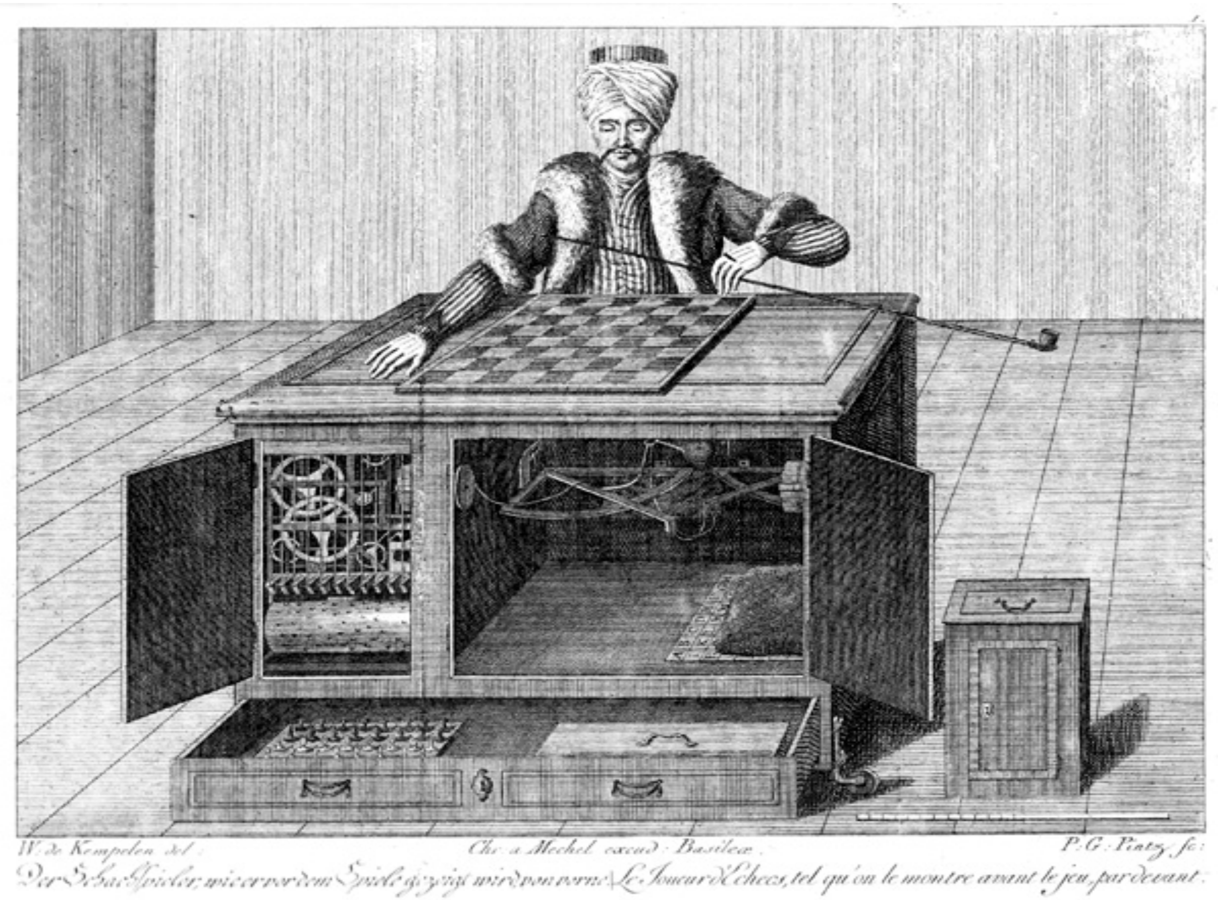
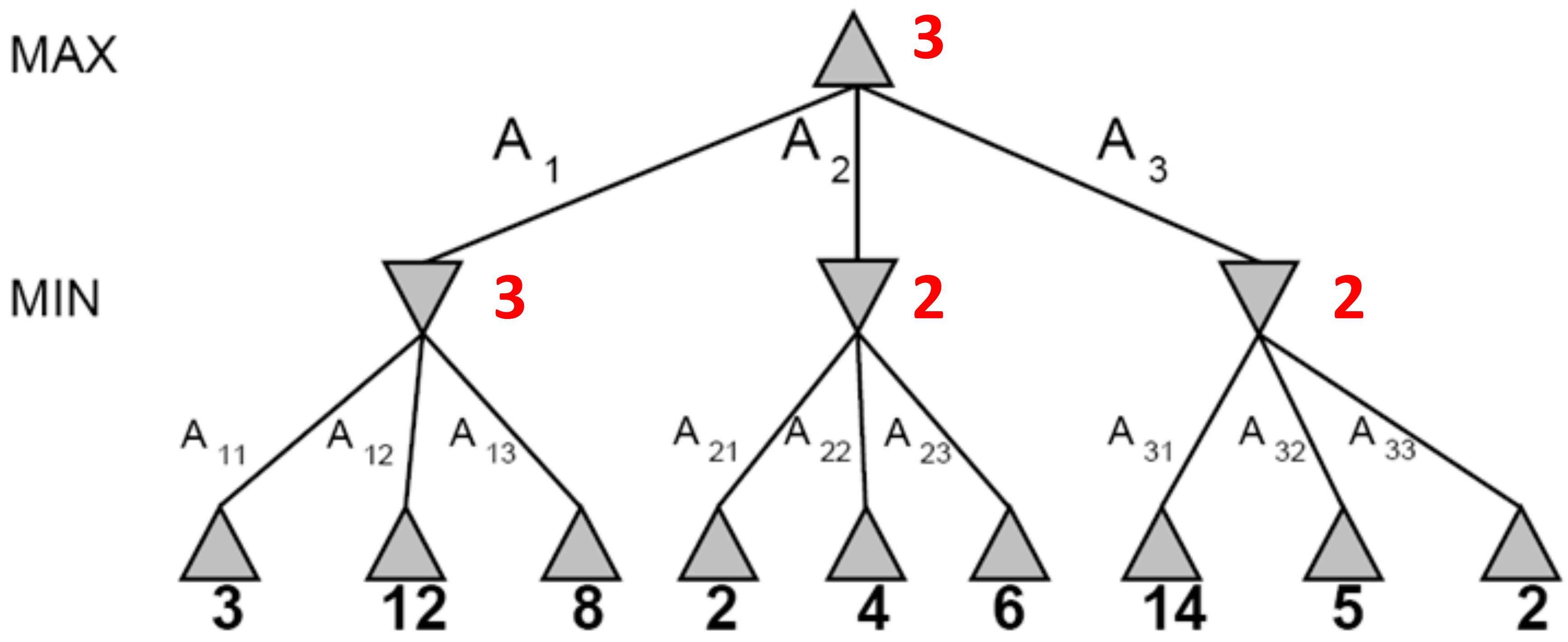
CS440/ECE448 Lecture 11: Alpha-Beta Pruning; Limited Horizon

Slides by Mark Hasegawa-Johnson & Svetlana Lazebnik, 2/2020



By Karl Gottlieb von Windisch - Copper engraving from the book: Karl Gottlieb von Windisch, Briefe über den Schachspieler des Hrn. von Kempelen, nebst drei Kupferstichen die diese berühmte Maschine vorstellen. 1783.Original Uploader was Schaelss (talk) at 11:12, 7. Apr 2004., Public Domain, https://commons.wikimedia.org/w/index.php?curid=424092

Minimax Search

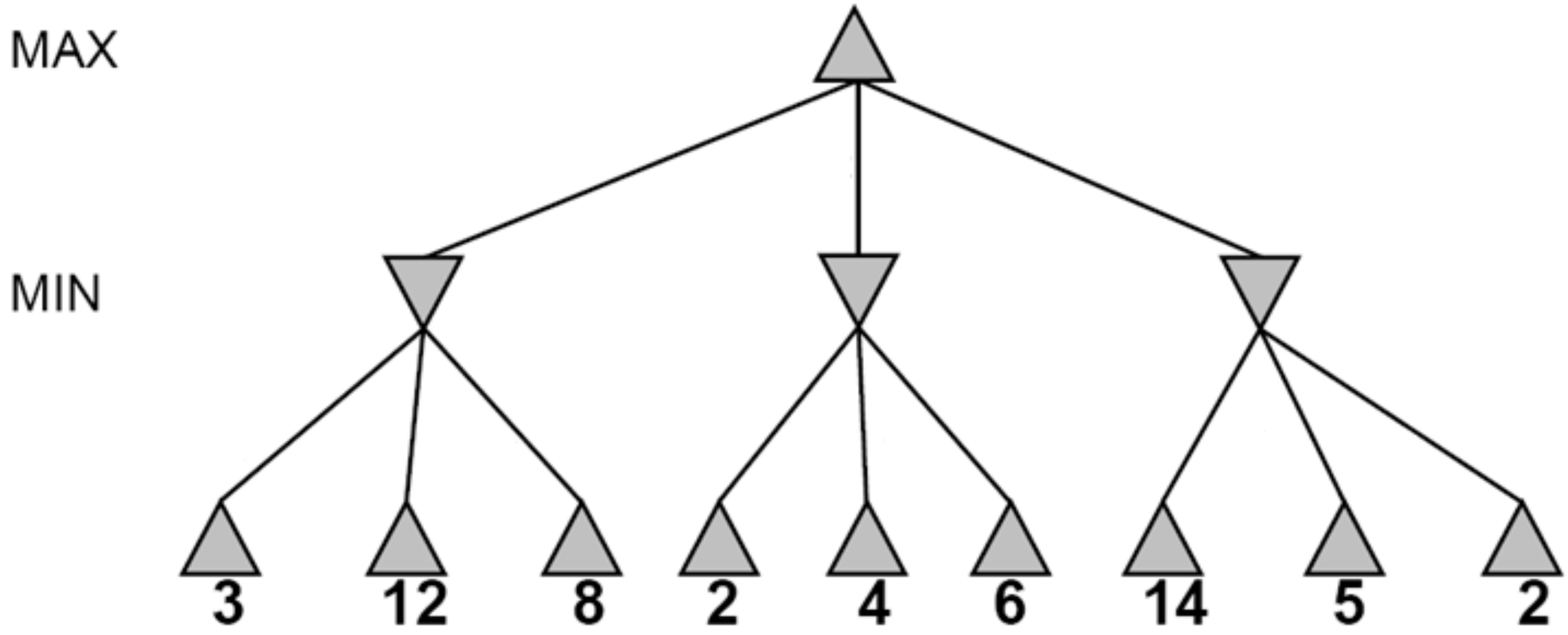


- **Minimax**(*node*) =
  - Utility(*node*) if *node* is terminal
  - $\max_{action}$ **Minimax**(Succ(*node, action*)) if *player* = MAX
  - $\min_{action}$ **Minimax**(Succ(*node, action*)) if *player* = MIN

Alpha-Beta Pruning

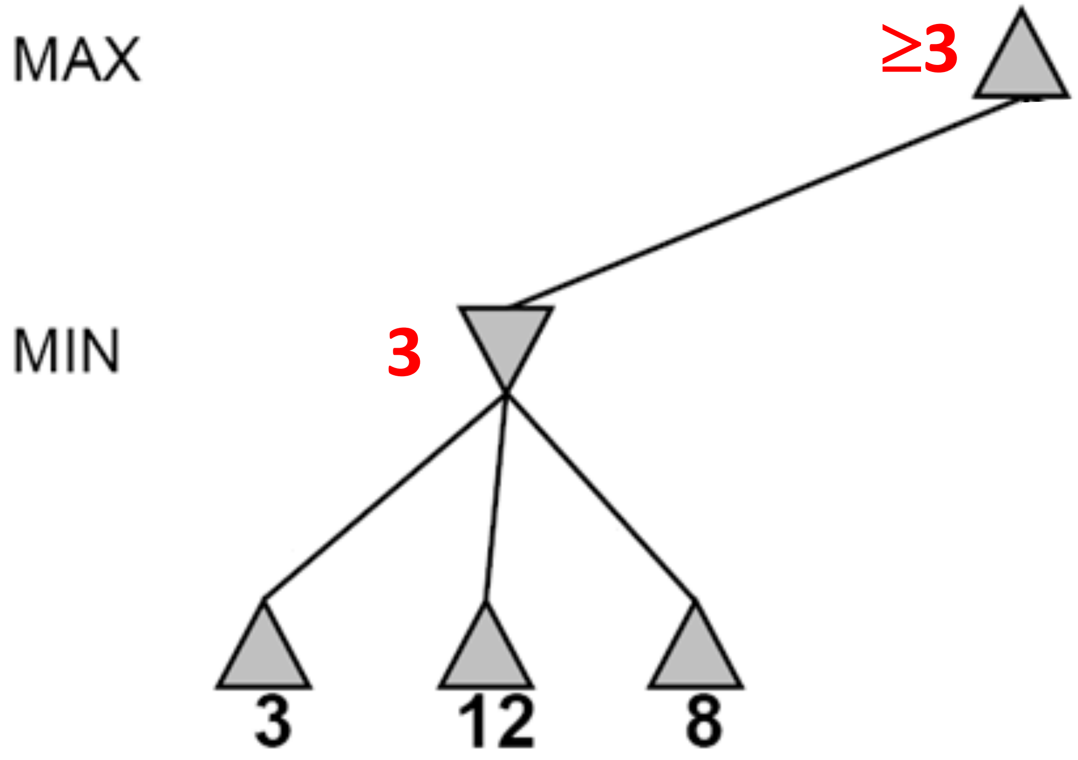
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



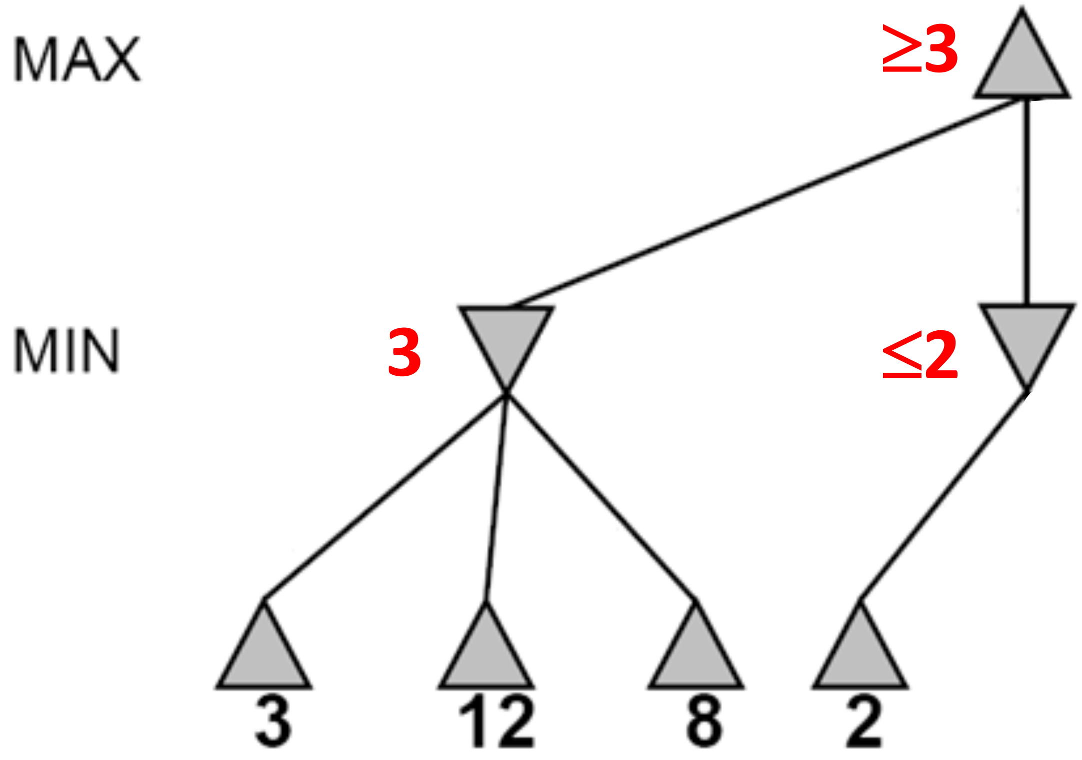
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



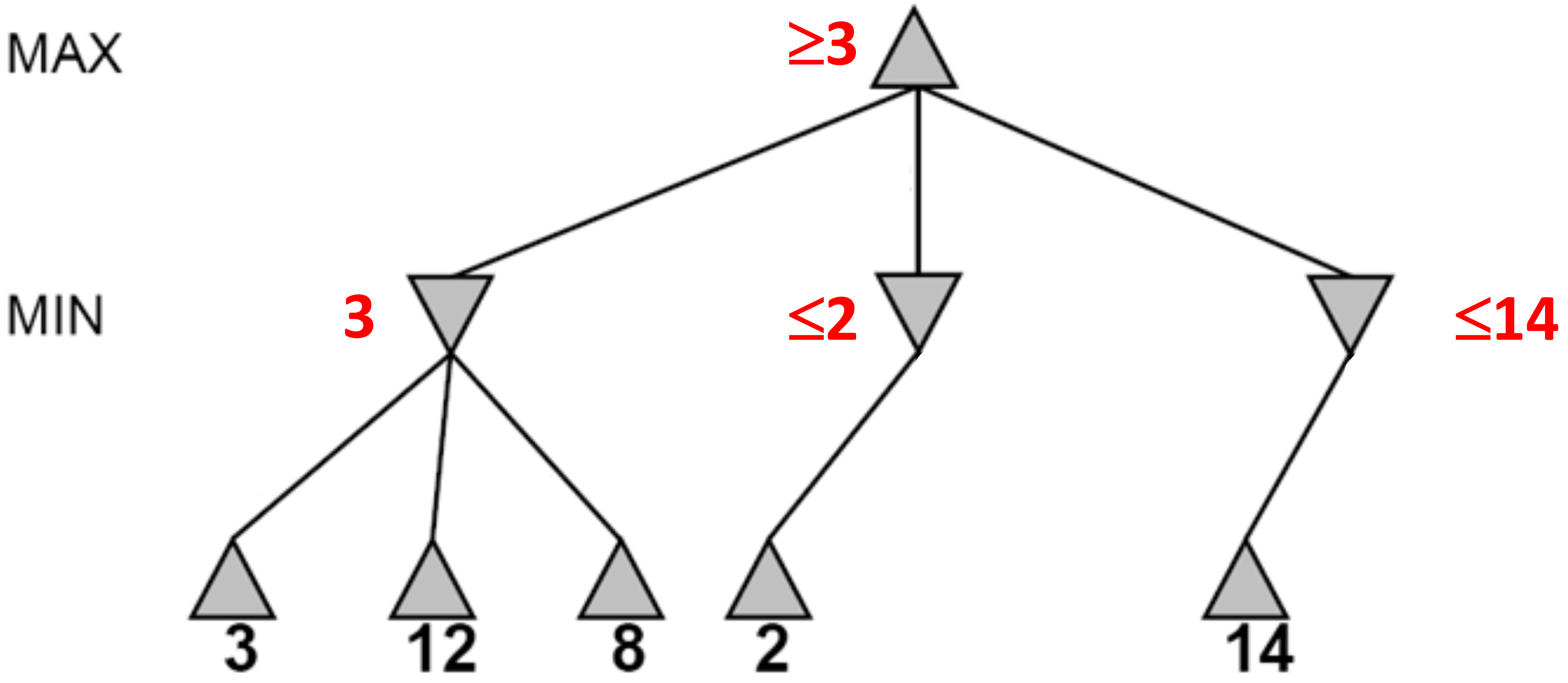
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



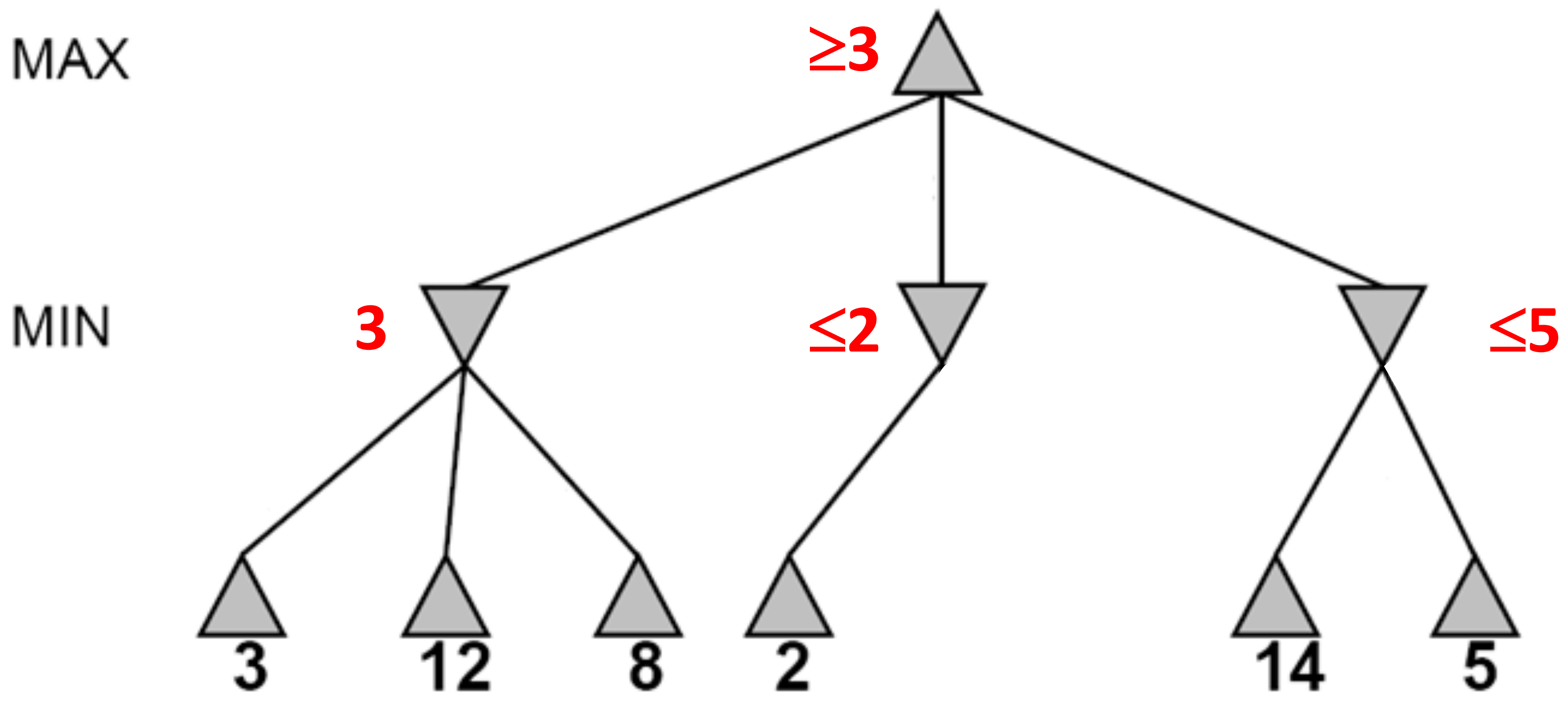
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



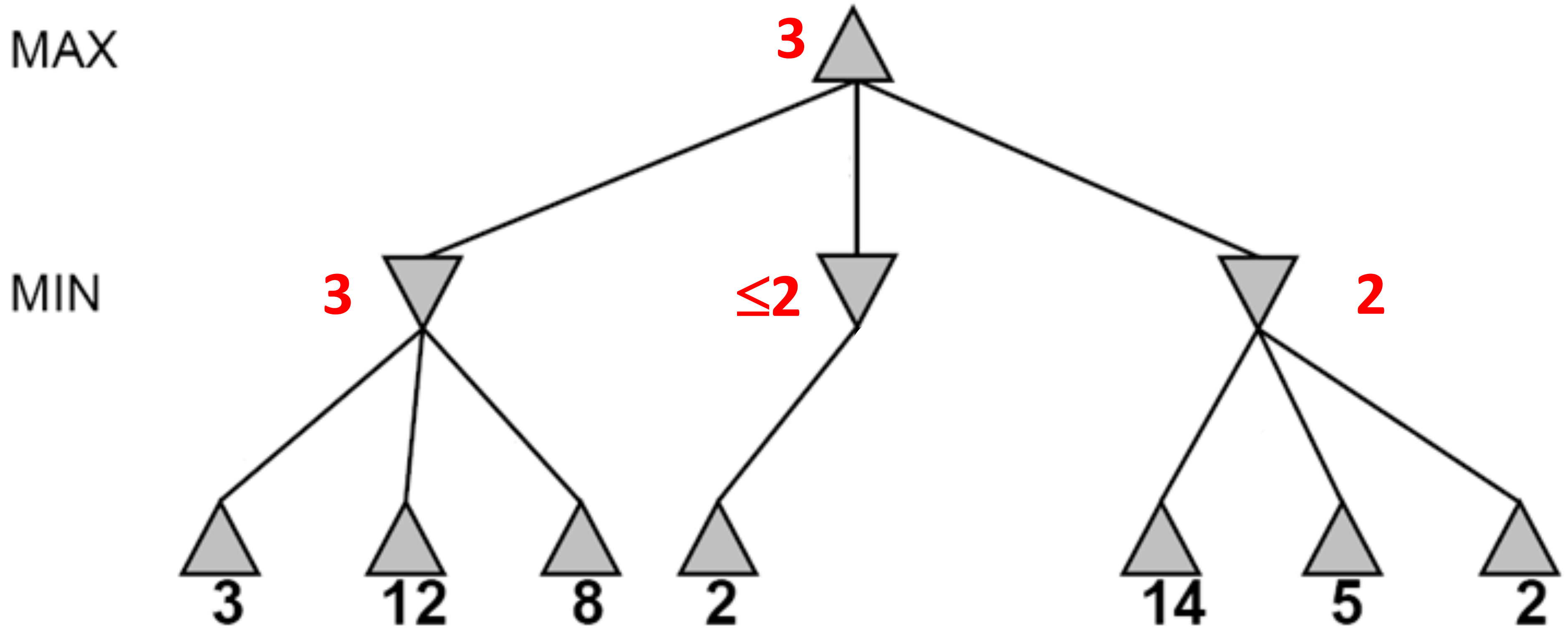
Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



Alpha-beta pruning

- It is possible to compute the exact minimax decision without expanding every node in the game tree



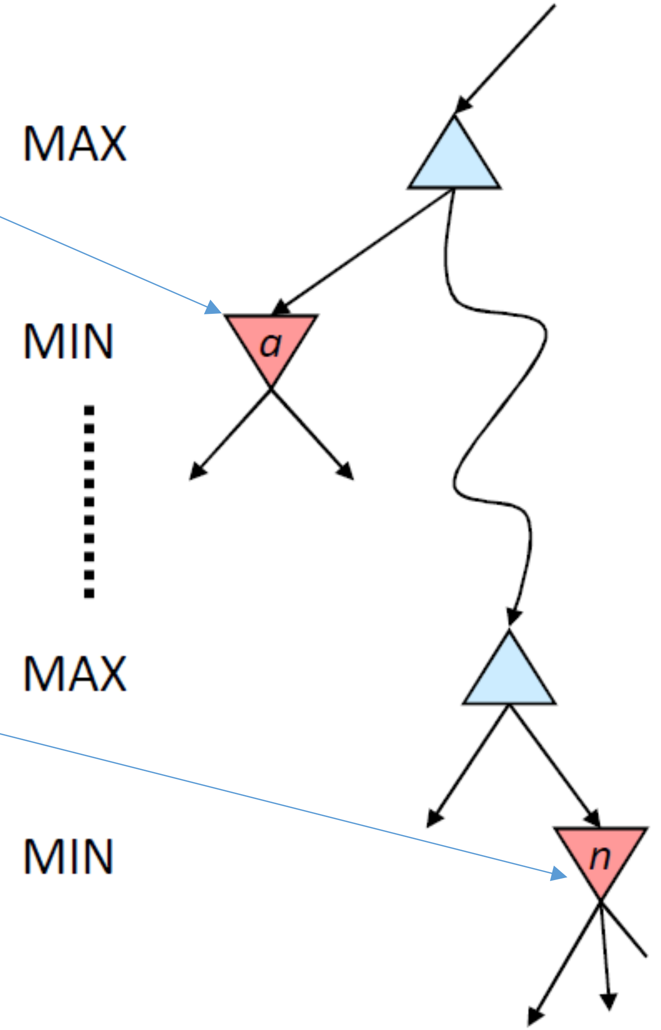
Alpha-Beta Pruning

Key point that I find most counter-intuitive:

- If MIN discovers that, at a particular node in the tree, she can make a move that’s REALLY REALLY GOOD for her…
- She can assume that MAX will never let her reach that node.
- … and she can prune it away from the search, and never consider it again.

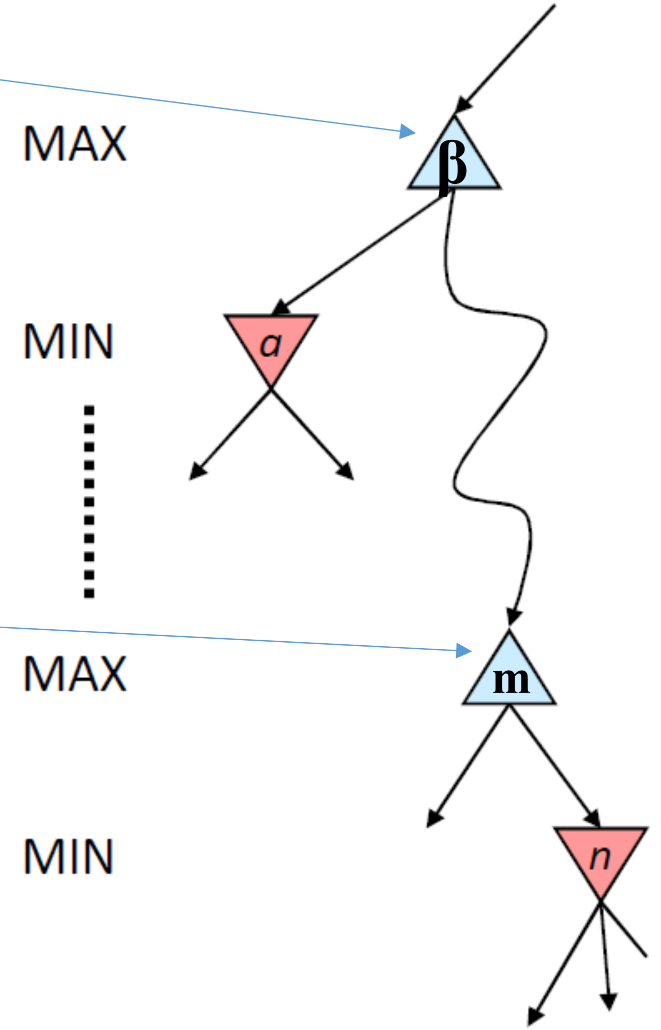
Alpha pruning: Nodes MIN can't reach

- **α** is the value of the best choice for the MAX player found so far at any choice point above node *n*
- More precisely: **α** is the highest number that MAX knows how to force MIN to accept
- We want to compute the MIN-value at *n*
- As we loop over *n*'s children, the MIN-value decreases
- If it drops below **α**, MAX will never choose *n*, so we can ignore *n*'s remaining children



Beta pruning: Nodes MAX can't reach

- **β** is the value of the best choice for the **MIN** player found so far at any choice point above node *m*
- More precisely: **β** is the lowest number that **MIN** know how to force **MAX** to accept
- We want to compute the MAX-value at *m*
- As we loop over *m*'s children, the MAX-value increases
- If it rises above **β**, MIN will never choose *m*, so we can ignore *m*'s remaining children



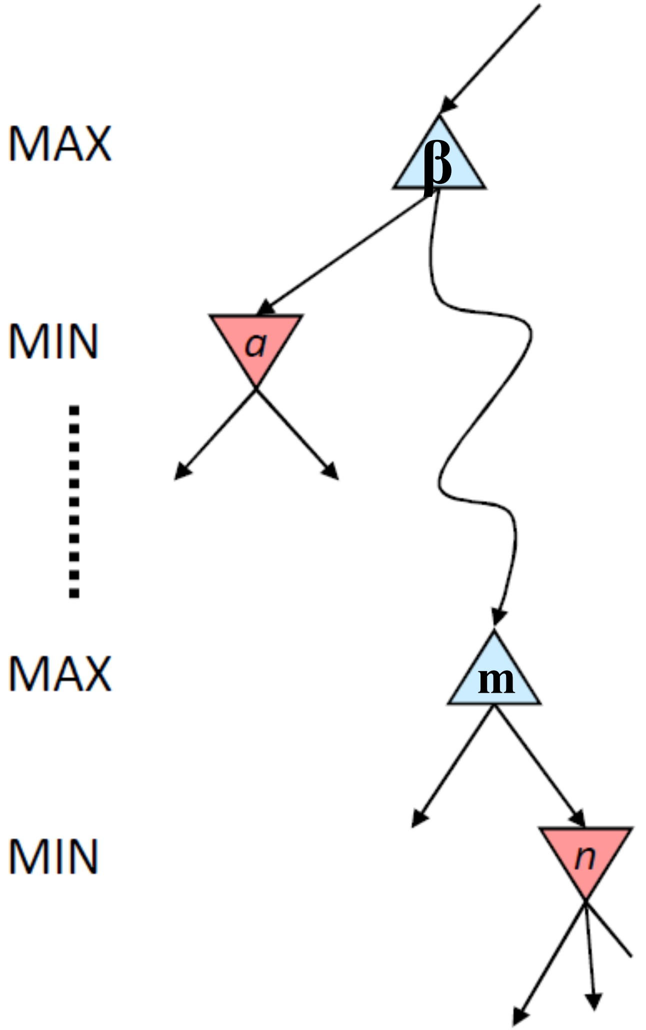
Alpha-beta pruning

An unexpected result:

- **α** is the highest number that MAX knows how to force MIN to accept
- **β** is the lowest number that **MIN** know how to force **MAX** to accept

So

$$\alpha \leq \beta$$



Alpha-beta pruning

Function *action* = **Alpha-Beta-Search**(*node*)

v = **Min-Value**(*node*, $-\infty$, ∞)

return the *action* from *node* with value *v*

α: best alternative available to the Max player

β: best alternative available to the Min player

Function *v* = **Min-Value**(*node*, *α*, *β*)

if Terminal(*node*) return Utility(*node*)

v = $+\infty$

for each *action* from *node*

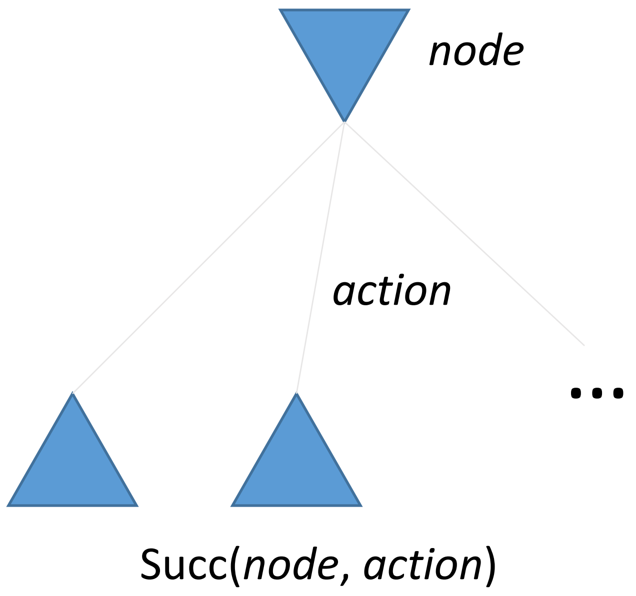
v = Min(*v*, **Max-Value**(Succ(*node*, *action*), *α*, *β*))

if $v \leq \alpha$ return *v*

β = Min(*β*, *v*)

end for

return *v*



Alpha-beta pruning

Function *action* = **Alpha-Beta-Search**(*node*)

v = **Max-Value**(*node*, $-\infty$, ∞)

return the *action* from *node* with value v

α: best alternative available to the Max player

β: best alternative available to the Min player

Function v = **Max-Value**(*node*, α, β)

if Terminal(*node*) return Utility(*node*)

$v = -\infty$

for each *action* from *node*

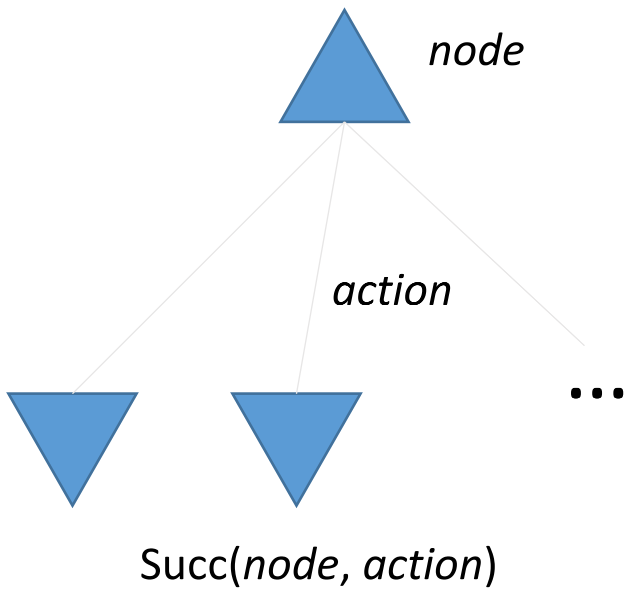
v = Max(v, **Min-Value**(Succ(*node*, *action*), α, β))

if $v \geq \beta$ return v

α = Max(α, v)

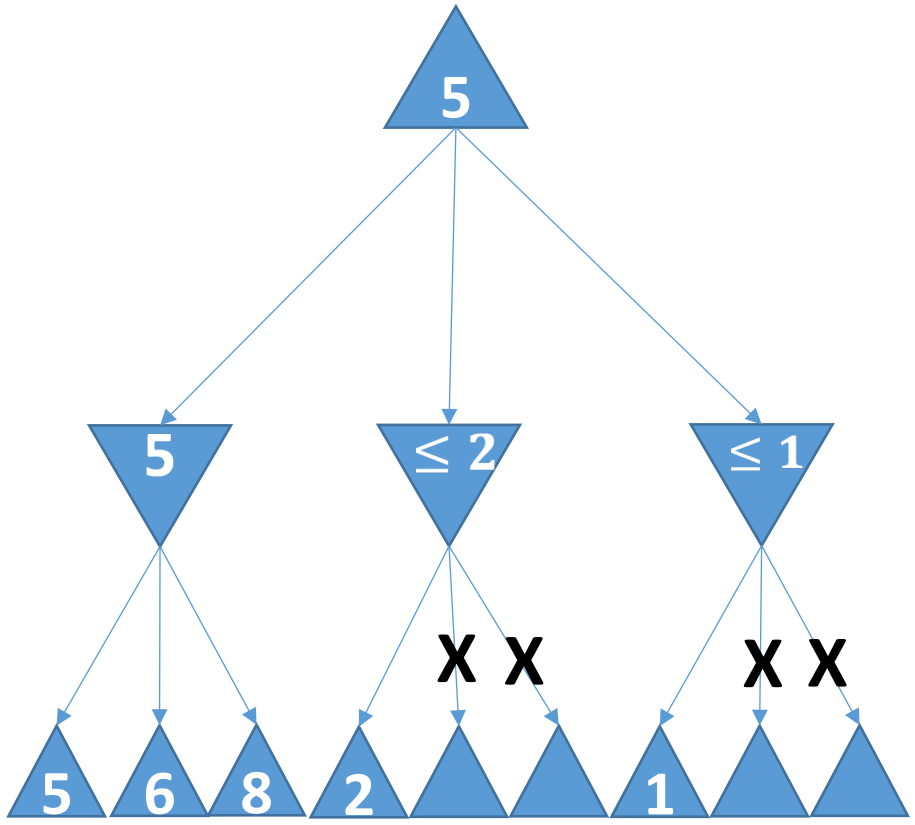
end for

return v



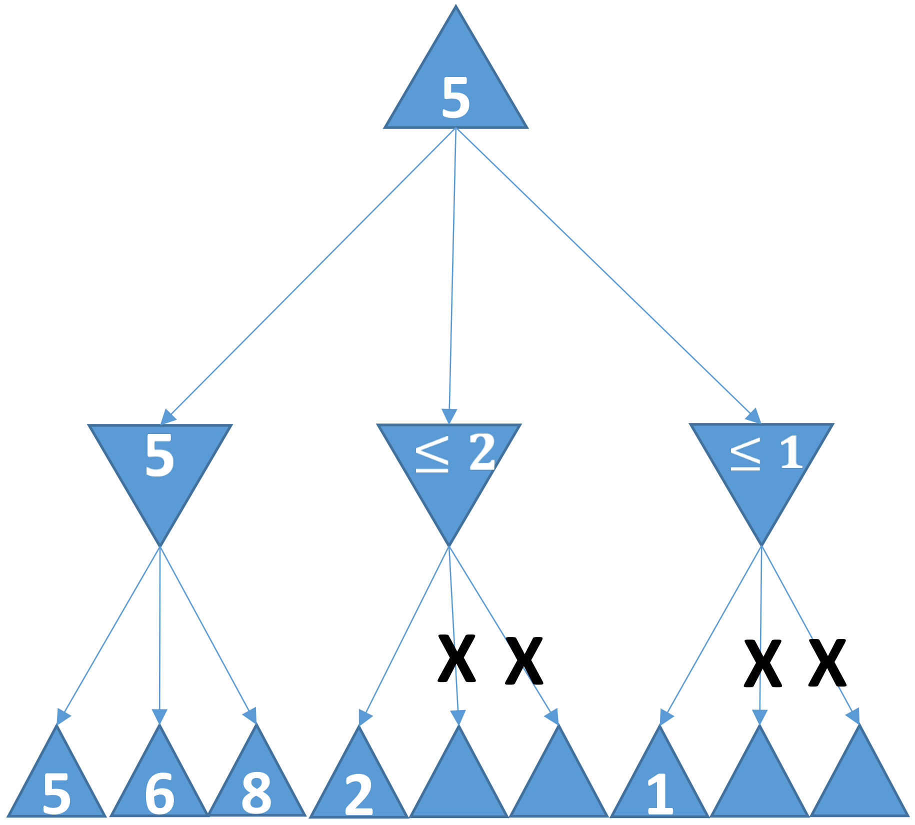
Alpha-beta pruning is optimal!

- Pruning does not affect final result



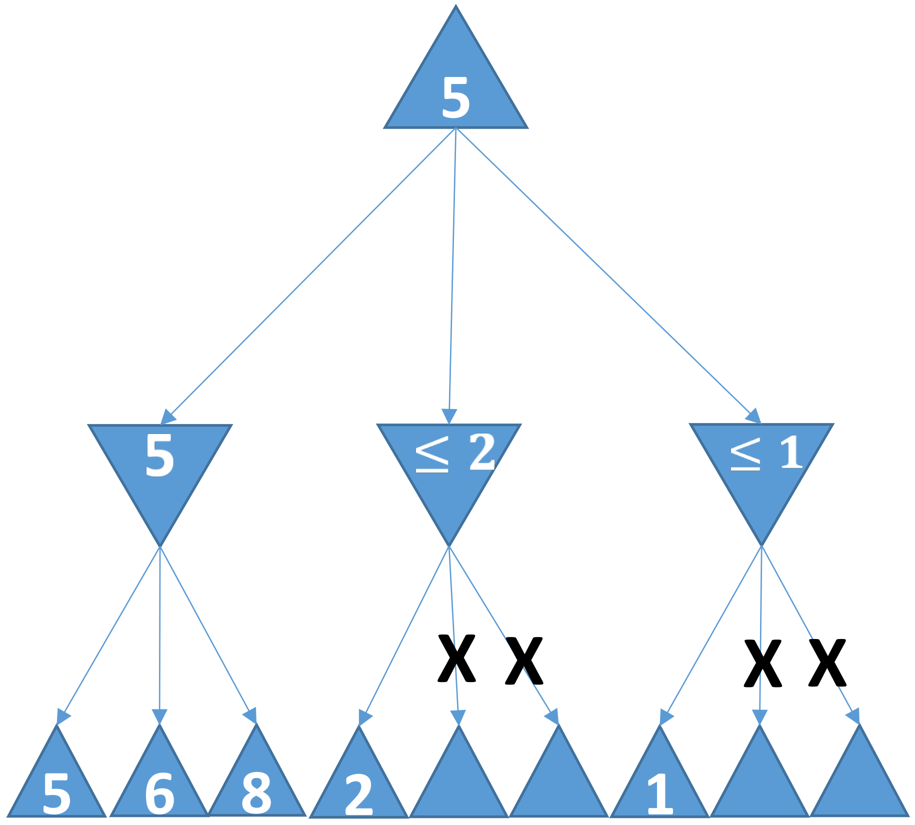
Alpha-beta pruning: Complexity

- Amount of pruning depends on move ordering
  - Should start with the “best” moves (highest-value for MAX or lowest-value for MIN)
- With perfect ordering, I have to evaluate:
  - ALL OF THE GRANDCHILDREN who are daughters of my FIRST CHILD, and
  - The FIRST GRANDCHILD who is a daughter of each of my REMAINING CHILDREN



Alpha-beta pruning: Complexity

- With perfect ordering:
  - With a branching factor of b, I have to evaluate only $2b - 1$ of my grandchildren, instead of b^2.
  - So the total computational complexity is reduced from $O\{b^m\}$ to $O\left\{b^{\frac{m}{2}}\right\}$
  - Exponential reduction in complexity!
  - Equivalently: with the same computational power, you can search a tree that is twice as deep.



Limited-Horizon Computation

Games vs. single-agent search

- We don’t know how the opponent will act
  - The solution is not a fixed sequence of actions from start state to goal state, but a ***strategy*** or ***policy*** (a mapping from state to best move in that state)

Computational complexity…

- In order to decide how to move at node n, we need to search all possible sequences of moves, from n until the end of the game

Computational complexity...

- The branching factor, search depth, and number of terminal configurations are huge
  - In chess, branching factor ≈ 35 and depth ≈ 100, giving a search tree of $\mathbf{35^{100} \approx 10^{154}}$ nodes
  - Number of atoms in the observable universe ≈ 10^{80}
  - This rules out searching all the way to the end of the game

Limited-horizon computing

- Cut off search at a certain depth (called the "horizon")
  - With a 10 gigaflops laptop = 10^9 operations/second, you can compute a tree of about $10^9 \approx 35^6$, i.e., your horizon is just 6 moves.
  - Blue Waters has 13.3 petaflops = 1.3×10^{16}, so it can compute a tree of about $10^{16} \approx 35^{11}$, i.e., the entire Blue Waters supercomputer, playing chess, can only search a game tree with a horizon of about 11 moves into the future.
- Obvious fact: after 11 moves, nobody has won the game yet (usually)…
- so you don't know the TRUE value of any node at a horizon of just 11 moves.

Limited-horizon computing

The solution implemented by every chess-playing program ever written:

- Search out to a horizon of m moves (thus, a tree of size b^m).
- For each of those b^m terminal states S_i ($0 \leq i < b^m$), use some kind of **evaluation function** to estimate the probability of winning, $p(S_i)$.
- Then use minimax or alpha-beta to propagate those $p(S_i)$ back to the start node, so you can choose the best move to make in the starting node.
- At the next move, push the tree one step farther into the future, and repeat the process.

Evaluation functions

How can we estimate the evaluation function?

- Use a neural net (or maybe just a logistic regression) to estimate $p(S_i)$ from a training database of human vs. human games.
  - … or by playing two computers against one another.
- Most of the possible game boards in chess have never occurred in the history of the universe. Therefore we need to approximate $p(S_i)$ by computing some useful features of S_i whose values we have observed, somewhere in the history of the universe.
- Example features: # rooks remaining, position of the queen, relative positions of the queen & king, # steps in the shortest path from the knight to the queen.

Cutting off search

- **Horizon effect:** you may incorrectly estimate the value of a state by overlooking an event that is just beyond the depth limit
  - For example, a damaging move by the opponent that can be delayed but not avoided
- Possible remedies
  - **Quiescence search:** do not cut off search at positions that are unstable – for example, are you about to lose an important piece?
  - **Singular extension:** a strong move that should be tried when the normal depth limit is reached

Chess playing systems

- Baseline system: 200 million node evaluations per move, minimax with a decent evaluation function and quiescence search
  - 5-ply ≈ human novice
- Add alpha-beta pruning
  - 10-ply ≈ typical PC, experienced player
- Deep Blue: 30 billion evaluations per move, singular extensions, evaluation function with 8000 features, large databases of opening and endgame moves
  - 14-ply ≈ Garry Kasparov
- More recent state of the art (Hydra, ca. 2006): 36 billion evaluations per second, advanced pruning techniques
  - 18-ply ≈ better than any human alive?

Summary

- A zero-sum game can be expressed as a minimax tree
- Alpha-beta pruning finds the correct solution. In the best case, it has half the exponent of minimax (can search twice as deeply with a given computational complexity).
- Limited-horizon search is always necessary (you can't search to the end of the game), and always suboptimal.
  - Estimate your utility, at the end of your horizon, using some type of learned utility function
  - Quiescence search: don't cut off the search in an unstable position (need some way to measure "stability")
  - Singular extension: have one or two "super-moves" that you can test at the end of your horizon