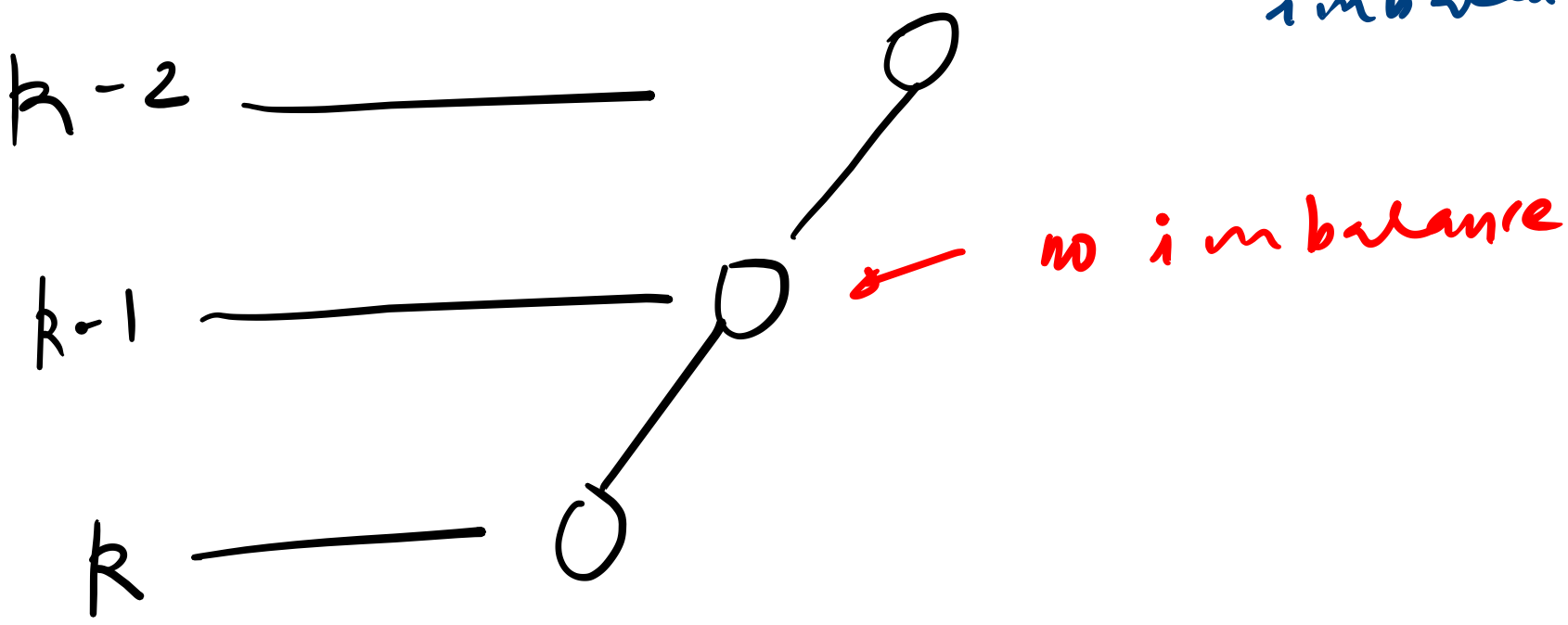
Property: Leaf closest to root is at level k

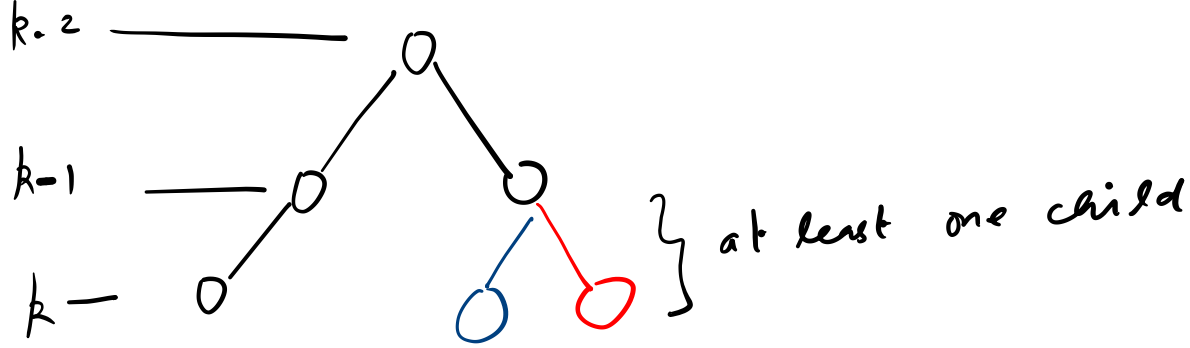
$\Rightarrow$ All nodes at level 1, 2, ..., k-2 have two children



← Should have two children
Otherwise we will have imbalance

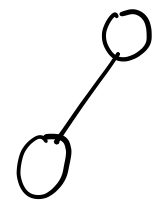
← no imbalance

we are traversing adversarially

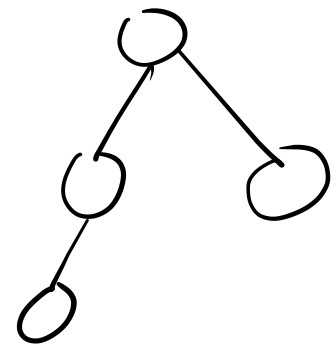


Start arguing from root

root alone
not possible
×



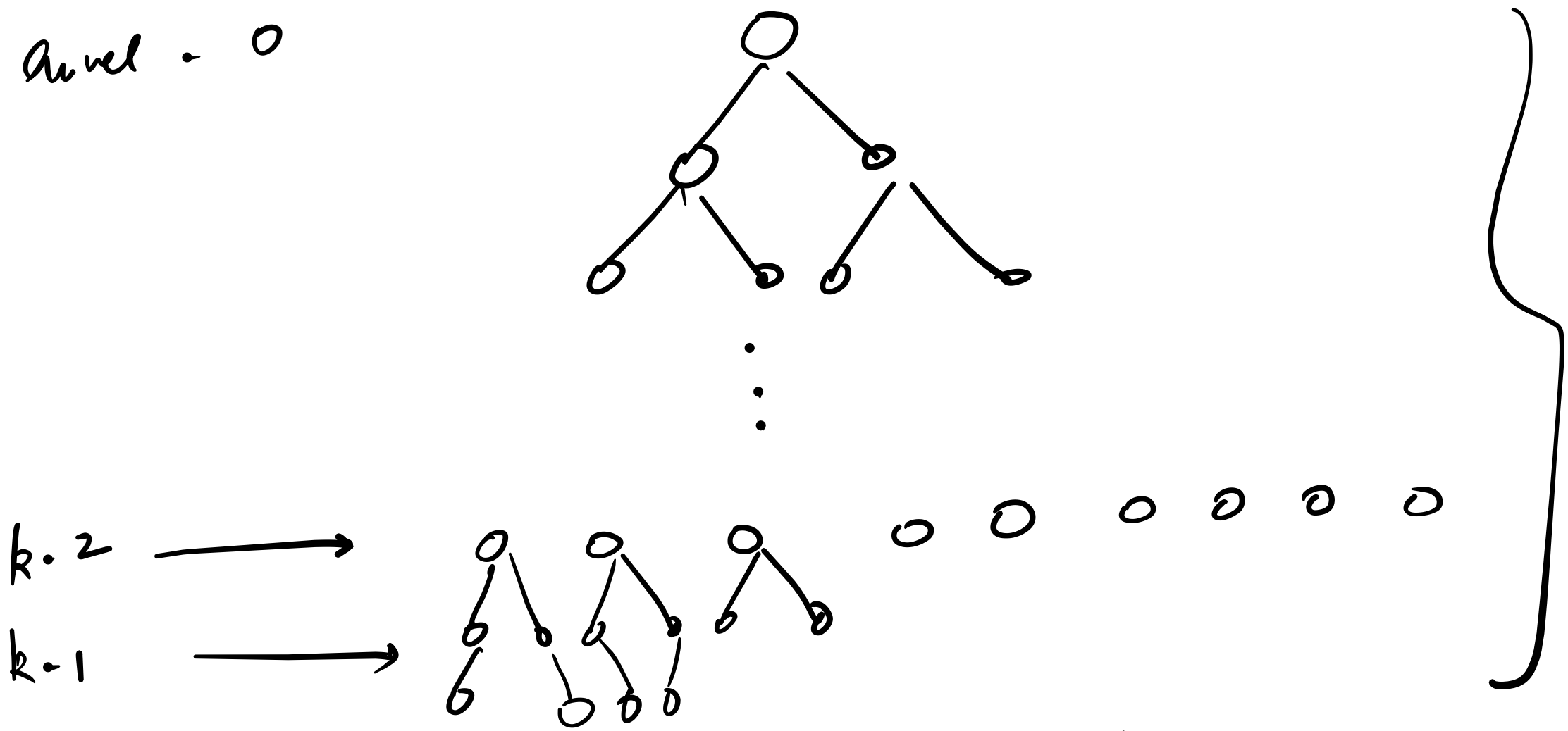
leaf is at level 2
× because at level k leaf is



$\Rightarrow$ AVL is a complete binary tree with

level $k-1$

level $= 0$



$$1 + 2 + \cdots + 2^{k-1} = 2^k - 1$$

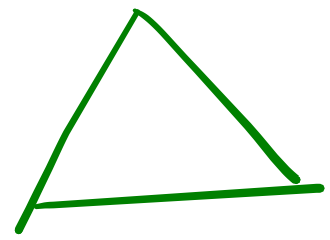
$$2^k - 1 \leq n \leq 2^{2k} - 1$$

Max level $= 2^k - 1$

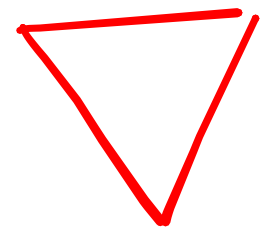
$1 + 2 + \cdots + 2^{2^k - 1} = (2^{2^k} - 1)$

Game Tree

(wants to maximise the score) max-player



(minimise the score) min-player



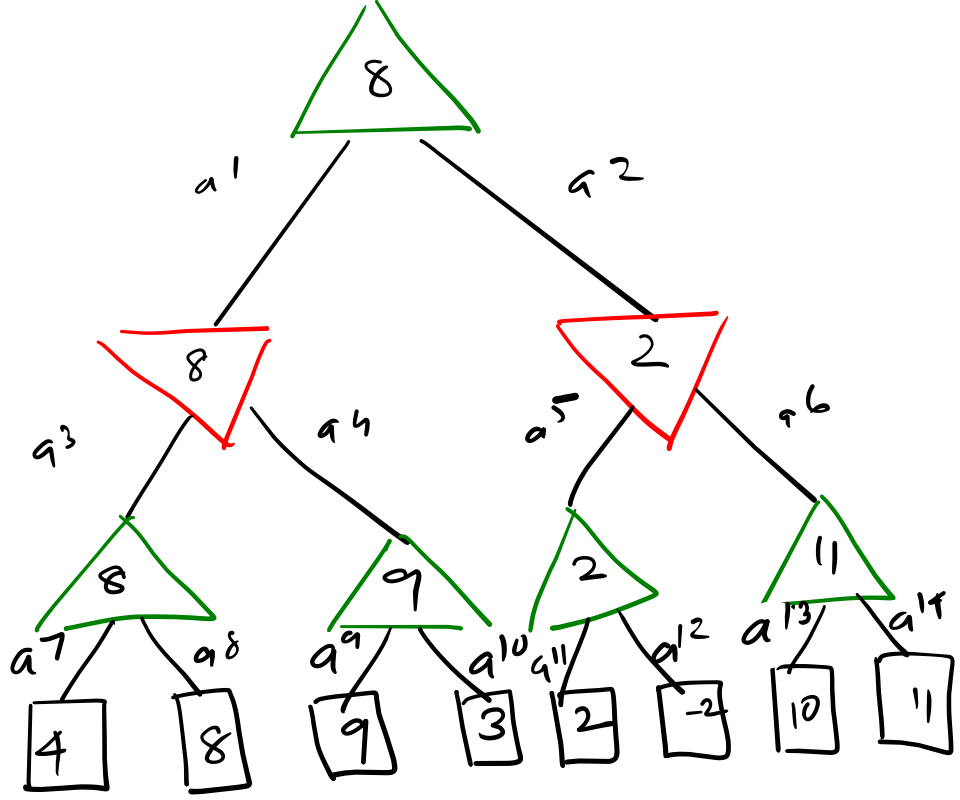
(Russell & Norvig Artificial Intelligence a modern approach

Example: Chess (players take alternate turns)

Each player has an action set A_s

where 's' is the state of the game

board configuration



Evaluate

Assume: Evaluate (s)

Score = White Value - Black Value

pawn	1
knight	3
Bishop	3.5
Rook	5
Queen	8 or 9

white value = 1 × # pawns + 3 × # knights + 3.5 × # bishops + 5 × # rooks + 8 × # queen

say branching factor is b, after depth d

d, total leaves $= \underbrace{b \times \cdots \times b}_{d \text{ times}} = b^d$

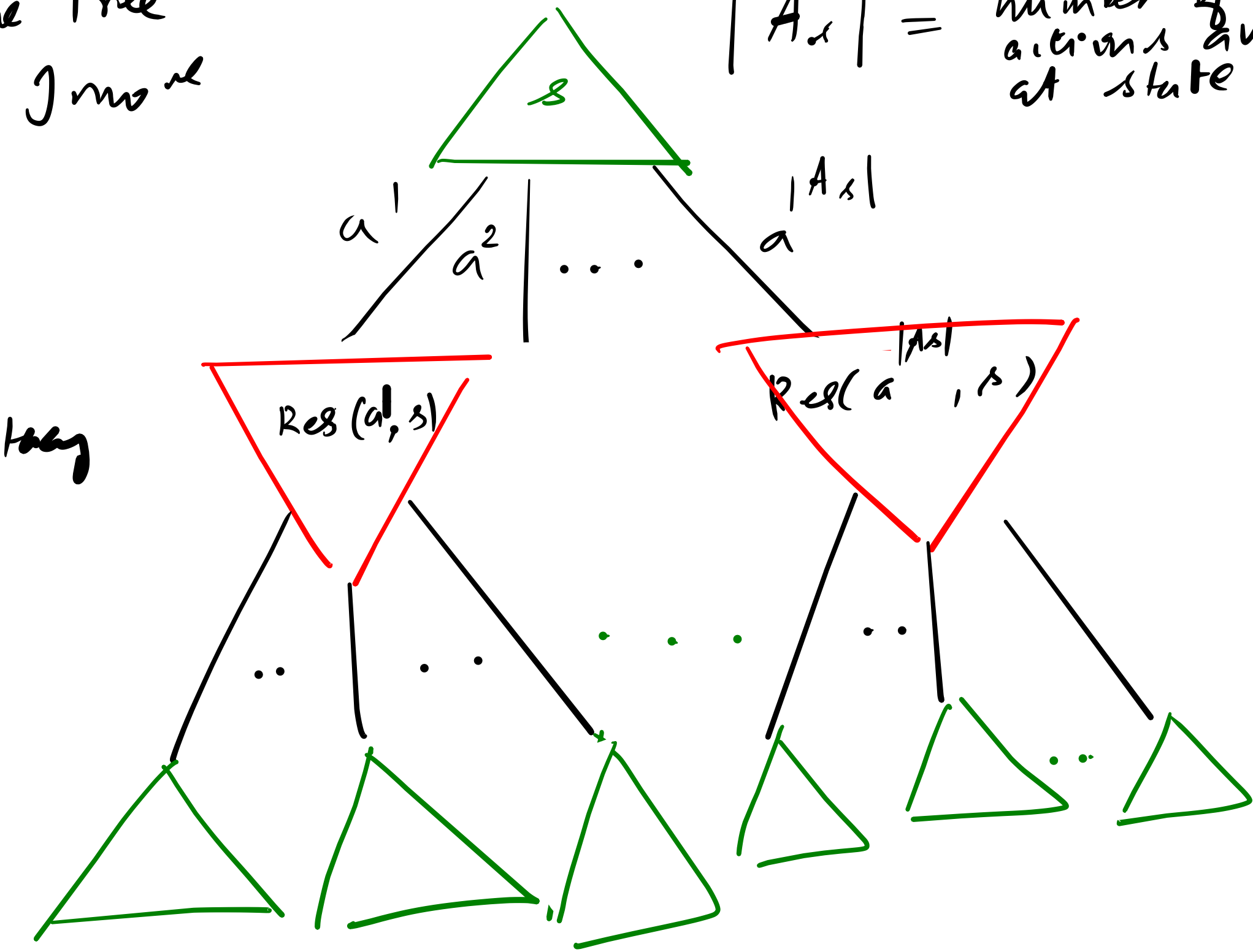
say $b = 10$ moves at each state

$d = 5$ steps ahead $\Rightarrow$ leaves $= 10^5$

Game Tree

If I move

$|A_s|$ = number of actions available at state s



then I may move

then I move

Min-Max Algo

max-player (state)

for all a in A_{state} depth

next.state ← Result (state, a)

value(a) ← min-eval (next.state, 2)

return value = $\max_a$ value(a) / action = $\arg\max_a$ value(a)

Min-Max Algo

max-player (state)

for all a in A_{state}

```
min-eval(state, depth):
    if depth > max-depth
        evaluate(state)
    if state is a terminal state
        evaluate(state)

    val ← -∞
    for al a in A_state
        next.state = Result(state, a)
        val ← min{val, max-val(next-state, depth+1)}

    return val
```

```
max-eval (state, depth):
    if depth > max-depth
        evaluate (state)
    if state is a terminal state
        evaluate (state)

    Val ← + ∞
    for al a in A_state
        next.state = Result(state, a)
        Val ← max { Val, min-val
                    (next-state,
                     depth+1) }

    return val
```