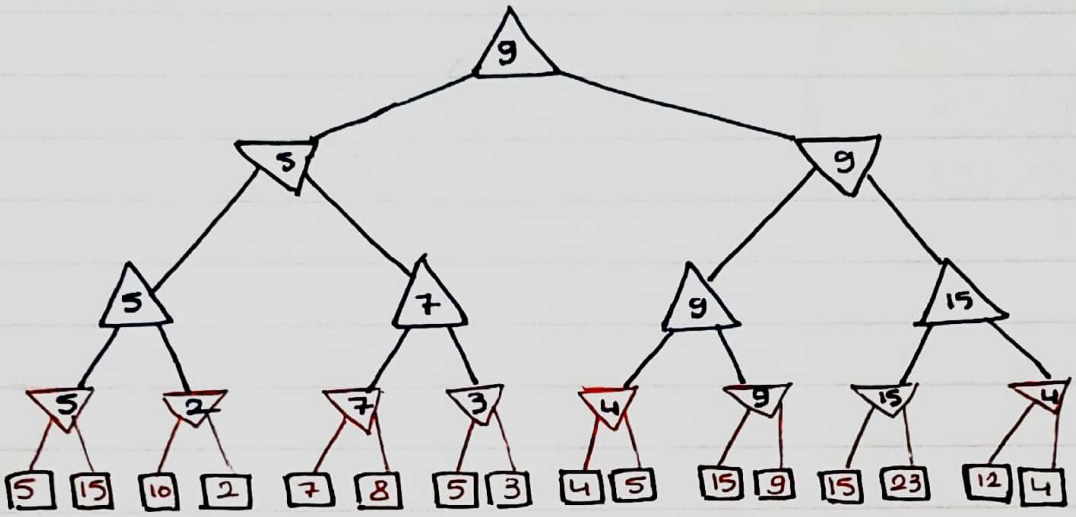
MI Sheet 6

1. A tree with branching factor 2 and depth 4 has the following values for its leaves
   → given 16 values
   ⟶ naturally follows that depth is $\log_b 16 = 4$

   • to draw, it's wise to start with the last level

△→▽→△→▽→□



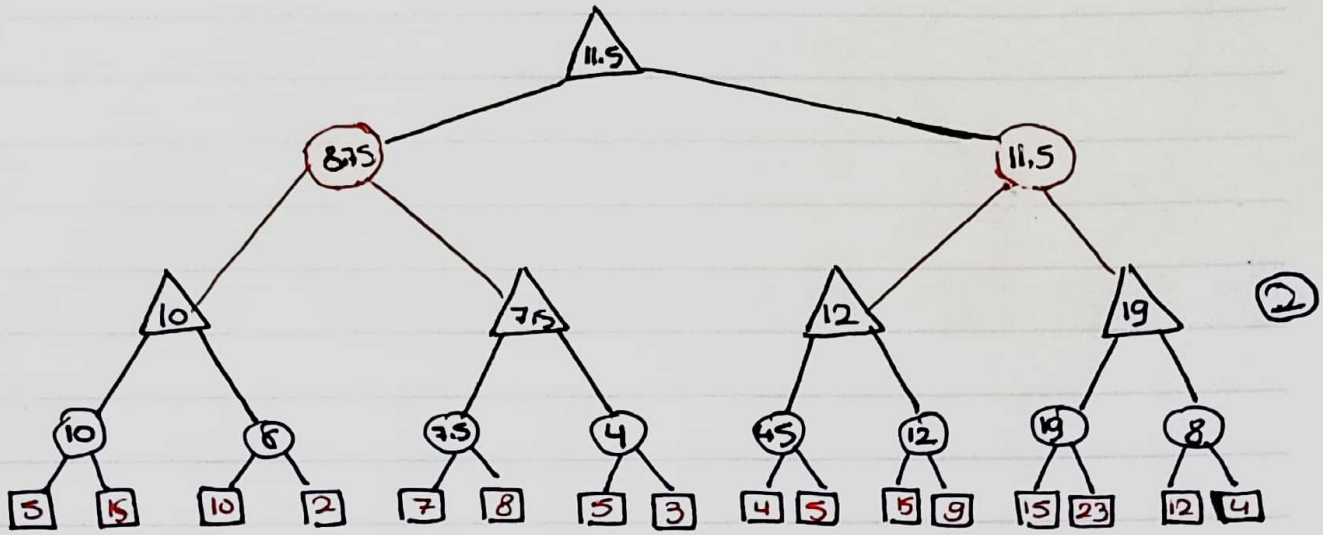
} only given

→ For α-β Pruning check [4, MI Lecture 6 P2]

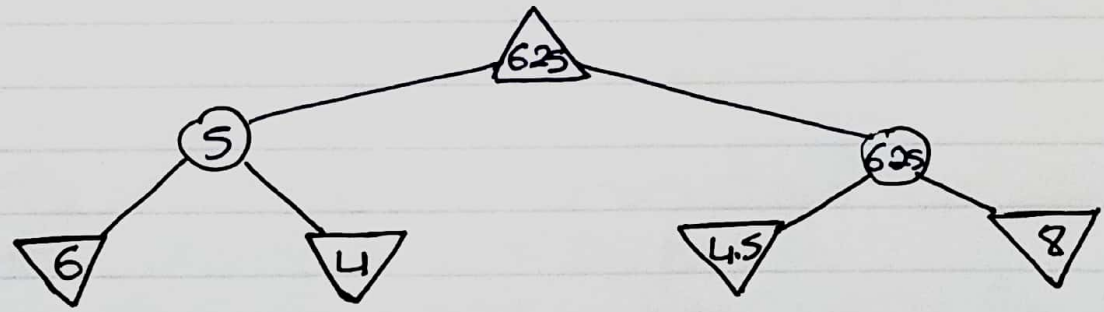
• Note that there are 3 methods
tutorial{ → Book's method (doesn't check $\alpha \geq \beta$ explicitly)
→ Popular online alternative (equivalent to it, checks ① $\alpha \geq \beta$ explicitly)
→ Professor's method (like ① but with less overhead for humans) // written lecture

• Prof. mentioned using hers or the book's method is okay (perhaps, ① is as well fine)

2. Repeat but replace 2nd Player with Pair Coin
   → Just take avg. instead of min (assumes equal Prob)
   → Called Expectimax (not expectiminimax)

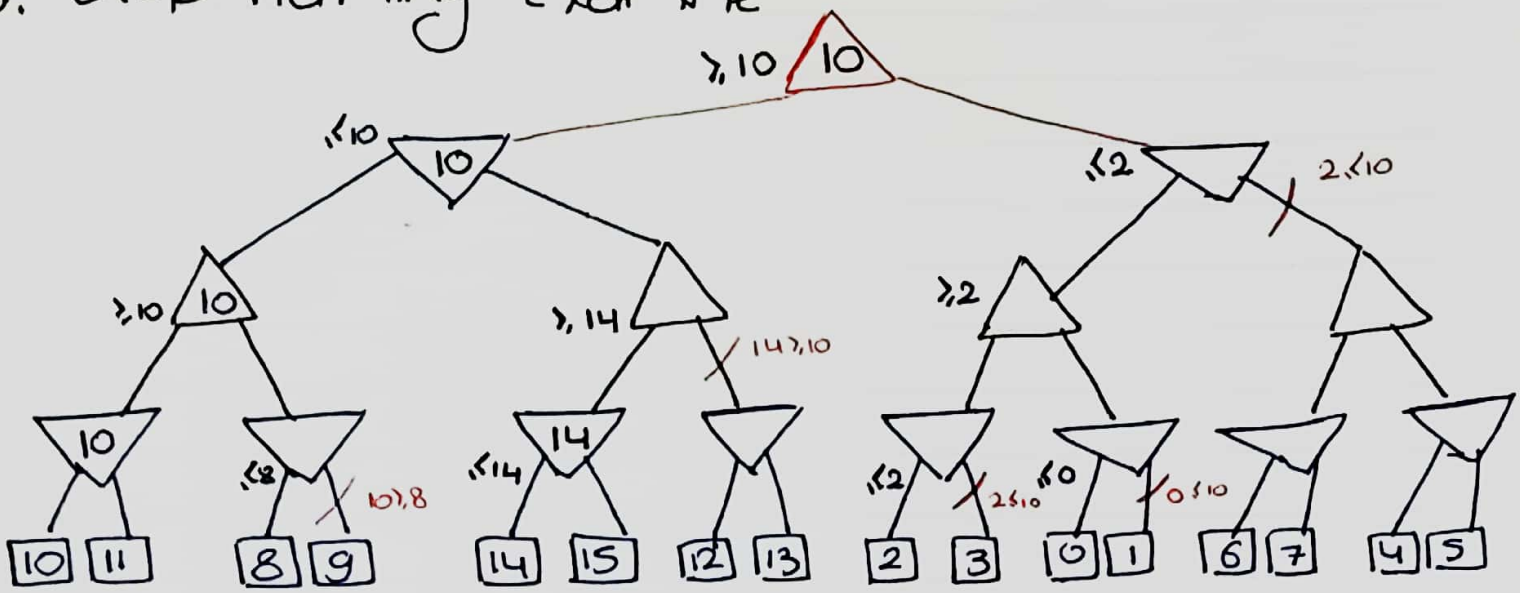


- there's another α-β Pruning example [3, MI Lecture 6 P2]
- try replacing layer ② with min nodes and solve again

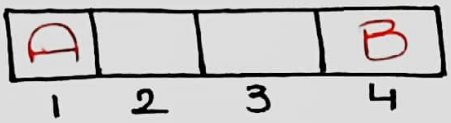


→ Now it's expectiminimax

3. α-β Pruning Example



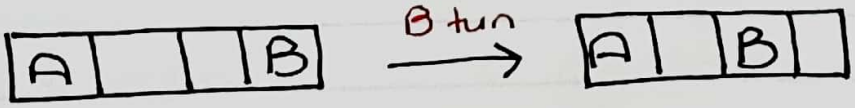
4.



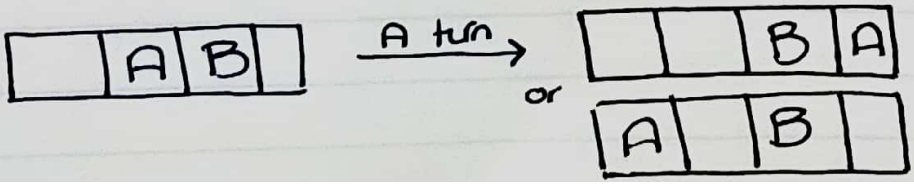
- given is the initial state of a simple game
- Player A moves 1st
- in each turn the Player must move its token to an adjacent slot (left or right)
- in case the other Player is in the next slot then the current Player can jump over them to next slot
- the game ends when one Player reaches the opposite end of the board.
  - → if A reaches 4, final utility for it is +1
  - → if B reaches 1, final utility for A is -1
    - recall everything is relative to max (A)

* Example turns for demonstration

tuple represent



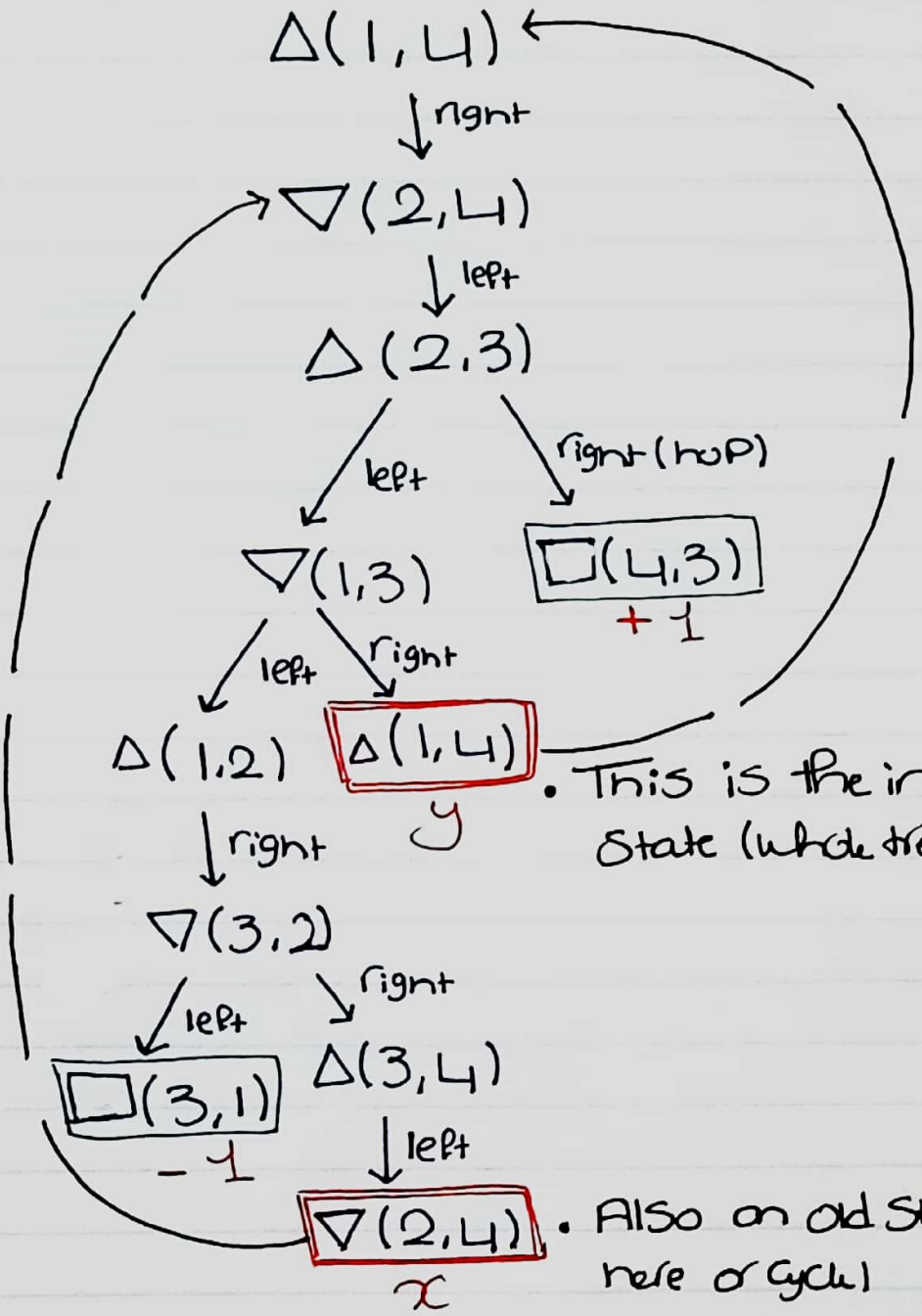
(1,4)→(1,3)



(2,3) → (4,3)
↘ (1,3)

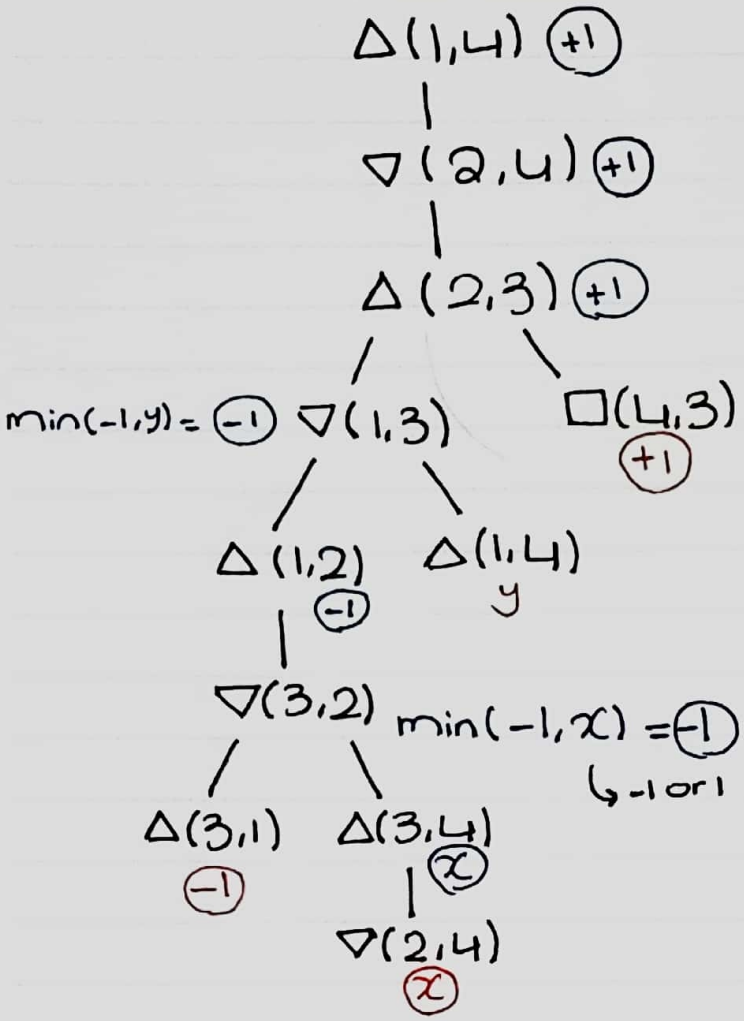
- By representing the State with (S_A, S_B)
  → terminal States are $(4, S_B)$ and $(S_A, 1)$

→ Clearly have UP to 2 actions at any State
↳ left or right



- This is the initial game State (whole tree repeats here or cycle)
- Also an old State (Subtree repeats here or cycle)

- Now how do we apply minimax
  → the final utilities of x, y can clearly either be -1 or 1 each (assuming game won't last forever)
  i.e., for sure $x \in \{-1, 1\}$ and $y \in \{-1, 1\}$



• Have even concluded $x=y=1$

* We were able to deal with x, y by exploiting that if each $\in\{-1,1\}$ then
  $\min(-1,x) = -1$
  $\min(-1,y) = -1$
  (likewise, $\max(1,x) = \max(1,y) = 1$ but we didn't need it)

C. the Standard minimax fails (will recurse infinitely) due to the repeating states (equivalently as game tree).
→ we were able to fix it by cutting off at the repeated states and using logic to compute the min/max values
• This however won't work for all game trees with loops (e.g. suppose $x \in \{-2,-1,1\}$ then we can no longer claim $\min(-1,x) = -1$, likewise $\max(-1,y) = y$ but propagating the y is not helpful for deciding final outcome from root)

* An alternative solution is relying on a heuristic and cut-off. (estimate utility @ loopy nodes)
→ Note: mini-max with cut-off doesn't have a problem with this.