Data Structures and Algorithms

CS245-2015S-P4

Two Player Games

David Galles

Department of Computer Science
University of San Francisco

P4-0: Overview

- Example games (board splitting, chess, Network)
- Min/Max trees
- Alpha-Beta Pruning
- Evaluation Functions

P4-1: Two player games

- Board-Splitting Game
  - Two players, V & H
  - V splits the board vertically, selects one half
  - H splits the board horizontally, selects one half
  - V tries to maximize the final value, H tries to minimize the final value

14	5	11	4
12	13	9	7
15	13	10	8
16	1	6	2

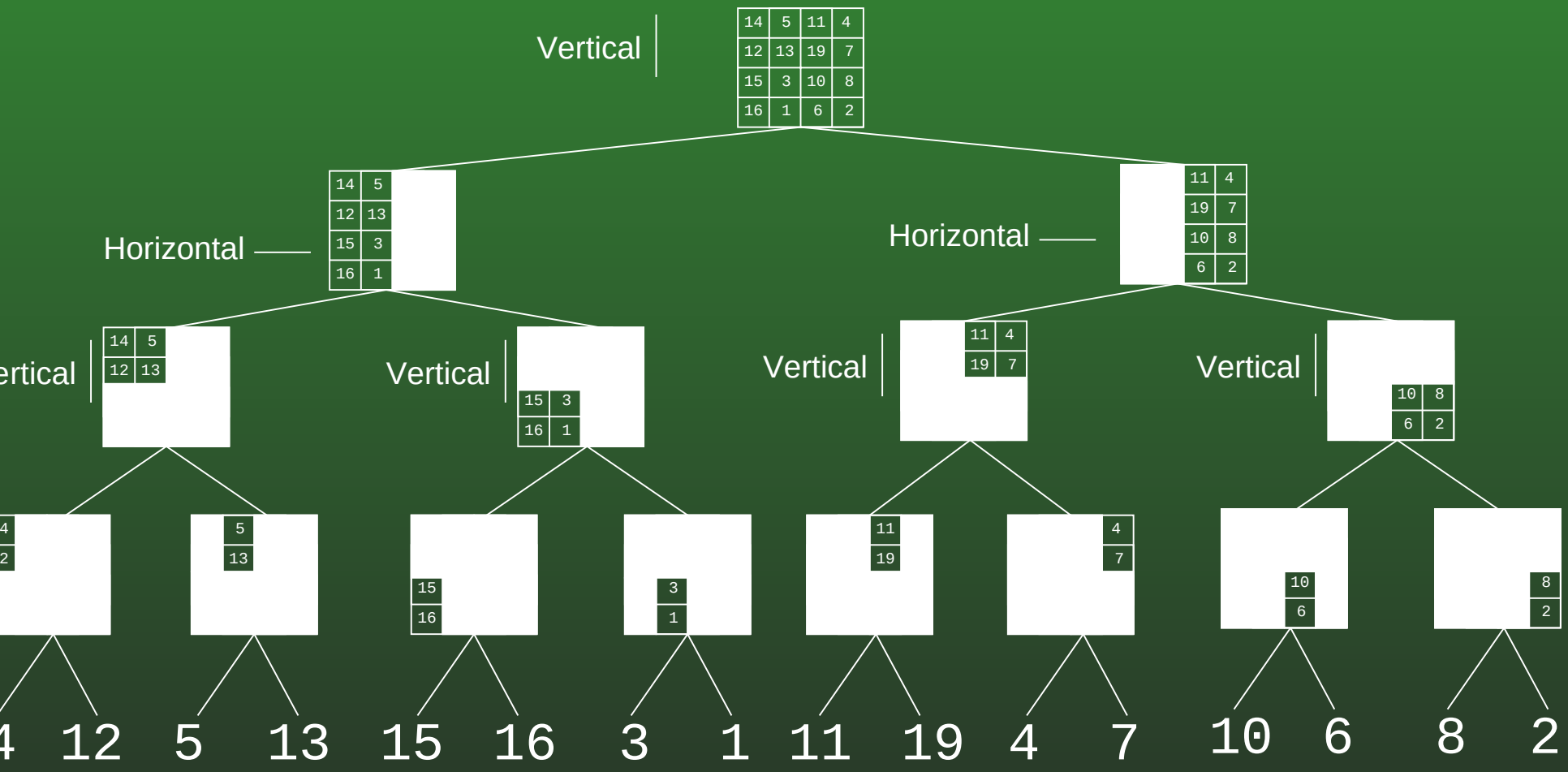
P4-2: Two player games

- Board-Splitting Game
  - We assume that both players are rational (make the best possible move)
  - How can we determine who will win the game?

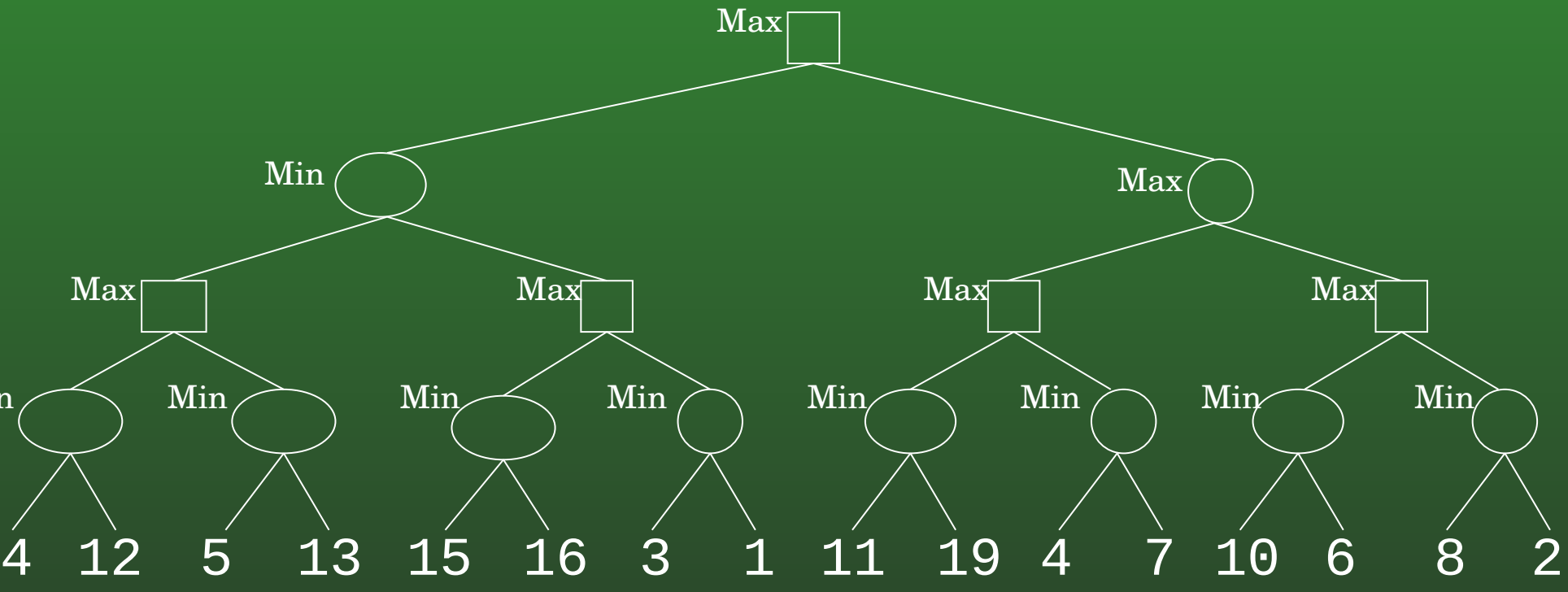
P4-3: Two player games

- Board-Splitting Game
  - We assume that both players are rational (make the best possible move)
  - How can we determine who will win the game?
    - Examine all possible games!

P4-4: Two player games



P4-5: Two player games



P4-6: Two player games

P4-7: Two player games

- Game playing agent can do this to figure out which move to make
  - Examine all possible moves
  - Examine all possible responses to each move
  - ... all the way to the last move
  - Caclulate the value of each move (assuming opponent plays perfectly)

P4-8: Minimax Algorithm

```
Max(node)
  if terminal(node)
     return utility(node)
  maxVal = MIN_VALUE
  children = successors(node)
  for child in children
     maxVal = max(maxVal, Min(child))
  return maxVal

Min(node)
  if terminal(node)
     return utility(node)
  minVal = MAX_VALUE
  children = successors(node)
  for child in children
     minVal = min(minVal, Max(child))
  return minVal
```

P4-9: Minimax Algorithm

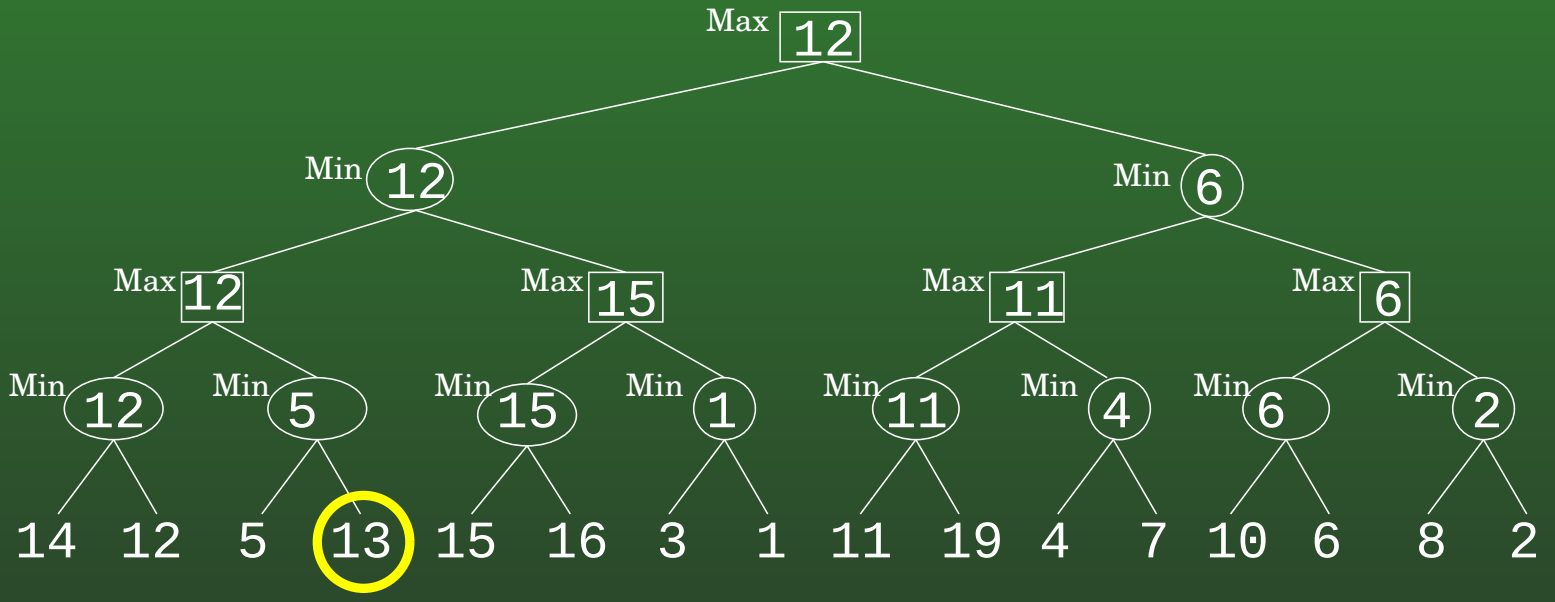
- Branching factor of b, game length of d moves, what are the time and space requirements for Minimax?

P4-10: Minimax Algorithm

- Branching factor of b, game length of d moves, what are the time and space requirements for Minimax?
  - Time: $O(b^d)$
  - Space: $O(d)$
- Not managable for any real games – chess has an average b of 35, can't search the entire tree
- Need to make this more managable

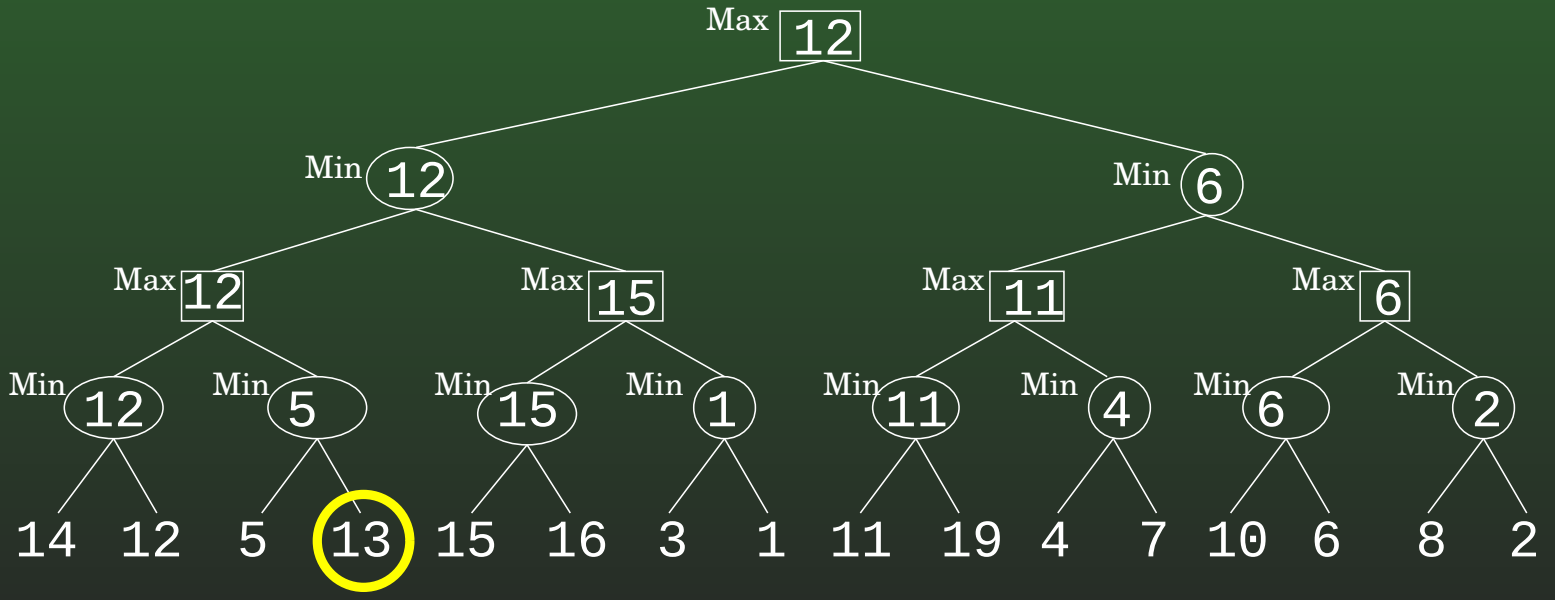
P4-11: Alpha-Beta Pruning

- Does it matter what value is in the yellow circle?



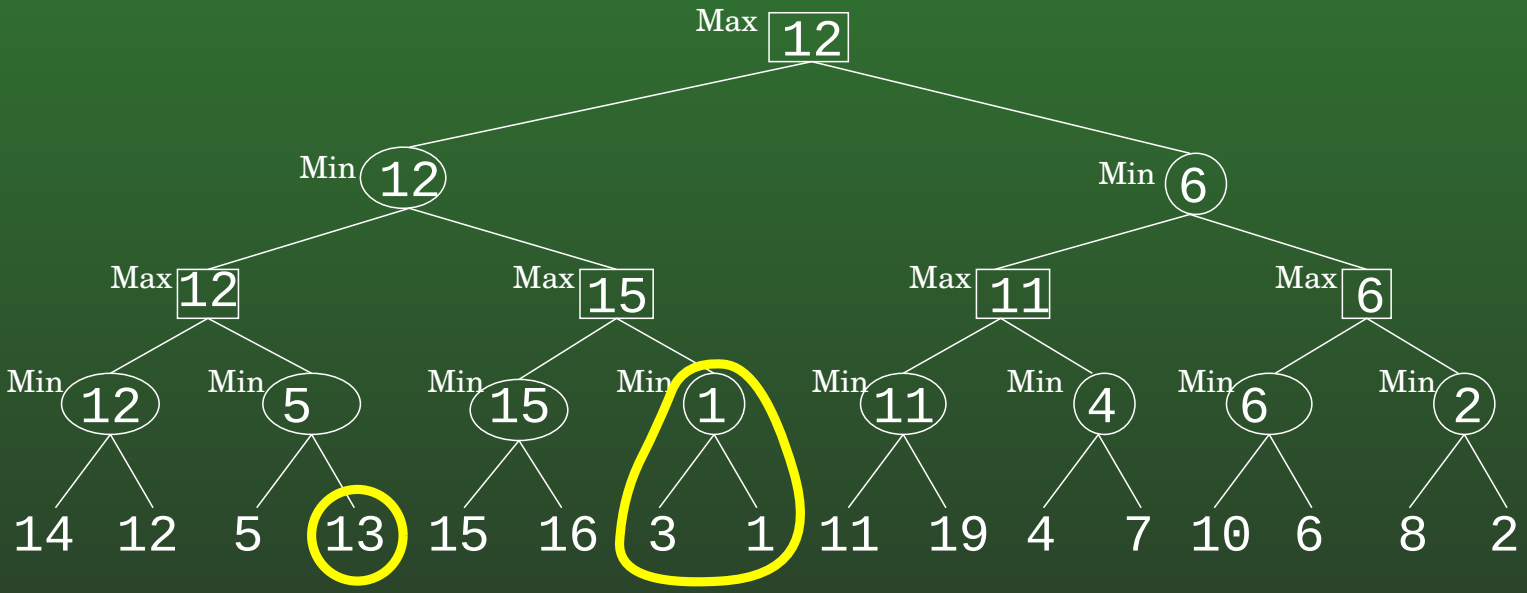
P4-12: Alpha-Beta Pruning

- If the yellow leaf has a value > 5, parent won't pick it
- If the yellow leaf has a value < 12, grandparent won't pick it
- To affect the root, value must be < 5 **and** > 12



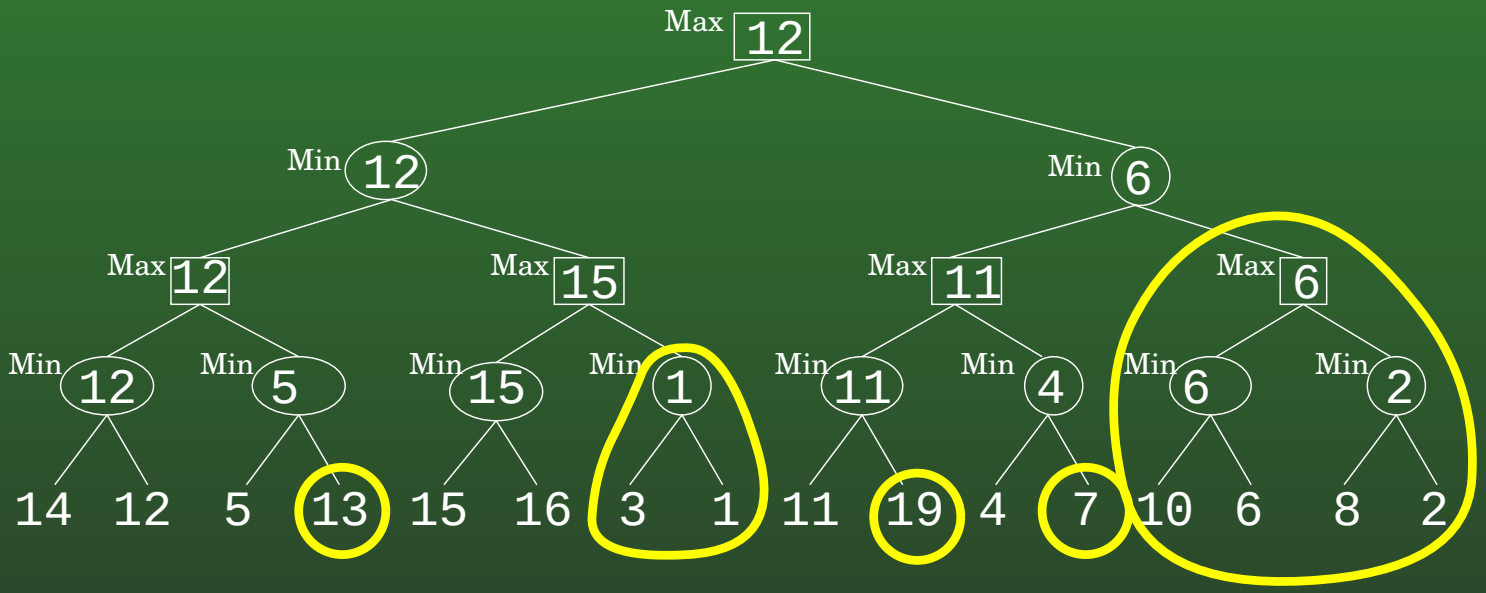
P4-13: Alpha-Beta Pruning

- Value of nodes in neither yellow circle matter. Are there more?

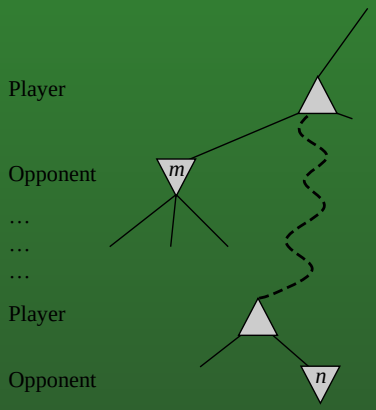


P4-14: Alpha-Beta Pruning

- Value of nodes in none of the yellow circles matter.



P4-15: Alpha-Beta Pruning

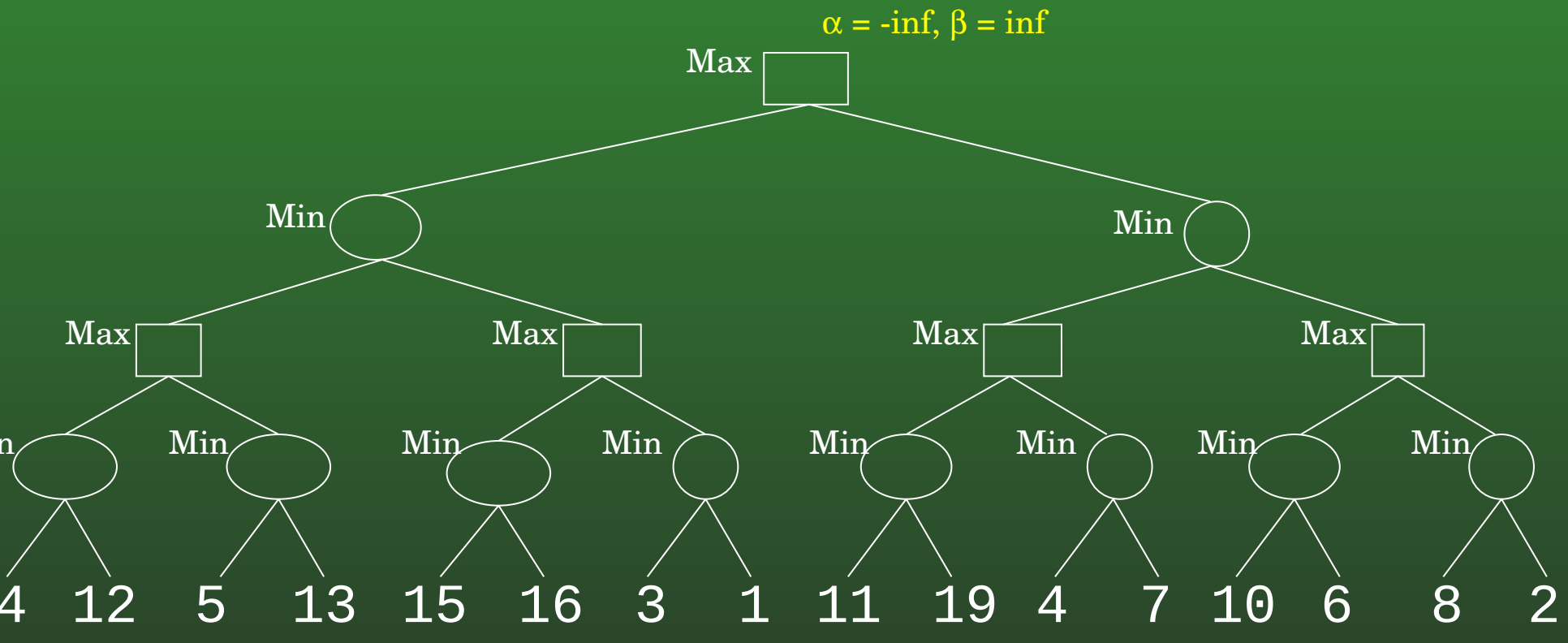


- If m is better than n for Player, we will never reach n
  - (player would pick m instead)

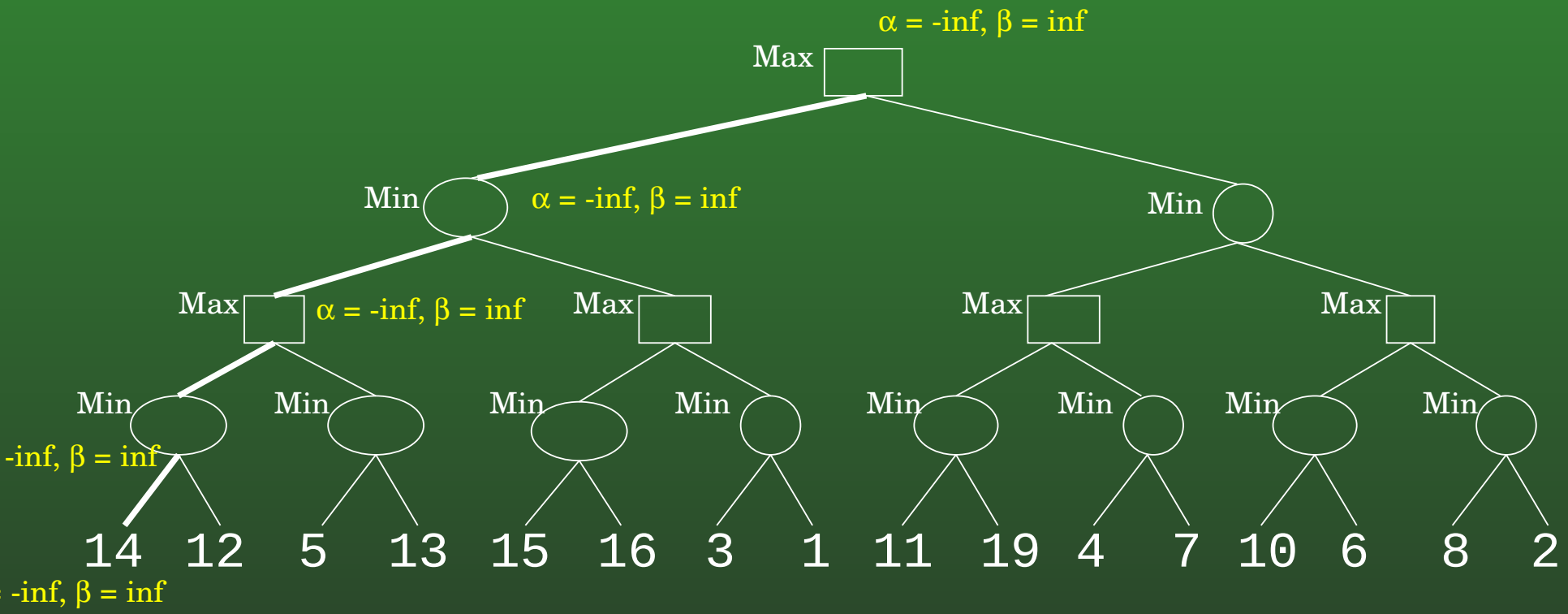
P4-16: Alpha-Beta Pruning

- Maintain two bounds, lower bound α, and an upper bound β
  - Bounds represent the values the node must have to possibly affect the root
- As you search the tree, update the bounds
  - Max nodes increase α, min nodes decrease β
- If the bounds ever cross, this branch cannot affect the root, we can prune it.

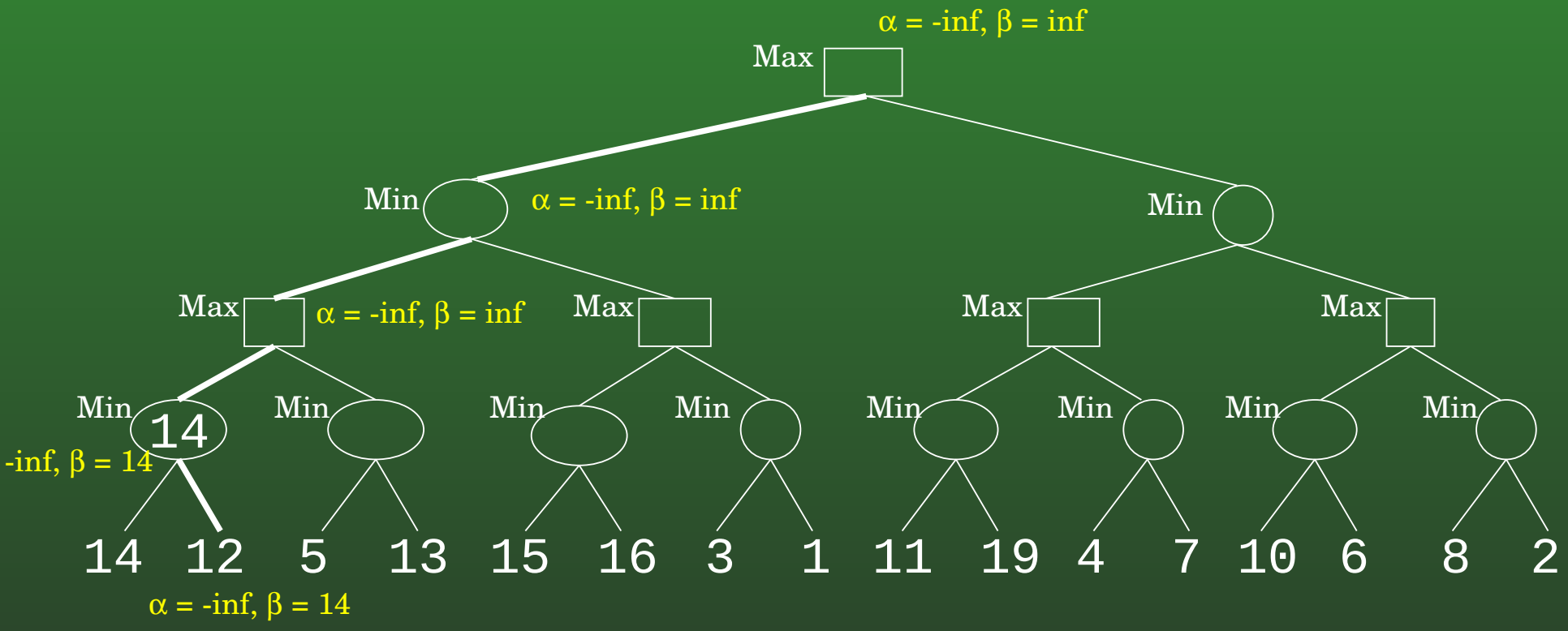
P4-17: Alpha-Beta Pruning



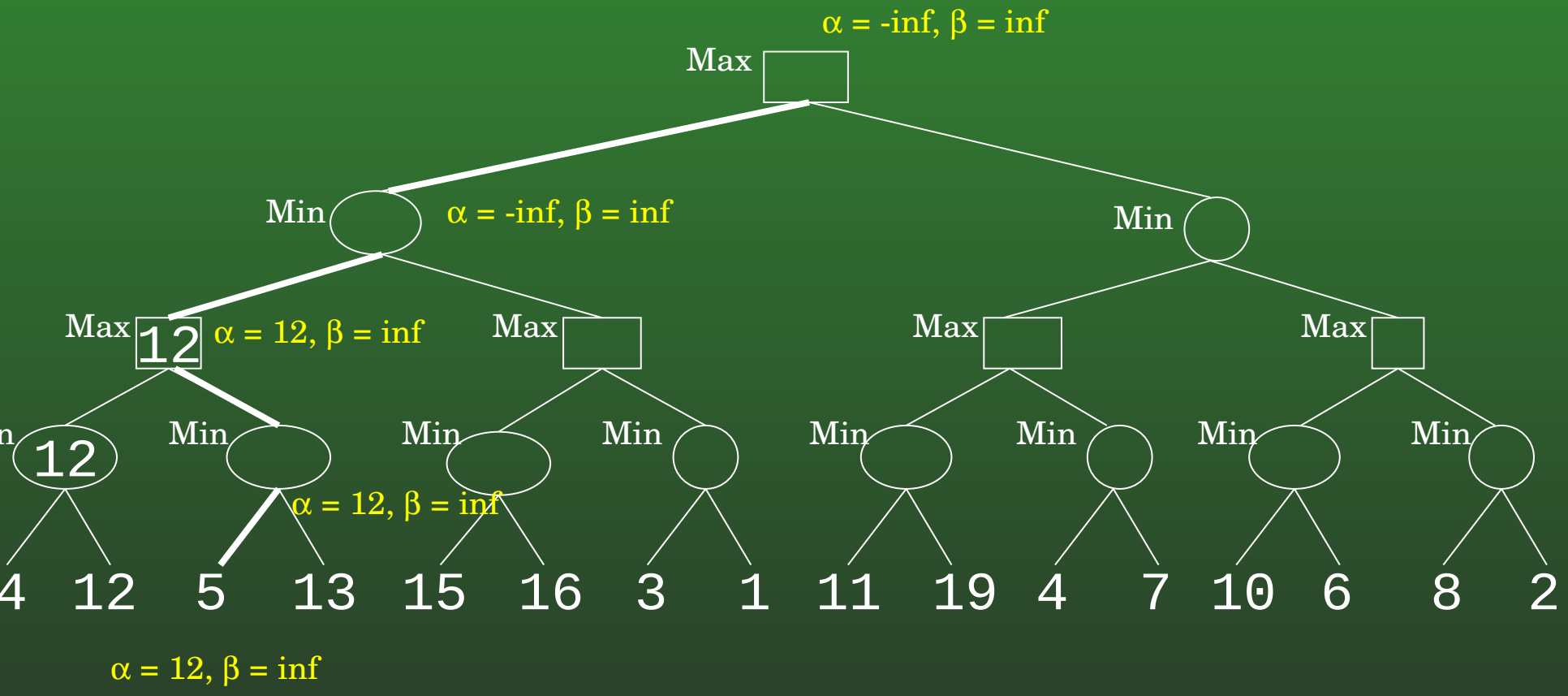
P4-18: Alpha-Beta Pruning



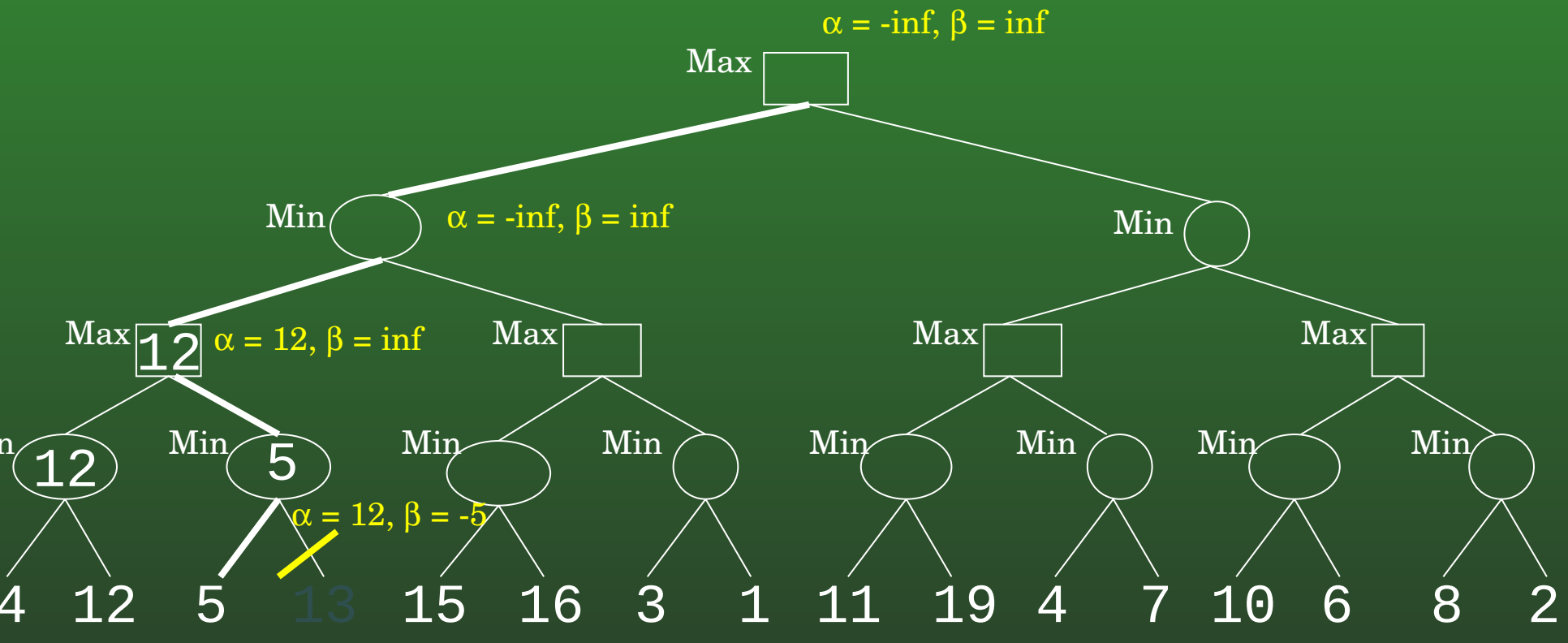
P4-19: Alpha-Beta Pruning



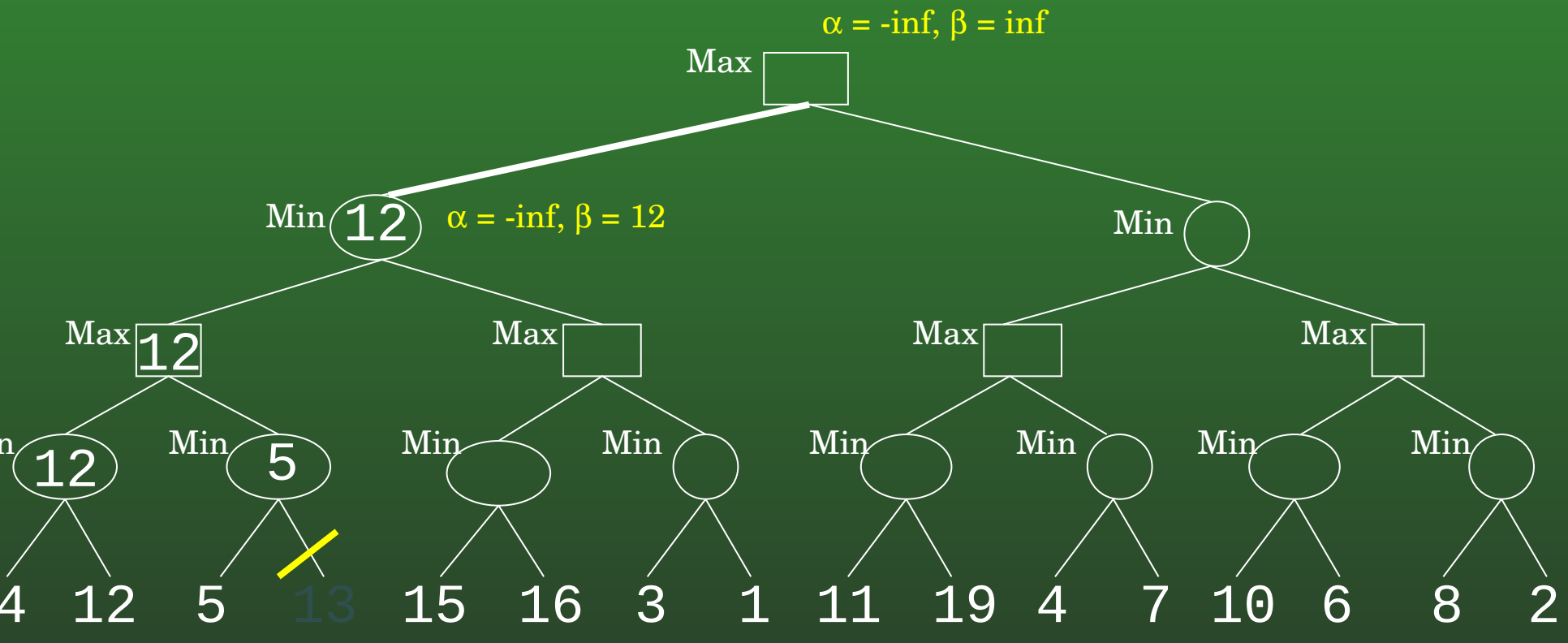
P4-20: Alpha-Beta Pruning



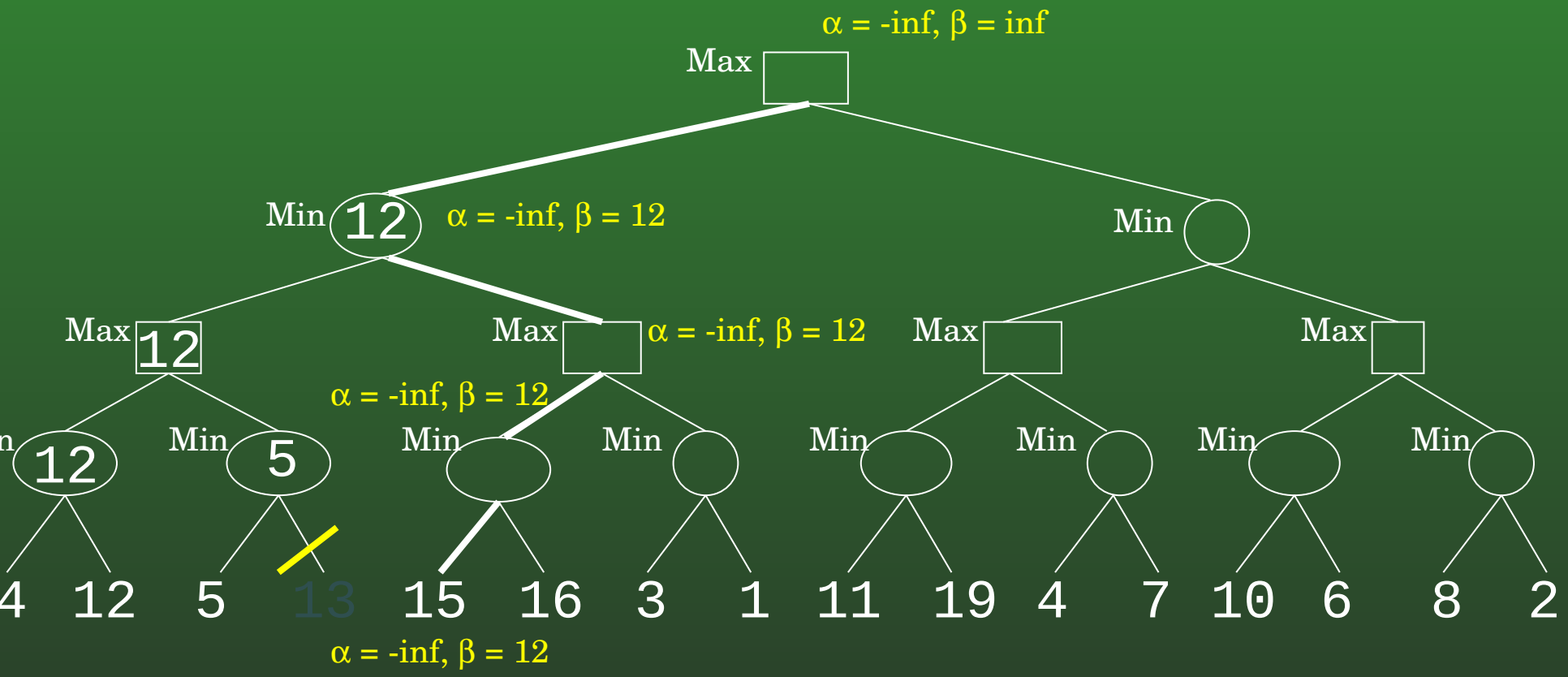
P4-21: Alpha-Beta Pruning



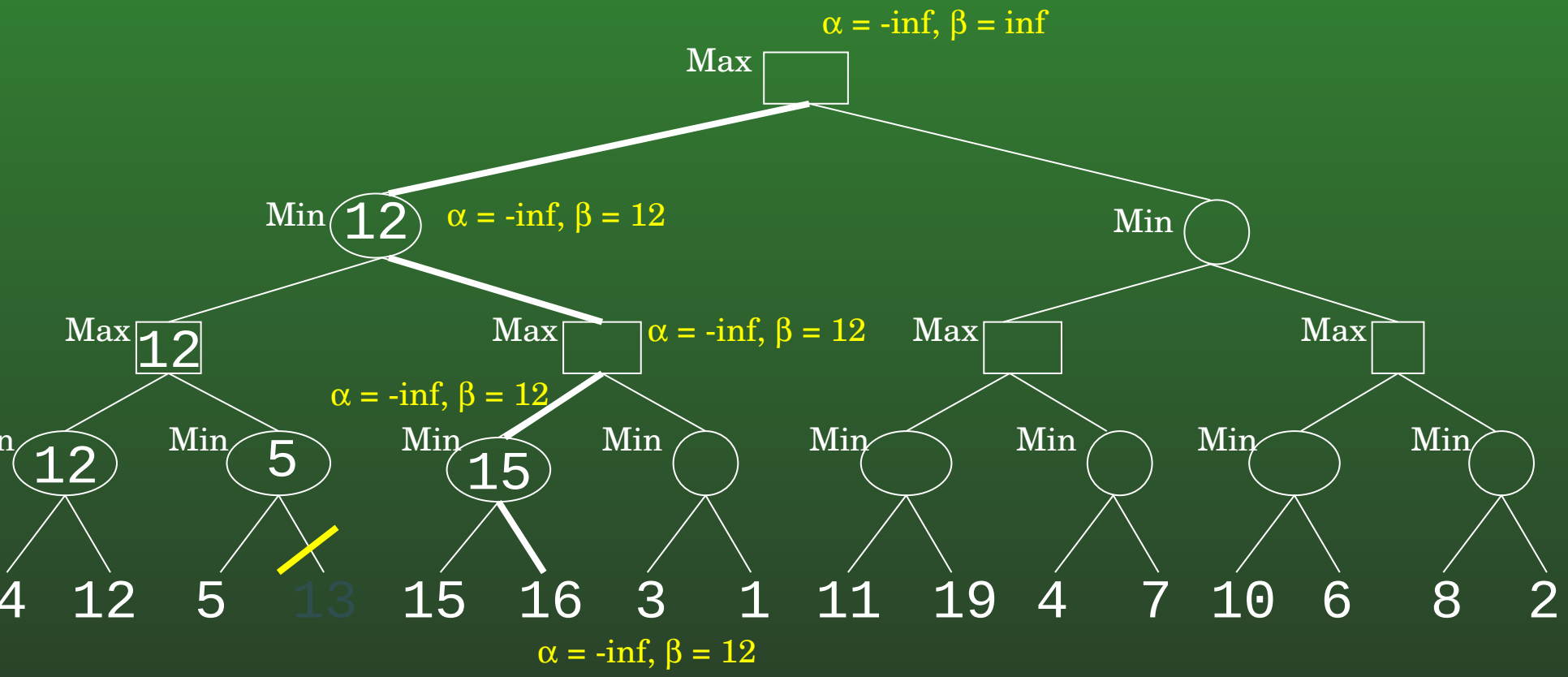
P4-22: Alpha-Beta Pruning



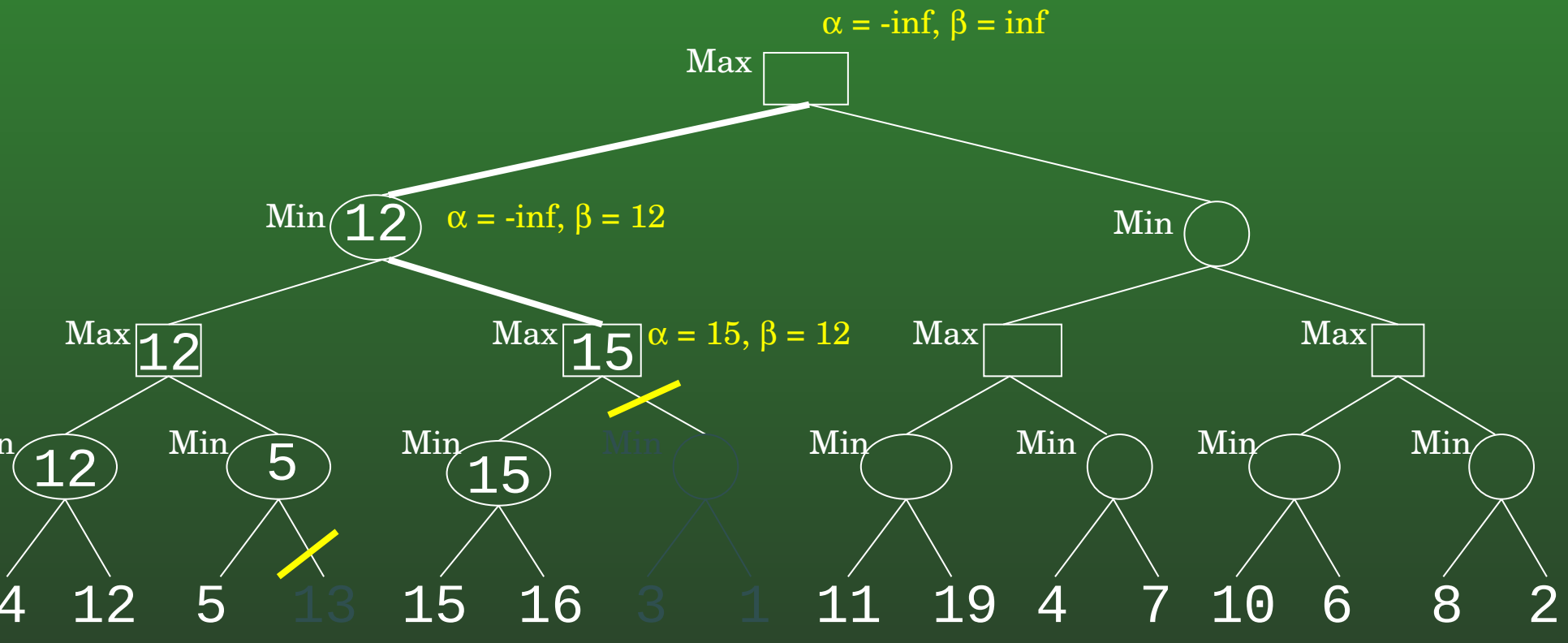
P4-23: Alpha-Beta Pruning



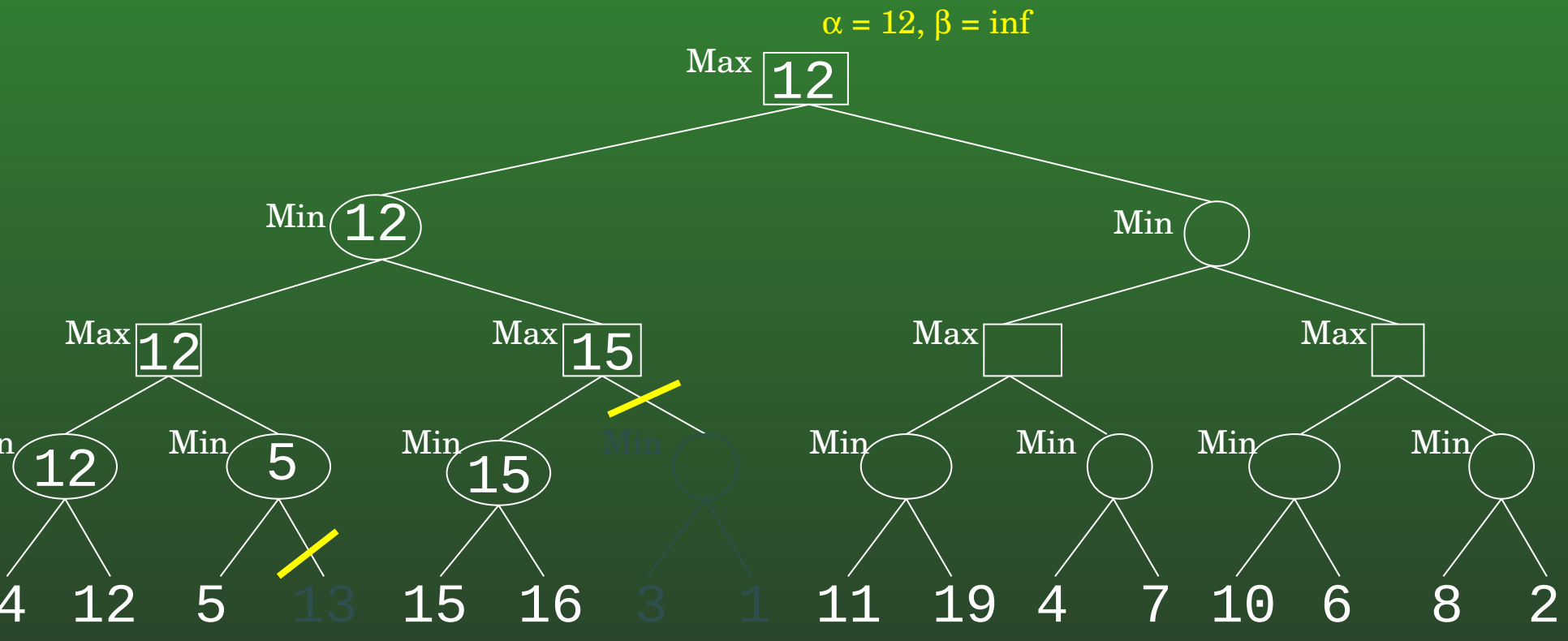
P4-24: Alpha-Beta Pruning



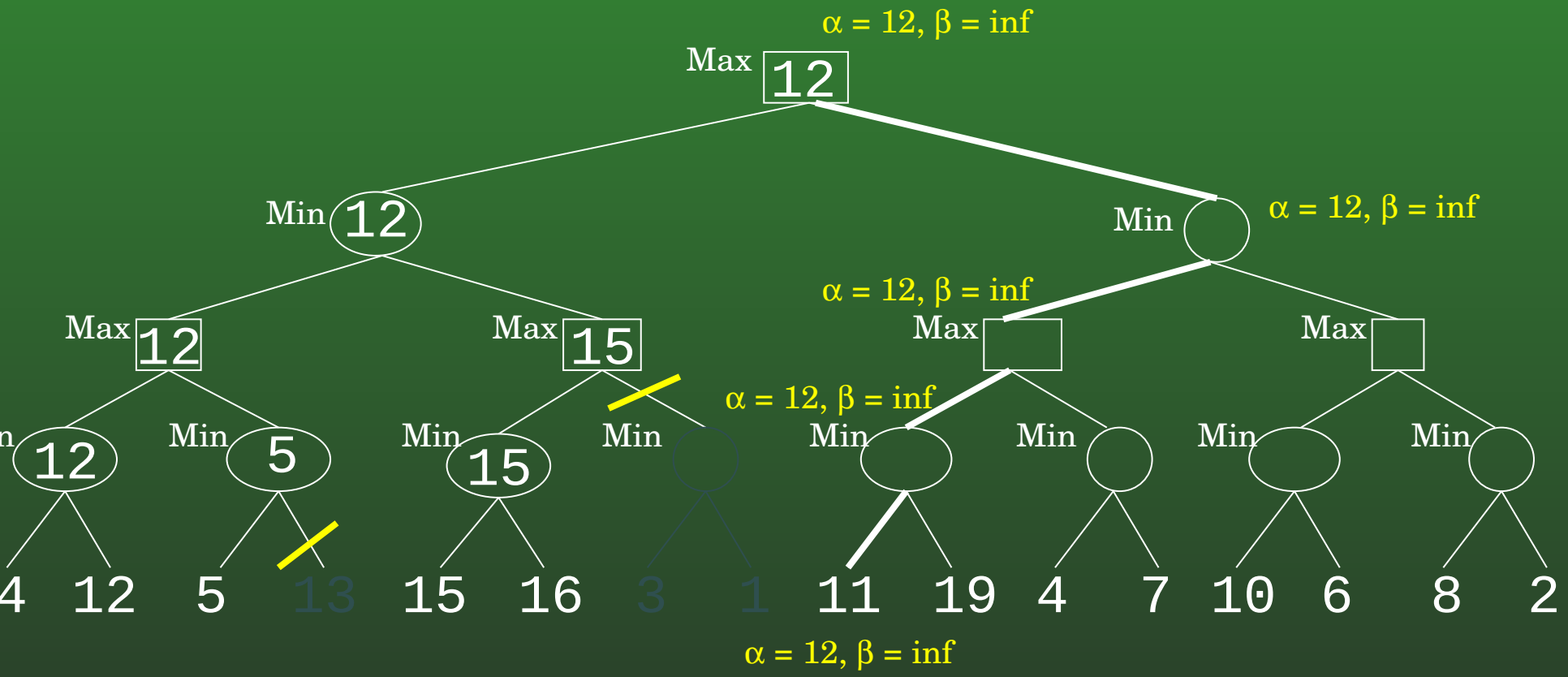
P4-25: Alpha-Beta Pruning



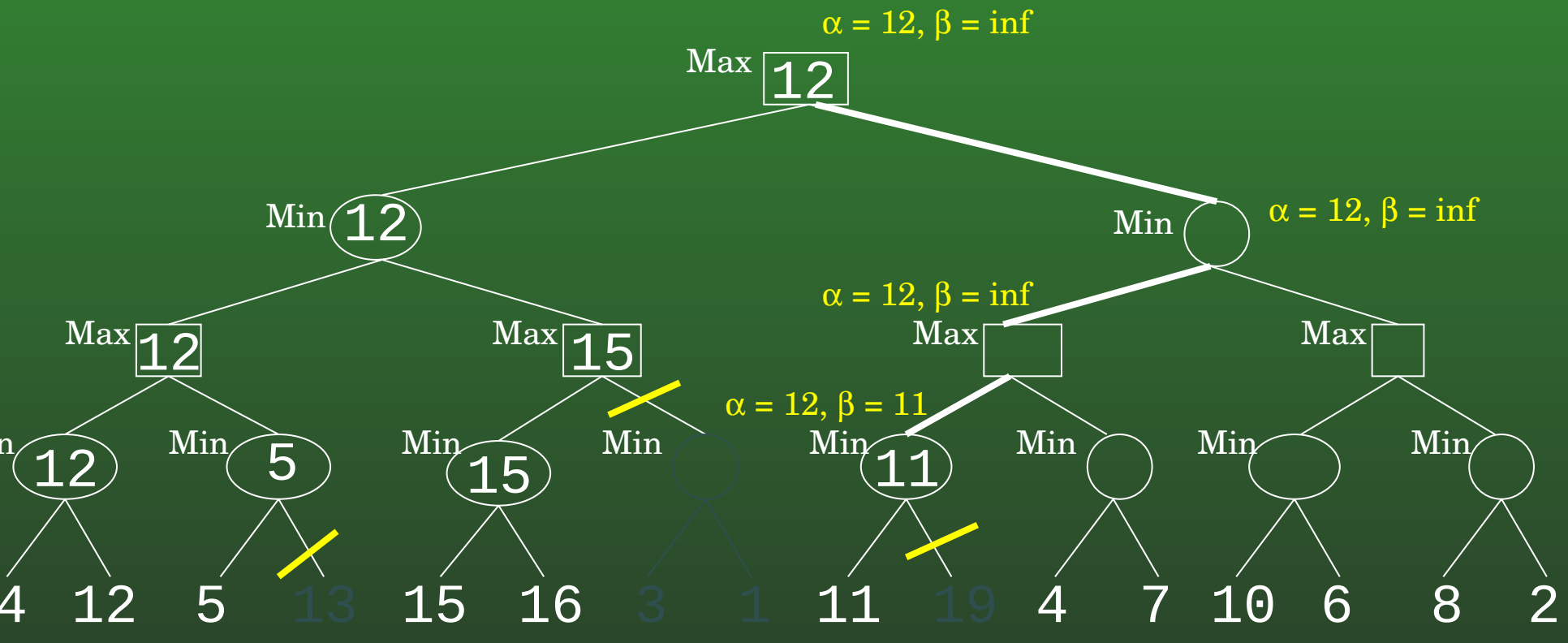
P4-26: Alpha-Beta Pruning



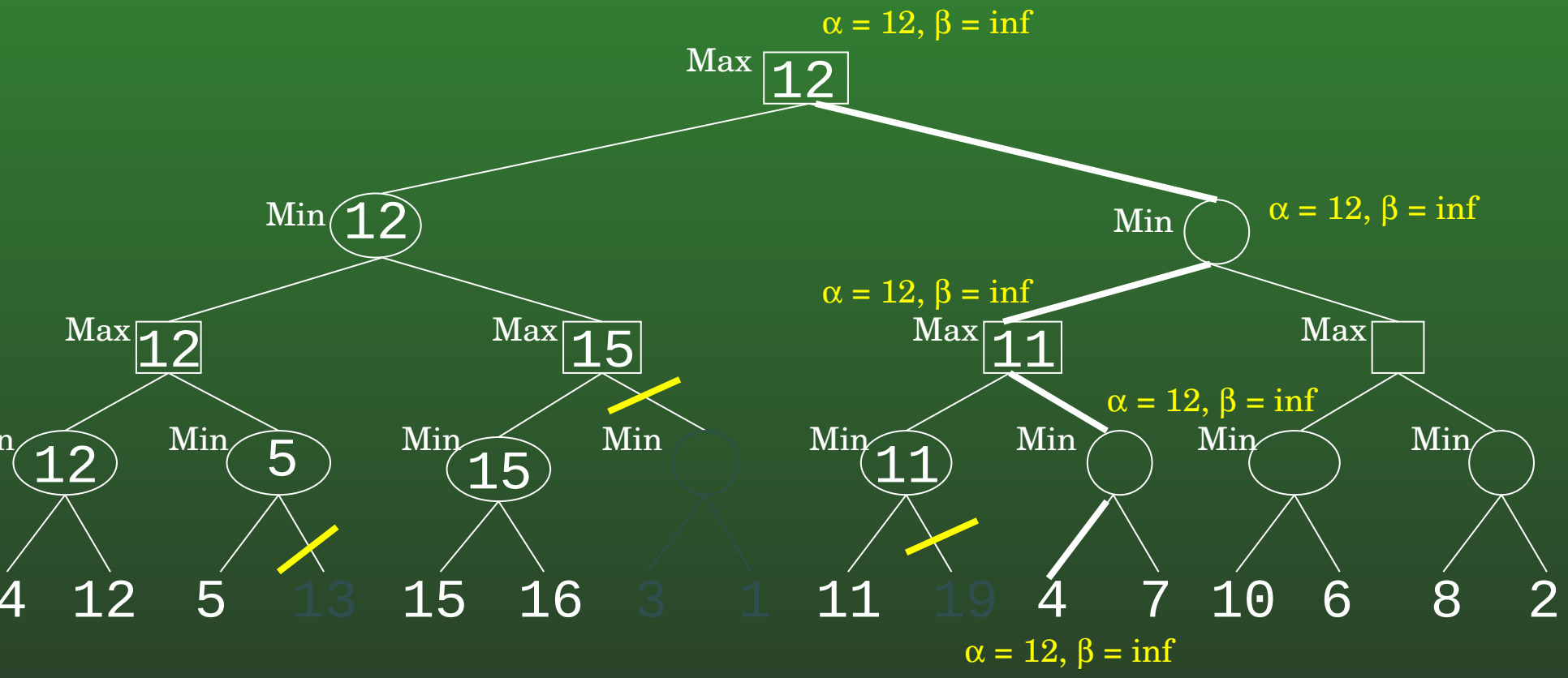
P4-27: Alpha-Beta Pruning



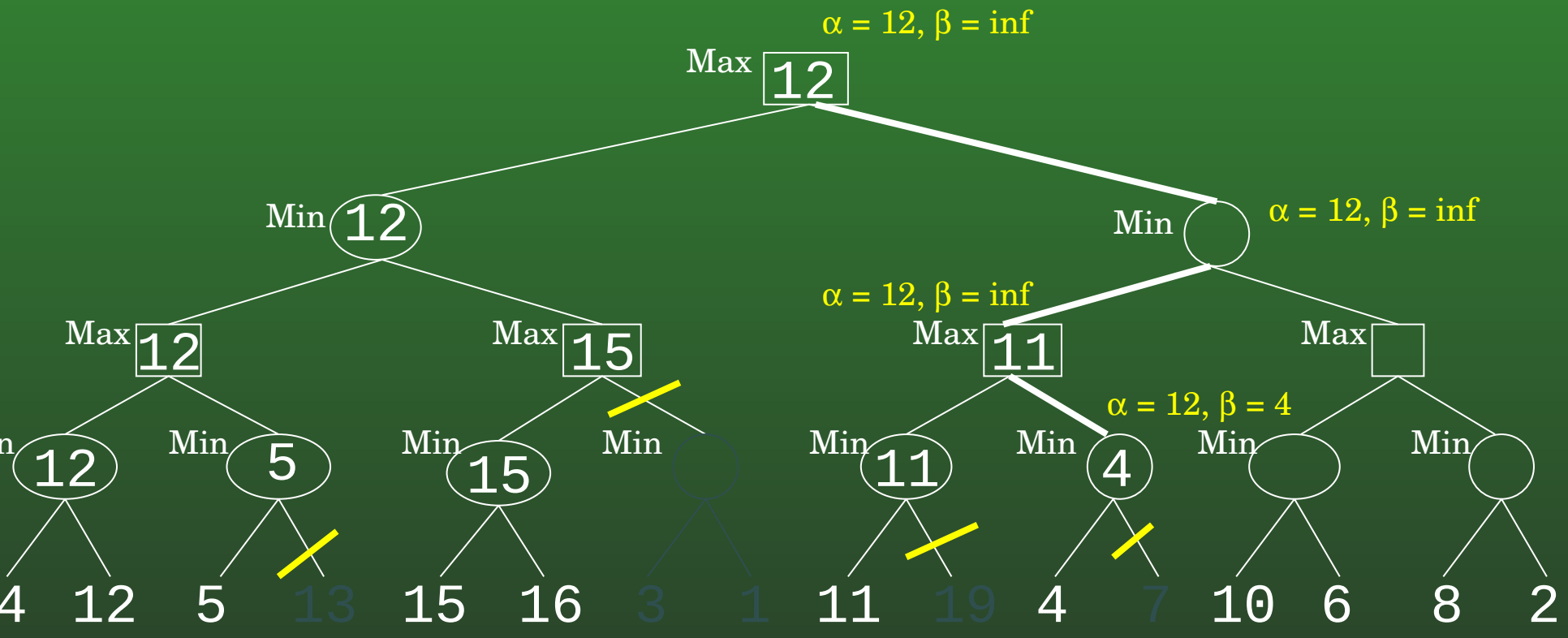
P4-28: Alpha-Beta Pruning



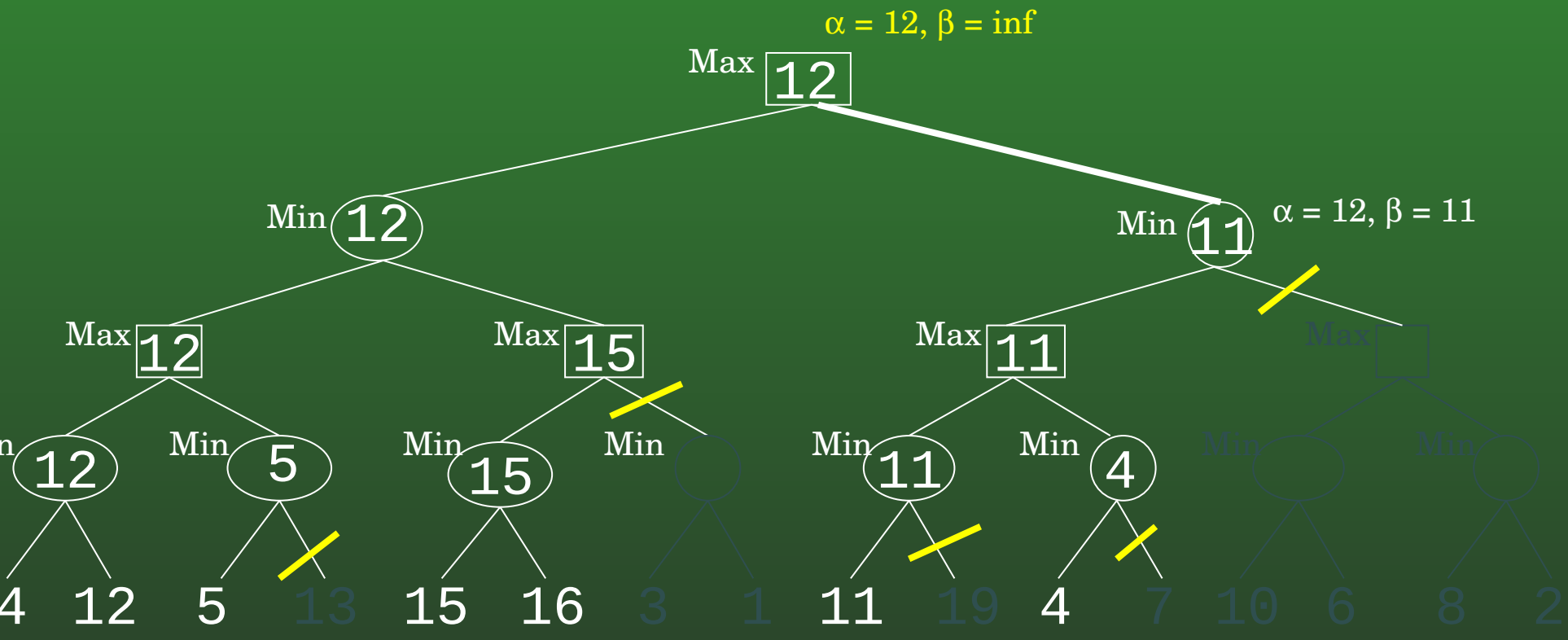
P4-29: Alpha-Beta Pruning



P4-30: Alpha-Beta Pruning



P4-31: Alpha-Beta Pruning



P4-32: Alpha-Beta Pruning

- We can cut large branches from the search tree
  - In the previous example, what would happen with similar values and a deeper tree?
- If we choose the order that we evaluate nodes (more on this in a minute ...), we can dramatically cut down on how much we need to search

P4-33: Evaluation Functions

- We can't search all the way to the bottom of the search tree
  - Trees are just too big
- Search a few levels down, use an evaluation function to see how good the board looks at the moment
- Back up the result of the evaluation function, as if it was the utility function for the end of the game

P4-34: Evaluation Functions

- Chess:
  - Material - value for each piece (pawn = 1, bishop = 3, etc)
    - Sum of my material - sum of your material
  - Positional advantages
    - King protected
    - Pawn structure
- Network:
  - Partial Networks
  - Number and length of partial networks

P4-35: Evaluation Functions

- If we have an evaluation function that tells us how good a move is, why do we need to search at all?
  - Could just use the evaluation function
- If we are only using the evalution function, does search do us any good?

P4-36: Evaluation Functions & α-β

- How can we use the evaluation function to maximize the pruning in alpha-beta pruning?

P4-37: Evaluation Functions & α-β

- How can we use the evaluation function to maximize the pruning in alpha-beta pruning?
  - Order children of max nodes, from highest to lowest
  - Order children of min node, from lowest to highest
  - (Other than for ordering, eval function is not used for interior nodes)
- With perfect ordering, we need to search only $b^{d/2}$ (instead of b^d) to find the optimal move – can search up to twice as far

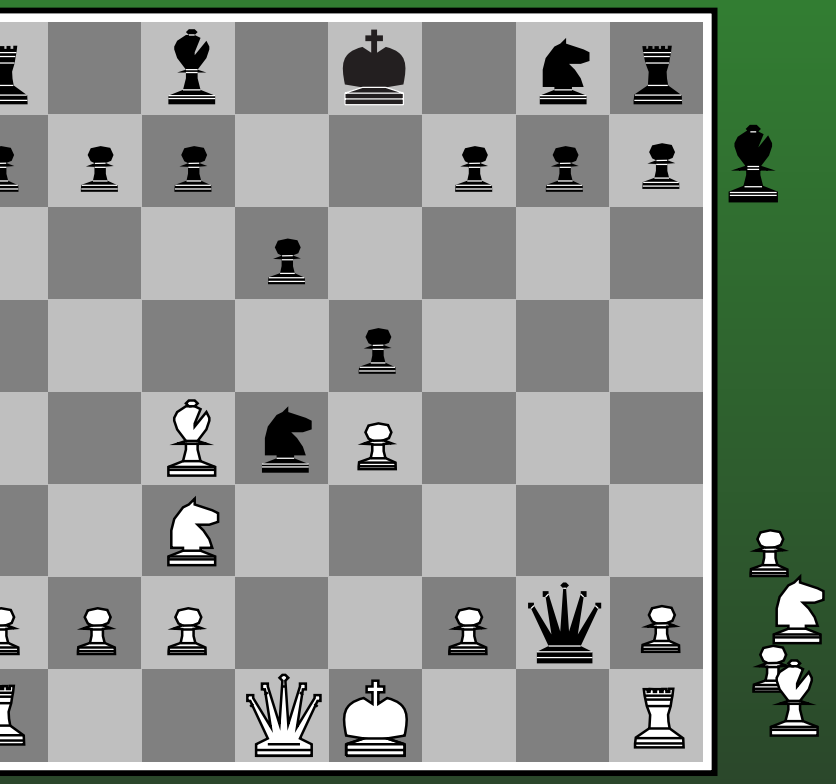
P4-38: Stopping the Search

- We can’t search all the way to the endgame
  - Not enough time
- Search a set number of moves ahead
  - Search Depth

P4-39: Stopping the Search

- We can't search all the way to the endgame
  - Not enough time
- Search a set number of moves ahead
  - What if we are in the middle of a piece trade?
  - In general, what if our opponent is about to capture one of our pieces

P4-40: Stopping the Search



(a) White to move

(b) White to move

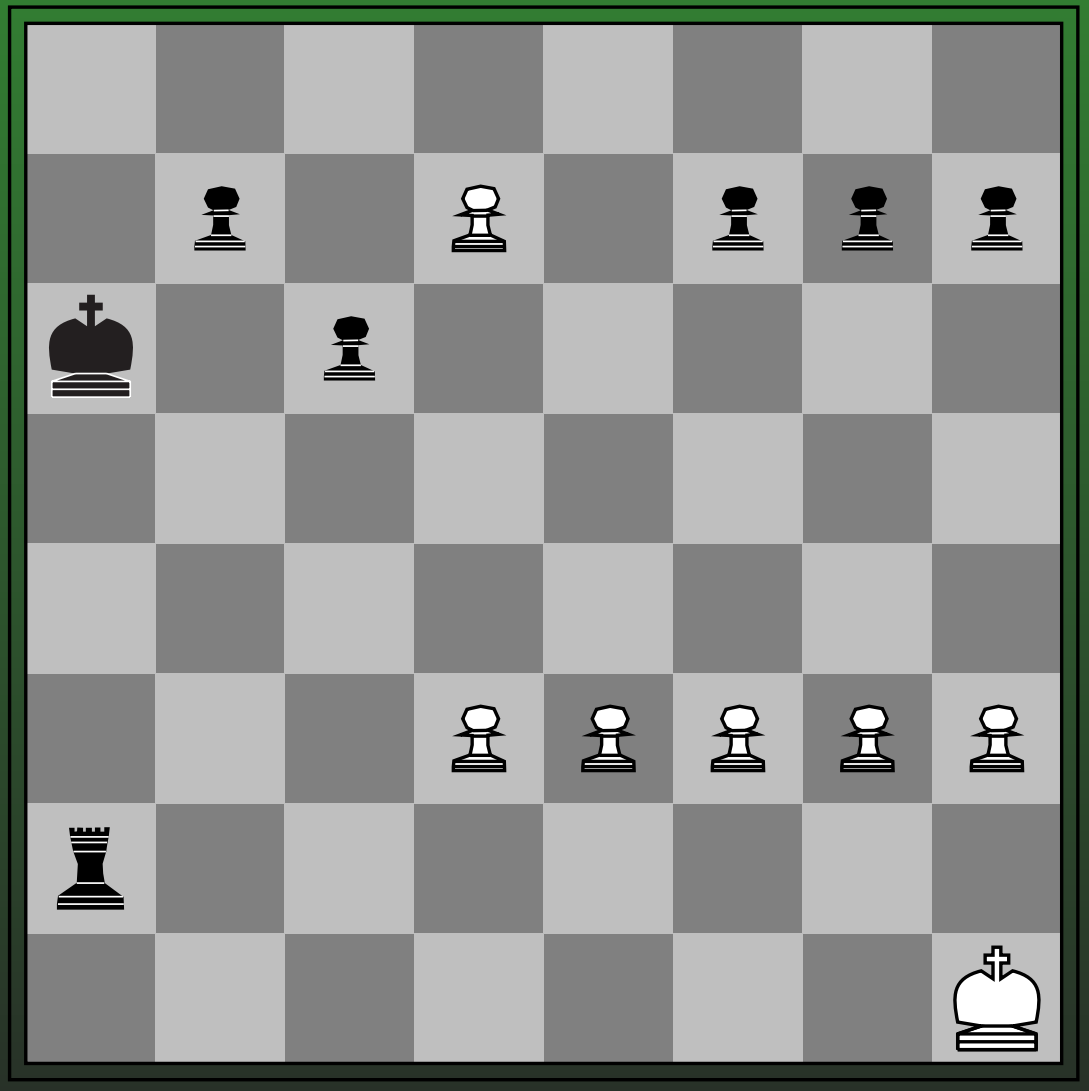
P4-41: Stopping the Search

- Quiescence Search
  - Only apply the evaluation function to nodes that do not swing wildly in value
  - If the next move makes a large change to the evaluation function, look ahead a few more moves
  - Not increasing the search depth for the entire tree, just around where the action is
  - To prevent the search from going too deep, may restrict the kinds of moves (captures only, for instance)

P4-42: Stopping the Search

- Horizon Problem
  - Sometimes, we can push a bad move past the horizon of our search
  - Not preventing the bad move, just delaying it
  - A position will look good, even though it is utlimately bad

P4-43: Horizon Problem



Black to move

P4-44: Horizon Problem

- Singular Extensions
  - When we are going to stop, see if there is one move that is clearly better than all of the others.
  - If so, do a quick “search”, looking only at the best move for each player
  - Stop when there is no “clearly better” move
  - Helps with the horizon problem, for a series of forced moves
- Similar to quiescence search