Computer Chess

- *Within 10 years a computer will be world chess champion*
  —Herbert Simon, 1957
- Deep Thought
  - developed by CMU and IBM
  - forerunner of Deep Blue
  - rated ~ 2500
  - won World Computer Chess Championship in 1989

- Chess Ratings
  - beginners: < 1000
  - grandmasters: 2500-2700
  - human world champions: ~ 2800 (Garry Kasparov: 2851)
  - current chess programs: > 3000

Computer Chess

- World champion Garry Kasparov beat Deep Thought decisively in 2 exhibition games in 1989
- Deep Thought 2
  - rated ~ 2600
- Deep Blue
  - developed at IBM Thomas J. Watson research center
  - massively parallel supercomputer with special-purpose chess hardware
  - capable of evaluating 200 million board positions per second
  - capable of looking ahead up to 40 “plies” (half-moves) in some situations
  - beat Garry Kasparov in a 6-game match on May 11, 1997
  - final score: 3.5 to 2.5

Deep Blue

IBM

Computer Chess

- Deep Fritz
  - PC with two Intel Core 2 Duo processors
  - capable of evaluating only 8 million positions per second
  - used more sophisticated heuristics
  - average search depth of 17-18 plies in the middlegame
  - drew an 8-game match against Vladimir Kramnik ("classical" world champion) in 2002
  - beat Kramnik (undisputed world champion) in 2006, 4 to 2
- Deep Junior
  - developed by Israeli computer scientists
  - drew a 6-game "rematch" against Garry Kasparov in 2003
  - beat Deep Fritz in 2007, 4 to 2

Computer Chess

Computer Chess

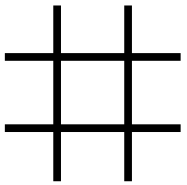
Computer Chess

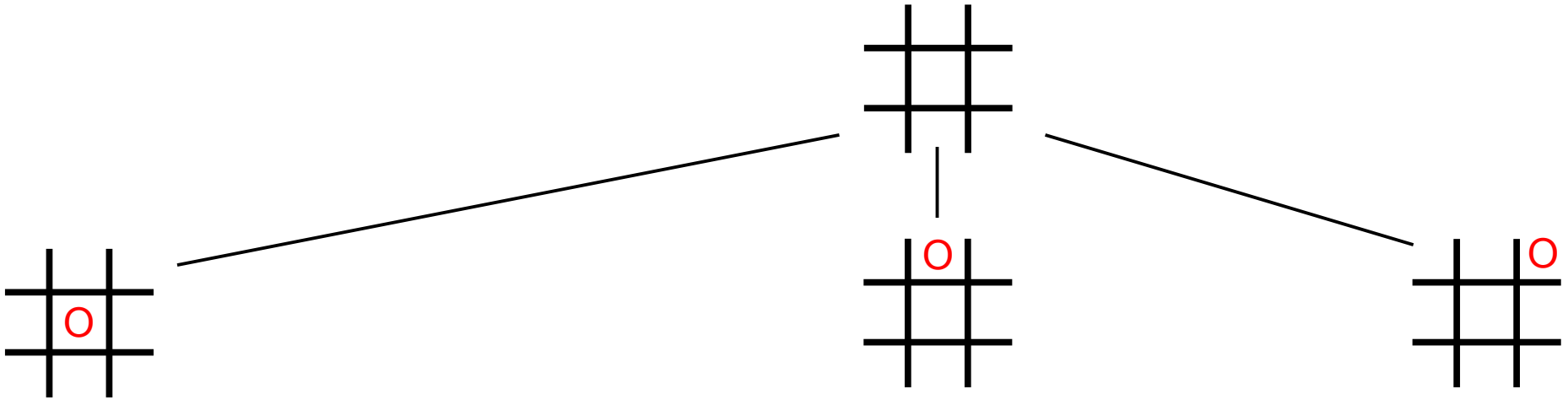
Computer Chess

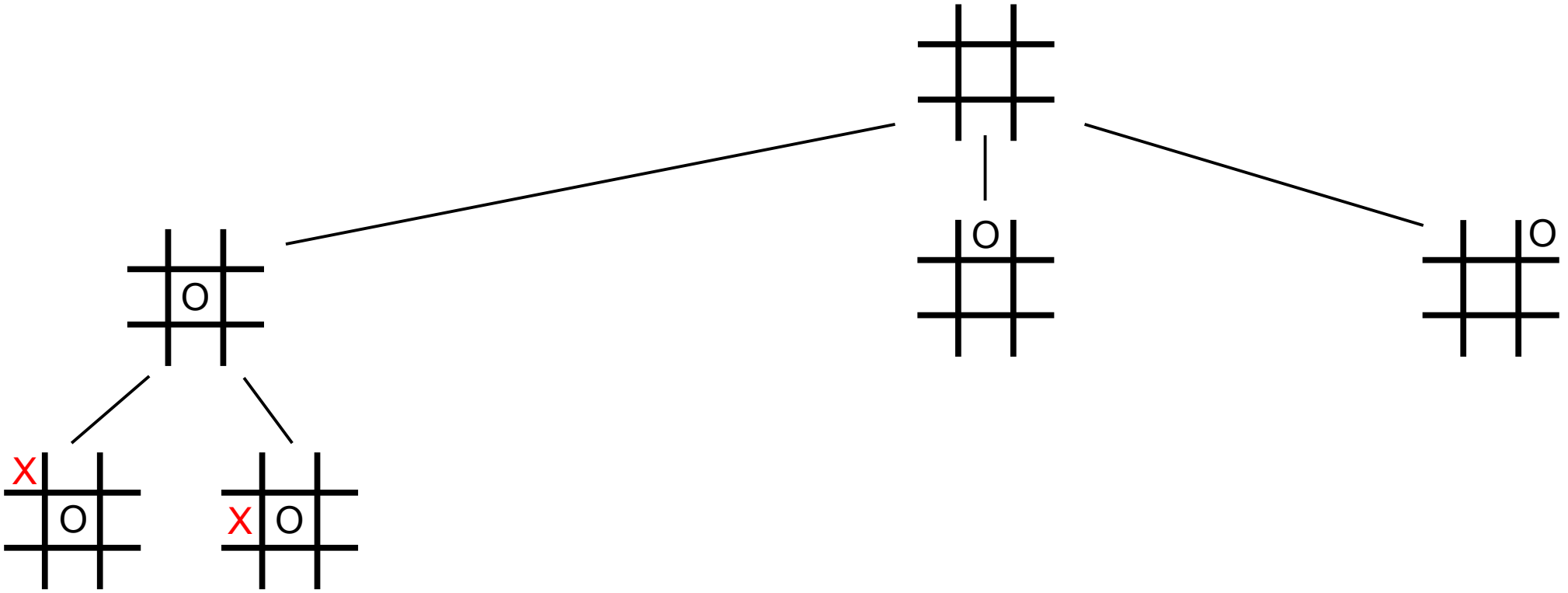
- Game tree “branching factor” is about 35
  - quickly leads to a combinatorial explosion
  - 3 levels down, already > 42,000 branches
- Claude Shannon estimated chess game tree has ~ 10^{120} nodes
  - if a computer could examine 100 trillion nodes per second, it would still take around *a googol years* (10^{100}) to search!
  - age of the universe is much less than 10^{11} years
- Need to select branches to examine in an intelligent way
  - Human chess masters ignore almost all branches and selectively focus only on potentially very good ones, using pattern-matching
  - Chess supercomputers look ahead many levels using brute-force computation

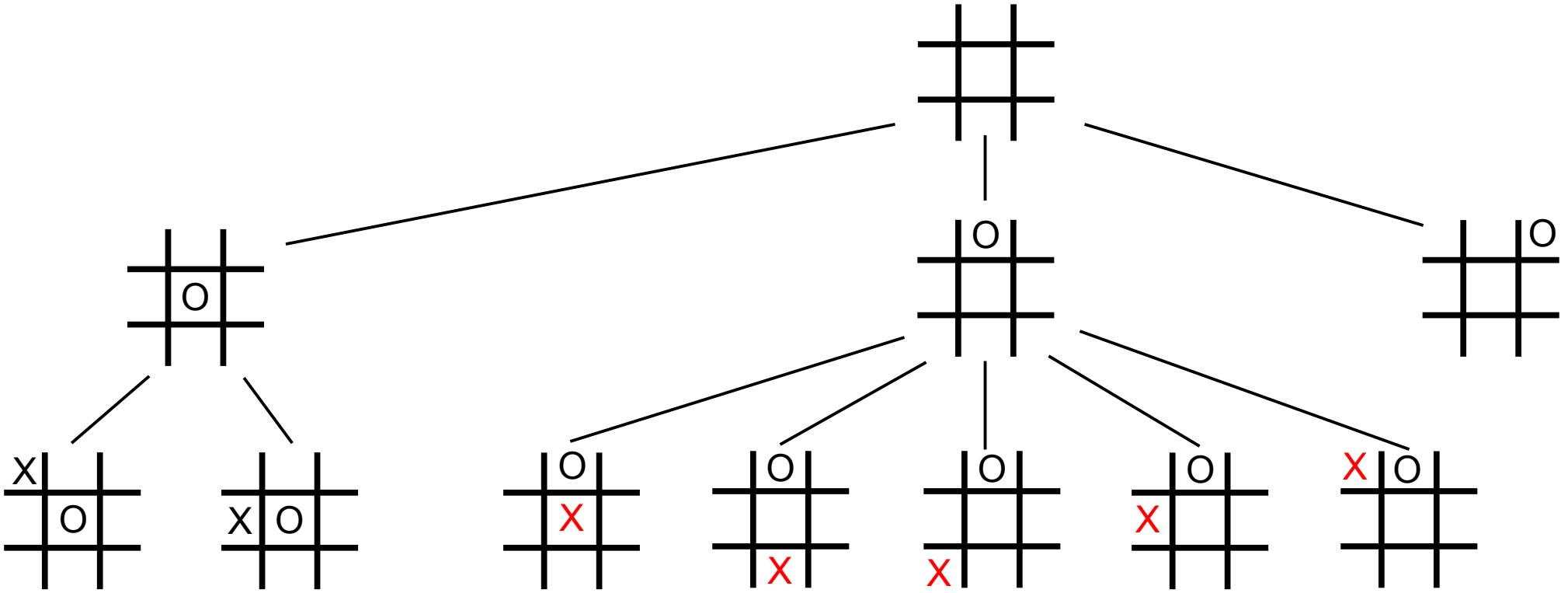
Computer Chess

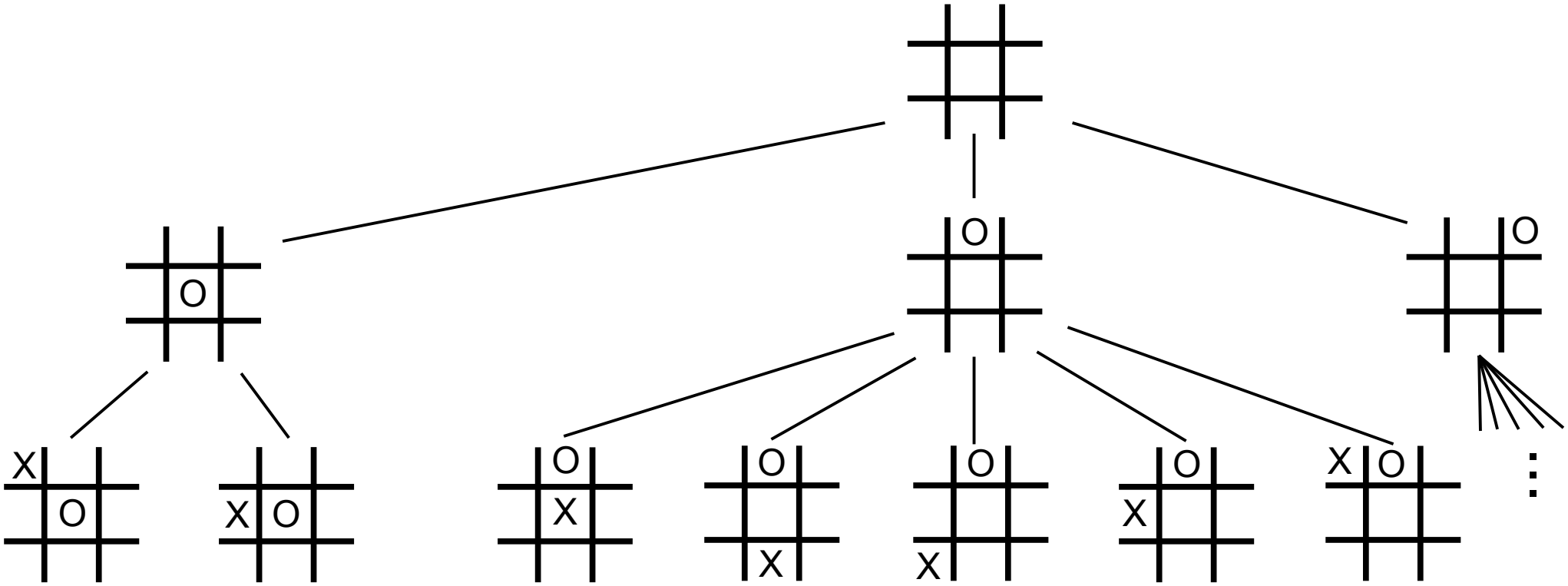
- Heuristic search techniques cannot be used directly
  - presence of opponent complicates the search
  - game tree is too large to “see” to the bottom
- Static evaluation function
  - numerically evaluates the strength of a board position from the viewpoint of a particular player
  - actual values are not as important as relative board ratings
  - example:
    - number of queens, rooks, knights, bishops, pawns
    - $9q + 5r + 3k + 3b + p$
- **Minimax algorithm** determines the best move at any point in the game, assuming that both players play *rationally*

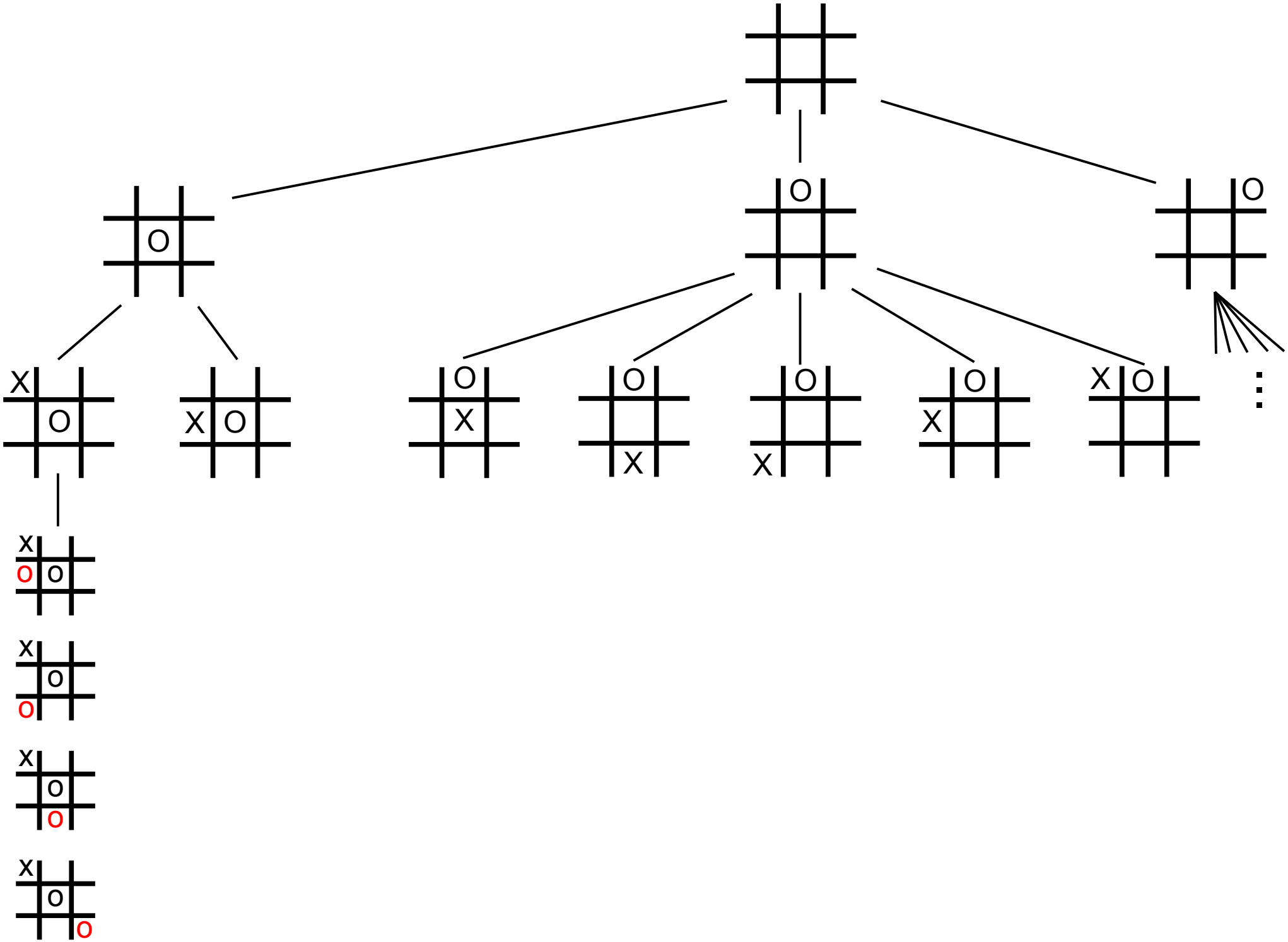


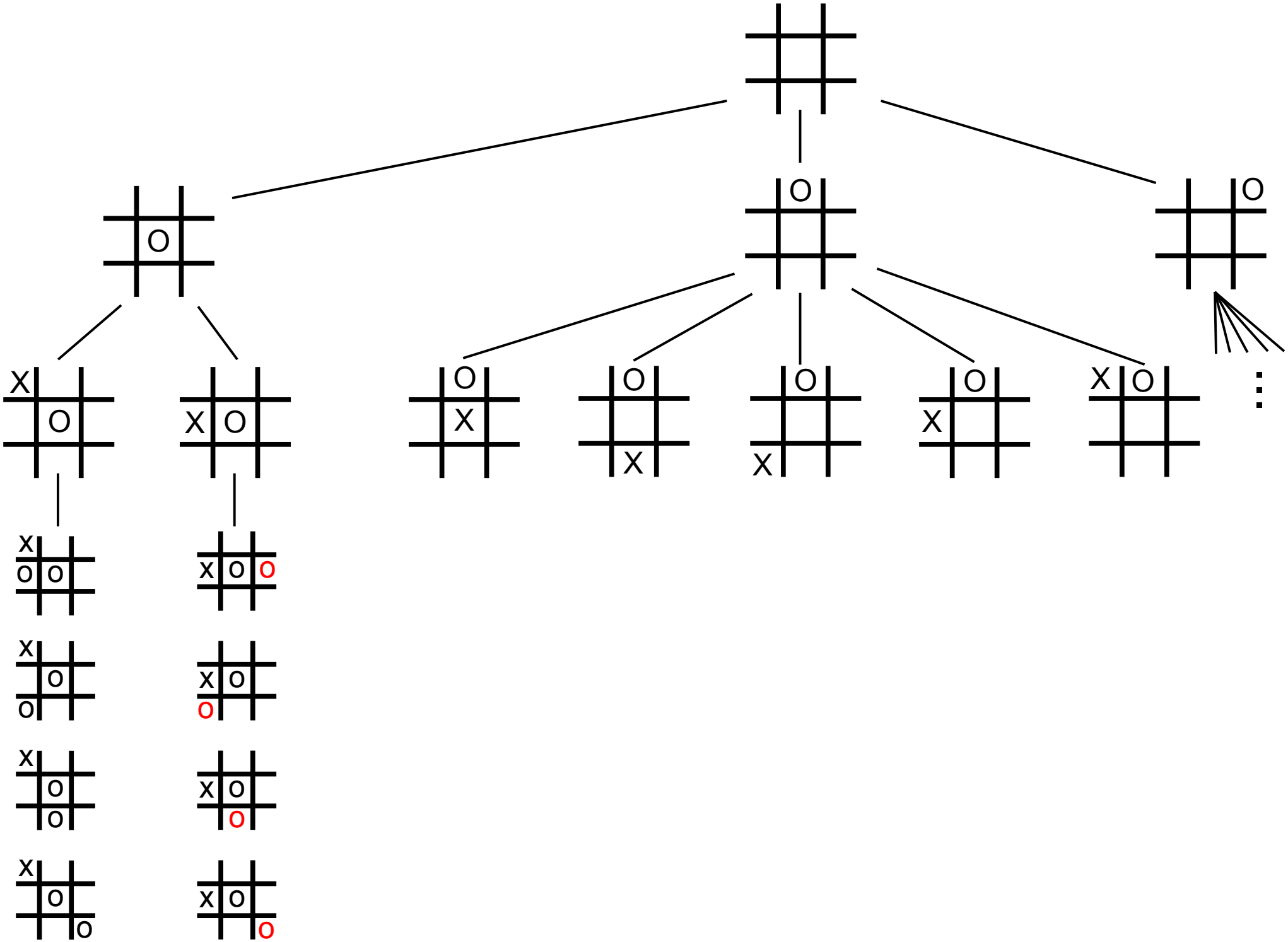

O
O
O

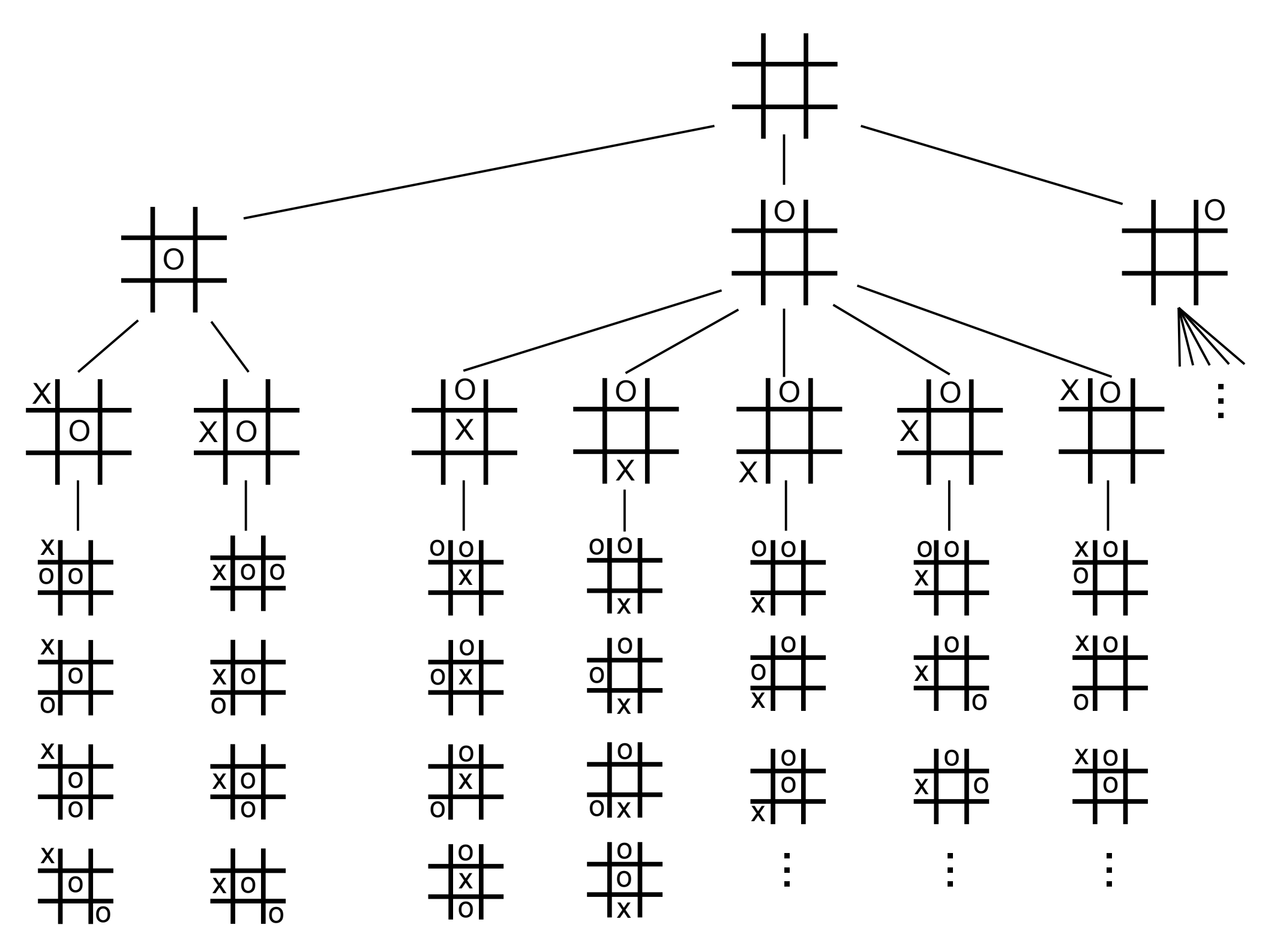

O
O
O
X
O
X
O

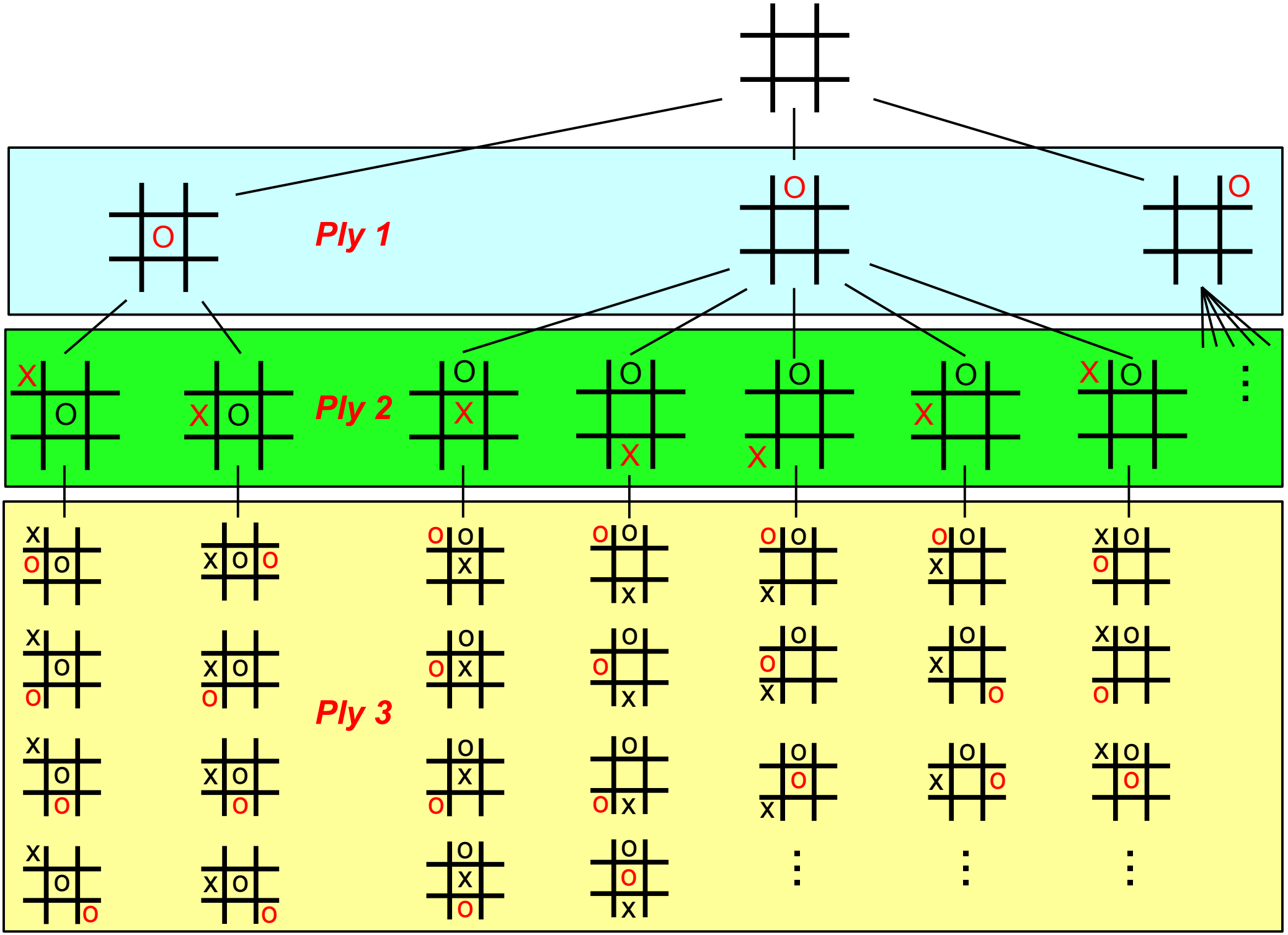

O
O
O
X
O
X
O
O
X
O
O
X
O
X
O
X
O
X
O


O
O
O
X
O
X
O
O
X
O
O
X
O
X
O
X
O
X
O


O
O
O
X
O
X
O
O
X
O
O
O
O
X
O
X
X
O
X
O
O
X
O
O
X
O
O
X
O
O



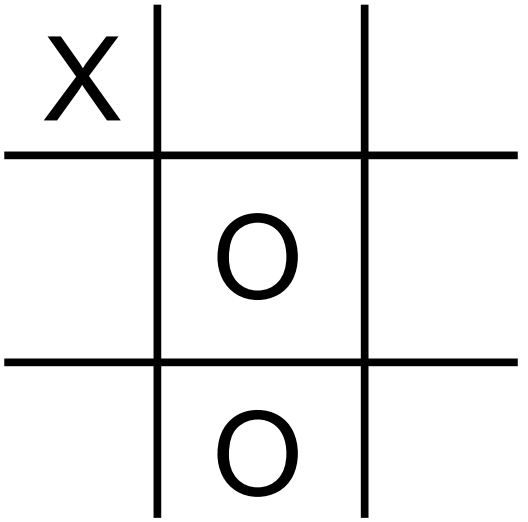



Ply 1
Ply 2
Ply 3

Static Evaluation Function

a = Number of ways to win by filling in 1 space
b = Number of ways to win by filling in 2 spaces

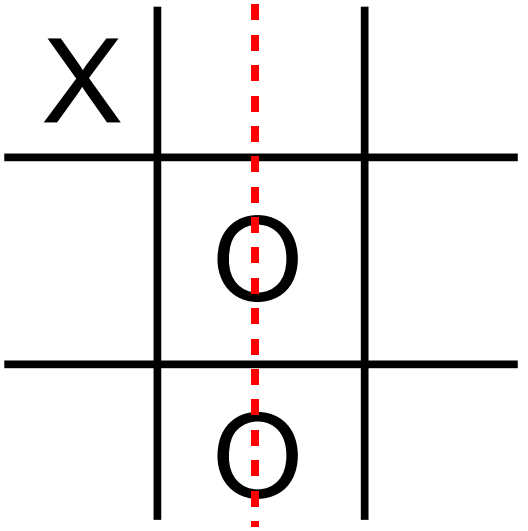
Evaluation function = 2*a* + *b*



Static Evaluation Function

a = Number of ways to win by filling in 1 space
b = Number of ways to win by filling in 2 spaces

Evaluation function = 2*a* + *b*

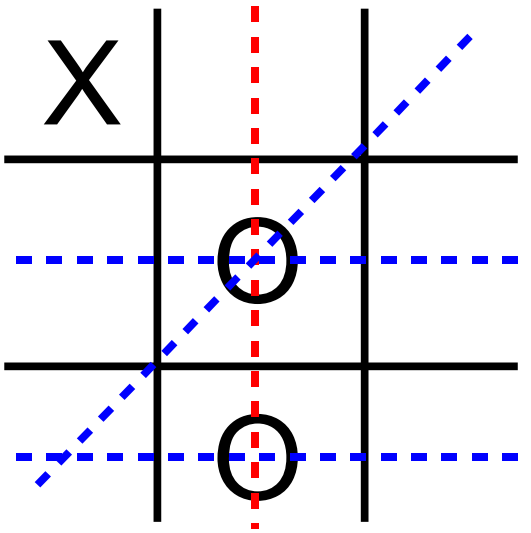


a = 1

Static Evaluation Function

a = Number of ways to win by filling in 1 space
b = Number of ways to win by filling in 2 spaces

Evaluation function = 2*a* + *b*



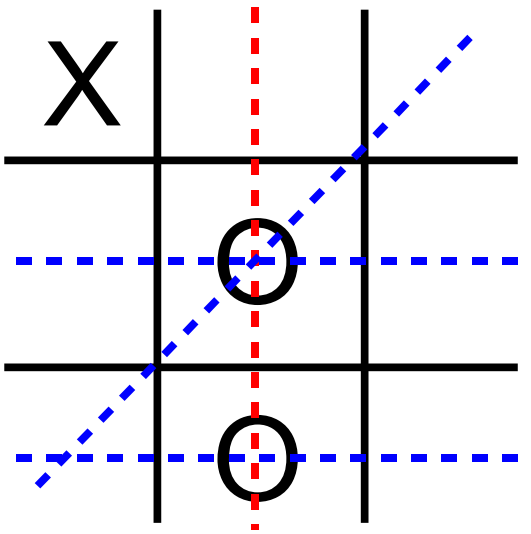
a = 1

b = 3

Static Evaluation Function

a = Number of ways to win by filling in 1 space
b = Number of ways to win by filling in 2 spaces

Evaluation function = 2*a* + *b*



a = 1

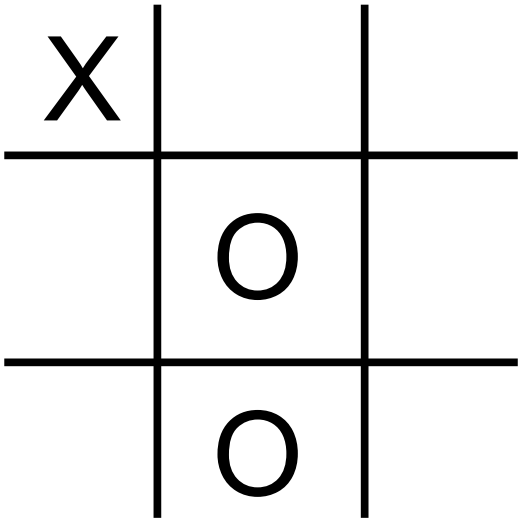
b = 3

2 × 1 + 3 = **5**

Static Evaluation Function

a = Number of ways to win by filling in 1 space
b = Number of ways to win by filling in 2 spaces

Evaluation function = 2*a* + *b*



From **O**'s perspective = 5

From **X**'s perspective = ?

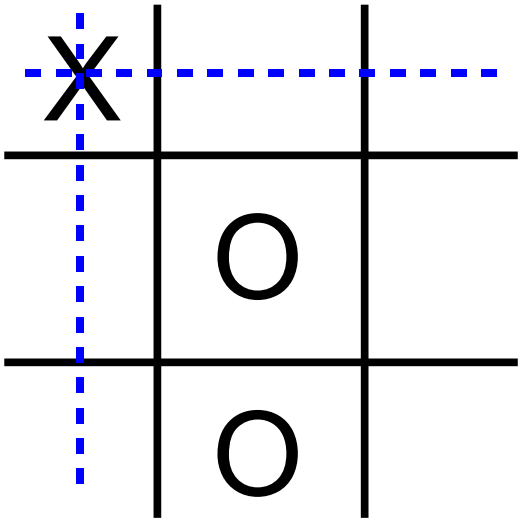
Static Evaluation Function

a = Number of ways to win by filling in 1 space
b = Number of ways to win by filling in 2 spaces

Evaluation function = 2*a* + *b*

From **O**'s perspective = 5

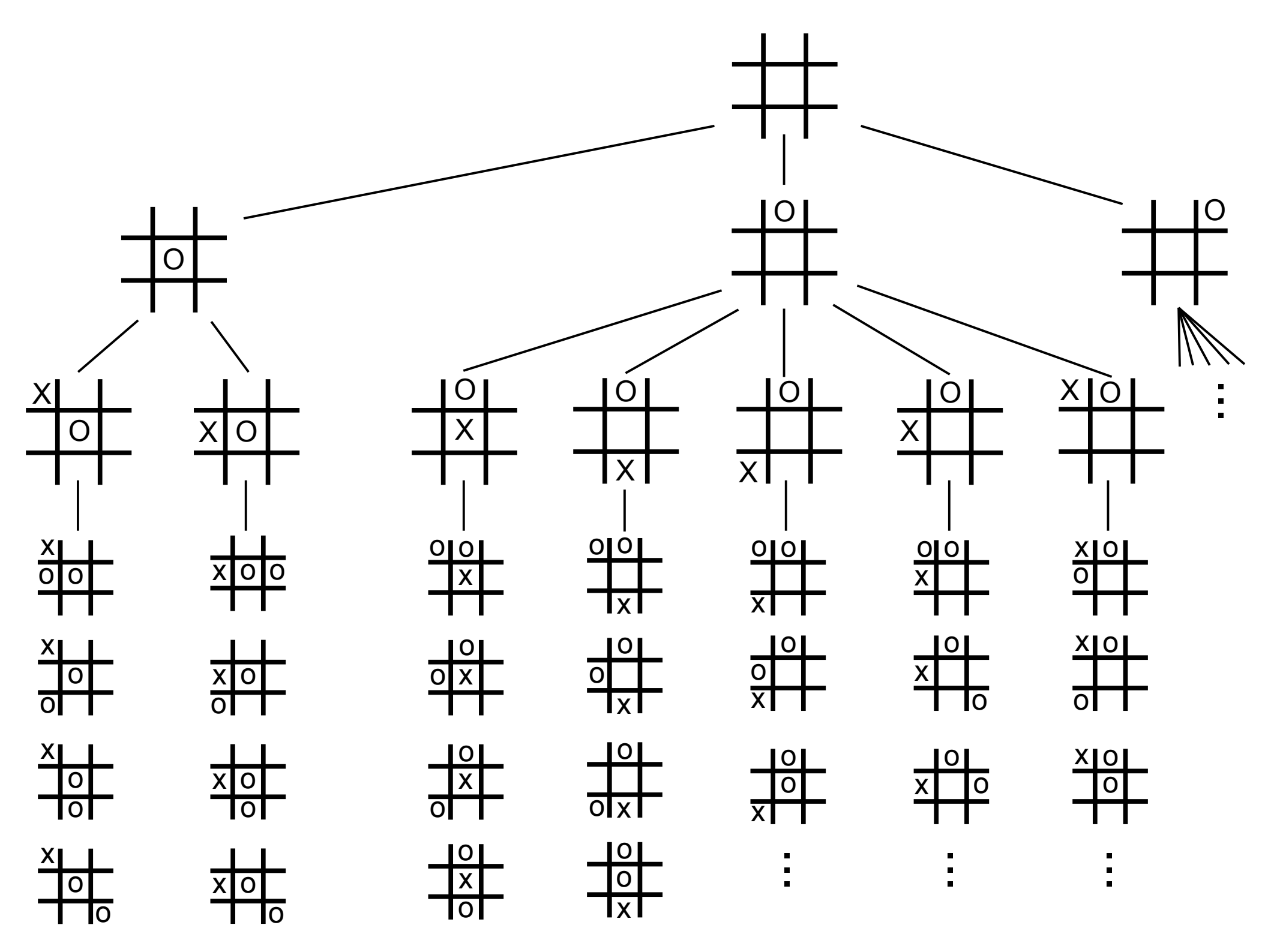
From **X**'s perspective = 2

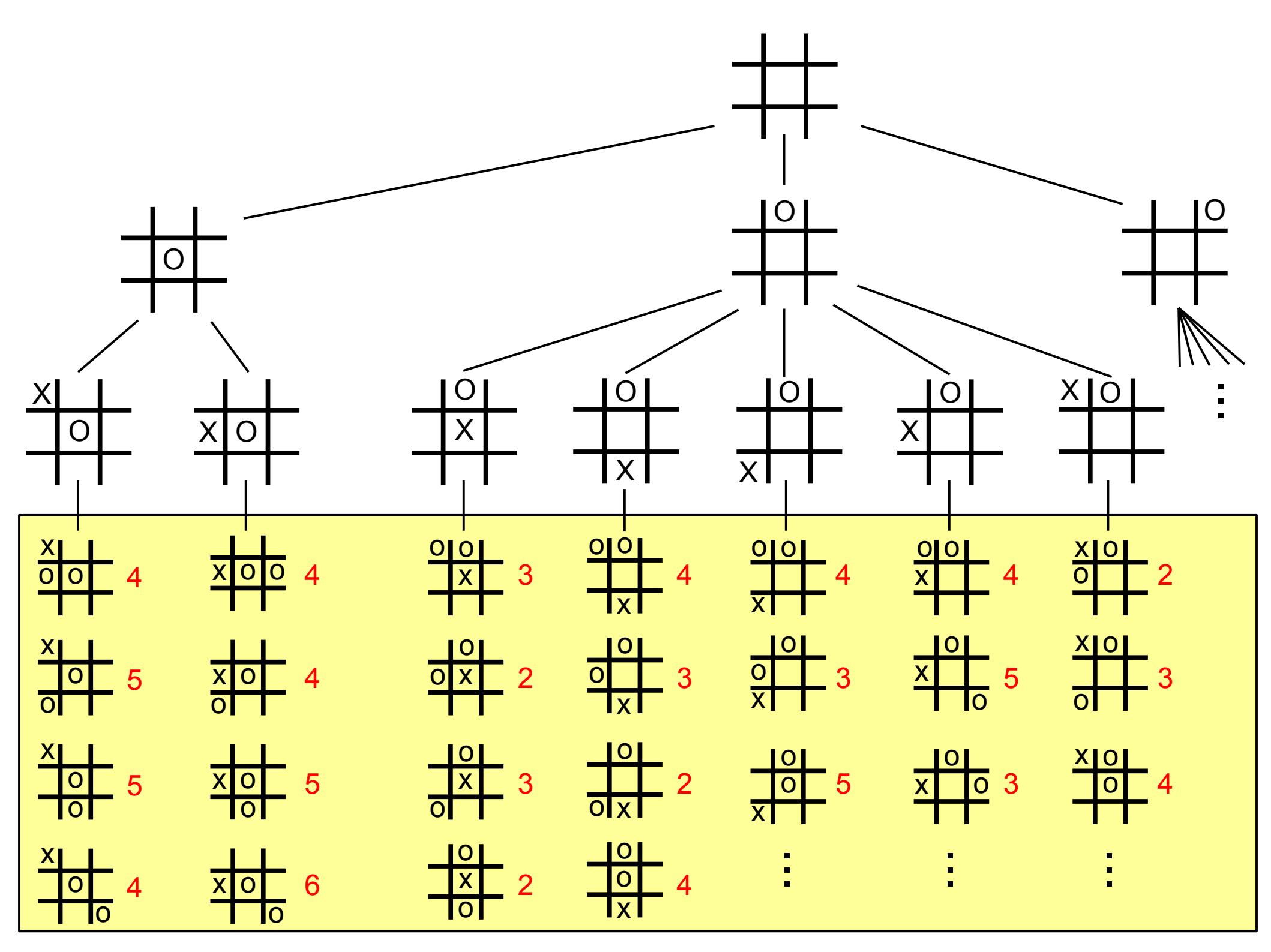


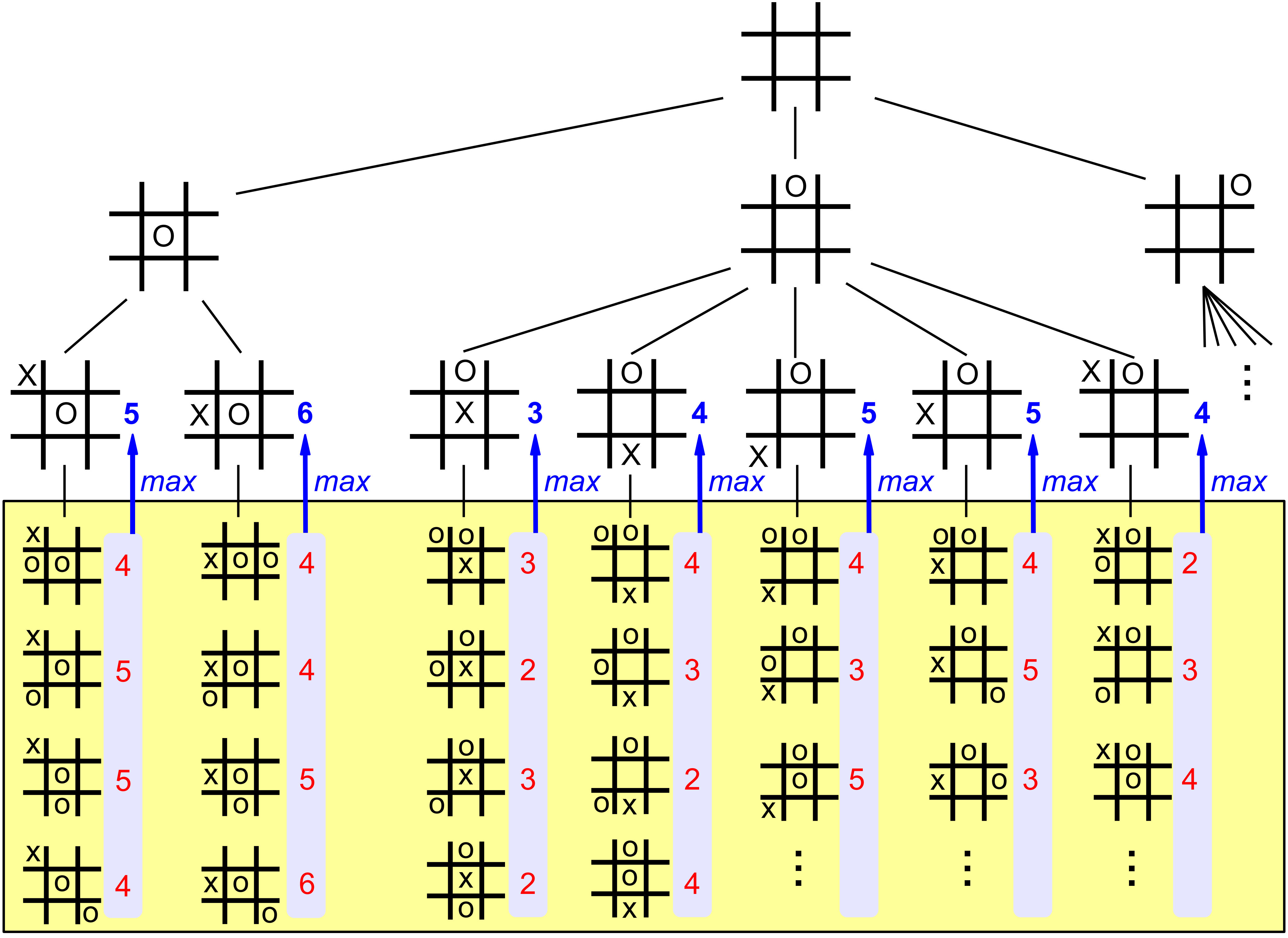
a = 0

b = 2

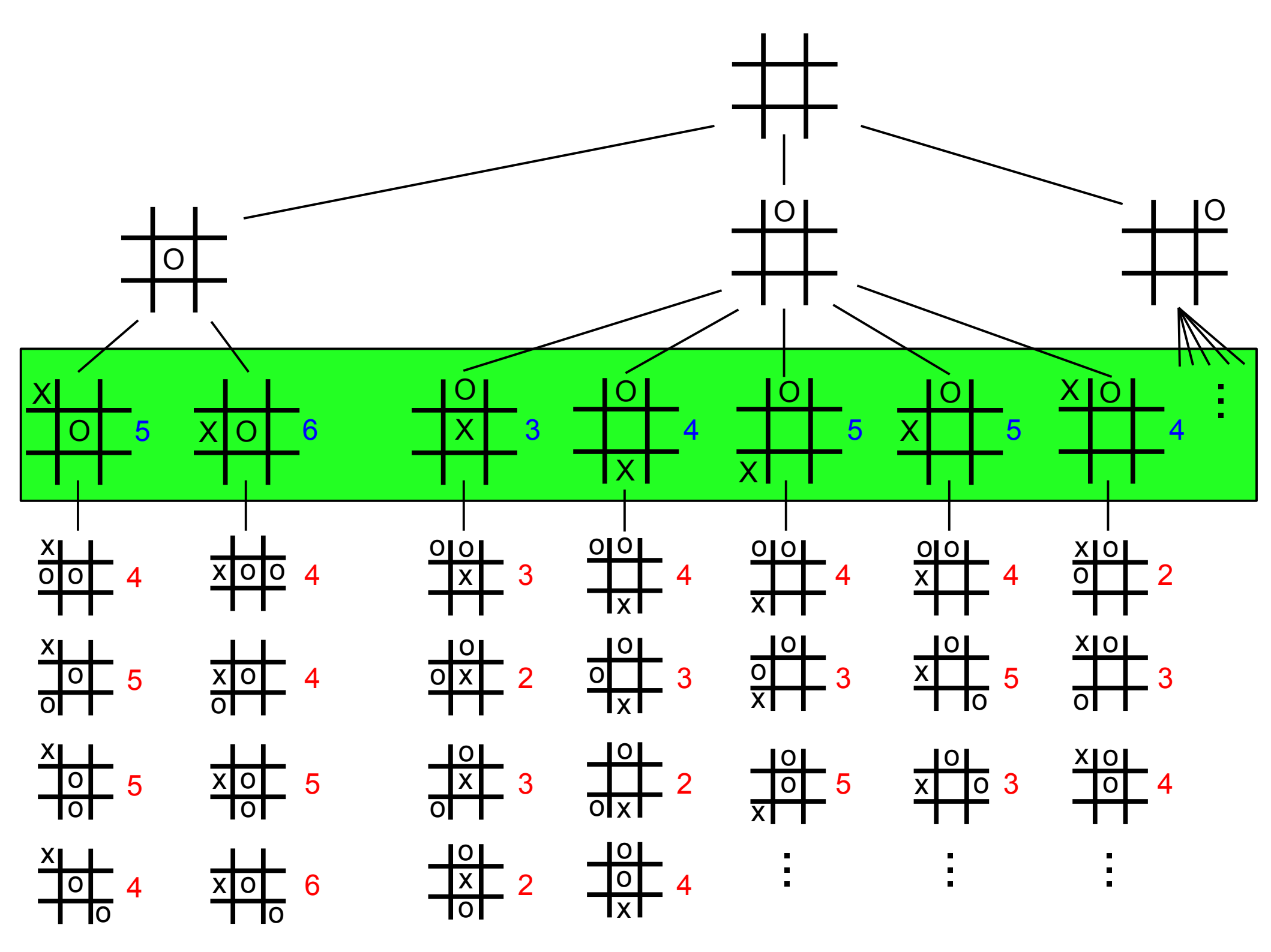
2 × 0 + 2 = **2**

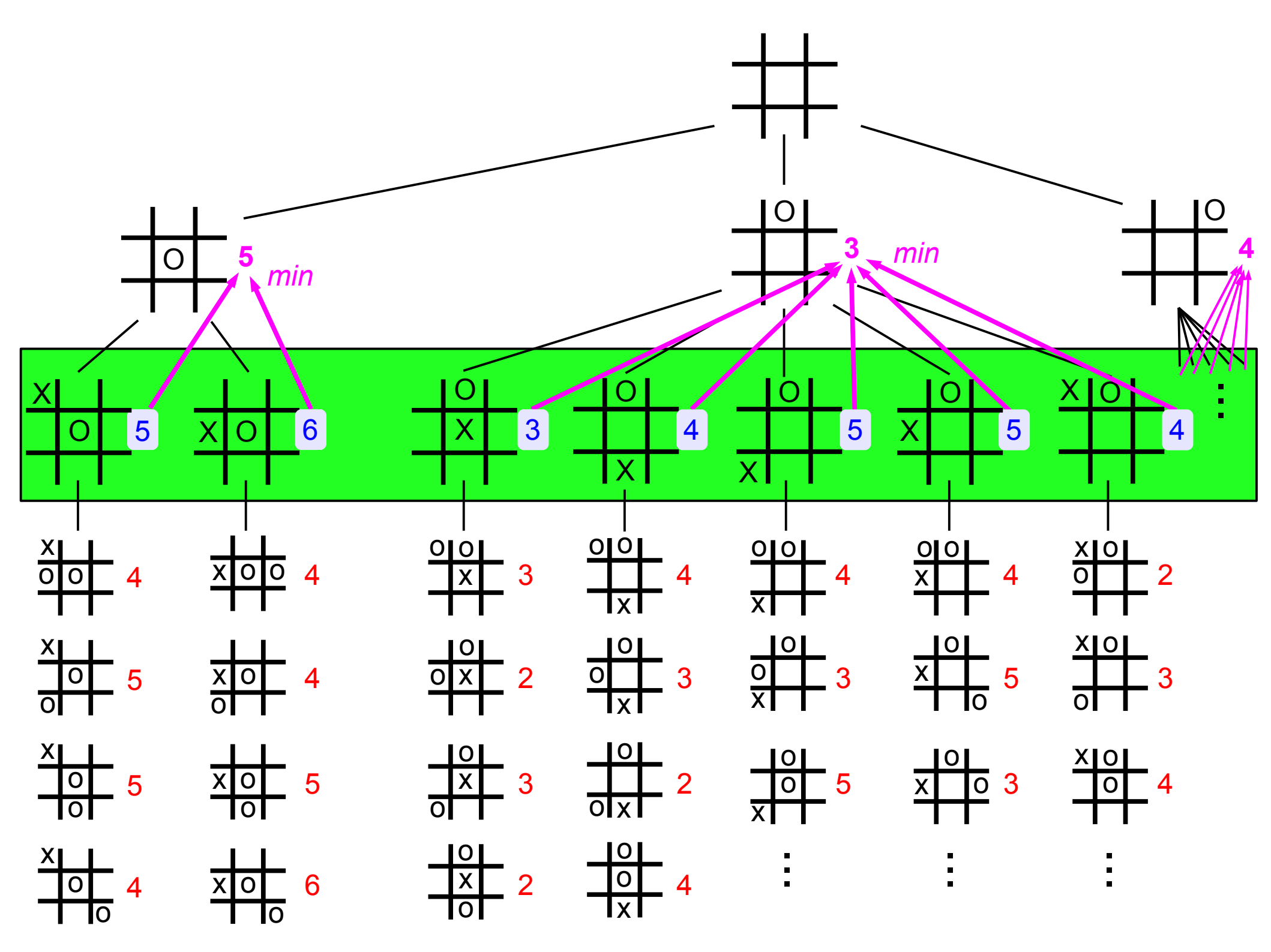


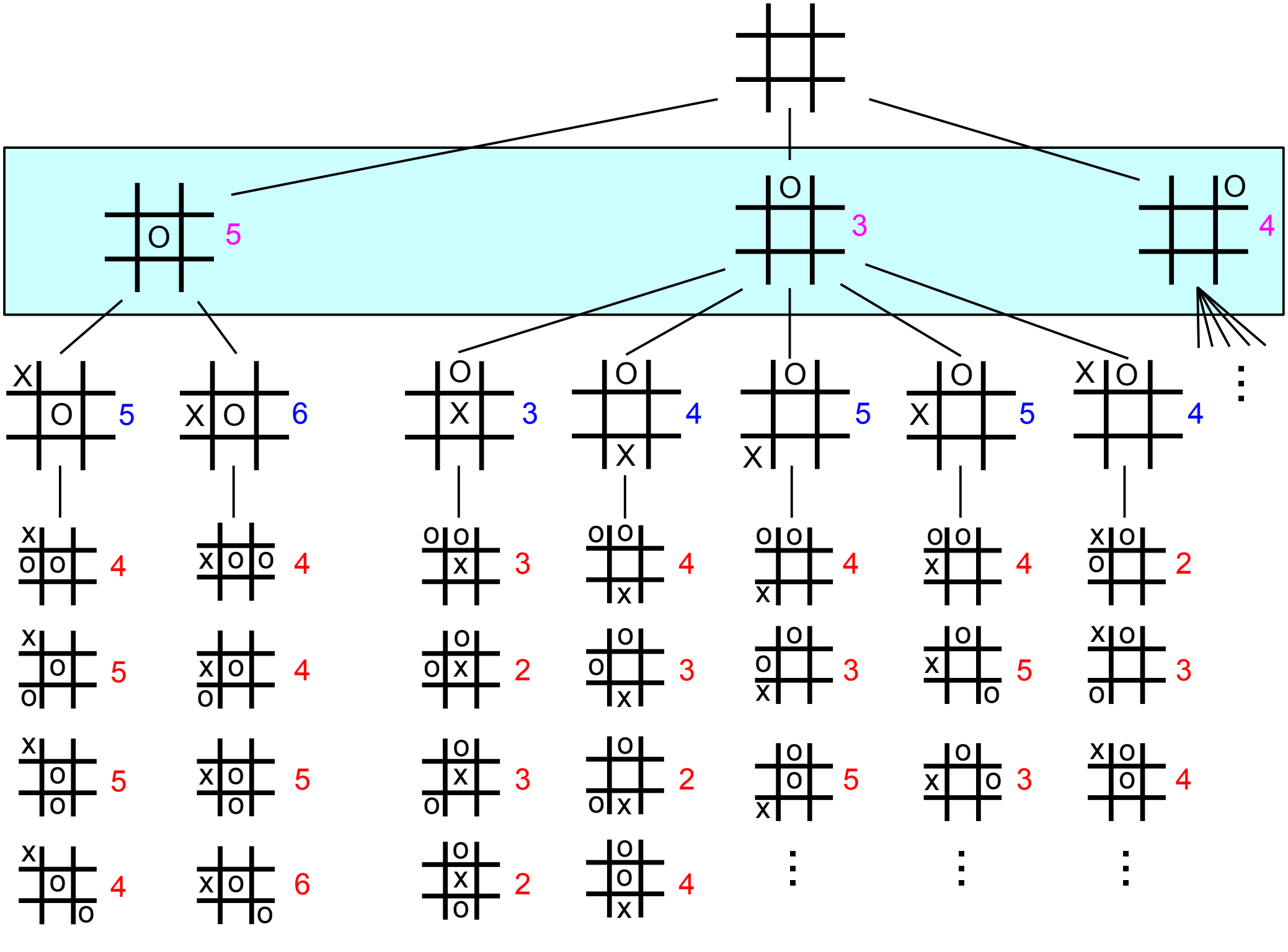

4 4 3 4 4 4 2
5 4 2 3 3 5 3
5 5 3 2 5 3 4
4 6 2 4

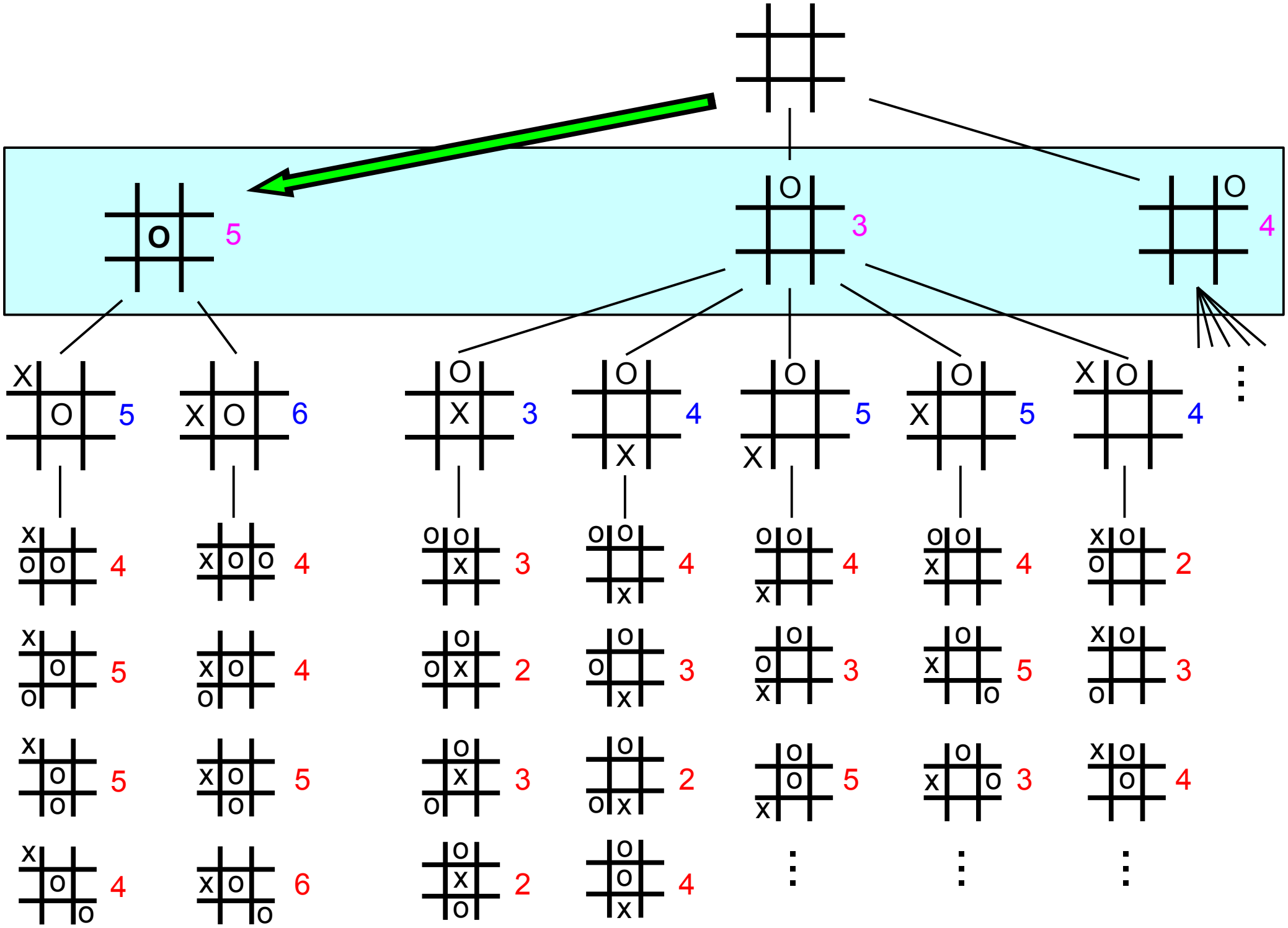


5
6
3
4
5
5
4
max
max
max
max
max
max
max
4
5
5
4
4
4
5
6
3
2
3
2
4
3
2
4
4
3
5
4
5
3
2
3
4


5
6
3
4
5
5
4
4
4
4
3
4
4
4
2
5
4
2
3
3
5
3
5
5
3
2
5
3
4
4
6
2
4


5
min
3
min
4
5
6
3
4
5
5
4
4
4
4
3
4
4
4
2
5
4
2
3
3
5
3
5
5
3
2
5
3
4
4
6
2
4

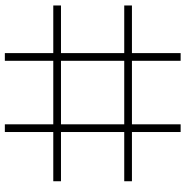

5
3
4
5
6
3
4
5
5
4
4
4
4
3
4
4
4
2
5
4
2
3
3
5
3
5
5
3
2
5
3
4
4
6
2
4

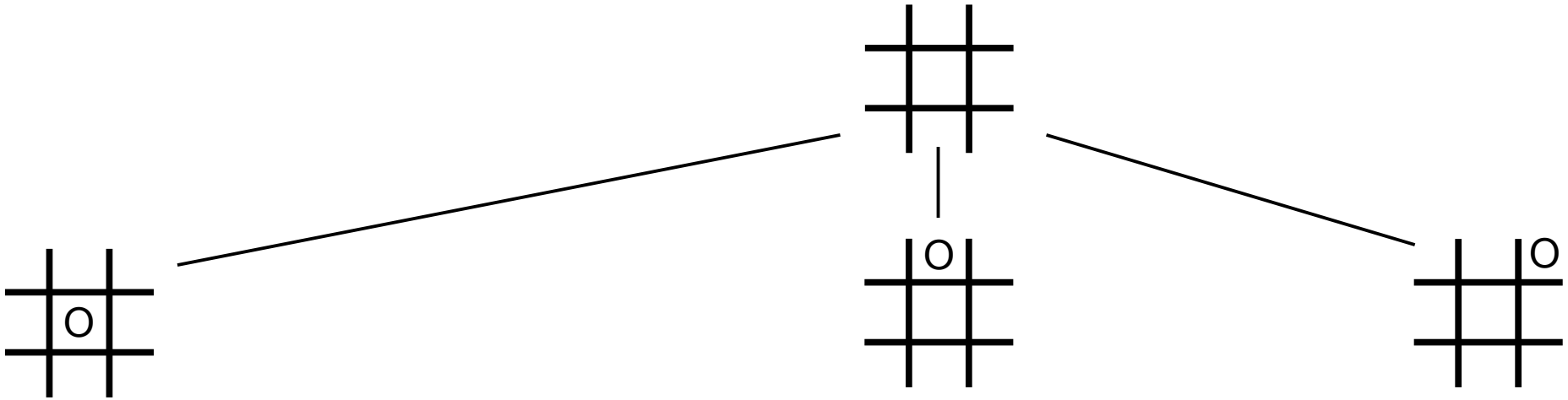

5
3
4
5
6
3
4
5
5
4
4
4
5
4
3
4
4
2
5
4
2
3
3
5
3
5
5
3
2
5
3
4
4
6
2
4

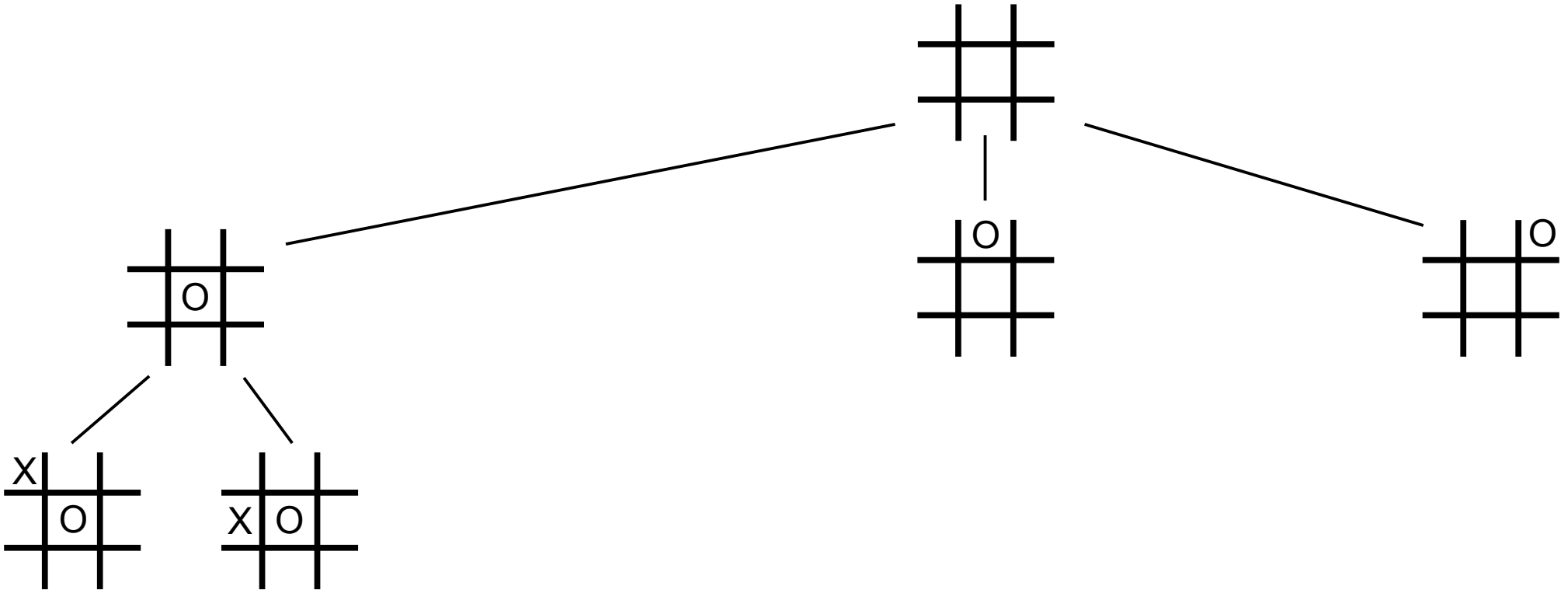
Pruning the Search Tree

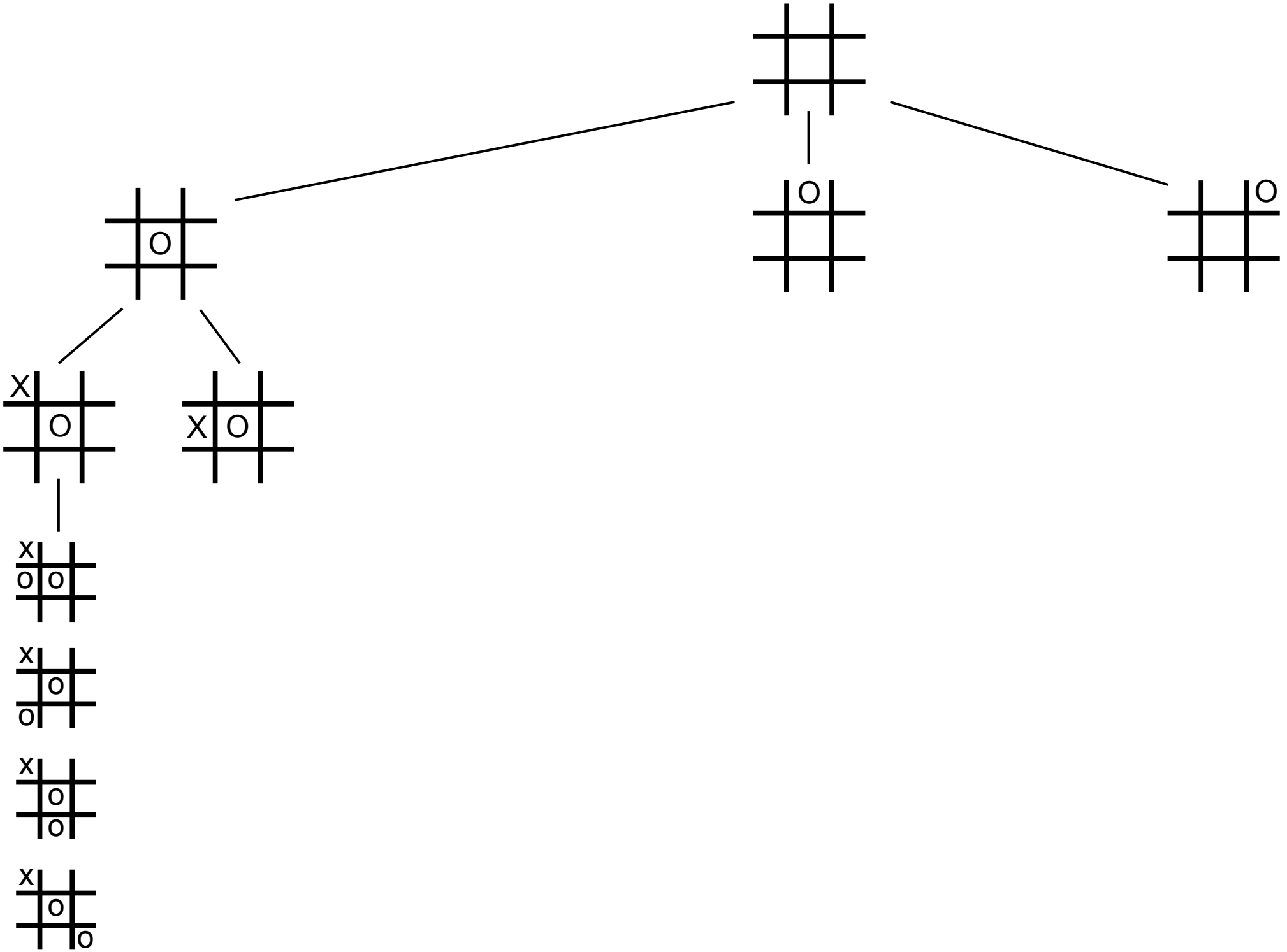
- In practice, we don't expand all nodes on each level at once
- Doing so is inefficient and may be unnecessary
- **Alpha-beta pruning** can significantly speed up the search
- We can avoid evaluating entire branches of the search tree

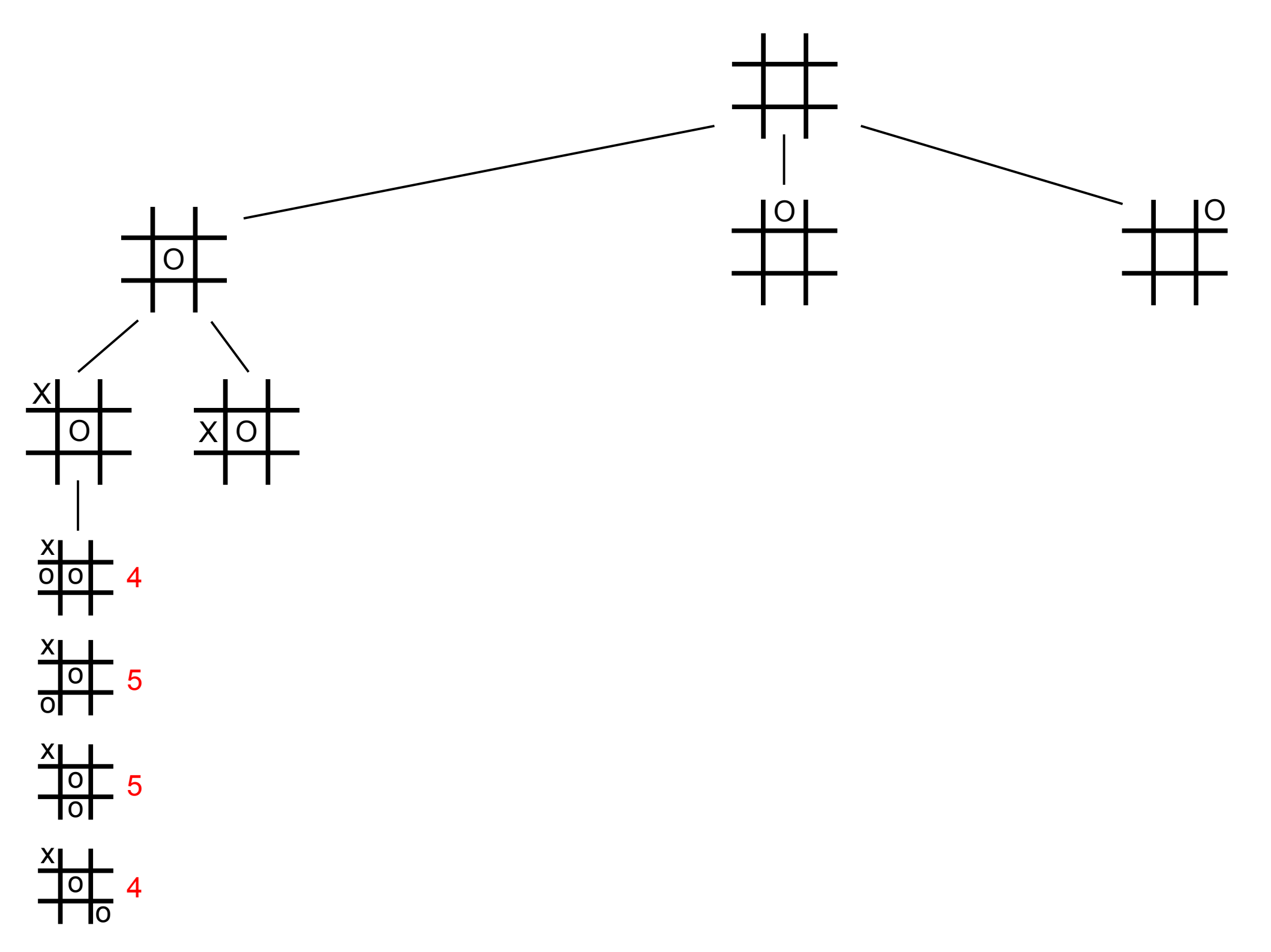
If an idea is surely bad, don't waste time analyzing just how bad it is

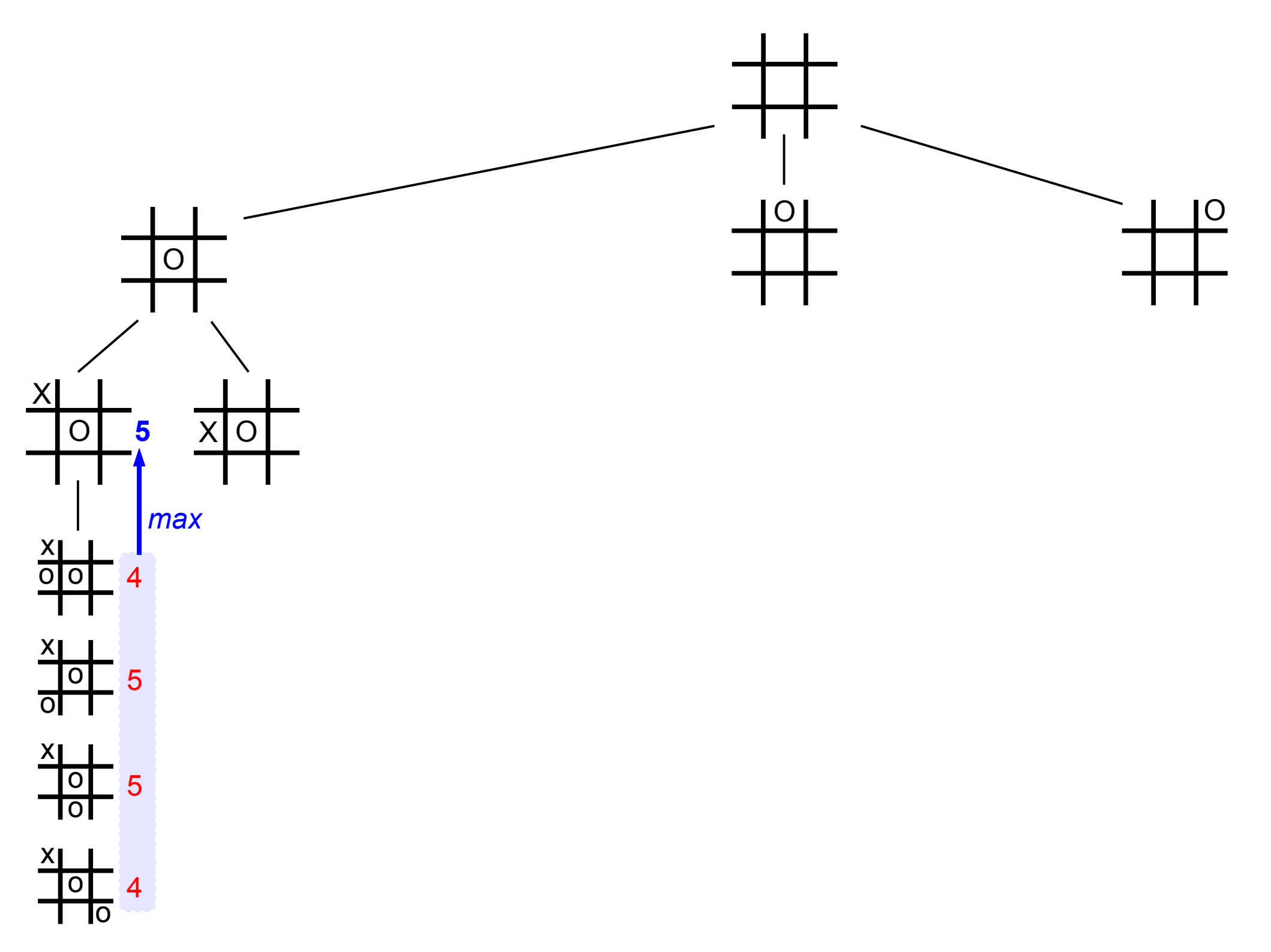


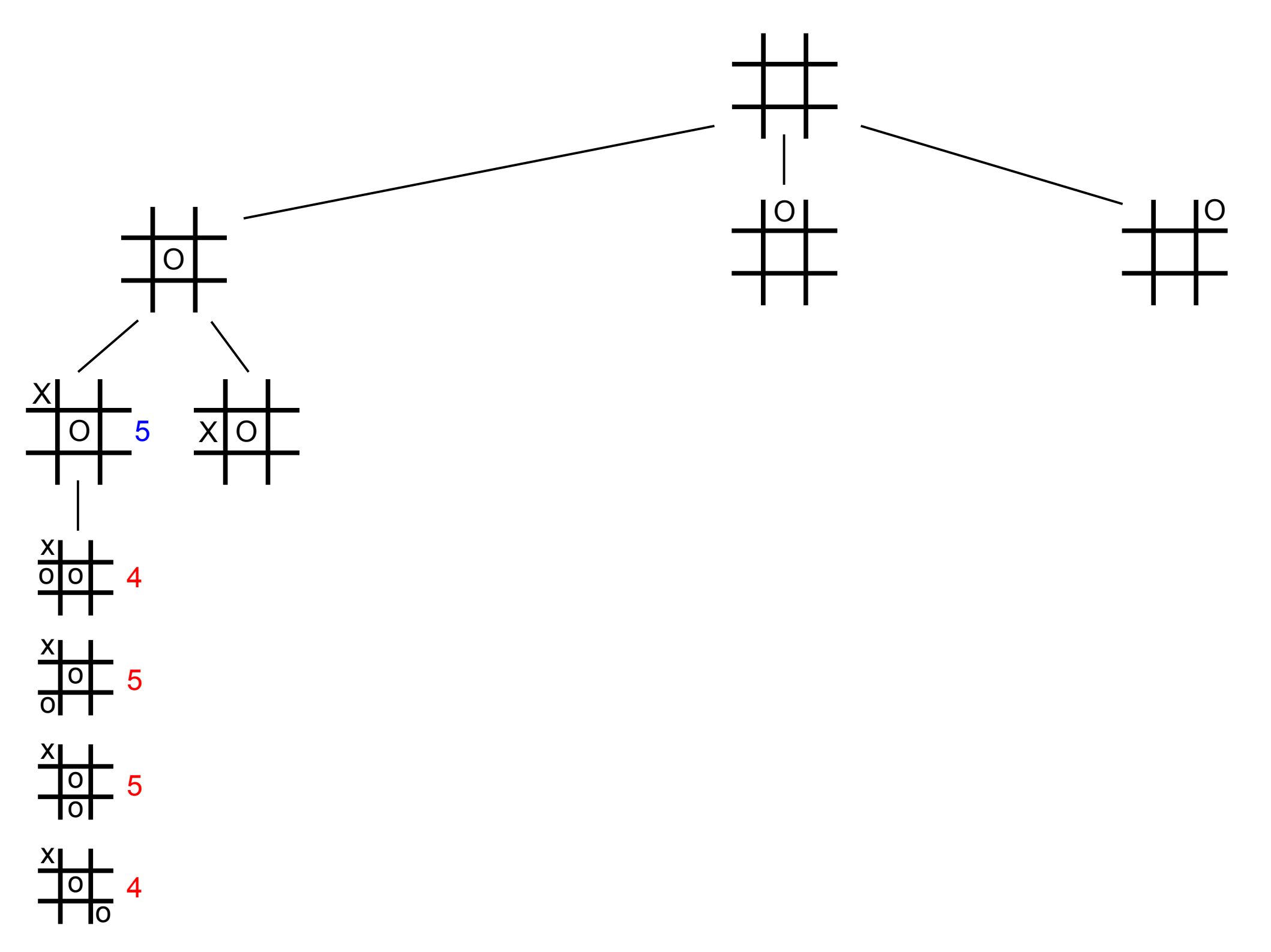

O
O
O

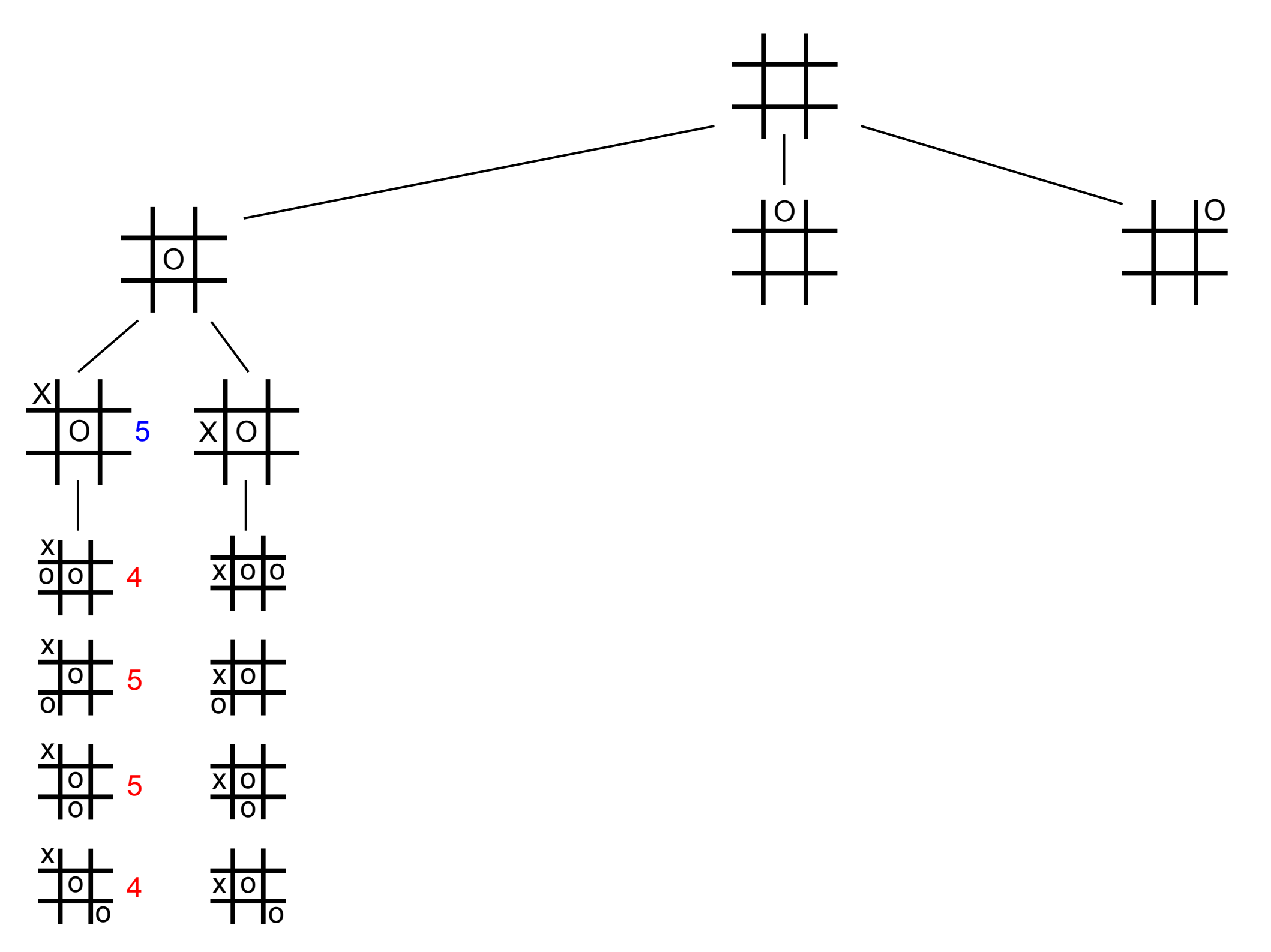

O
O
O
X
O
X
O

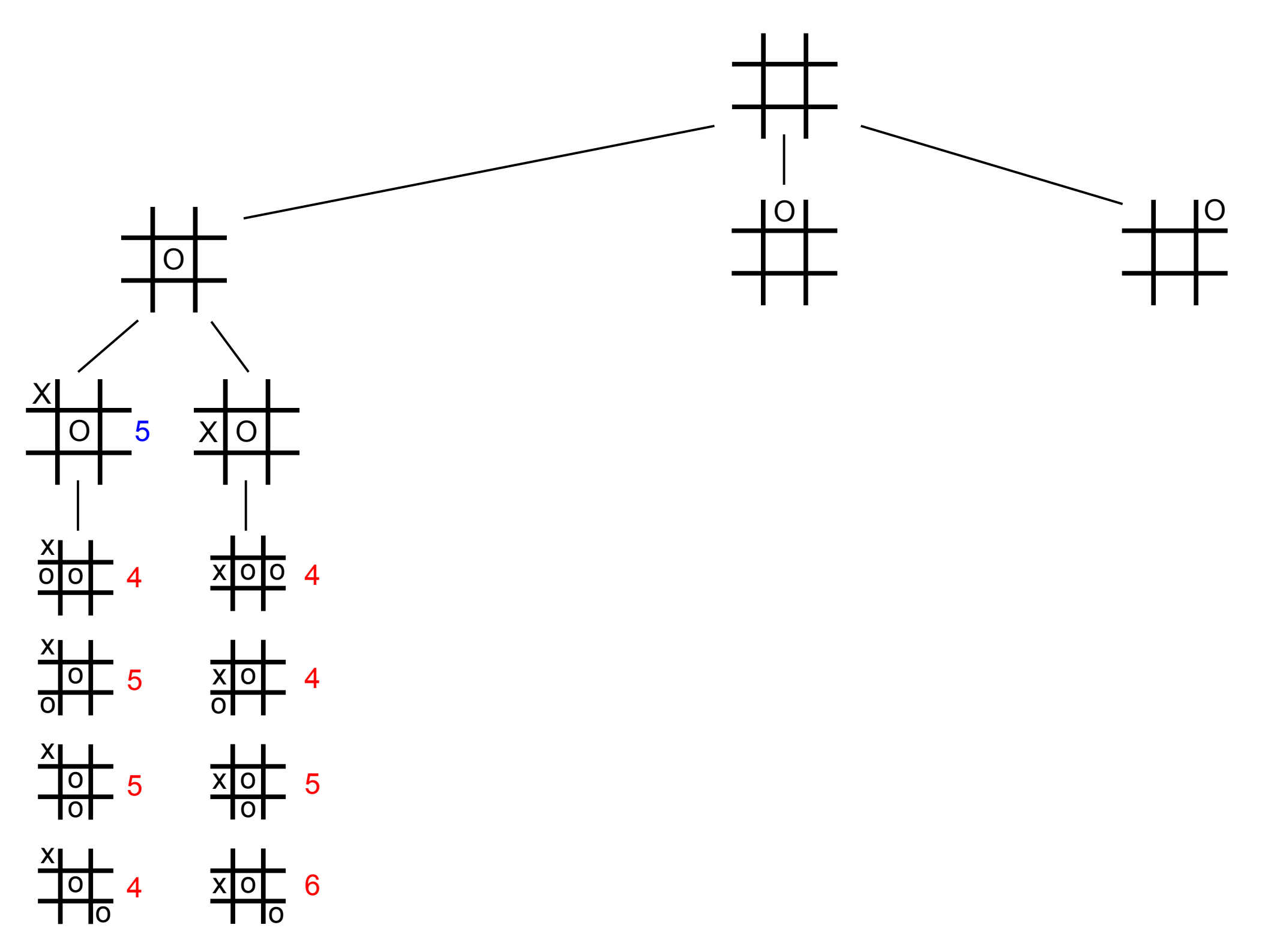

O
O
O
X
O
X
O
x
o
o
x
o
o
x
o
o
x
o
o

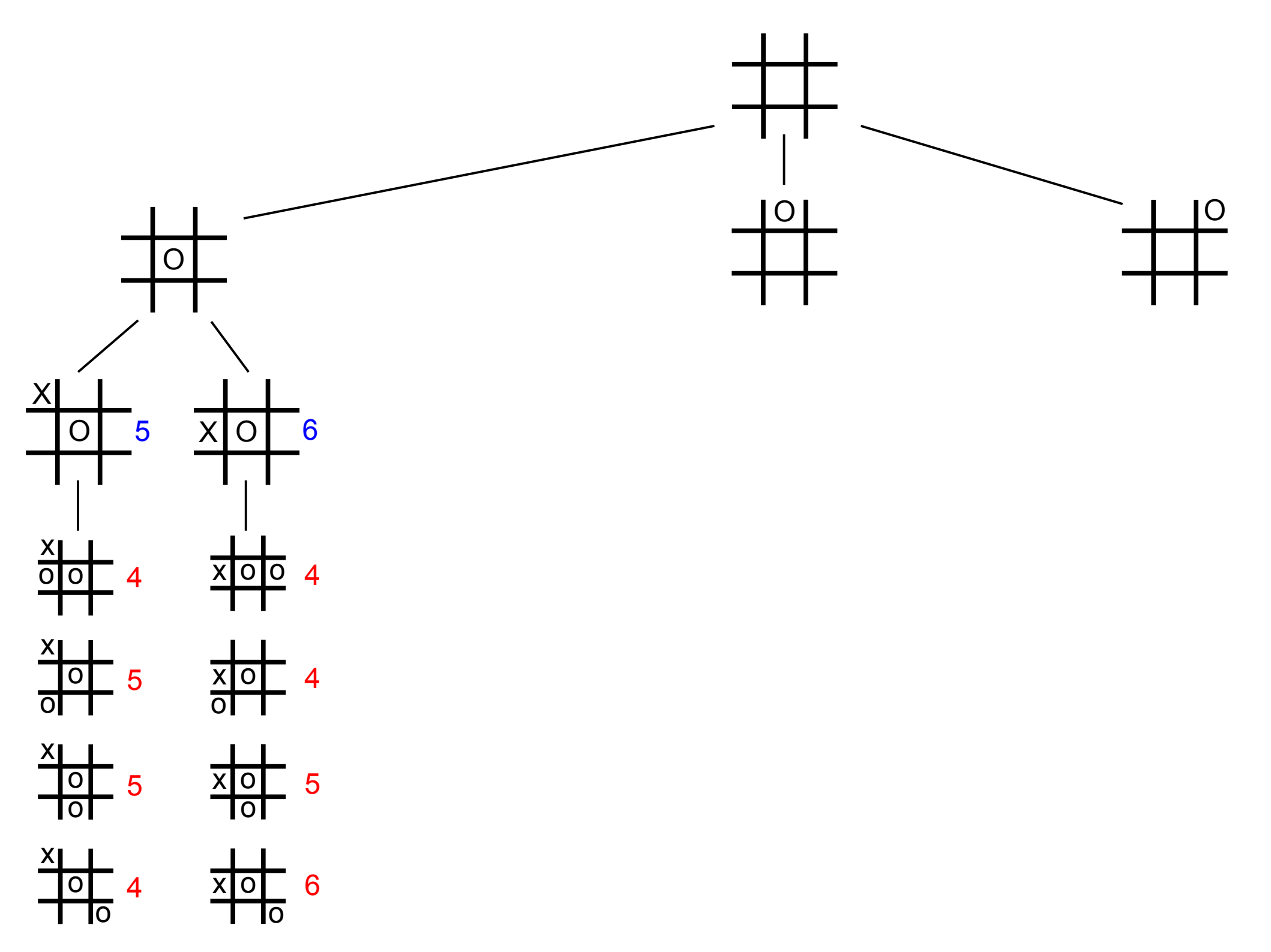

O
O
O
X
O
X
O
x
o
o
4
x
o
o
5
x
o
o
5
x
o
o
4

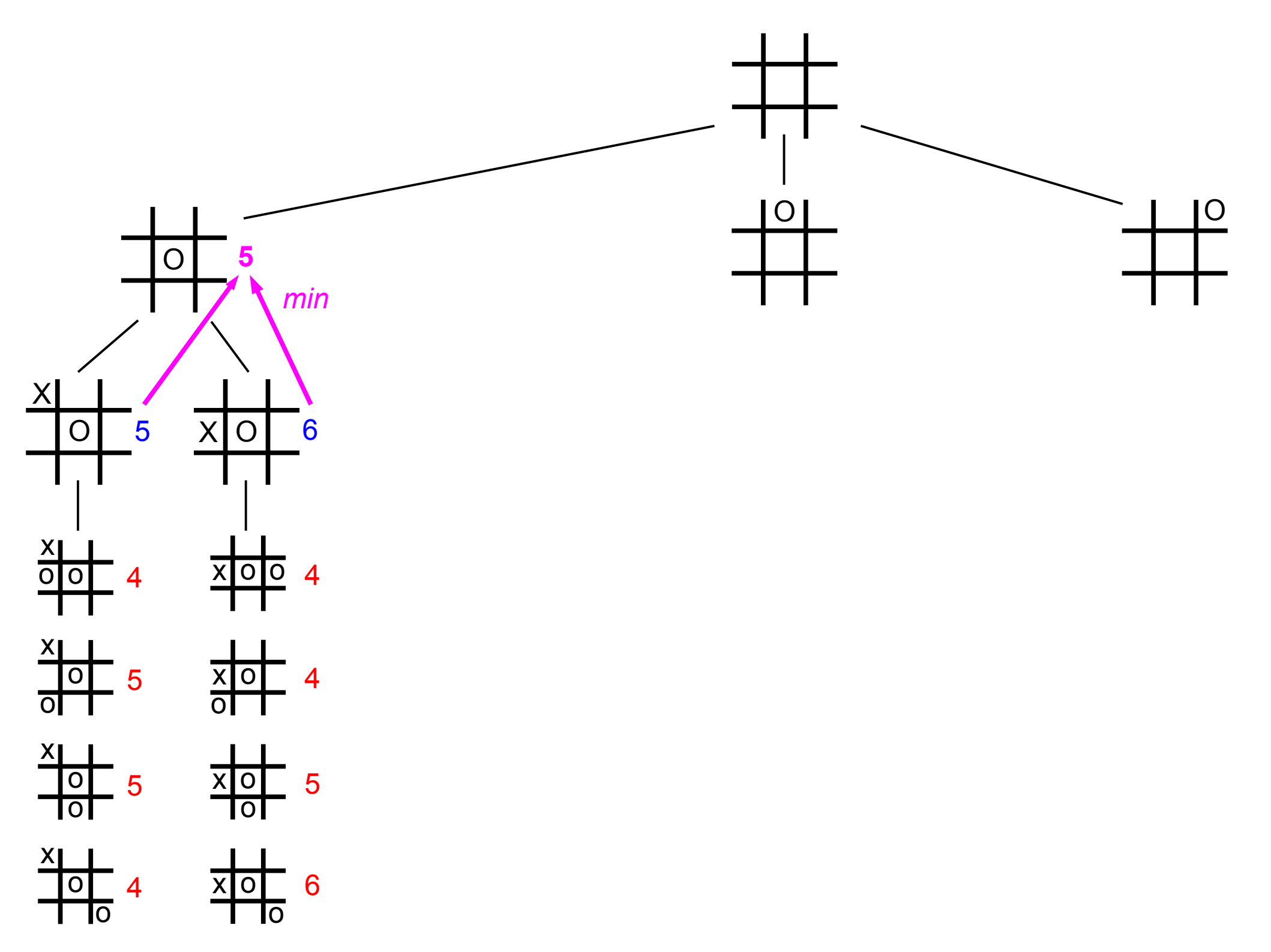

O
O
O
X
O
5
X
O
max
X
O
O
4
X
O
O
5
X
O
O
5
X
O
O
4

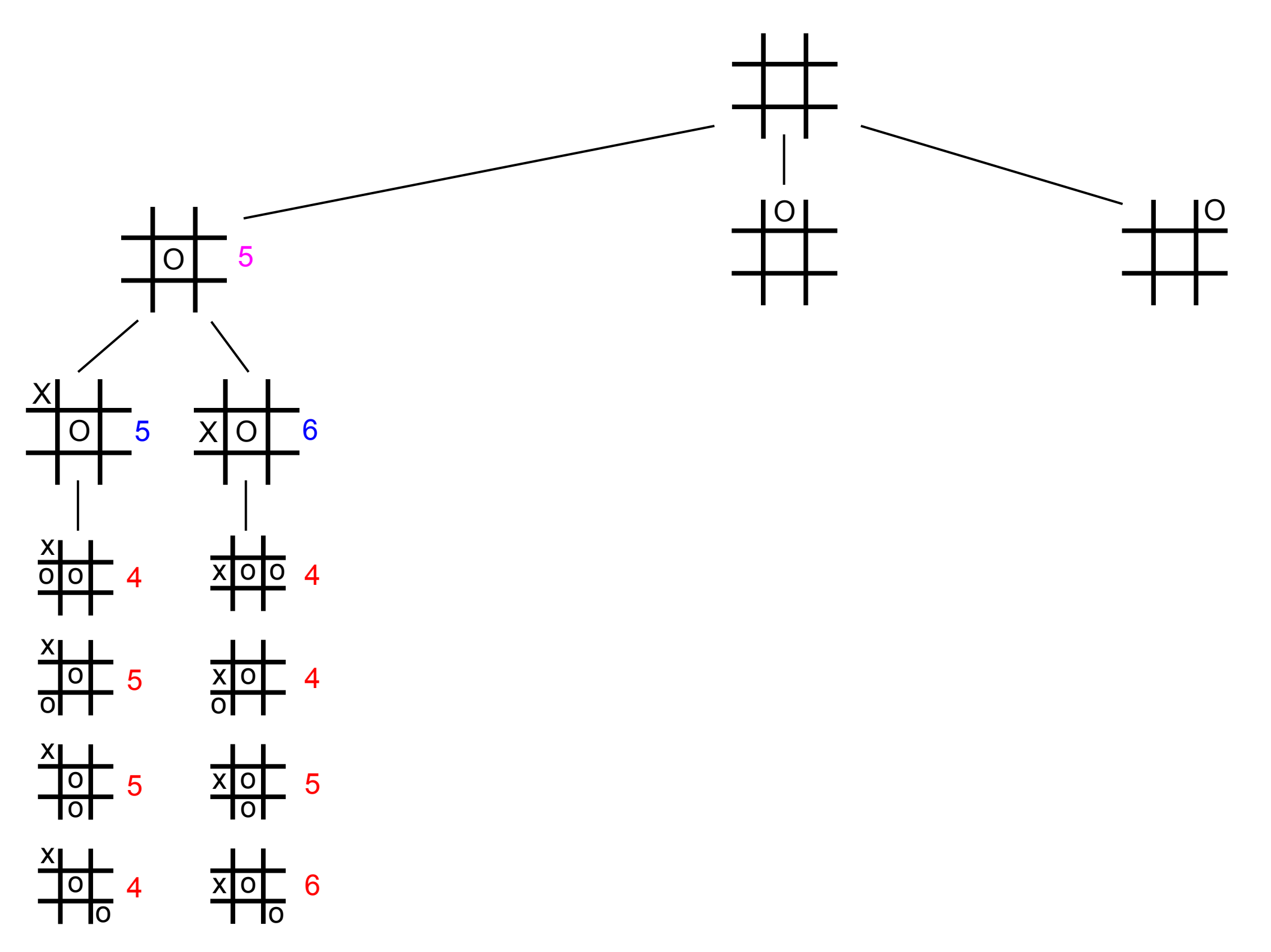

O
O
O
X
O
5
X
O
x
o
o
4
x
o
o
5
x
o
o
5
x
o
o
4

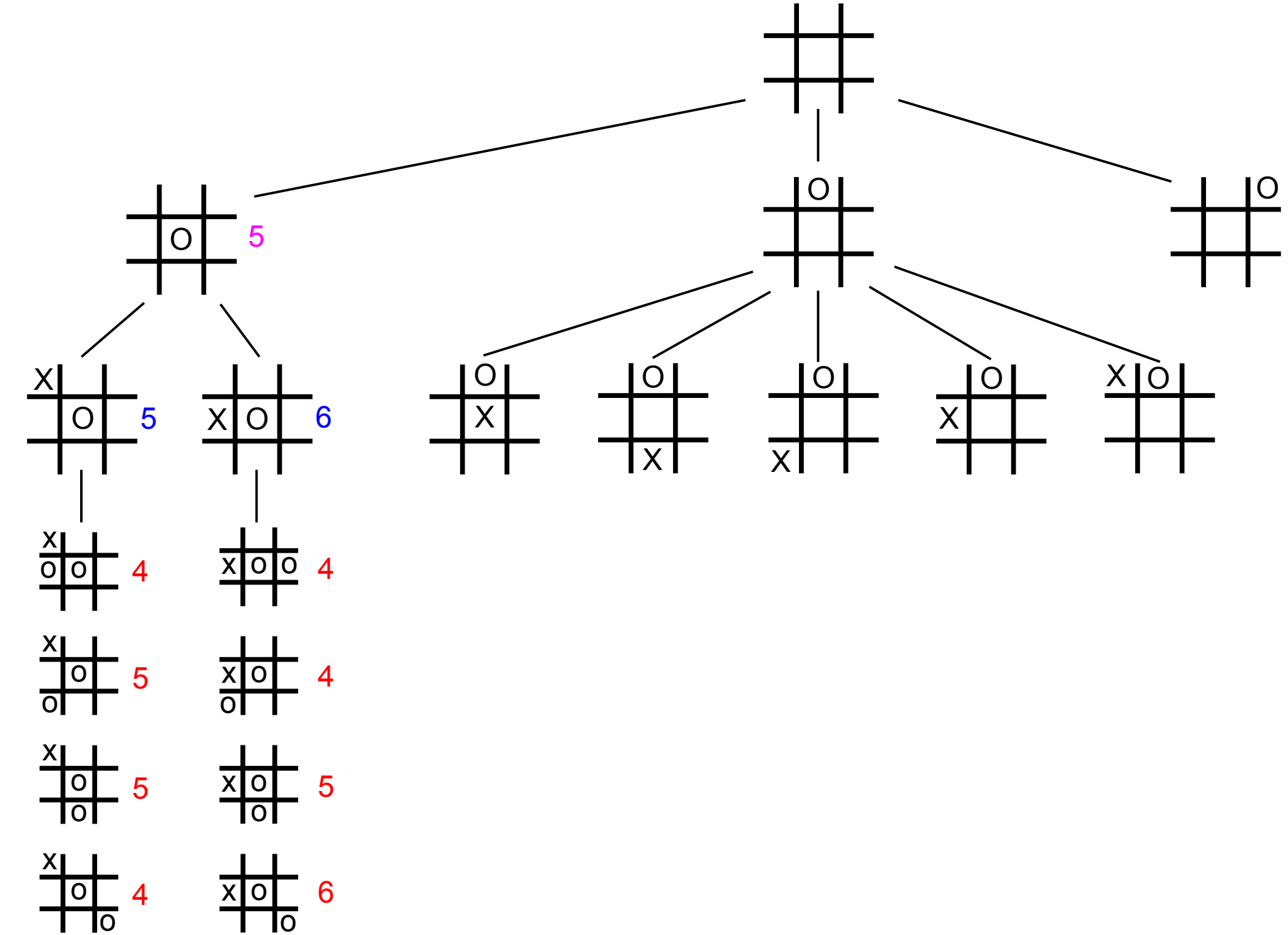

O
O
O
X
O
5
X
O
x
o
o
4
x
o
o
x
o
o
5
x
o
o
x
o
o
x
o
o
5
x
o
o
x
o
o
x
o
o
4
x
o
o
x
o
o

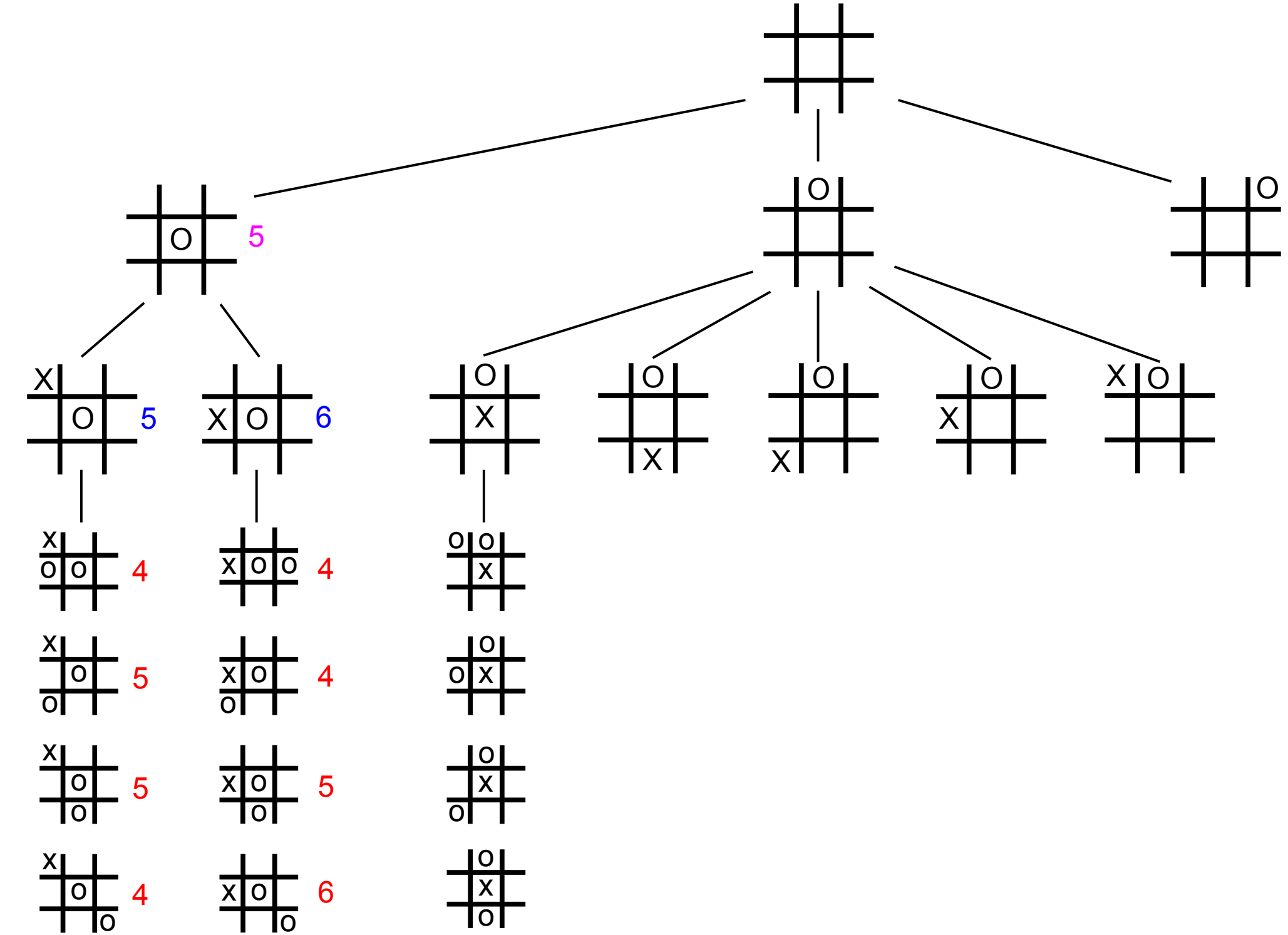

O
O
O
X
O
5
X
O
x
o
o
4
x
o
o
4
x
o
5
x
o
o
4
x
o
5
x
o
o
5
x
o
4
x
o
o
6

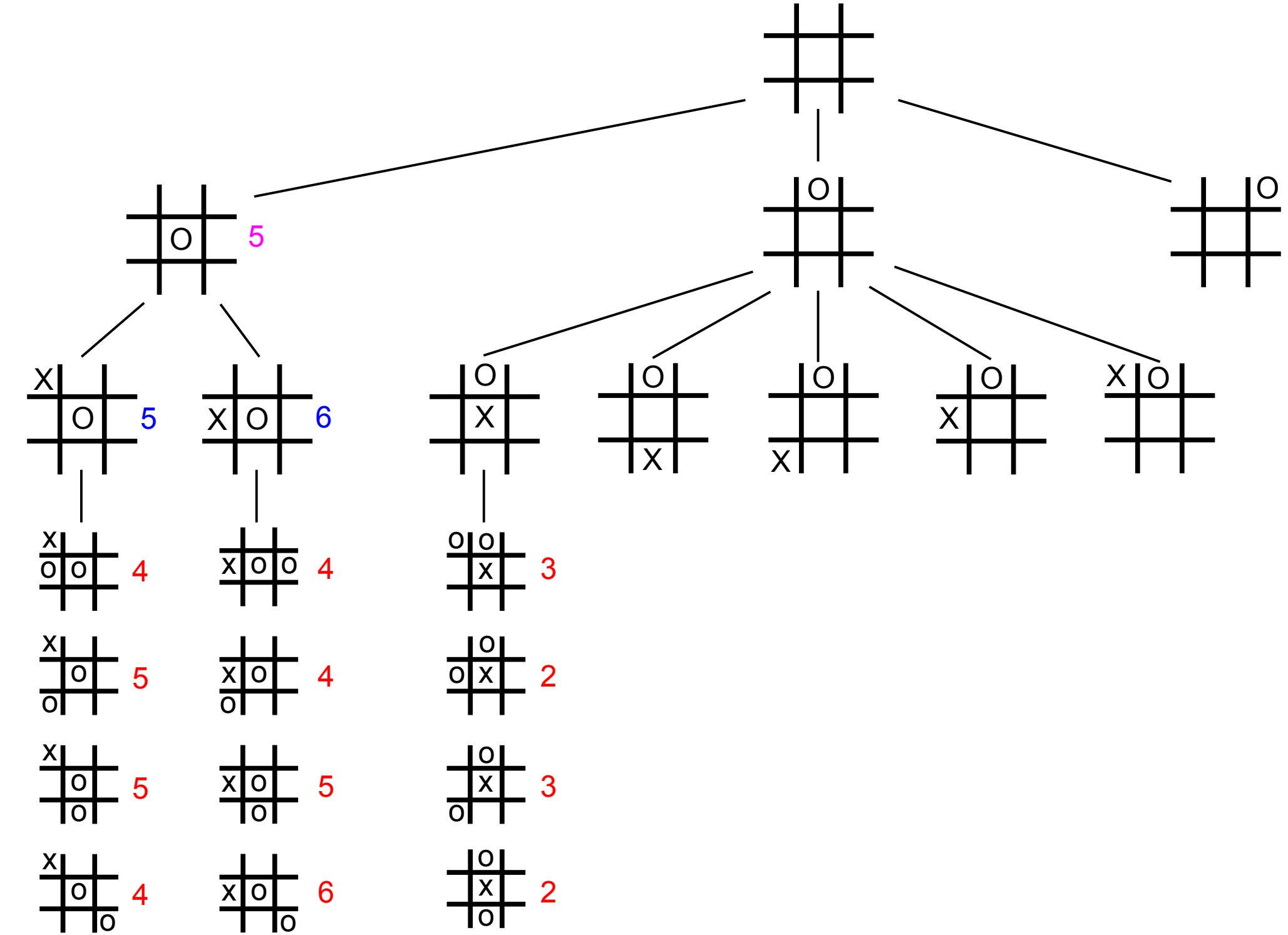

O
O
O
X
O
5
X
O
6
X
O
o
4
x
o
o
4
X
o
o
5
x
o
o
4
X
o
o
5
x
o
o
5
X
o
o
4
x
o
o
6

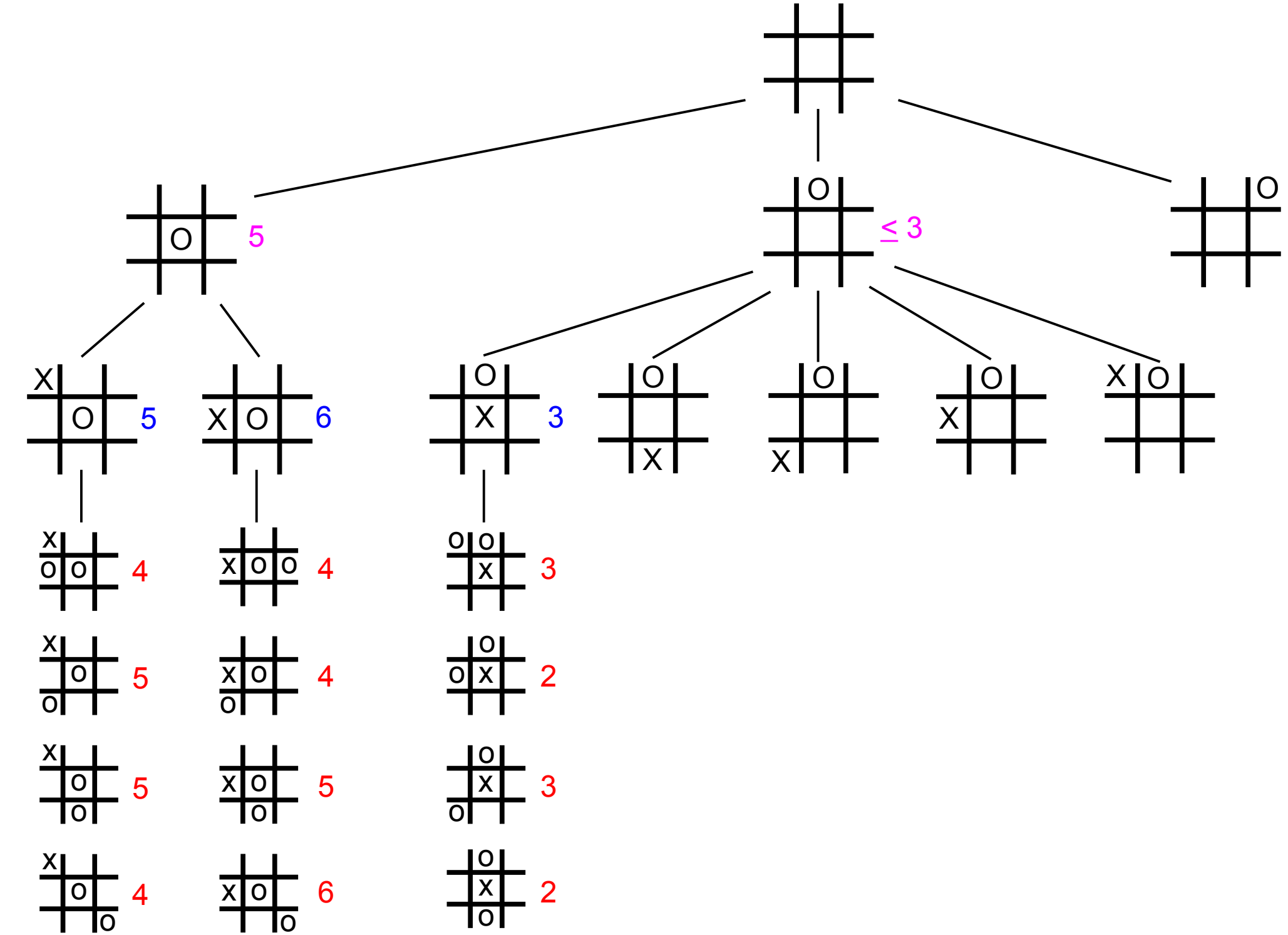

O
5
min
X
O
5
X O
6
X
O O
4
X O O
4
X
O
O
5
X O
O
4
X
O
O
5
X O
O
5
X
O
O
4
X O
O
6

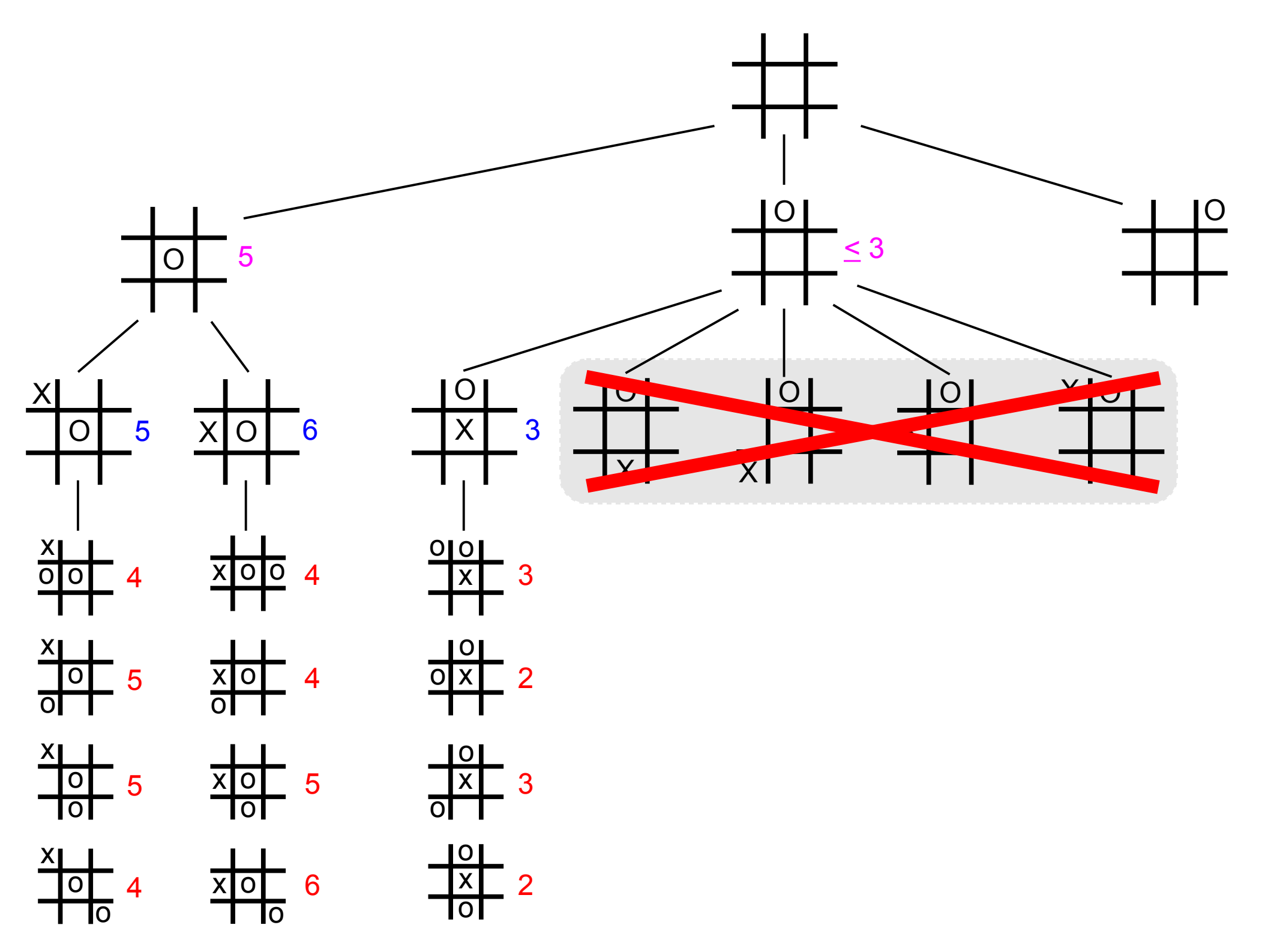

O
5
O
O
X
O
5
X
O
6
X
O
4
X
O
O
4
X
O
5
X
O
O
4
X
O
5
X
O
O
5
X
O
4
X
O
O
6

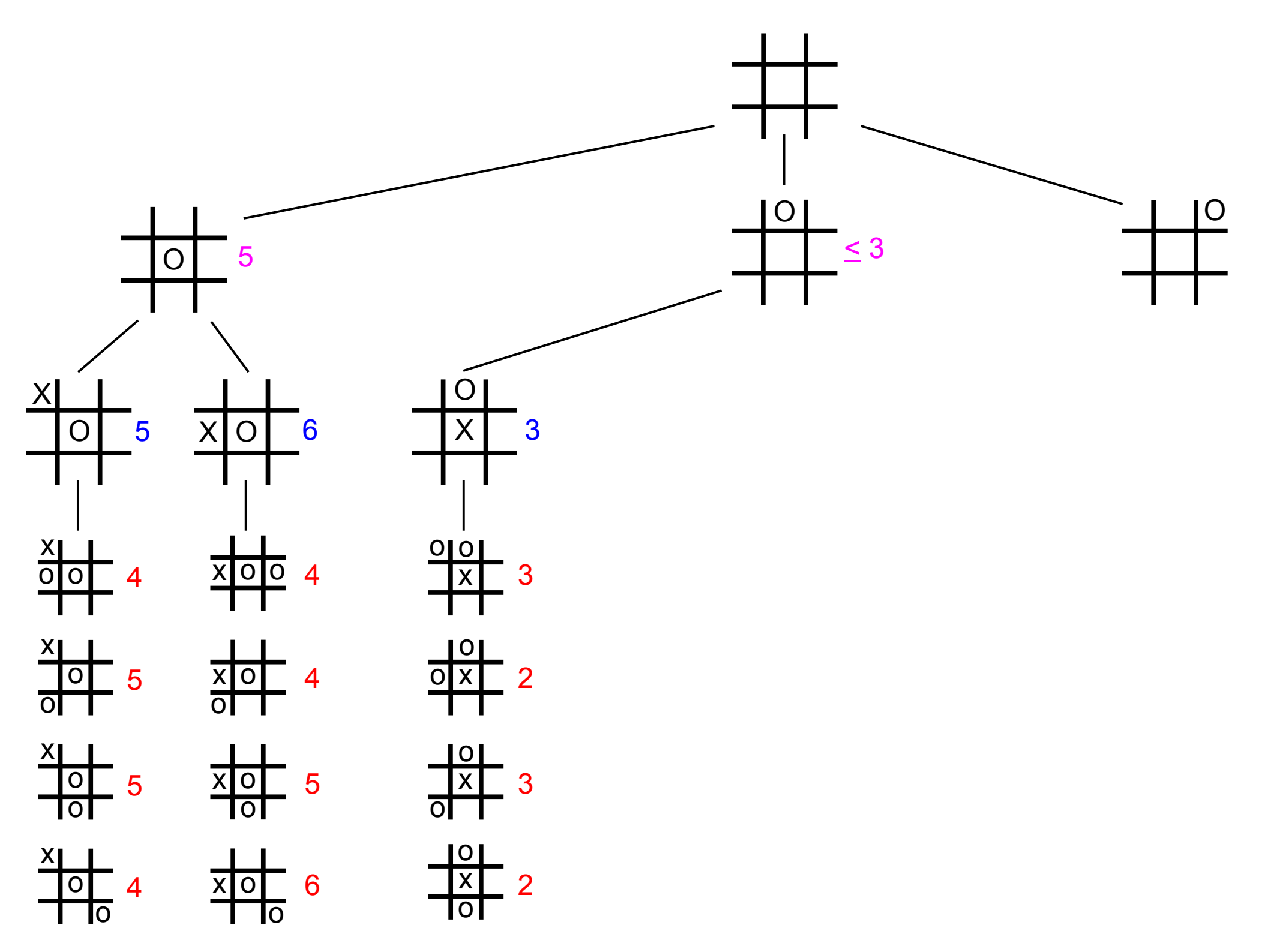

O
5
X
O
5
X O
6
O
X
O
X
O
X
O
X
X O
X
o o
4
x o o
4
x
o
o
5
x o
o
4
x
o
o
5
x o
o
5
x
o
o
4
x o
o
6

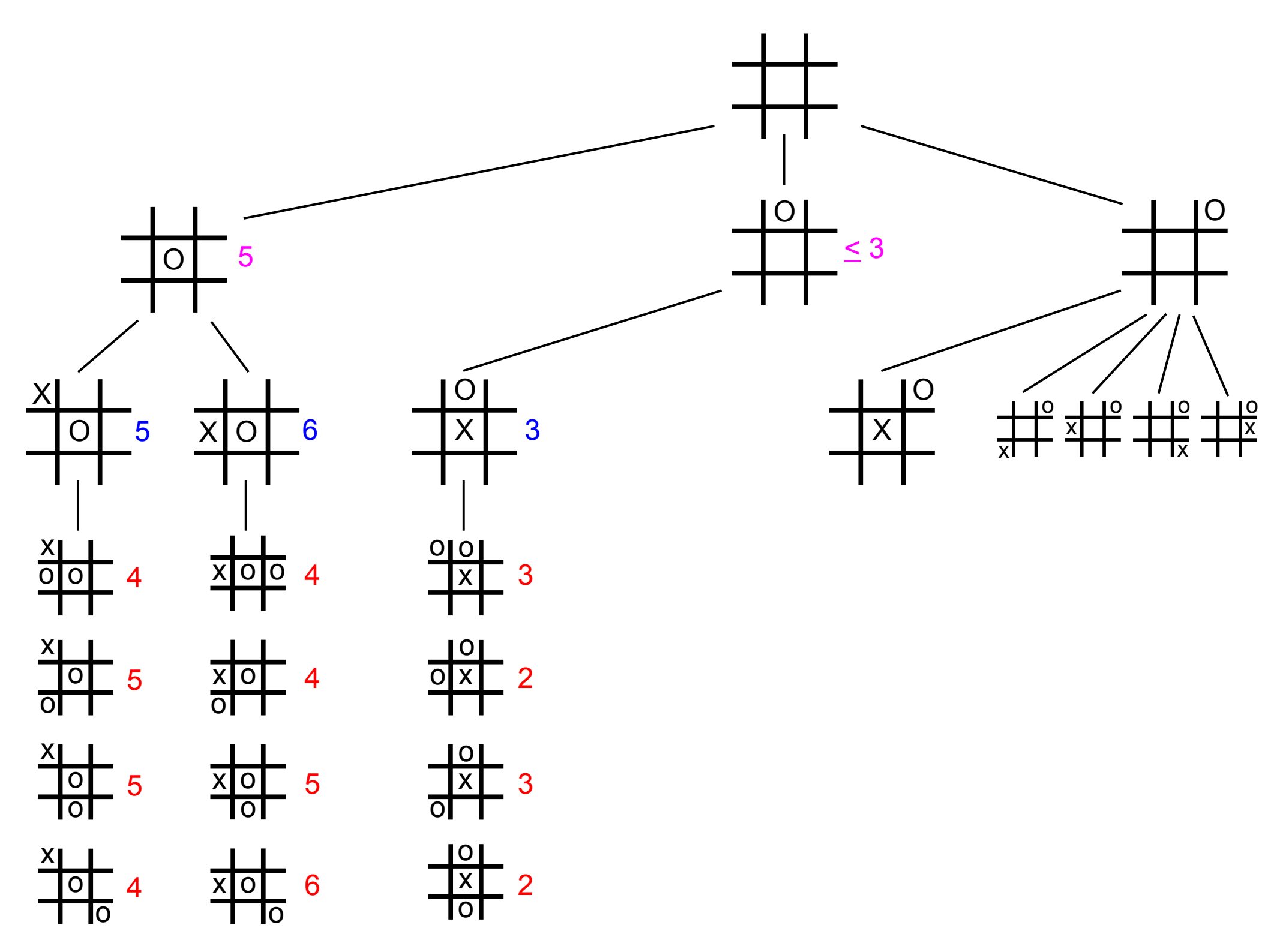

O
5
O
O
X
O
5
X
O
6
O
X
O
X
O
X
O
X
O
X
X
O
o
o
x
4
x
o
o
4
o
o
x
x
o
5
x
o
o
4
o
o
x
x
o
5
x
o
o
5
o
x
o
x
o
4
x
o
o
6
o
x
o

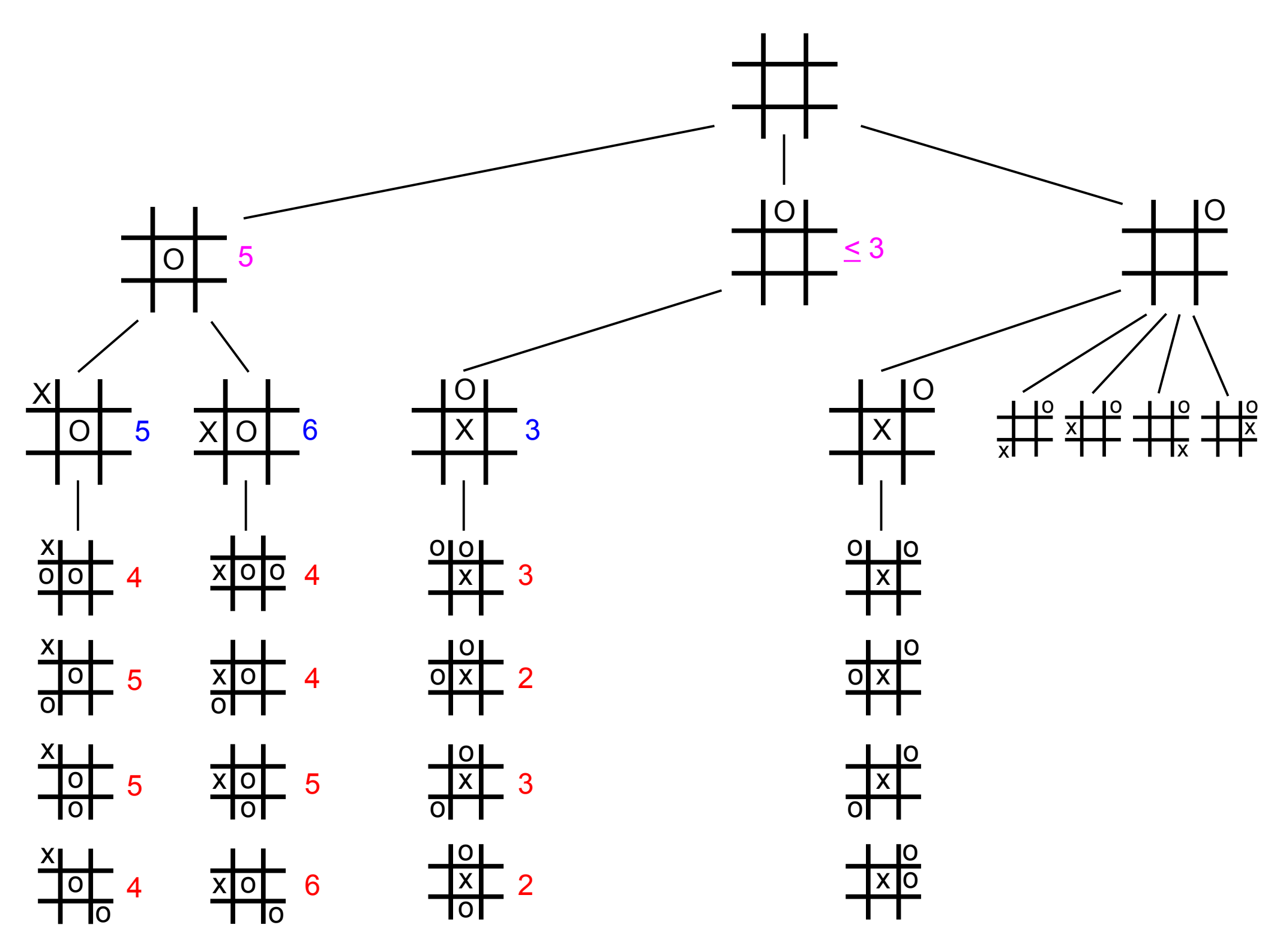

O
5
O
O
X
O
5
X
O
6
O
X
O
X
O
X
O
X
X
O
o
o
x
o
4
x
o
o
4
o
o
x
3
x
o
o
5
x
o
o
4
o
o
x
2
x
o
o
5
x
o
o
5
o
x
o
3
x
o
o
4
x
o
o
6
o
x
o
2

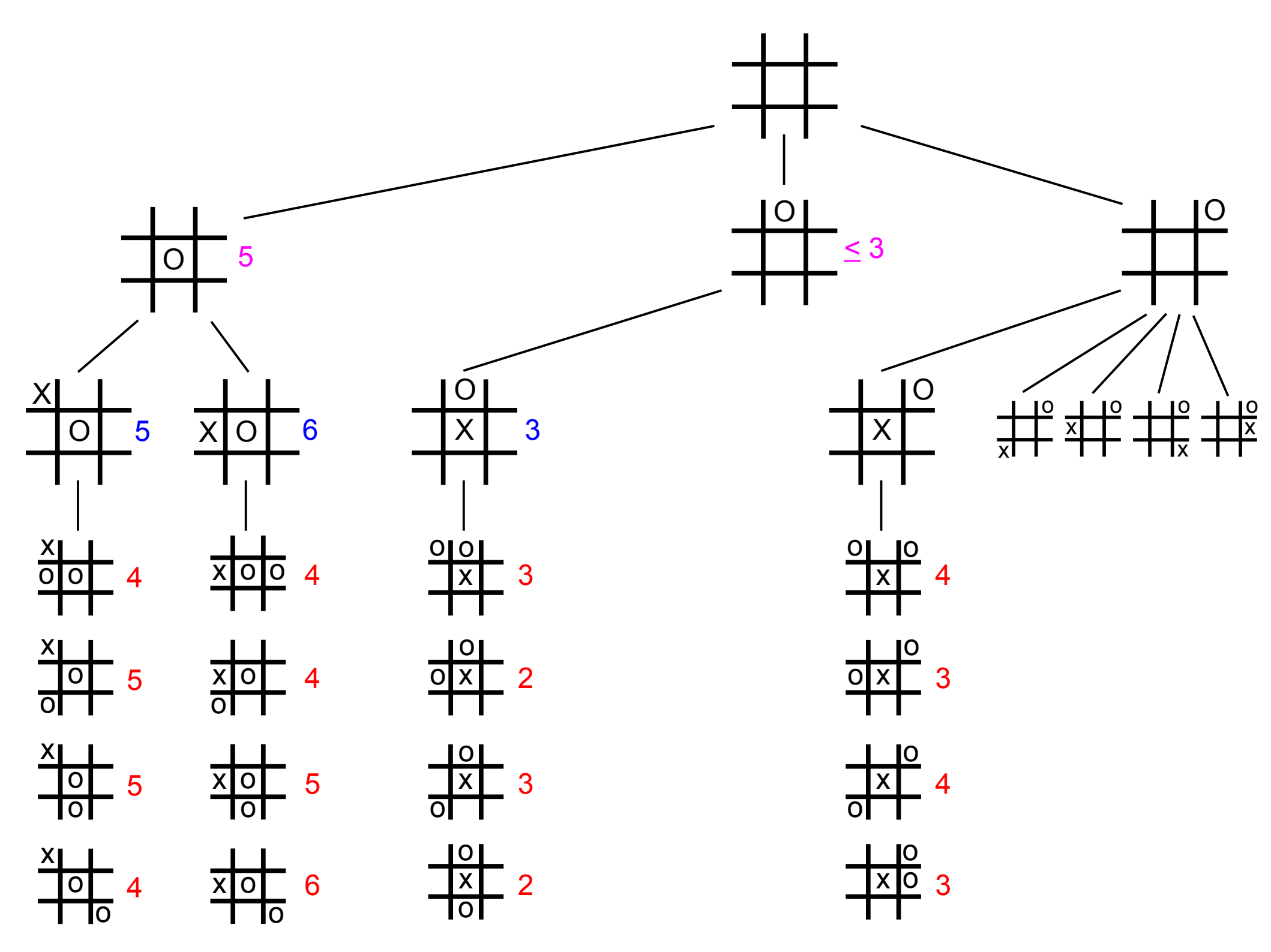

5
≤ 3
5
6
3
4
4
3
5
4
2
5
5
3
4
6
2

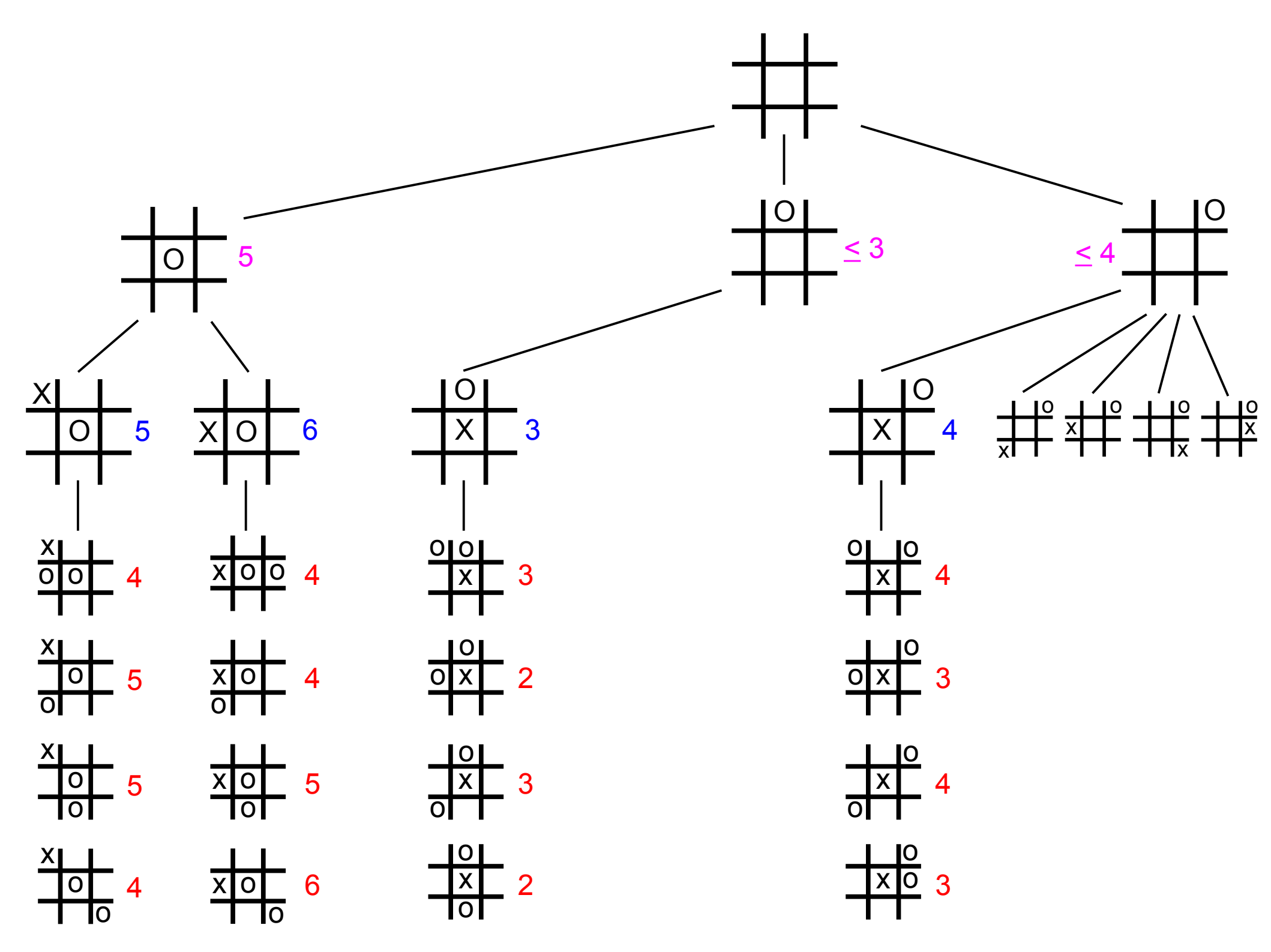

5
≤ 3
5
6
3
4
4
3
5
4
2
5
5
3
4
6
2

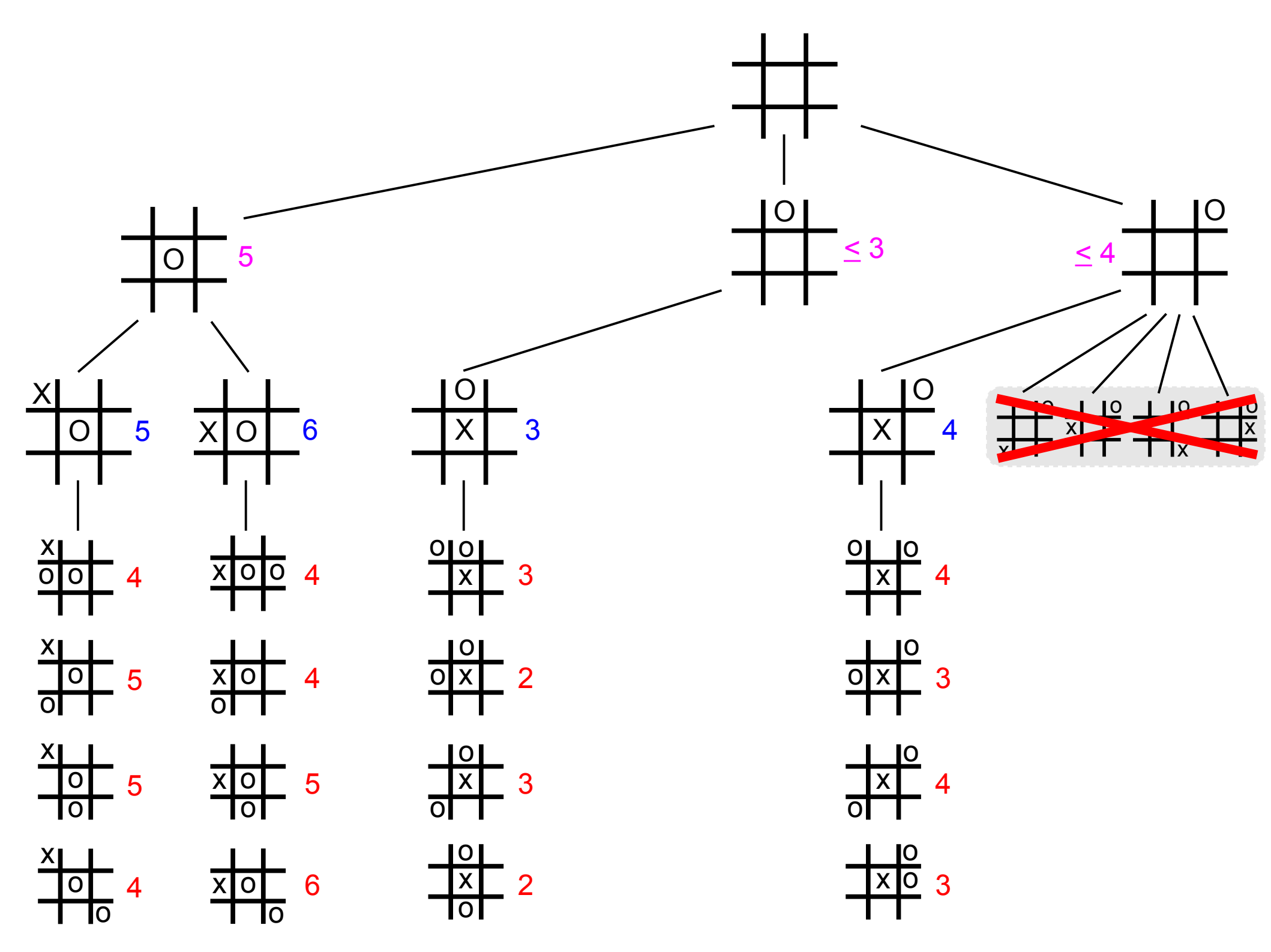

O
5
O
≤ 3
O
X
O
5
X
O
6
O
X
3
X
O
O
4
X
O
O
4
O
O
X
3
X
O
O
5
X
O
O
4
O
O
X
2
X
O
O
5
X
O
O
5
O
X
O
3
X
O
O
4
X
O
O
6
O
X
O
2

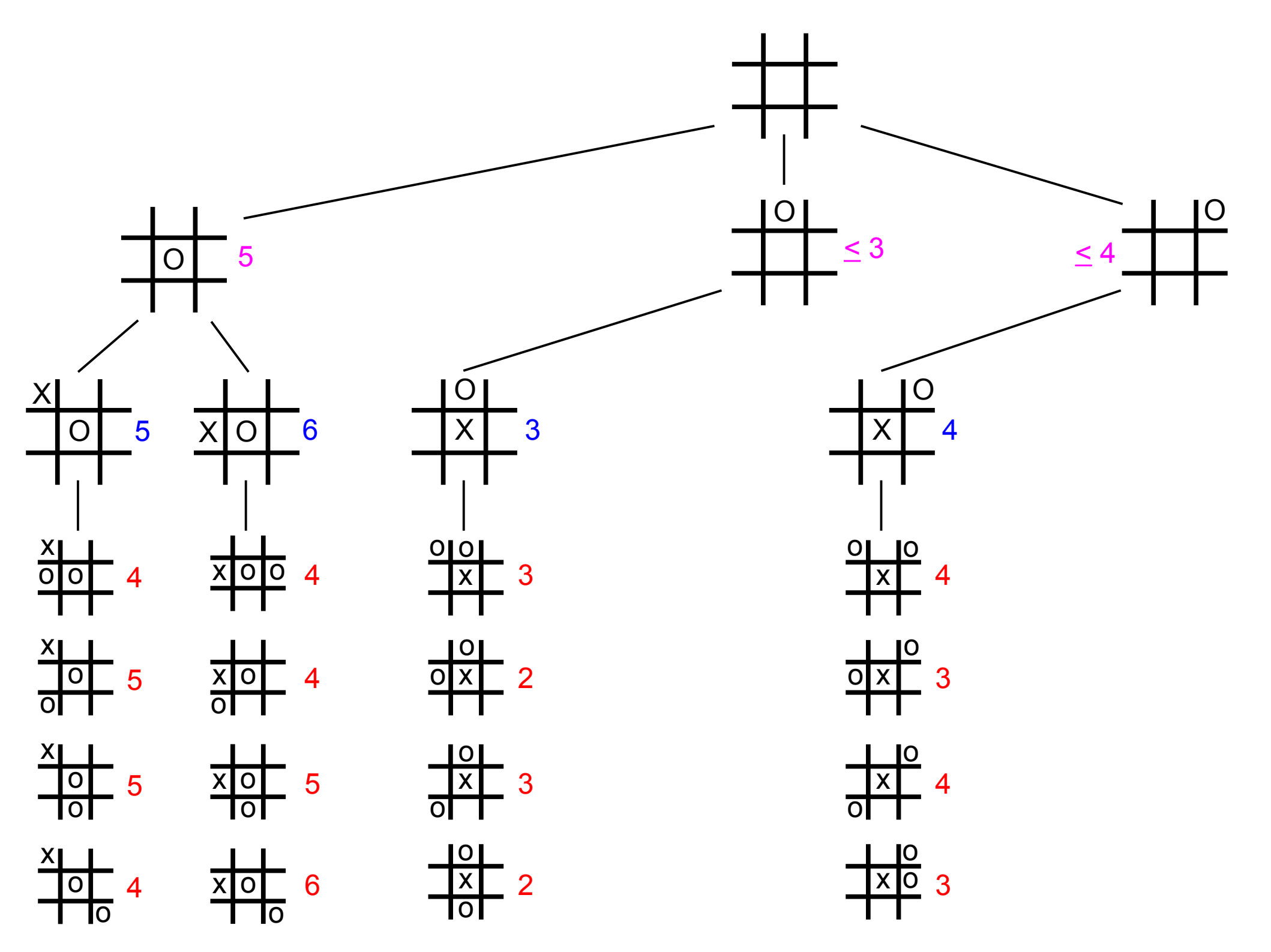

O
5
O
≤ 3
O
X
O
5
X
O
6
O
X
3
O
X
O
x
o
x
o
x
o
x
x
o
4
x
o
o
4
o
o
x
3
x
o
5
x
o
o
4
o
o
x
2
x
o
o
5
x
o
o
5
o
x
o
3
x
o
4
x
o
o
6
o
x
o
2

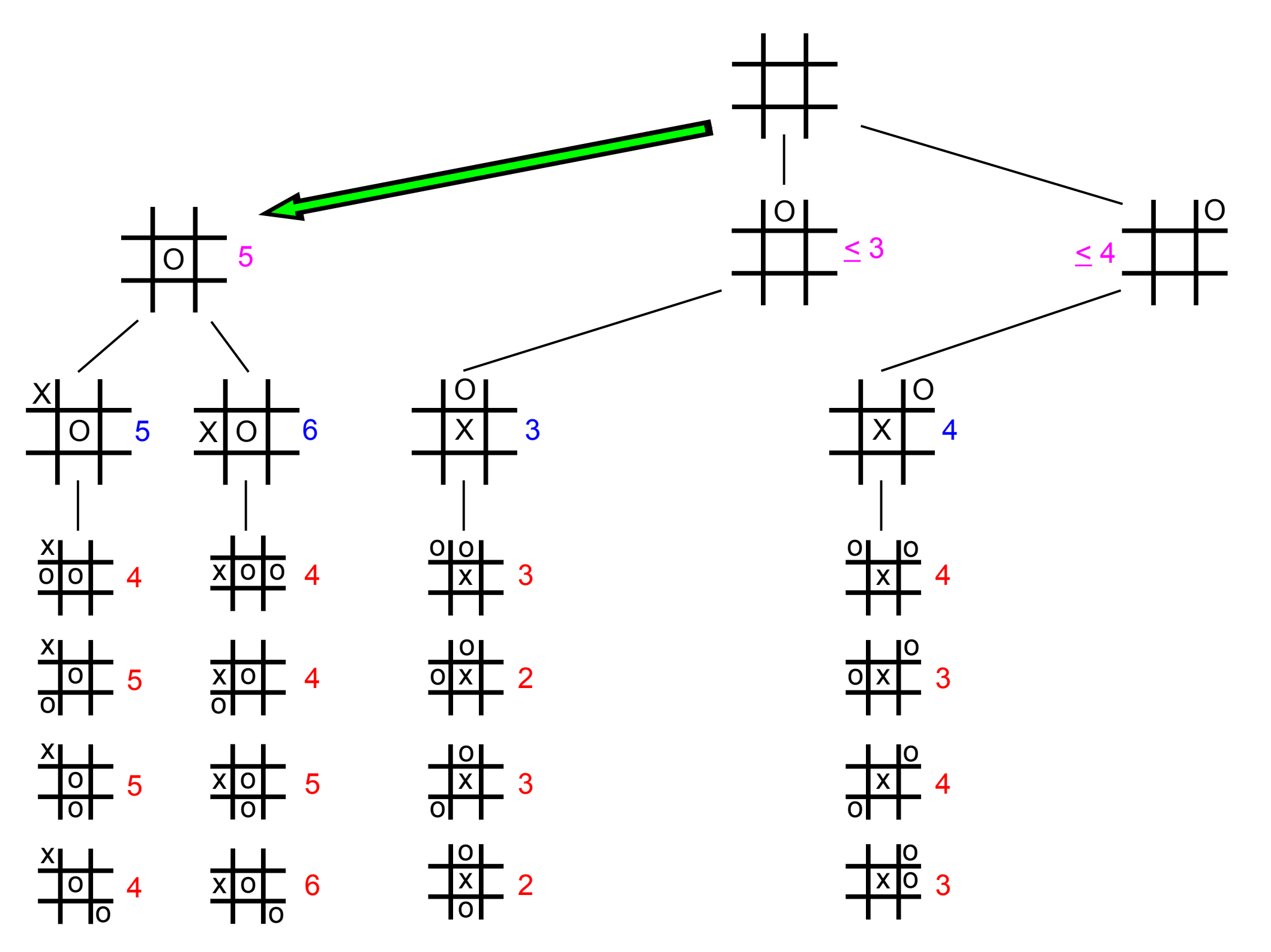

5
≤ 3
5
6
3
4
4
3
5
4
2
5
5
3
4
6
2


O
5
O
≤ 3
O
X
O
5
X
O
6
O
X
3
O
X
O
4
o
x
o
x
o
x
o
x
x
o
o
4
x
o
o
4
o
o
x
3
x
o
o
5
x
o
o
4
o
x
o
2
o
o
x
3
x
o
o
5
x
o
o
5
o
x
o
3
o
o
x
4
x
o
o
4
x
o
o
6
o
x
o
2
o
o
x
3


O
5
O
≤ 3
O
≤ 4
X
O
5
X
O
6
O
X
3
O
X
4
o
x
o
x
o
x
o
x
x
o
o
4
x
o
o
4
o
o
x
3
x
o
o
5
x
o
o
4
o
x
o
2
o
o
x
3
x
o
o
5
x
o
o
3
o
x
o
4
x
o
o
4
x
o
o
6
o
x
o
2
o
x
o
3


5
≤ 3
≤ 4
5
6
3
4
4
4
3
5
5
2
5
5
3
4
6
2
3

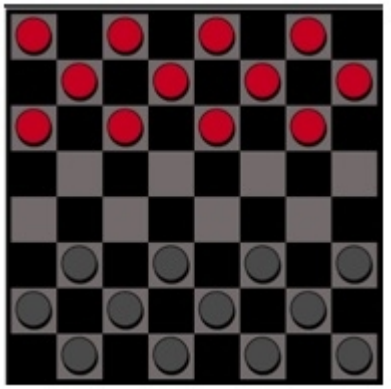

O
5
O
≤ 3
O
≤ 4
X
O
5
X
O
6
O
X
3
O
X
4
X
O
O
4
X
O
O
4
O
X
3
O
O
X
2
O
O
X
3
O
O
X
4
O
O
X
O
4
X
O
O
5
X
O
O
5
O
O
X
3
O
O
X
2
X
O
O
4
X
O
O
6
O
X
O
2
O
X
O
3


O
5
O
≤ 3
O
≤ 4
X
O
5
X
O
6
O
X
3
O
X
4
X
O
O
4
X
O
O
4
O
X
3
O
O
X
2
O
O
X
3
O
O
X
4
X
O
O
5
X
O
O
5
O
X
O
3
O
X
O
4
X
O
O
4
X
O
O
6
O
X
O
2
O
X
O
3

Pruning the Search Tree

- In this example, we performed 16 static evaluations
- Without alpha-beta pruning, we would have performed 66
- This represents a savings of 75%
- Amount of pruning depends on the order that nodes are expanded
- This example shows the best case scenario
- Effective branching factor reduces from b to $\sqrt{b}$
- This means that alpha-beta can look twice as far ahead as minimax for the same cost
- However, the search is still exponential even in the best case

Checkers



- More manageable than chess
  - average branching factor ~ 8
  - only 2 piece types
  - only 32 squares
  - still ~ 500 billion billion possible board positions
- Arthur Samuel's Checkers program
  - developed at IBM in 1959
  - first successful machine learning program
  - learned to play checkers better than Samuel himself
  - used evaluation functions, minimax, and heuristics

Chinook

- Checkers program developed by Jonathan Schaeffer and colleagues at the University of Alberta, Canada
- Uses minimax, alpha-beta pruning, various heuristics, and a large database covering hundreds of billions of opening moves and endgames (all endgames with 8 pieces or less)
- Examines game tree to ~ 20 ply
- Unlike Samuel's program, does not learn
- Marion Tinsley: world champion for 40 years, best player ever
  - in 1990, beat Chinook 7.5 to 6.5
  - in 1992, beat Chinook in World Checkers Championship (won 4, lost 2, drew 33)
  - in 1994, played Chinook in 6 games, all drawn, then resigned due to ill health

Chinook

- Chinook has never been beaten since then
- ...and never will be! In 2007 it was proven that the current version cannot be beaten, only drawn
- Board evaluation is a weighted function of:
  - piece count
  - king count
  - balance of the distribution of pieces across the board
  - number of trapped kings
  - etc.
- Search heuristics take into account the likelihood of a human player making mistakes on different pathways through the game tree

Go

- Still out of reach of AI programs
- 19 x 19 board
- Average branching factor ~ 360
- Search must be extremely selective
- Pattern recognition is very important
- Best programs can be trounced by human players
- $1 million prize for first program to beat a professional Go player