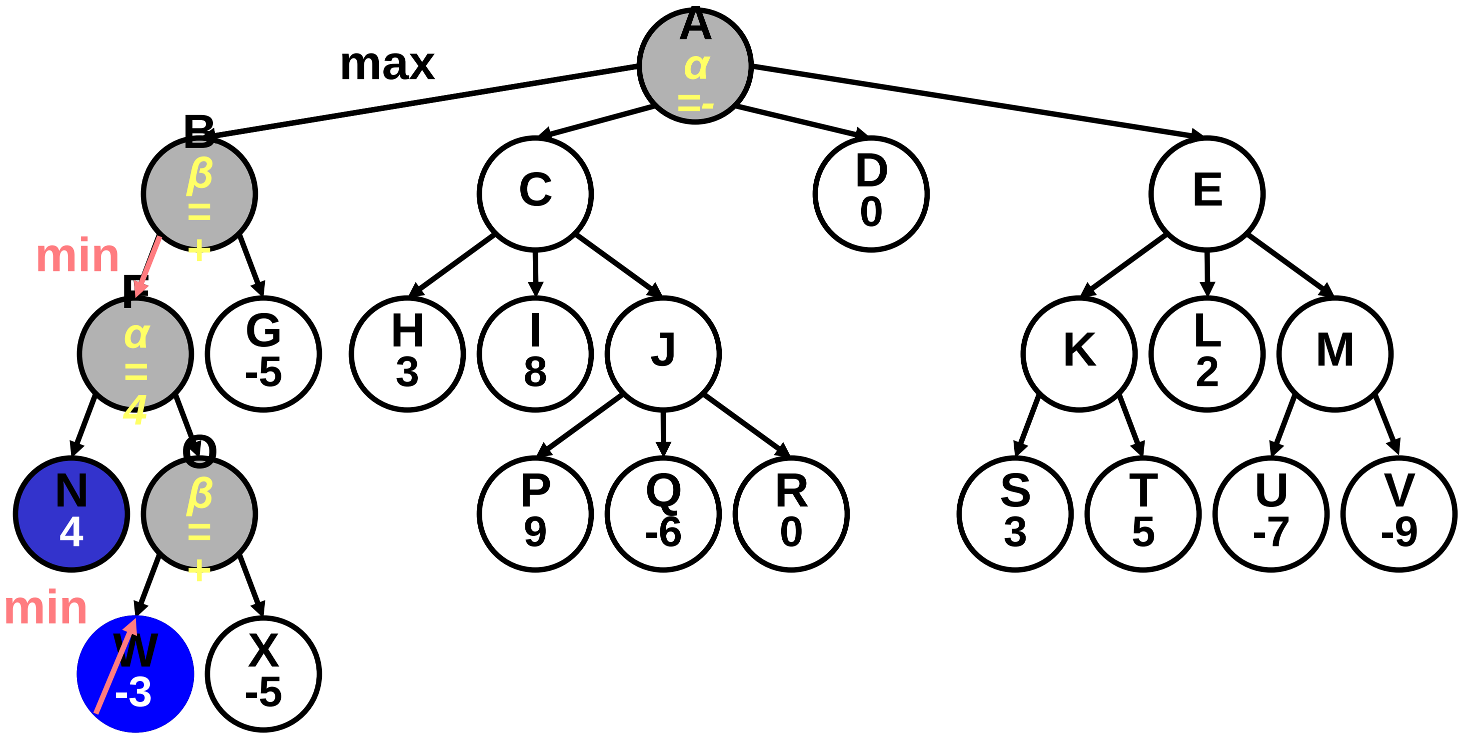
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

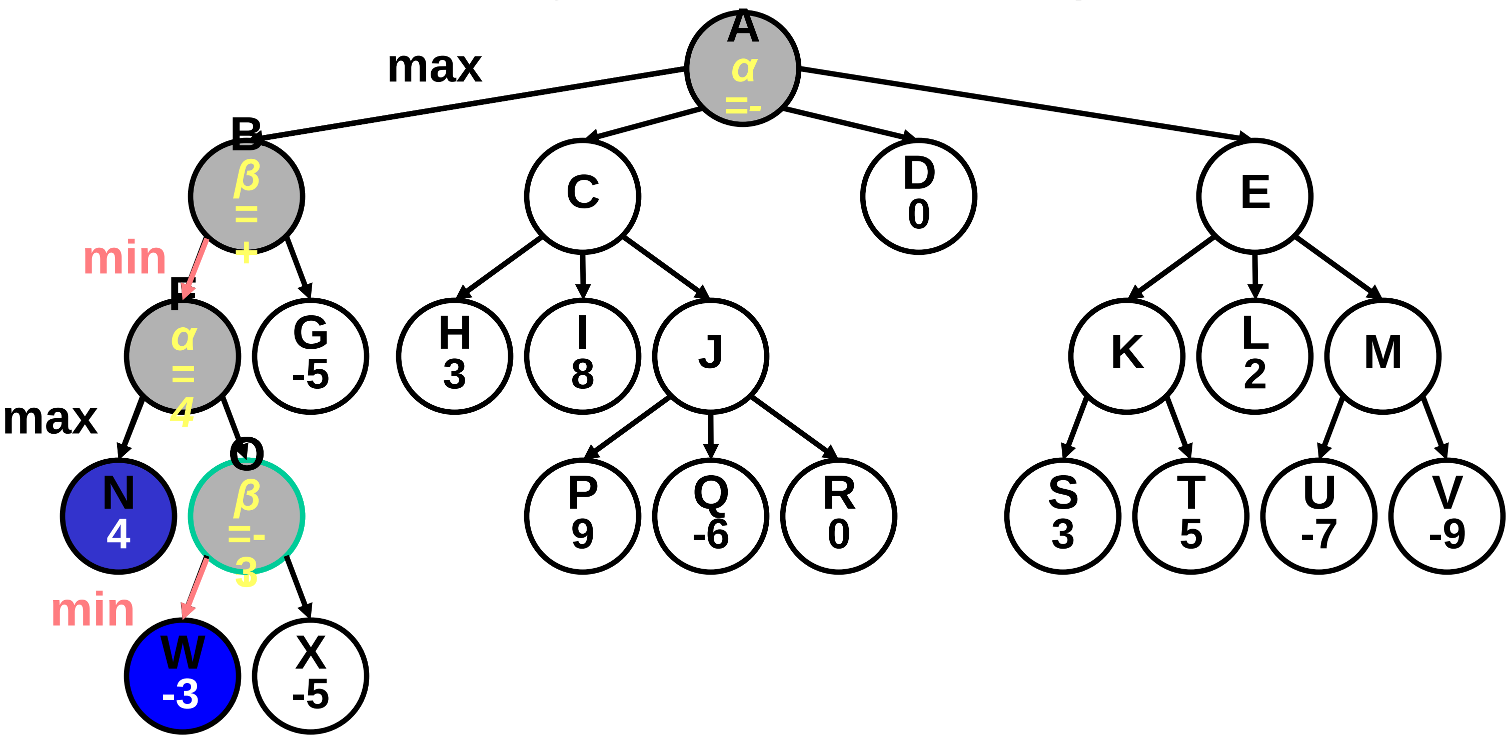
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

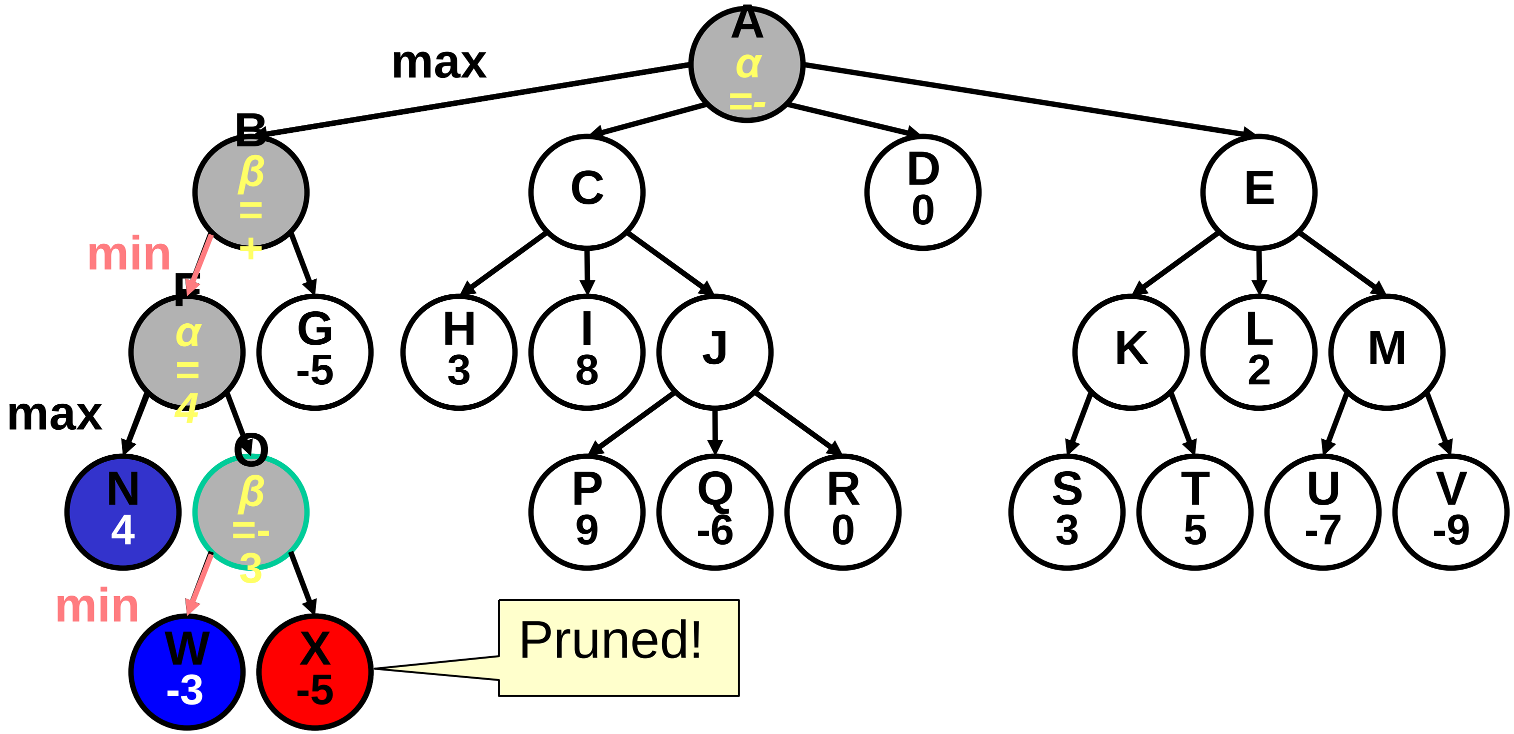
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

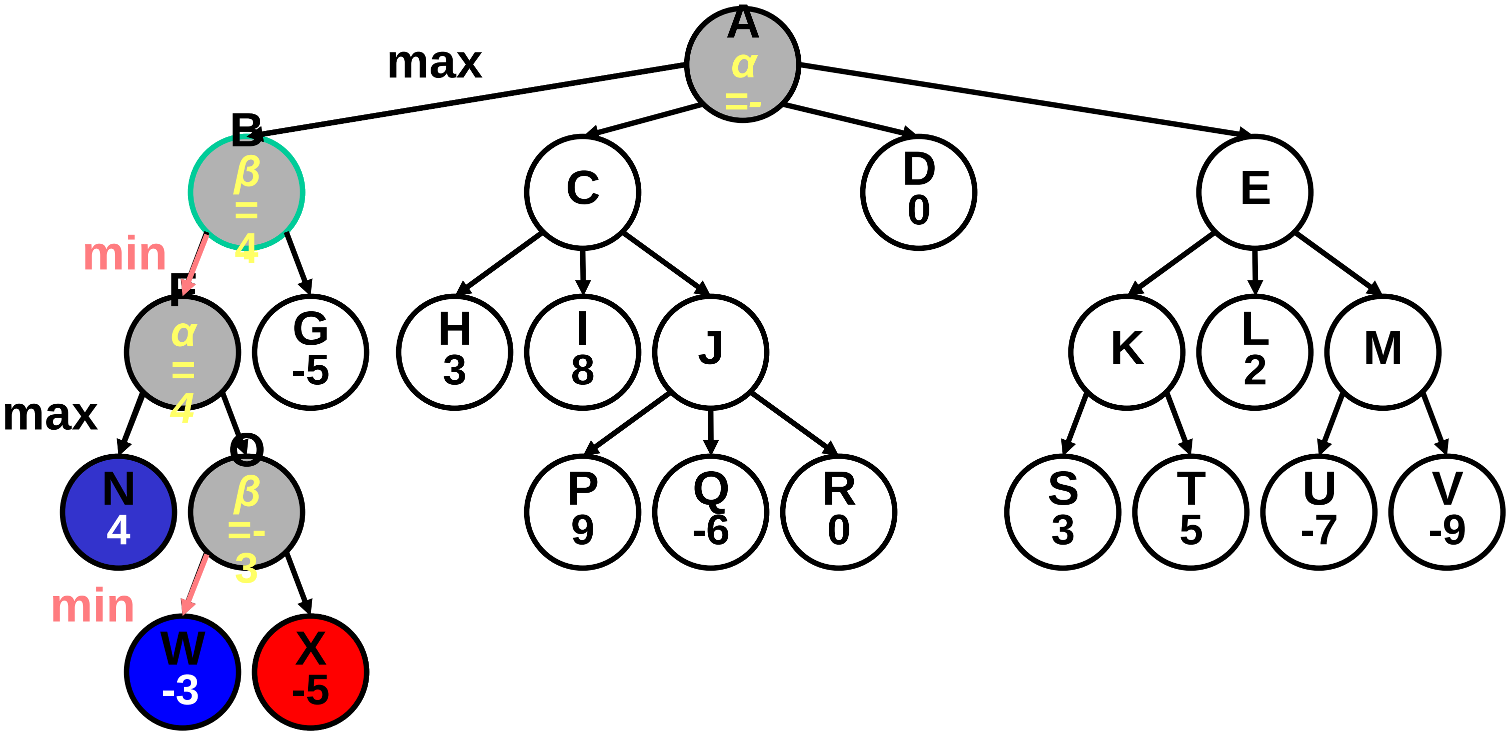
Alpha-beta pruning example

- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



Alpha-beta pruning example

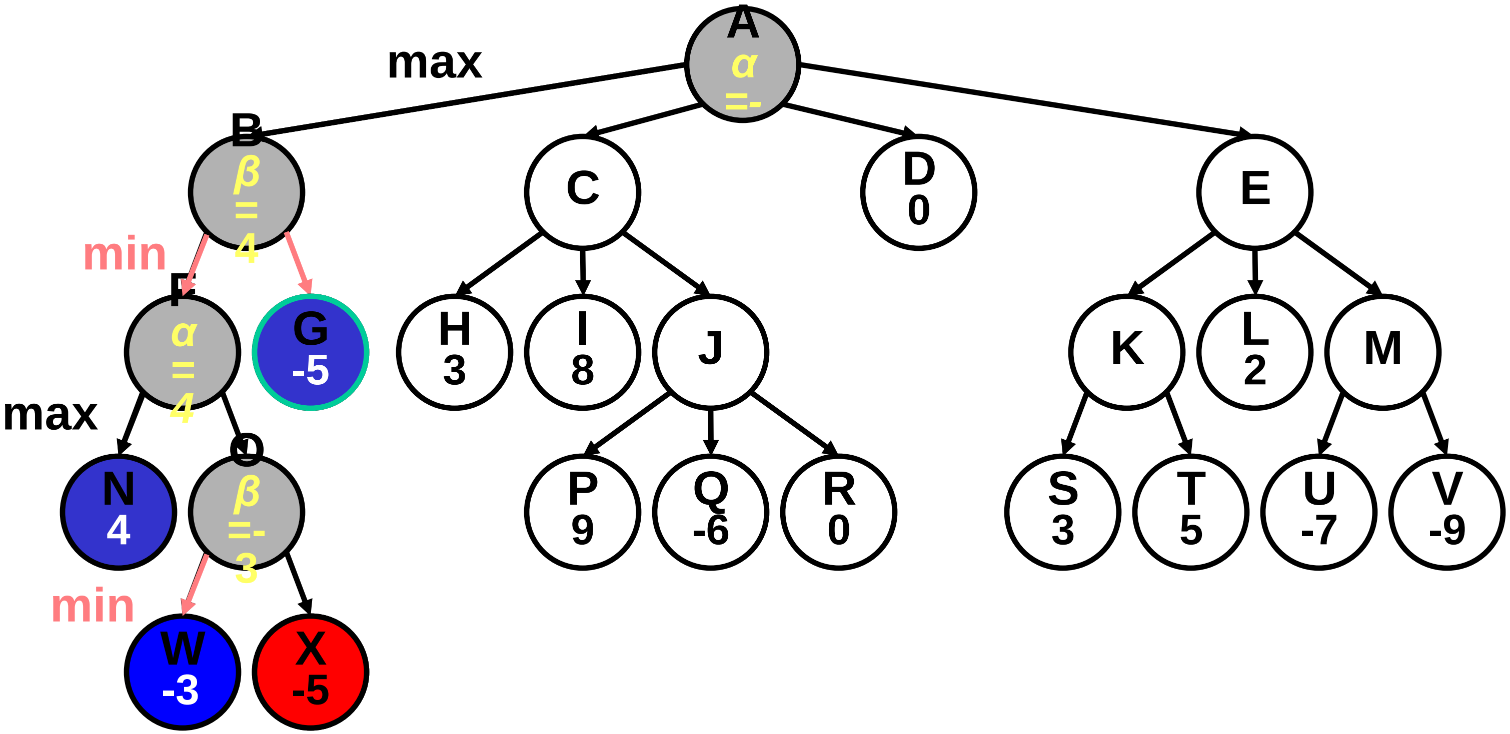
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

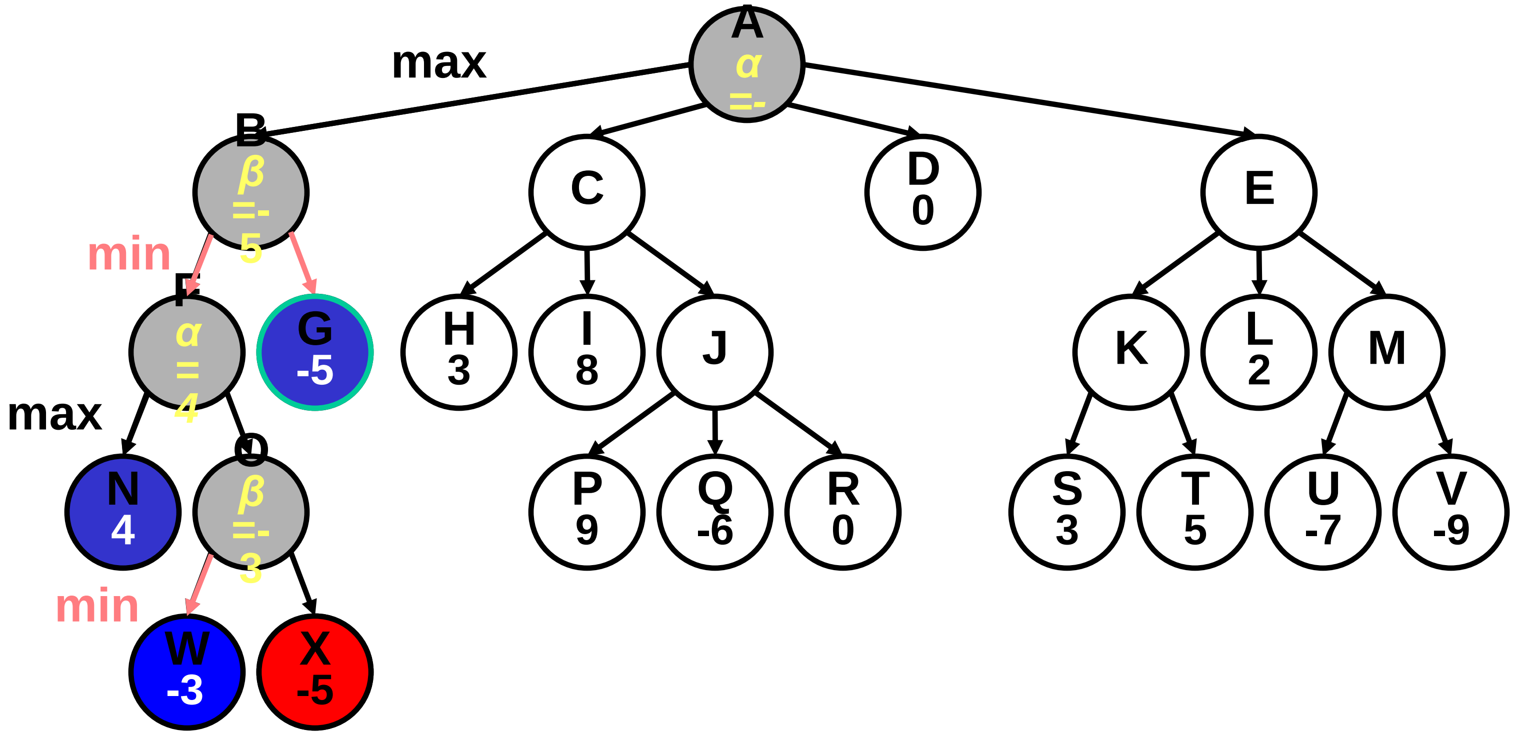
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

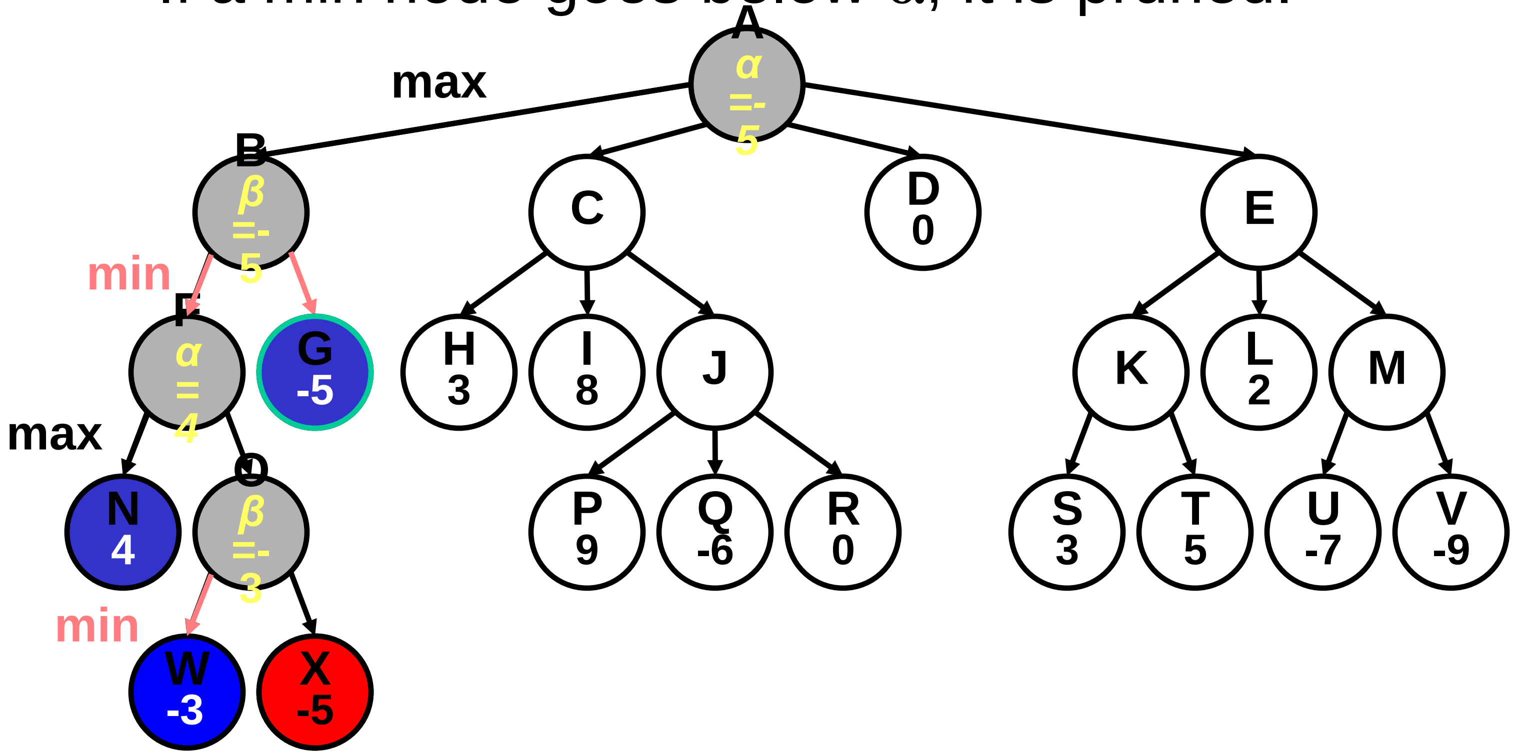
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

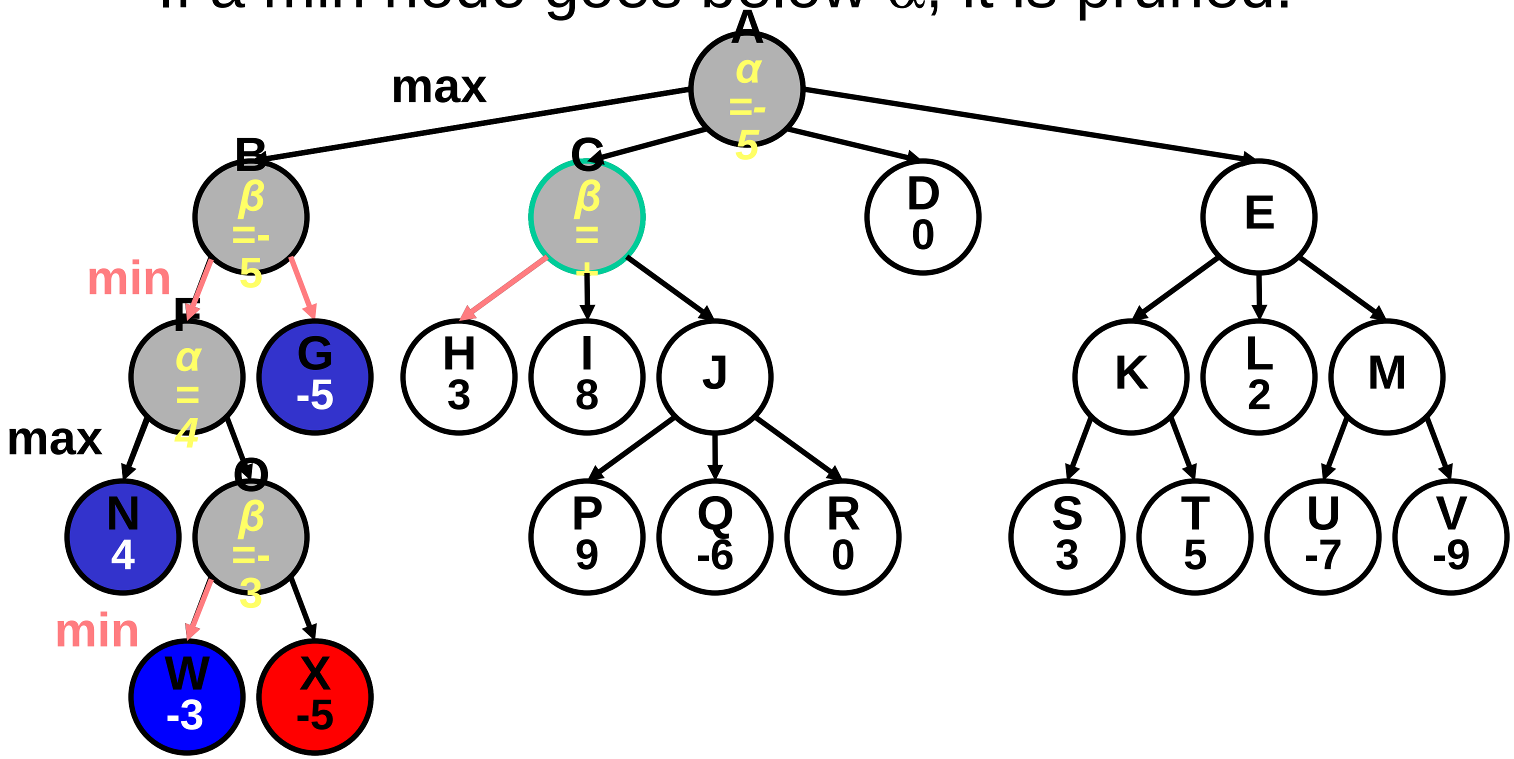
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

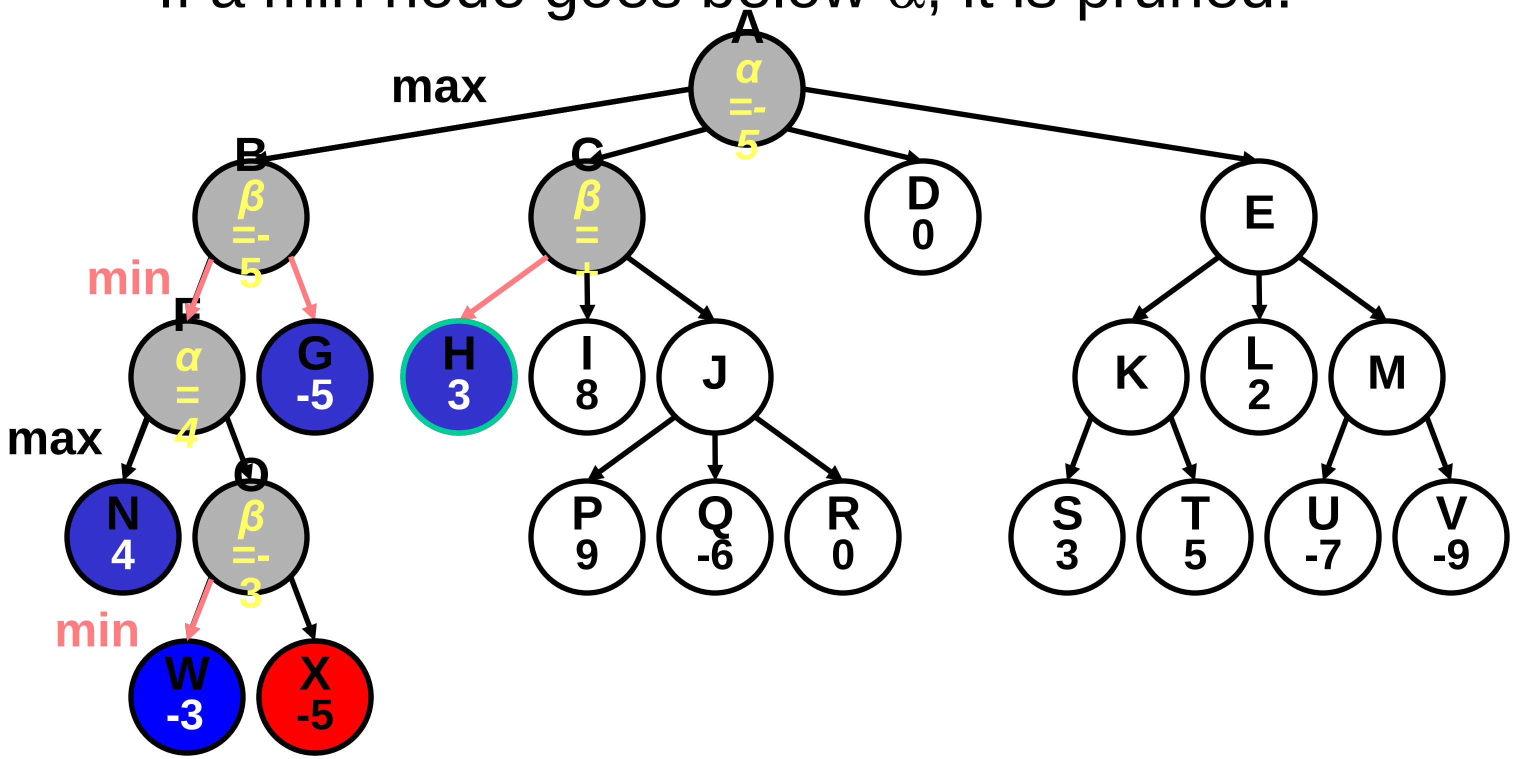
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

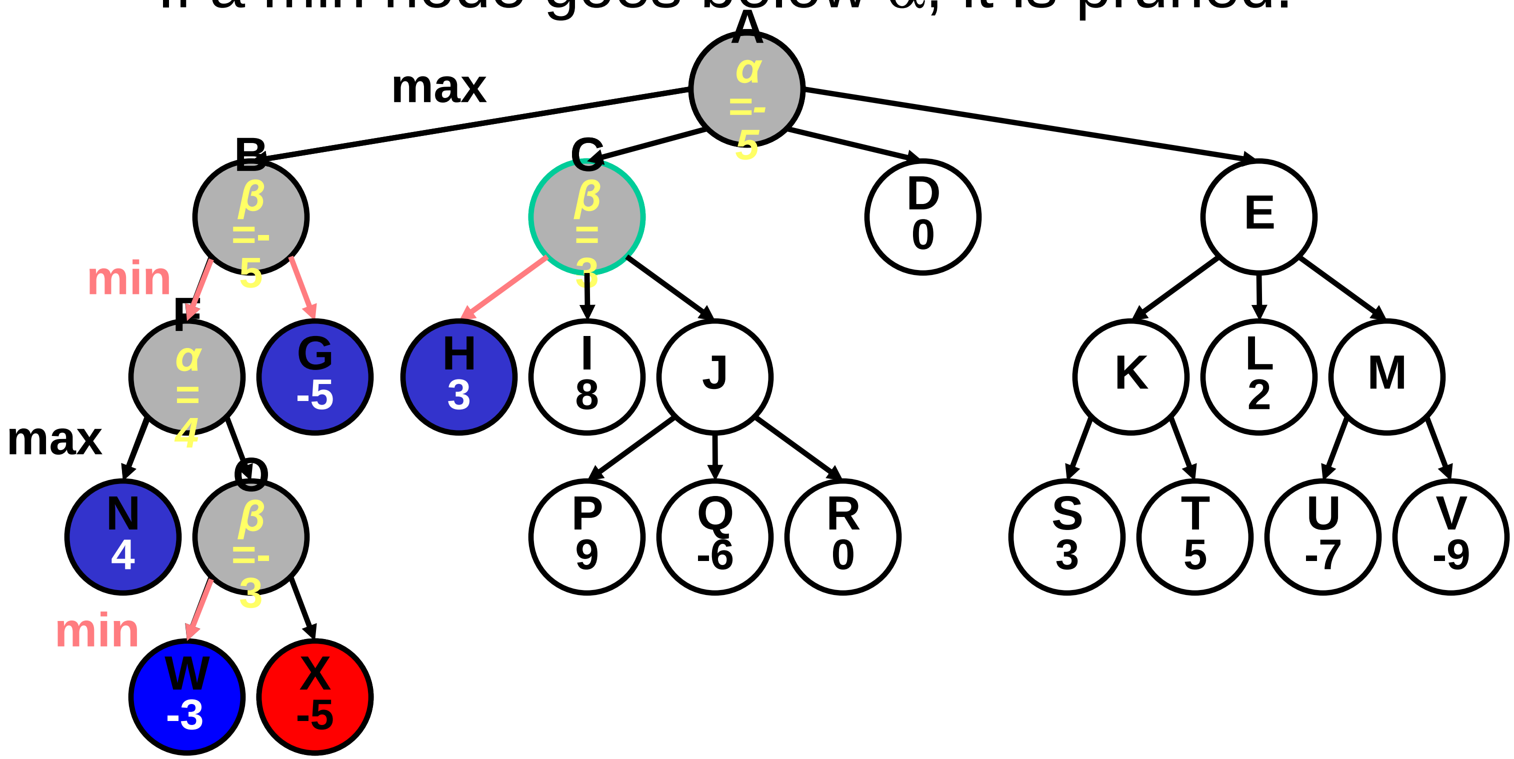
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

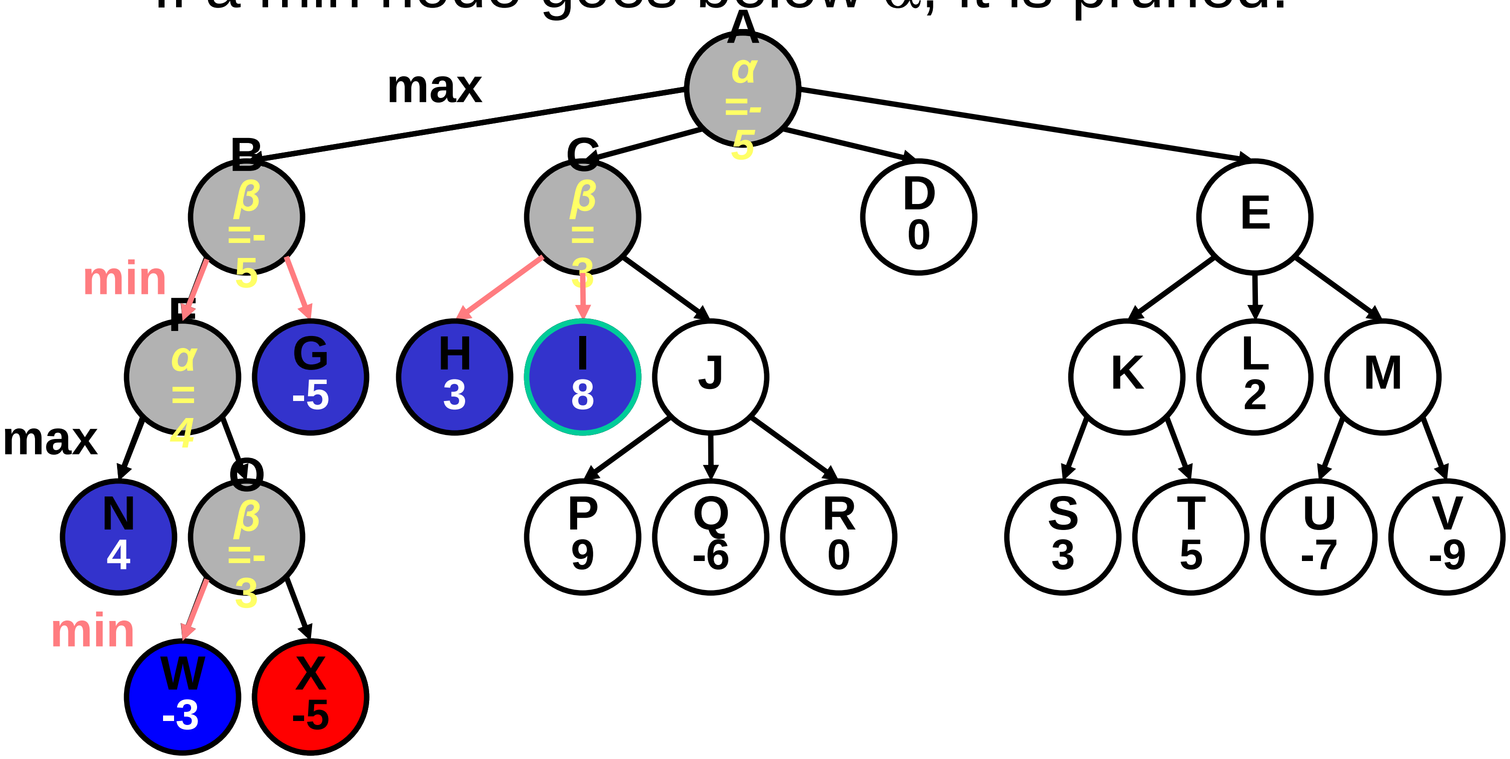
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

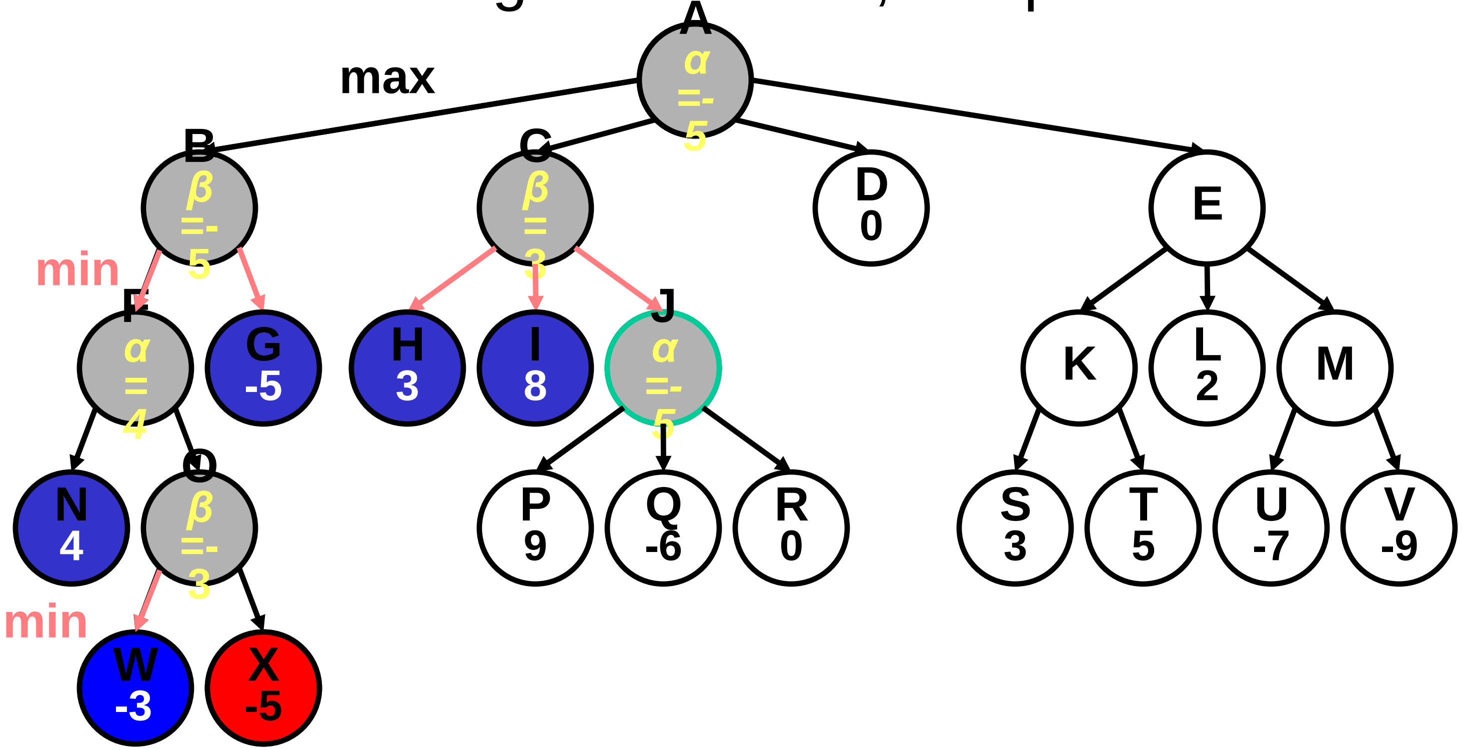
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

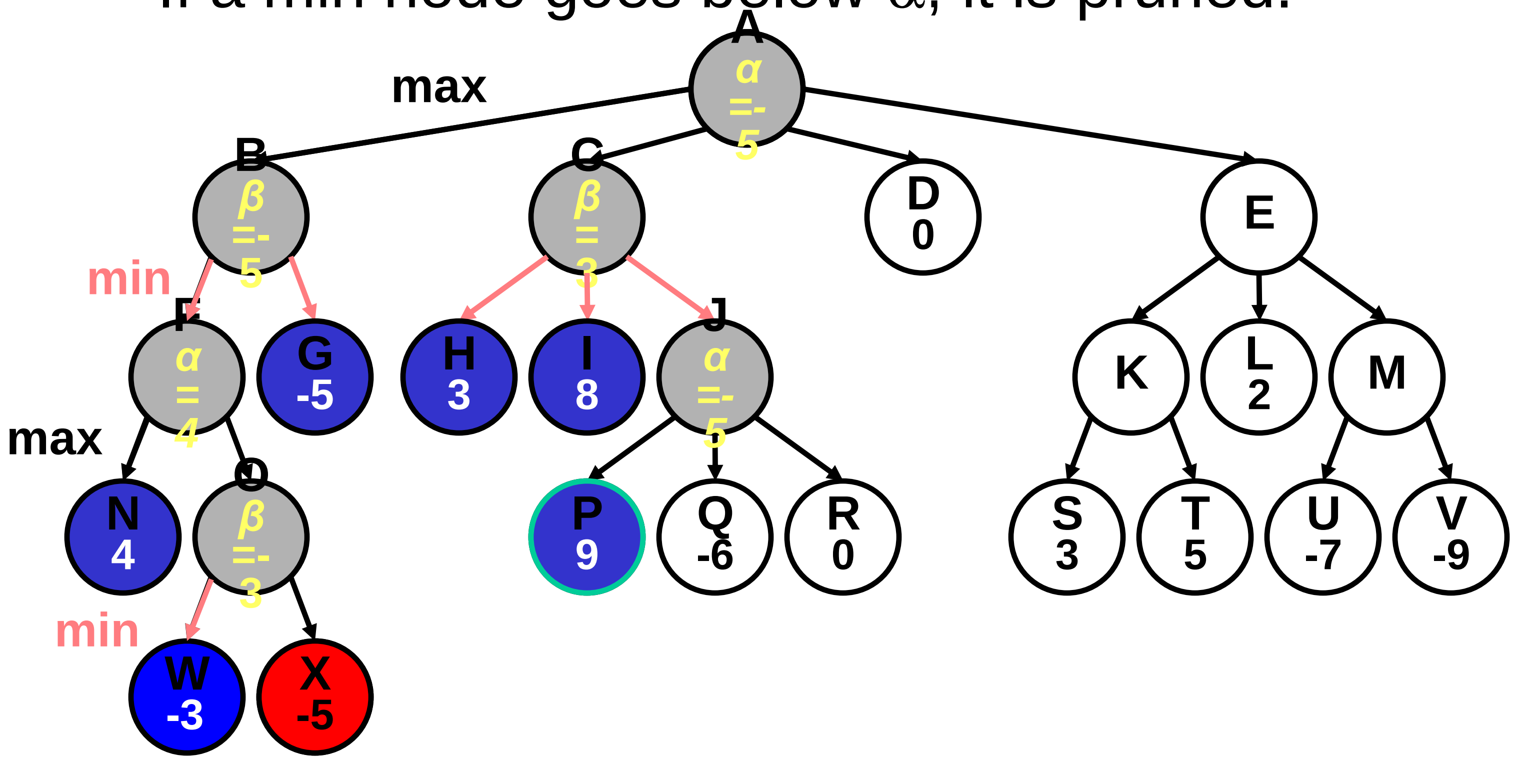
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

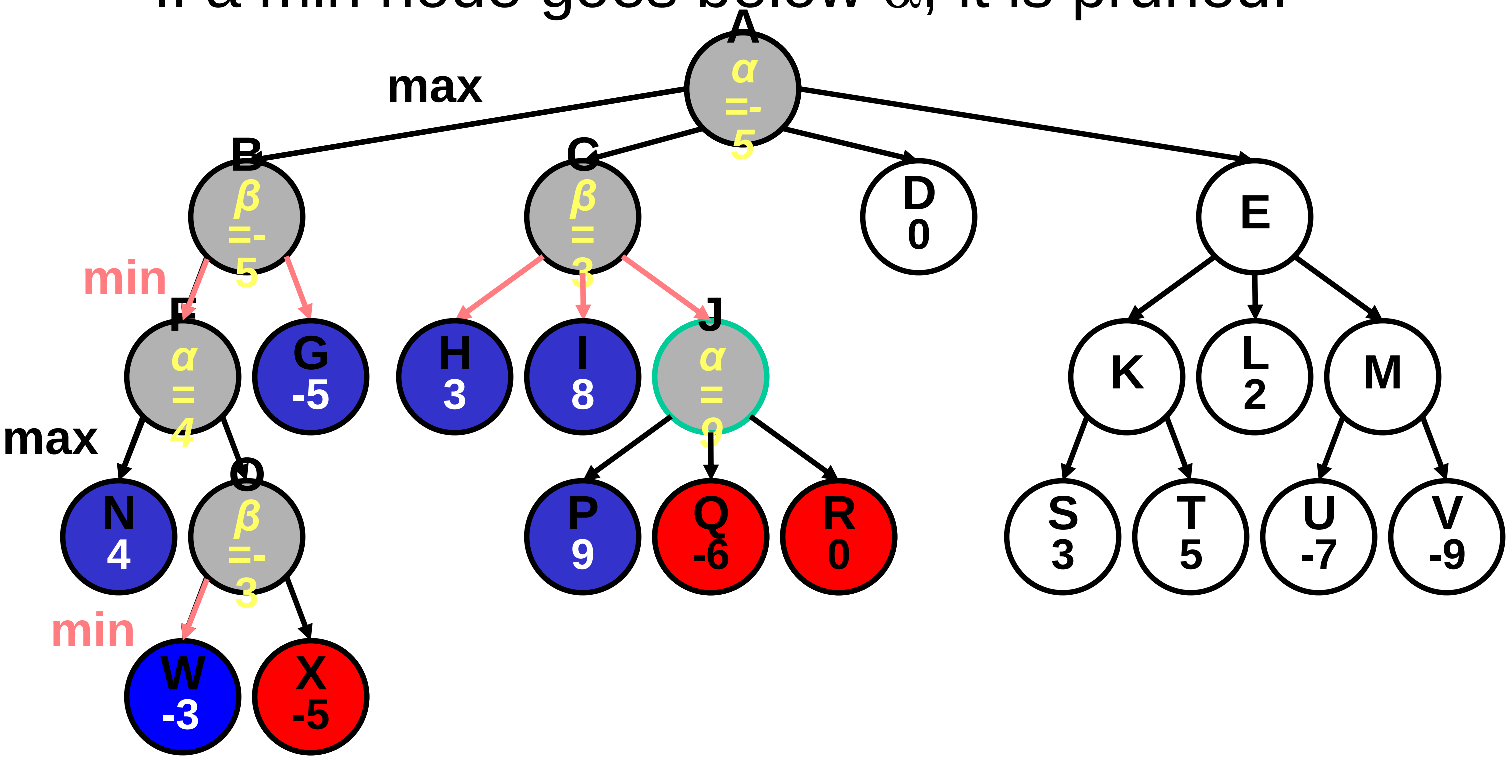
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

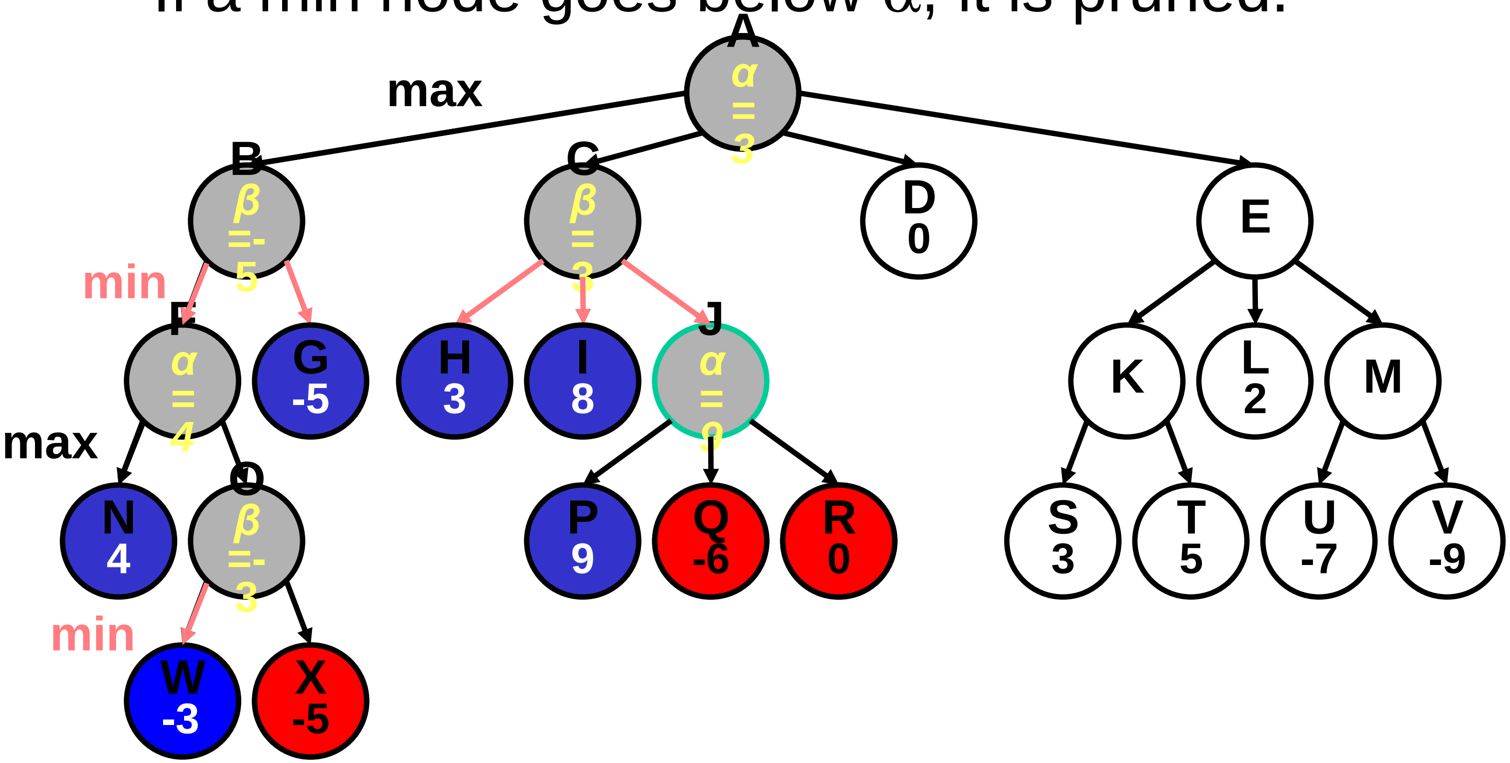
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

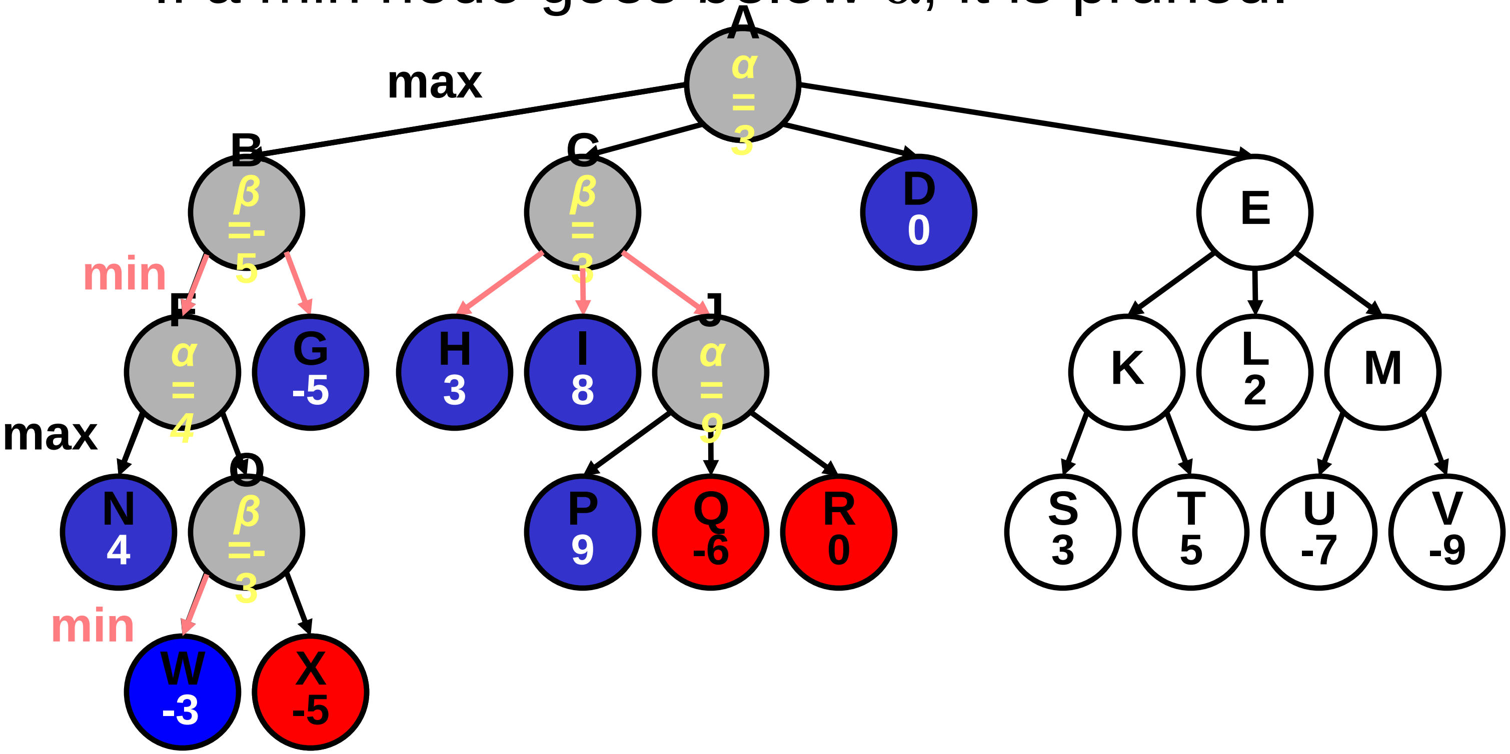
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

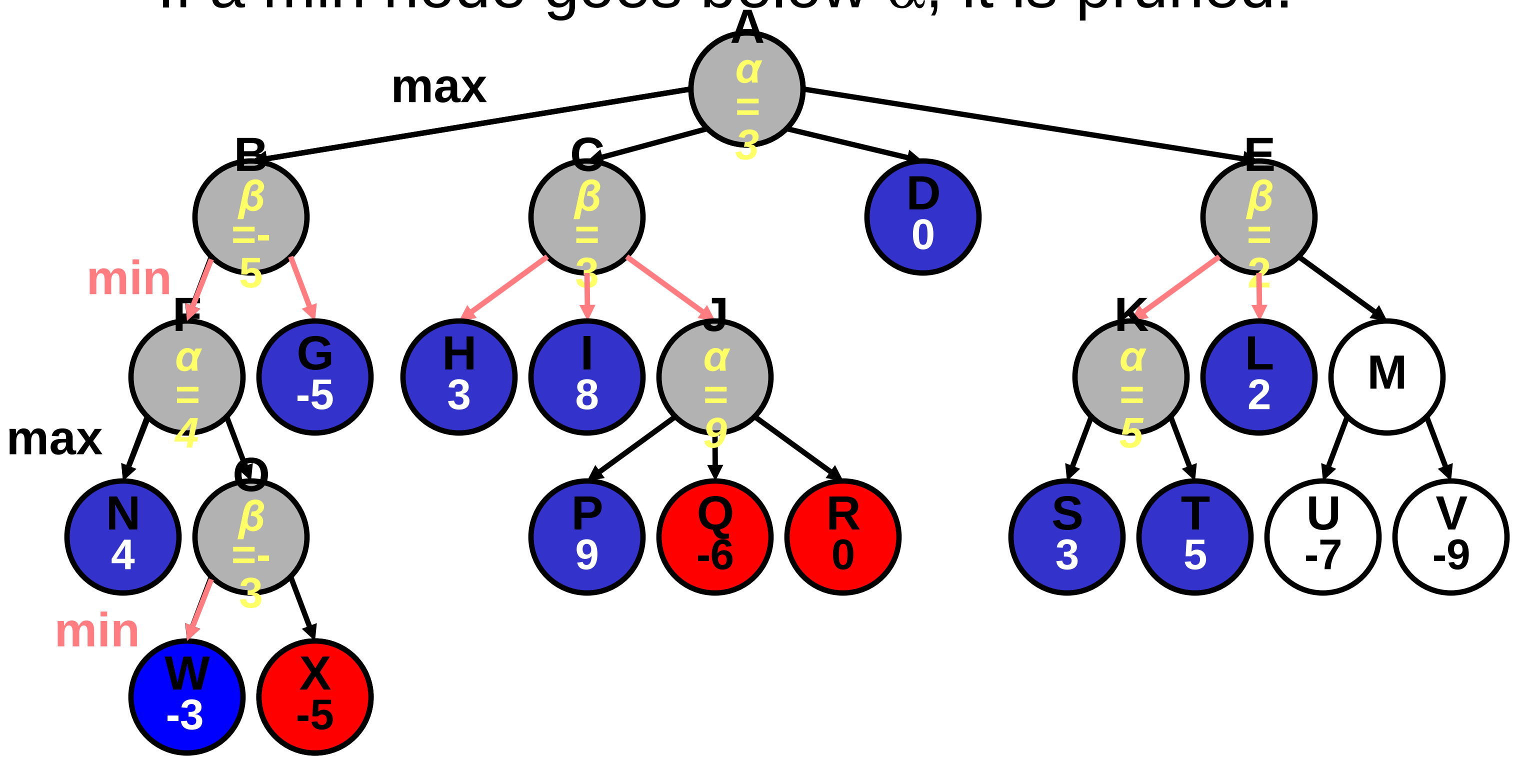
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

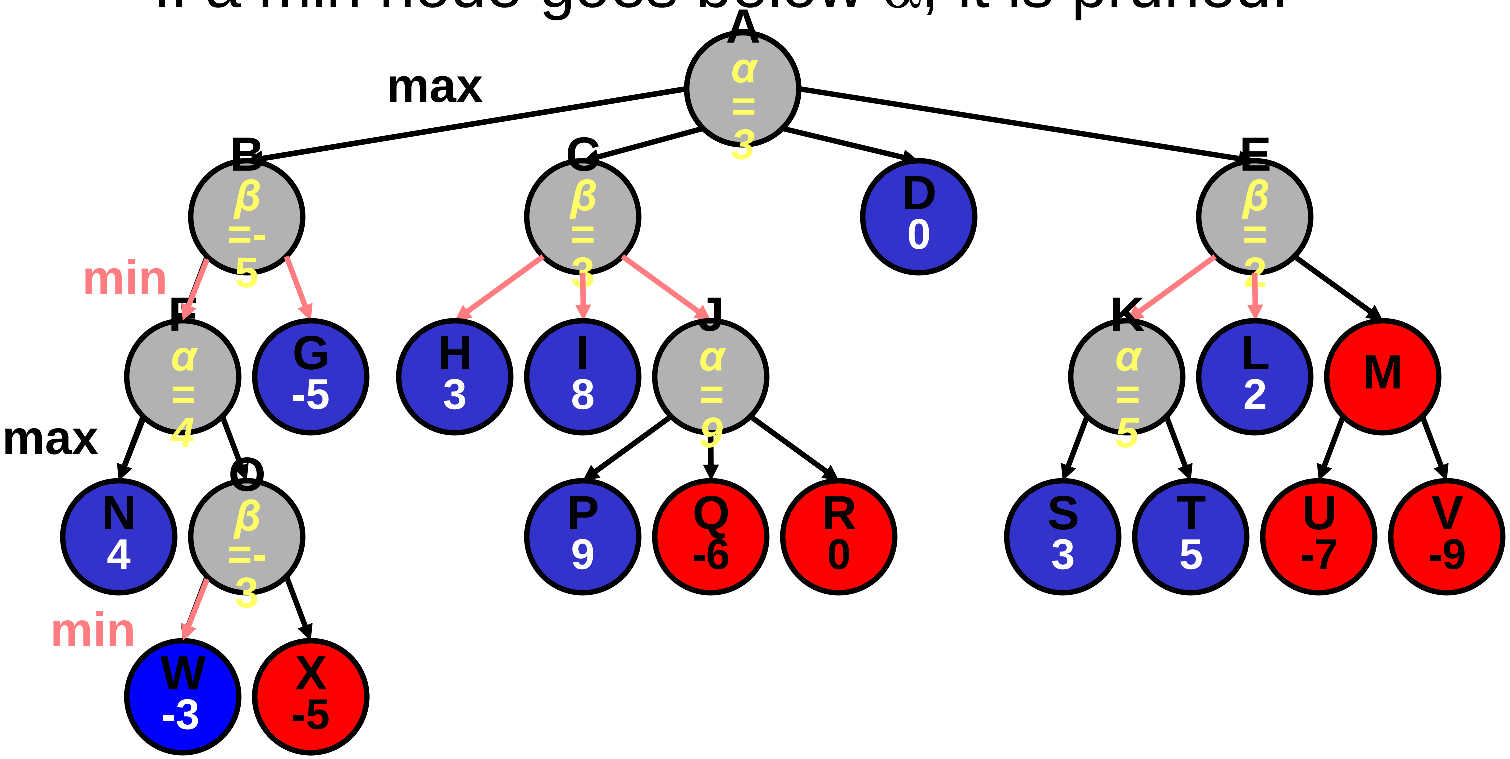
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

Alpha-beta pruning example

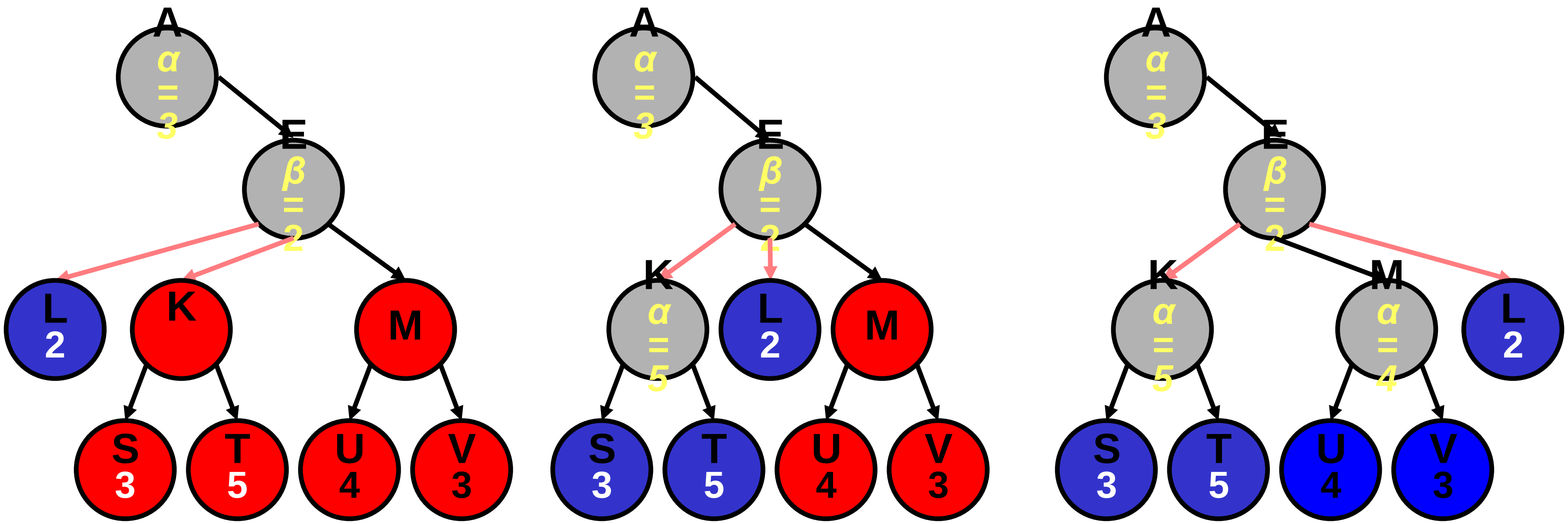
- Keep two bounds along the path
  - α: the best Max can do on the path
  - β: the best (smallest) Min can do on the path
- If a max node exceeds β, it is pruned.
- If a min node goes below α, it is pruned.



[Example from James Skrentny]

How effective is alpha-beta pruning?

- Depends on the order of successors!



- In the best case, the number of nodes to search is O($b^{m/2}$), the square root of minimax's cost.
- This occurs when each player's best move is the leftmost child.
- In DeepBlue (IBM Chess), the average branching factor was about 6 with alpha-beta instead of 35-40 without.
- The worst case is no pruning at all.

Game-playing for large games

- We've seen how to find game theoretic values. But it is too expensive for large games.
- What do real chess-playing programs do?
  - They can't possibly search the full game tree
  - They must respond in limited time
  - They can't pre-compute a solution

Game-playing for large games

- The most popular solution: heuristic evaluation functions for games
  - 'Leaves' are intermediate nodes at a depth cutoff, not terminals
  - Heuristically estimate their values
  - Huge amount of knowledge engineering (R&N 6.4)
  - Example: Tic-Tac-Toe:

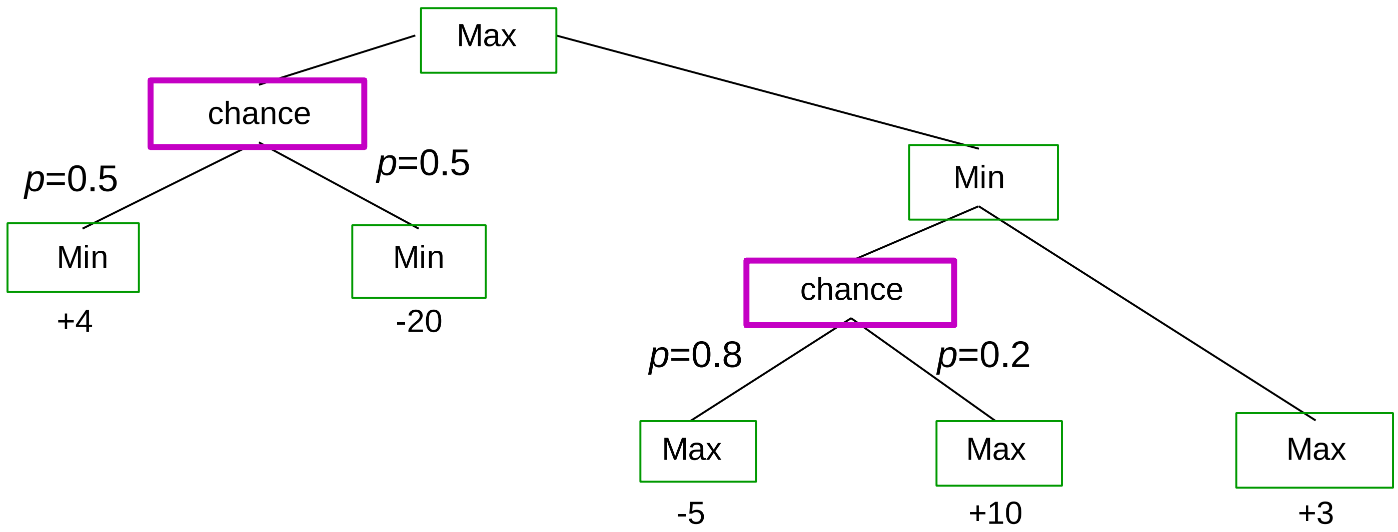
  (number of 3-lengths open for me)-(number of 3-lengths open for you)
- Each move is a new depth-cutoff game-tree search (as opposed to search the complete game-tree once).
- Depth-cutoff can increase using iterative deepening, as long as there is time left.

More on large games

- Battle the limited search depth
  - Horizon effect: things can suddenly get much worse just outside your search depth (‘horizon’), but you can’t see that
  - Quiescence / secondary search: select the most ‘interesting’ nodes at the search boundary, expand them further beyond the search depth
- Incorporate book moves
  - Pre-compute / record opening moves, end games

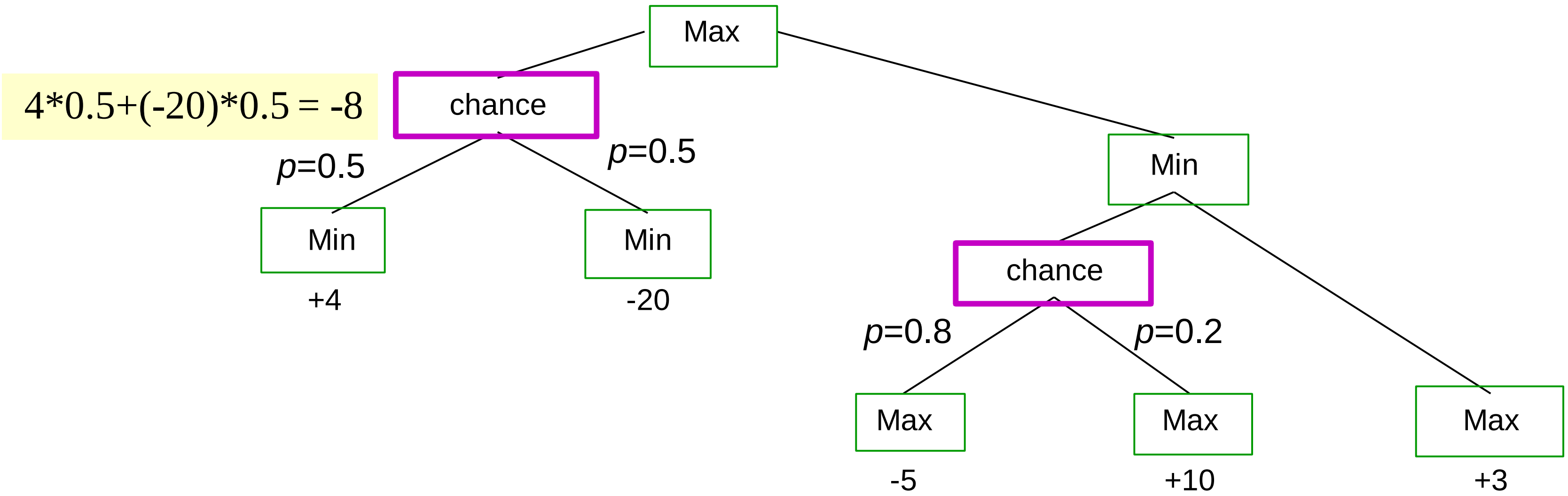
Two-player zero-sum discrete finite NONdeterministic games of perfect information

- There is an element of chance (coin flip, dice roll, etc.)
- "Chance node" in game tree, besides Max and Min nodes. Neither player makes a choice. Instead a random choice is made according to the outcome probabilities.



Solving non-deterministic games

- Easy to extend minimax to non-deterministic games
- At chance node, instead of using max() or min(), compute the average (weighted by the probabilities).



- What's the value for the chance node at right?
- What action should Max take at root?
- The play will be optimal. In what sense?

What you should know

- What is a two-player zero-sum discrete finite deterministic game of perfect information
- What is a game tree
- What is the minimax value of a game
- Minimax search
- Alpha-beta pruning
- Basic understanding of very large games
- How to extend minimax to non-deterministic games