Artificial Intelligence
Adversarial search

More about

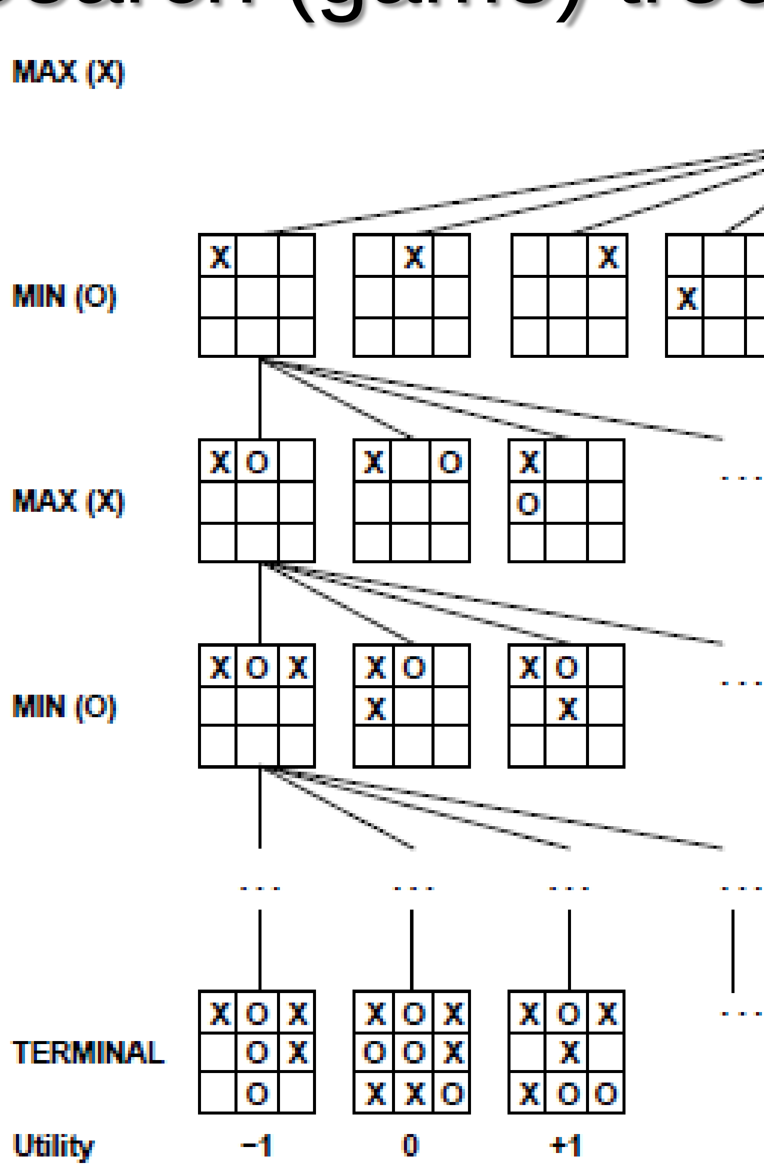
Textbook, Chapter 6, Adversarial Search

Adversarial search – two-player games

Problem:
- not only we act, the opposing agent also acts
- we do not know its actions
- we must be prepared for every contingency with a strategy
- generally zero-sum games (win = -loss)
- ability to make *some decision even when calculating the optimal* decision is infeasible
- games penalize inefficiency severely.

Model:
- Max, Min players (Max moves first)
- initial state
- successor function
- terminal test (goal test)
- utility (objective, pay-off) function

Search (game) tree (tic-tac-toe)



e.g. Chess:

search state of the game:

ca. 10^{47} states

search (game) tree:

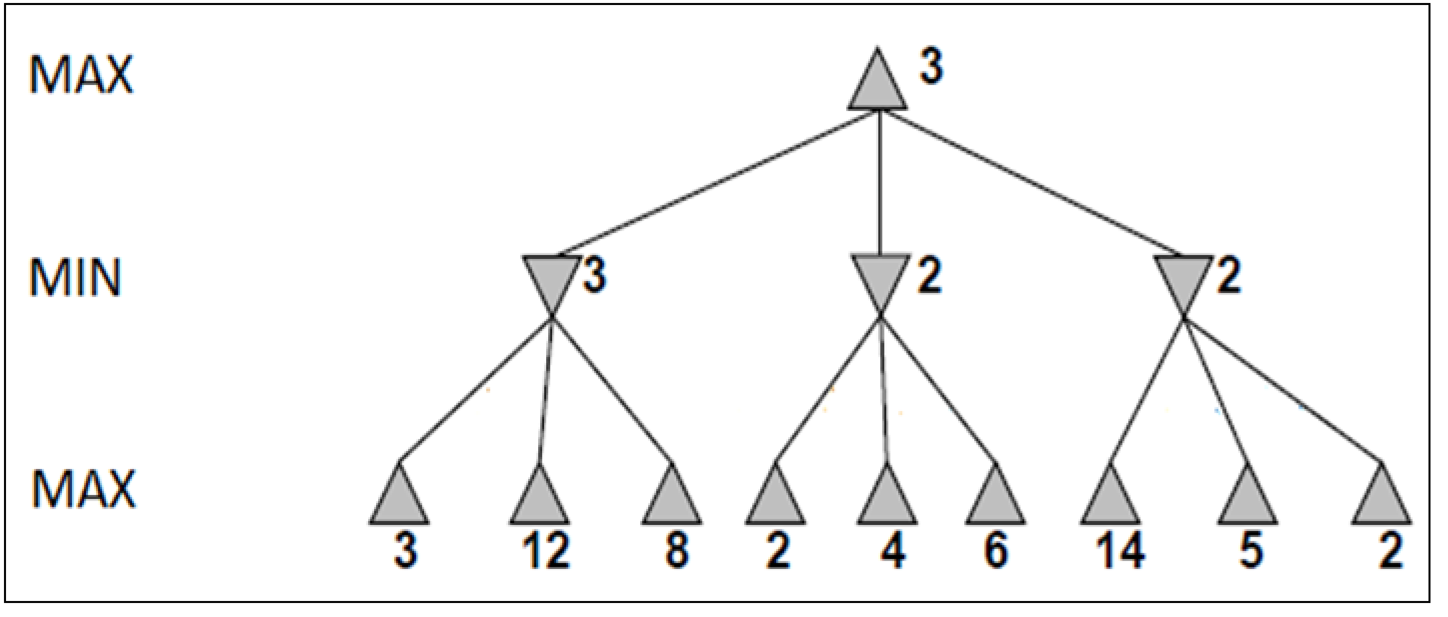
ca. 10^{123} games

(average) branching factor:

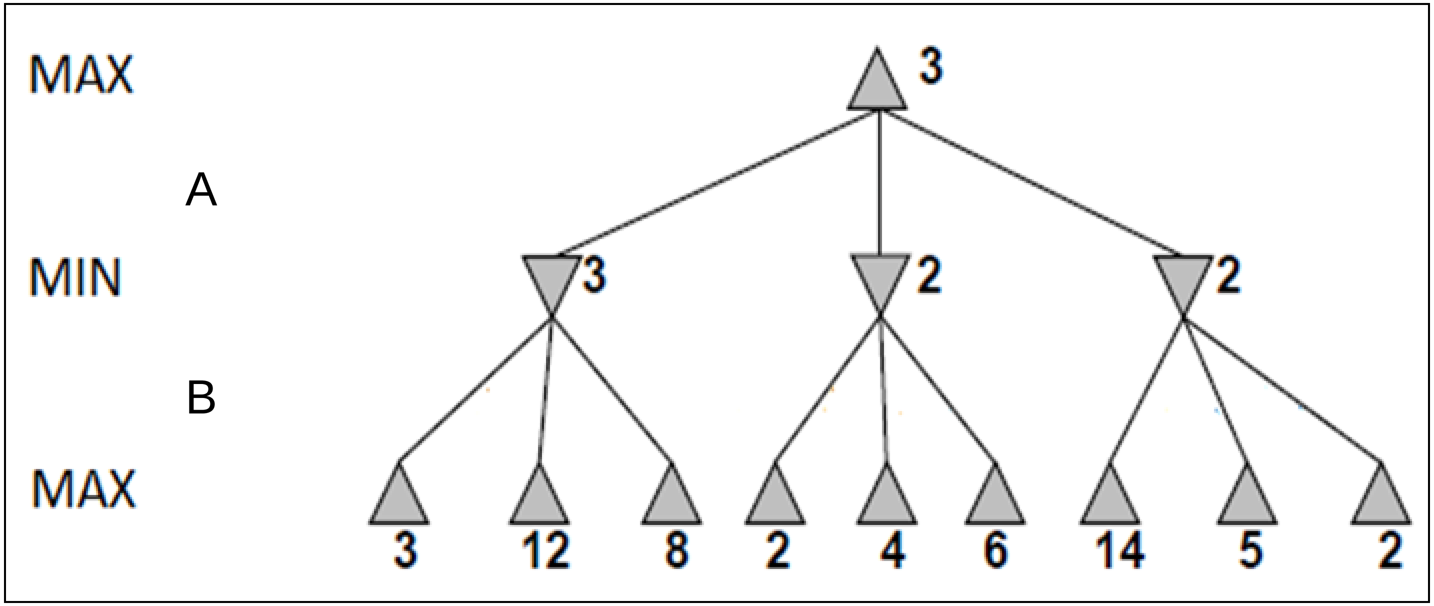
35

Optimal decision strategy - Minimax

- find the optimal strategy for Max assuming an infallible Min
- assumption: Both players play optimally!
- the optimal strategy can be determined by using the **minimax value of each node** (Zermelo 1912)



Optimal decision strategy - Minimax



$$\text{MINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in Actions(s)} \text{MINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \end{cases}$$

Optimal decision strategy - Minimax

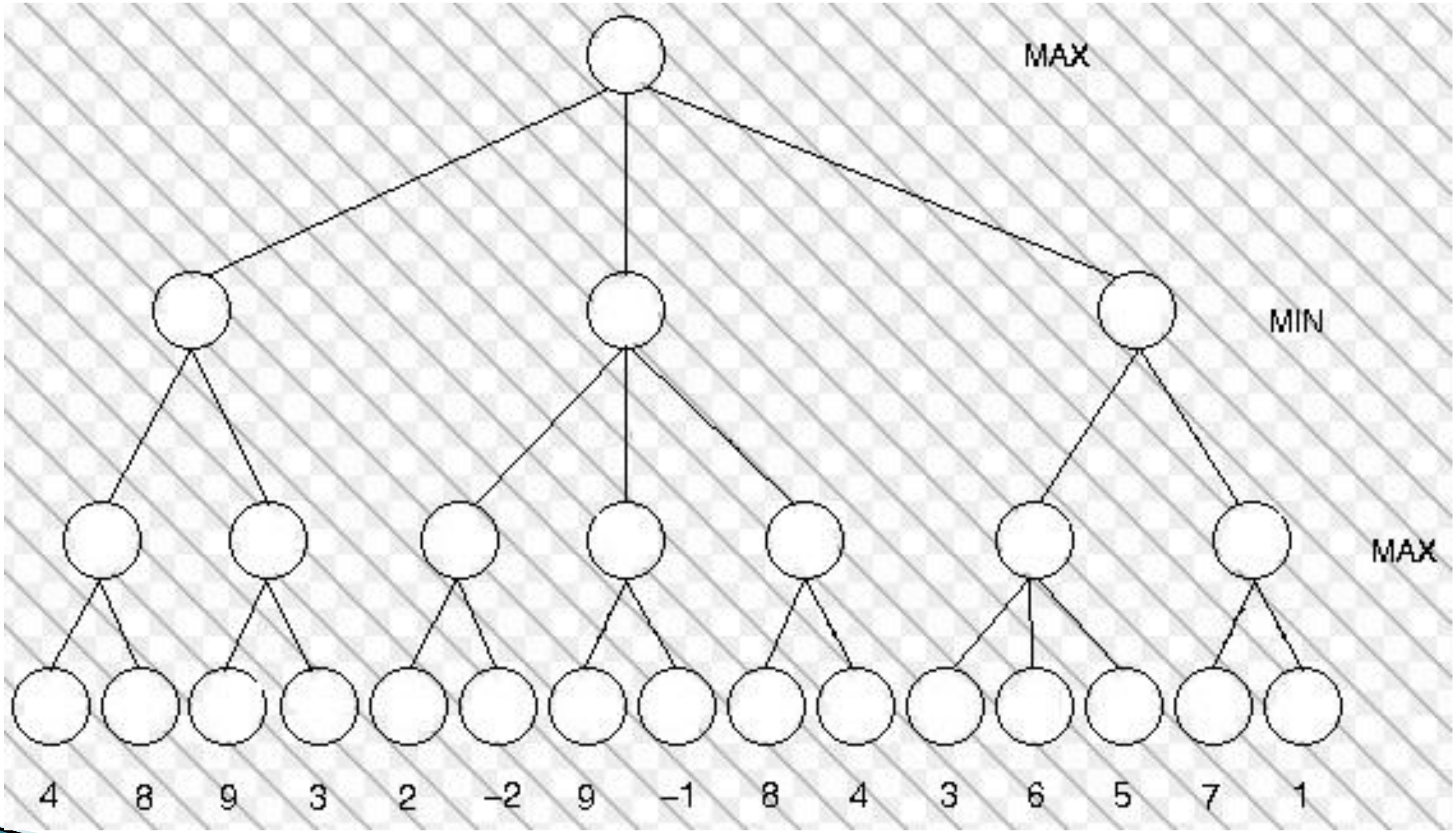
Minimax Algorithm
Complete depth-first exploration of the game tree

Complexity:	Time $O(b^d)$
	Space $O(bd)$
Complete?	Yes, if the tree is finite
Optimal?	Yes, against an optimal adversary

Otherwise?

(Chess: $b \approx 35$, $d \approx 80$? $O(10^{123})$ different games)
(really we need to search the whole tree?)

Minimax - example



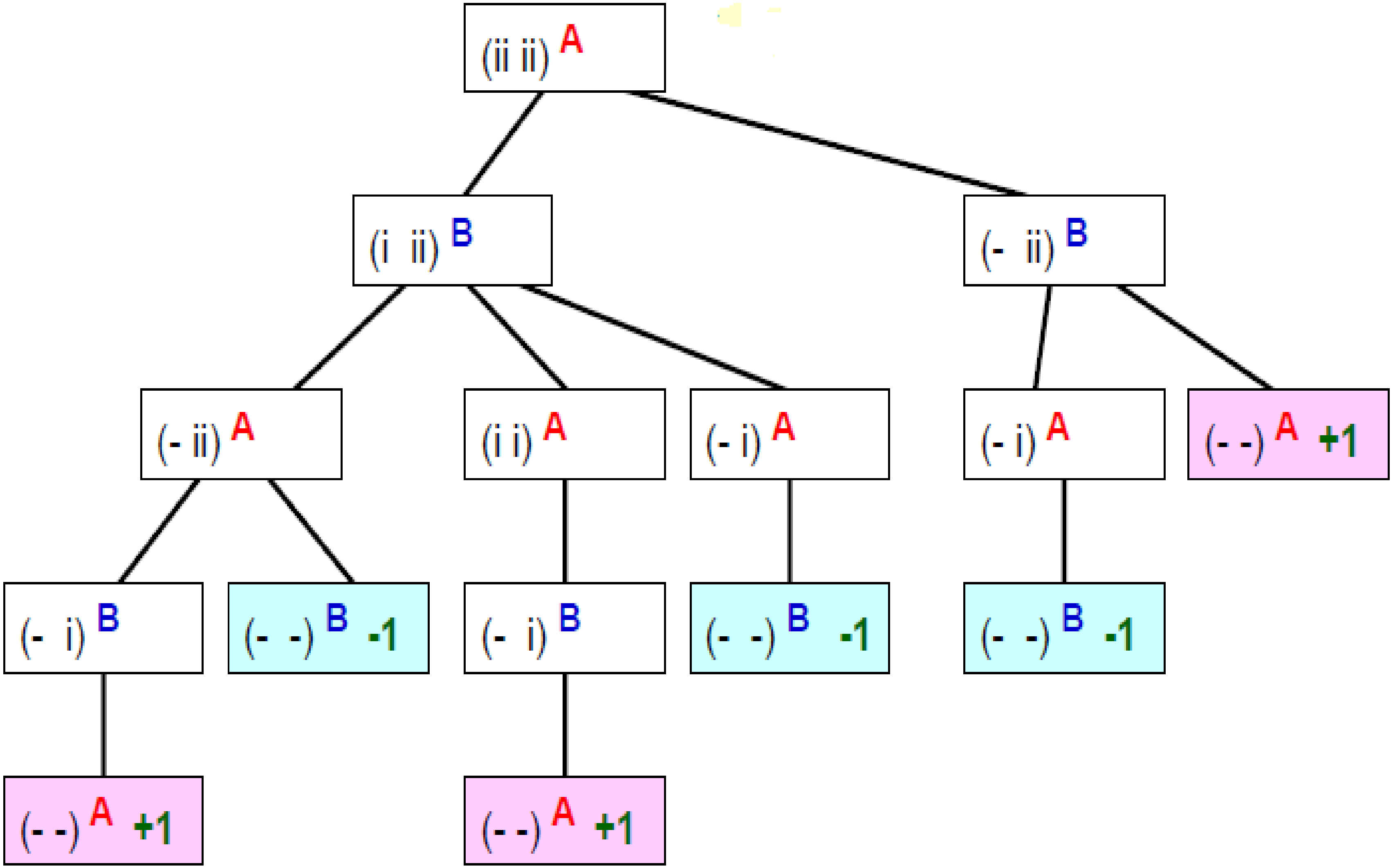
Nim

1. Two players take turns removing objects from distinct heaps or piles.
2. On each turn, a player must remove at least one object, and may remove any number of objects provided they all come from the same heap/pile
3. The goal of the game is to avoid taking the last object.

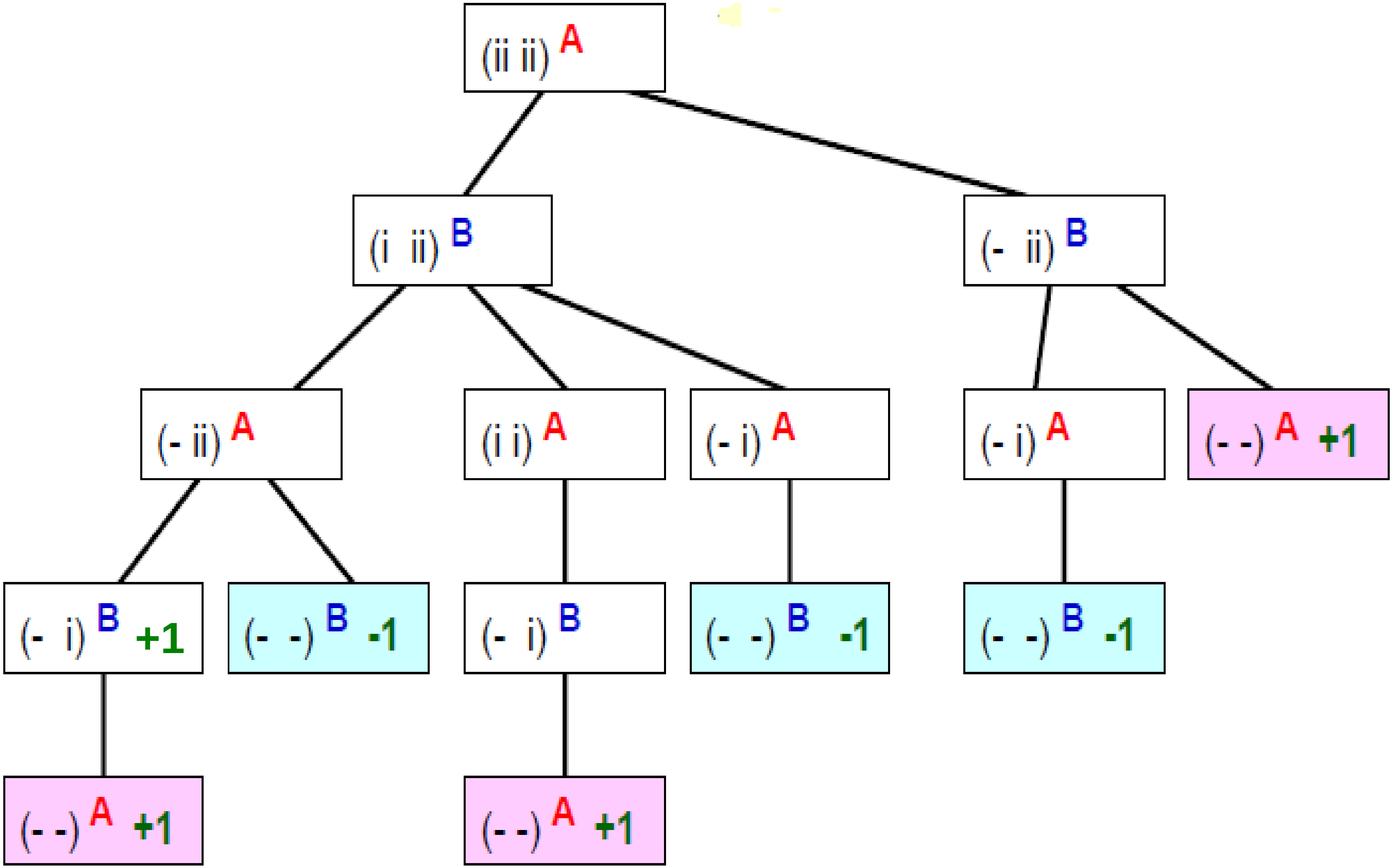
II-Nim

S =	a finite set of states (note: state includes information sufficient to deduce who is due to move)	(_ , _)-A (_ , i)-A (_ , ii)-A (i , i)-A (i , ii)-A (ii , ii)-A (_ , _)-B (_ , i)-B (_ , ii)-B (i , i)-B (i , ii)-B (ii , ii)-B		
I =	the initial state	(ii , ii)-A		
Succs =	a function which takes a state as input and returns a set of possible next states available to whoever is due to move	Succs(_,i)-A = { (_,_)-B }	Succs(_,i)-B = { (_,_)-A }
		Succs(_,ii)-A = { (_,_)-B , (_,i)-B }	Succs(_,ii)-B = { (_,_)-A , (_,i)-A }
		Succs(i,i)-A = { (_,i)-B }	Succs(i,i)-B = { (_,i)-A }
		Succs(i,ii)-A = { (_,i)-B (_,ii)-B (i,i)-B}	Succs(i,ii)-B = { (_,i)-A , (_,ii)-A (i,i)-A }
		Succs(ii,ii)-A = { (_,ii)-B , (i,ii)-B }	Succs(ii,ii)-B = { (_,ii)-A , (i,ii)-A }
T =	a subset of S. It is the terminal states	(_ , _)-A	(_ , _)-B
V =	Maps from terminal states to real numbers. It is the amount that A wins from B.	V(_ , _)-A = +1	V(_ , _)-B = -1

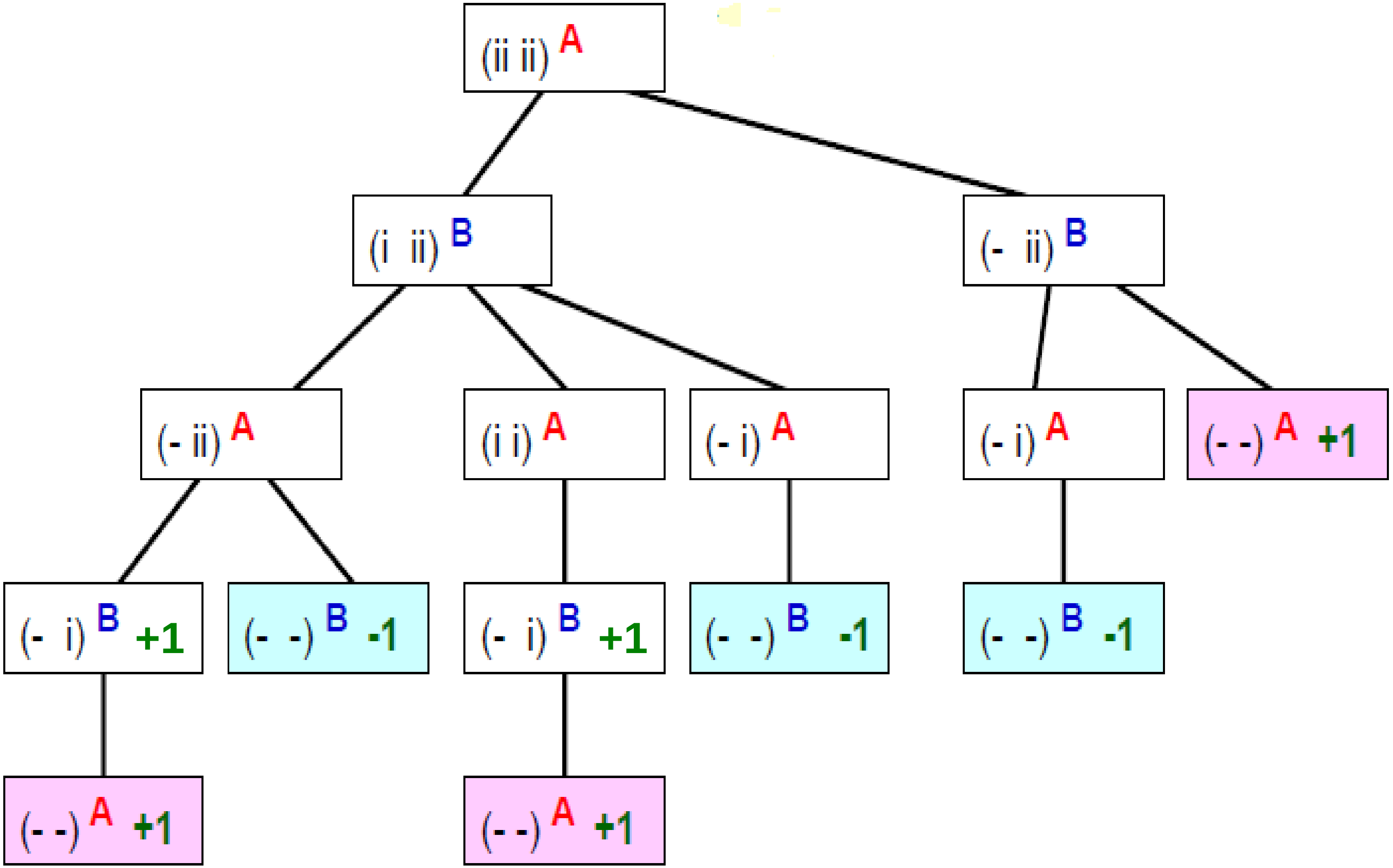
Nim



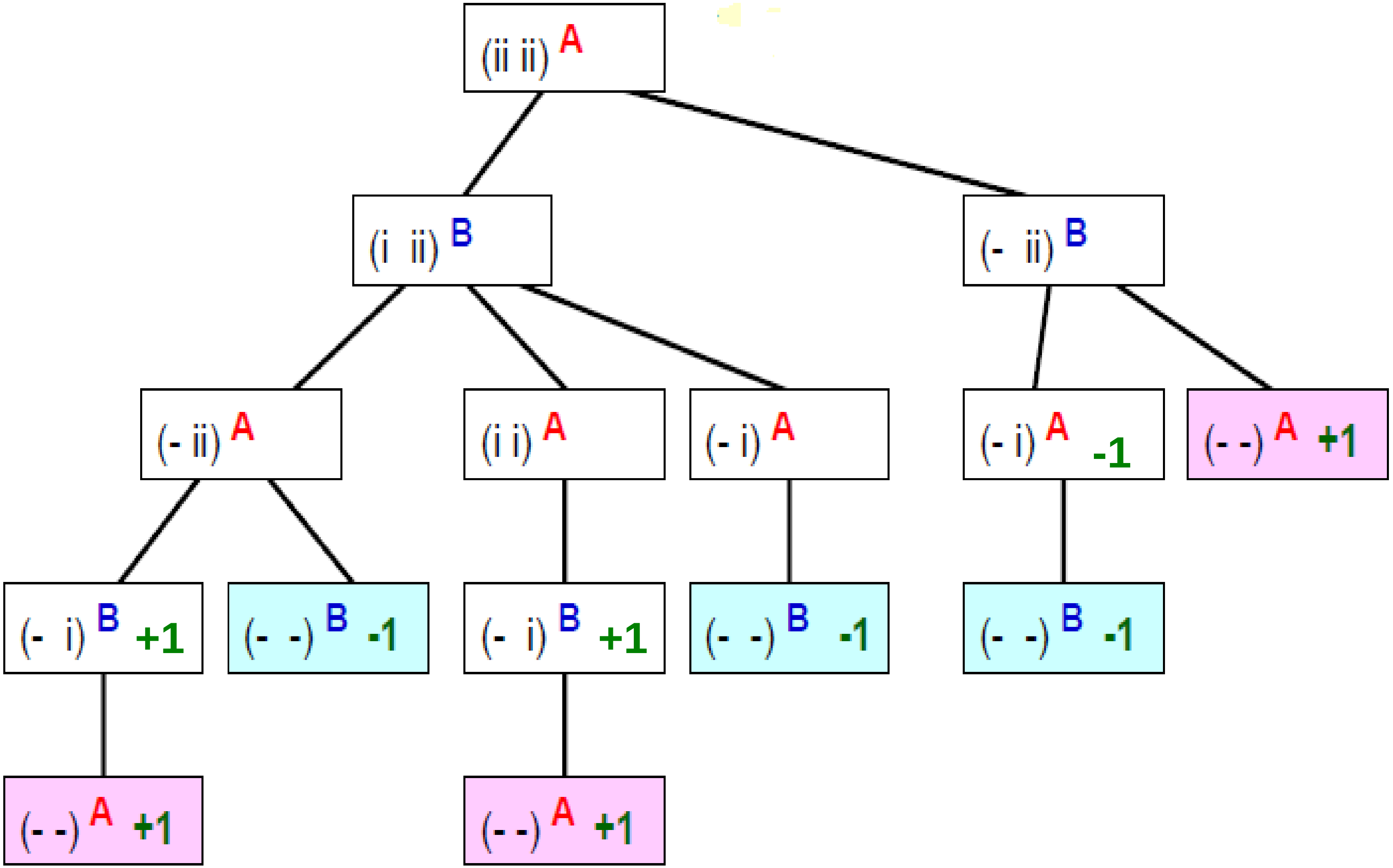
Nim



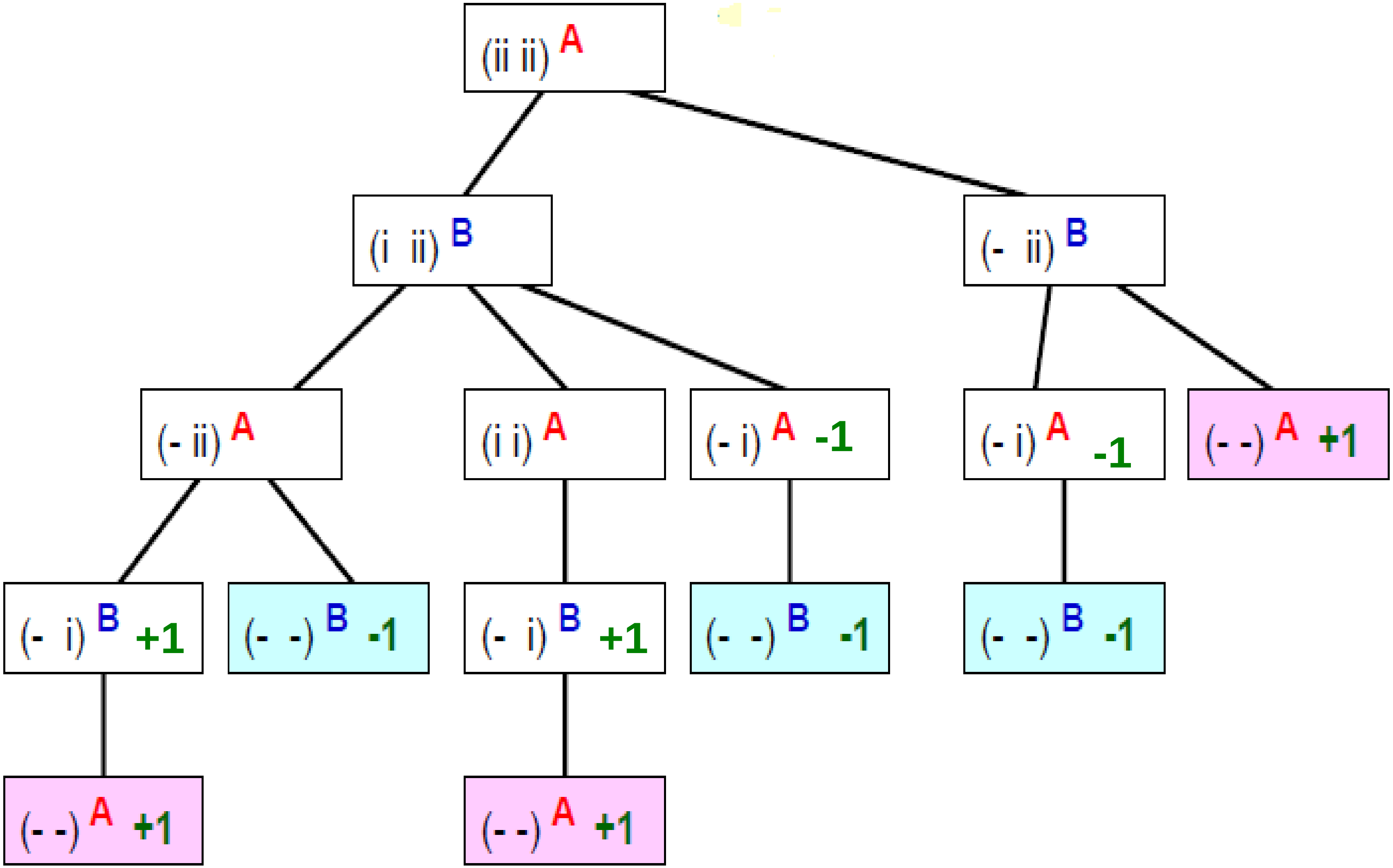
Nim



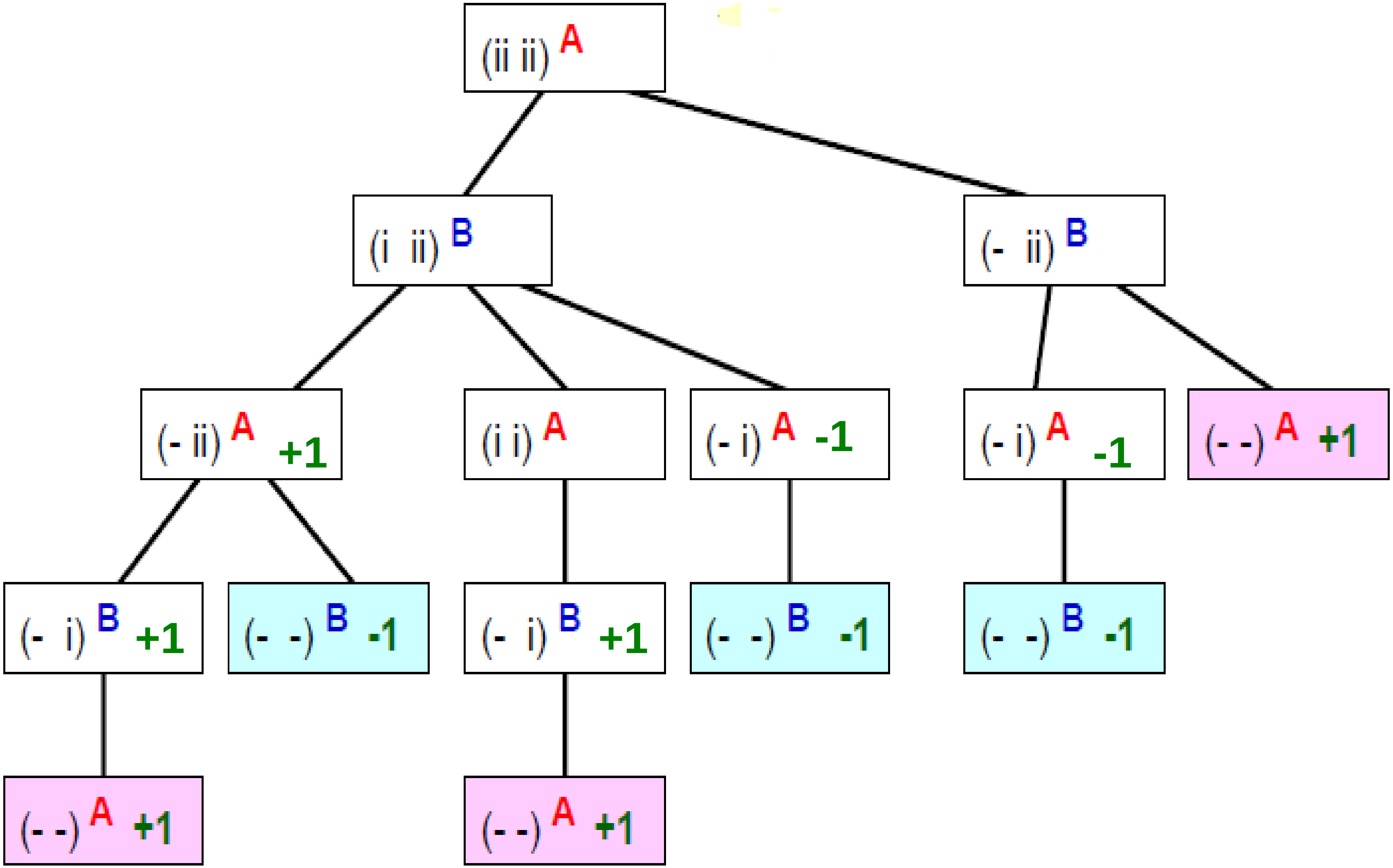
Nim



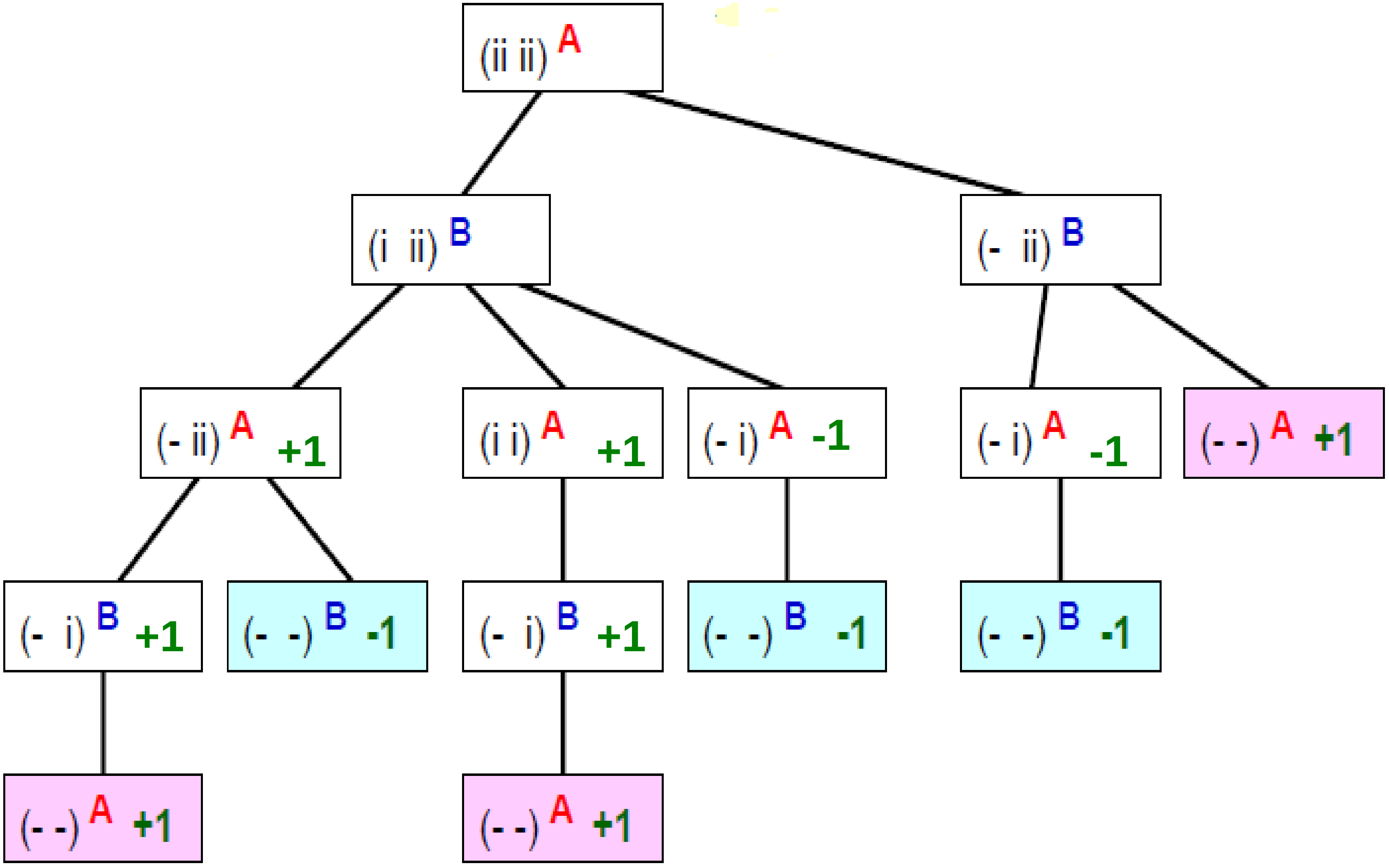
Nim



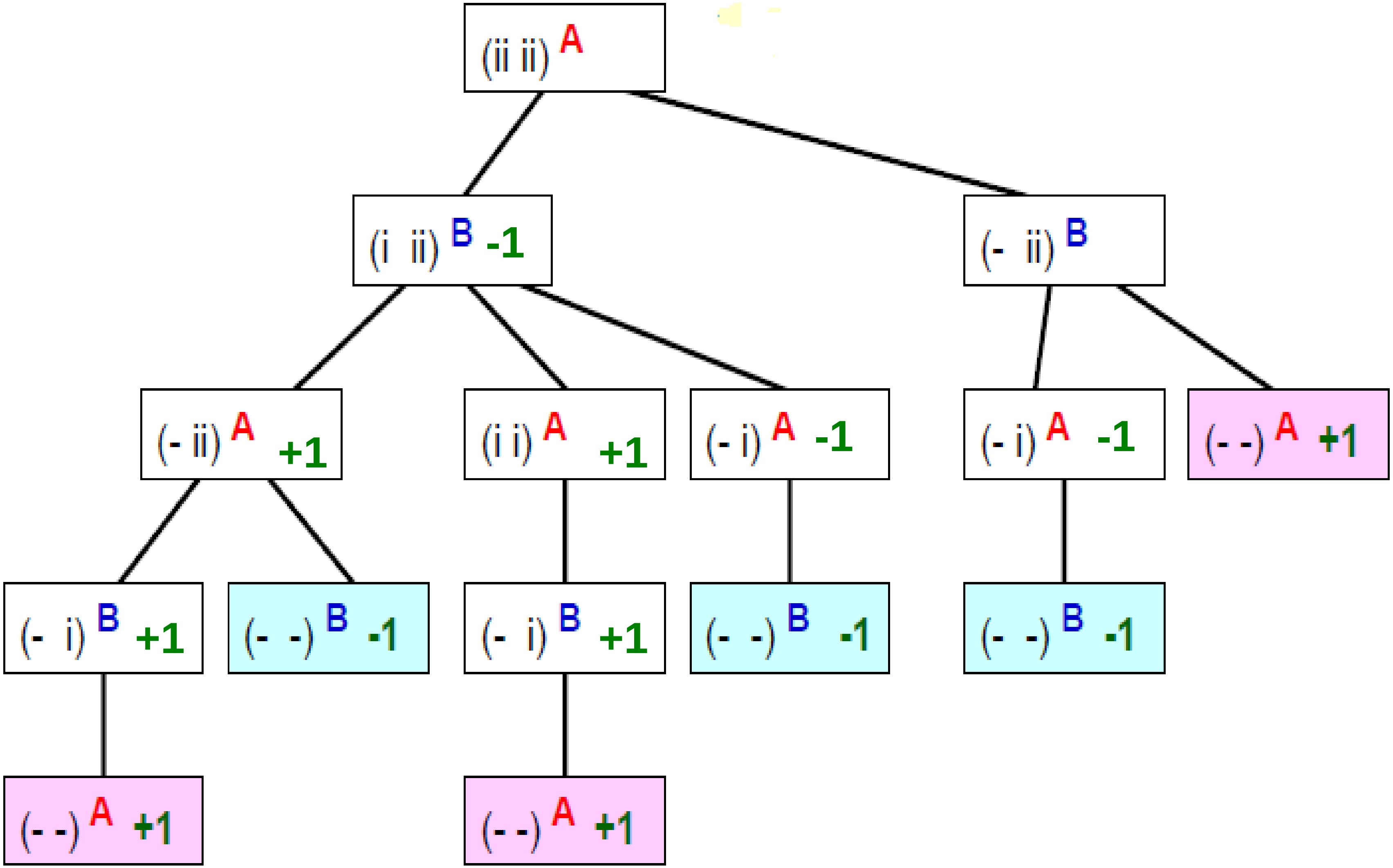
Nim



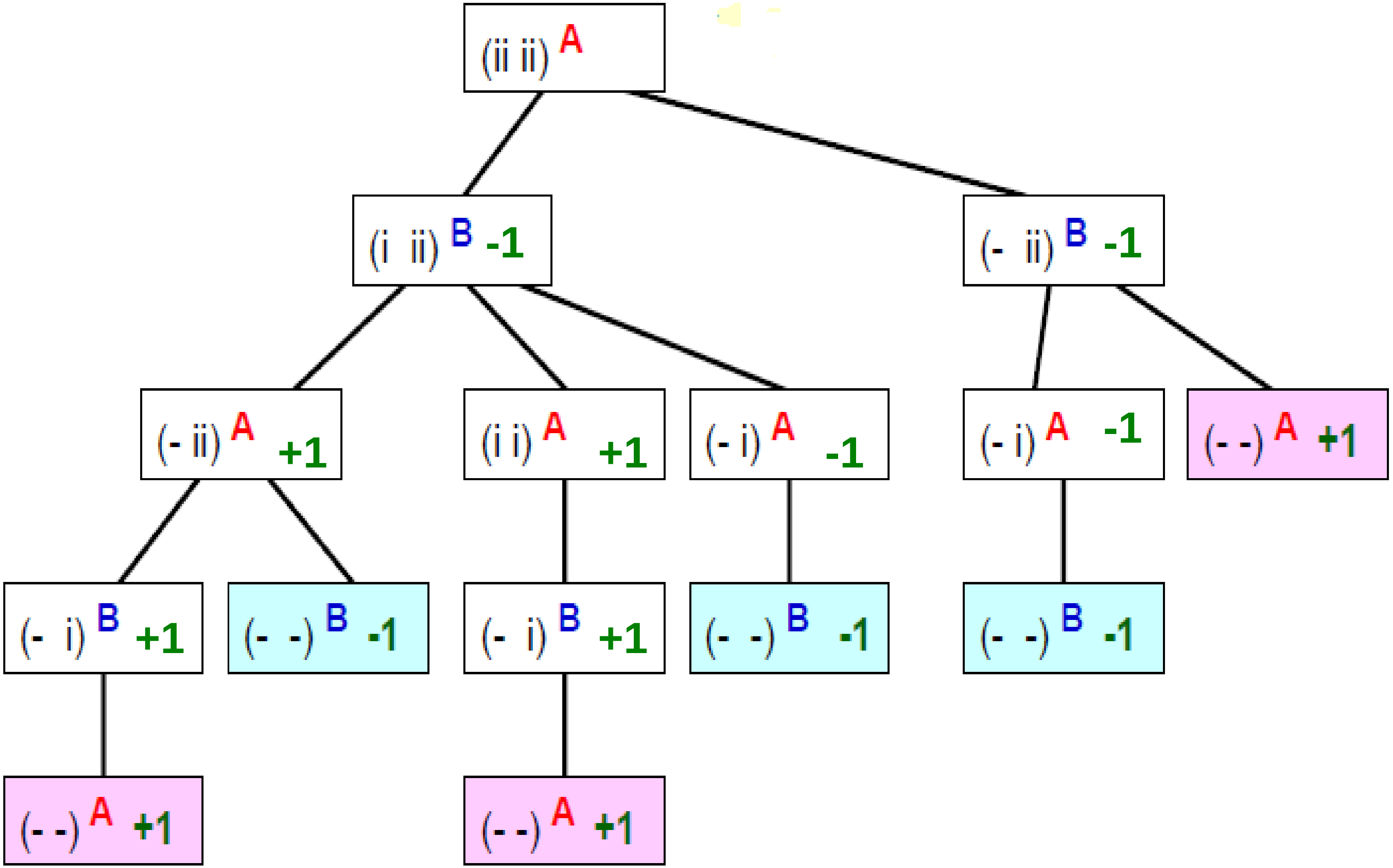
Nim



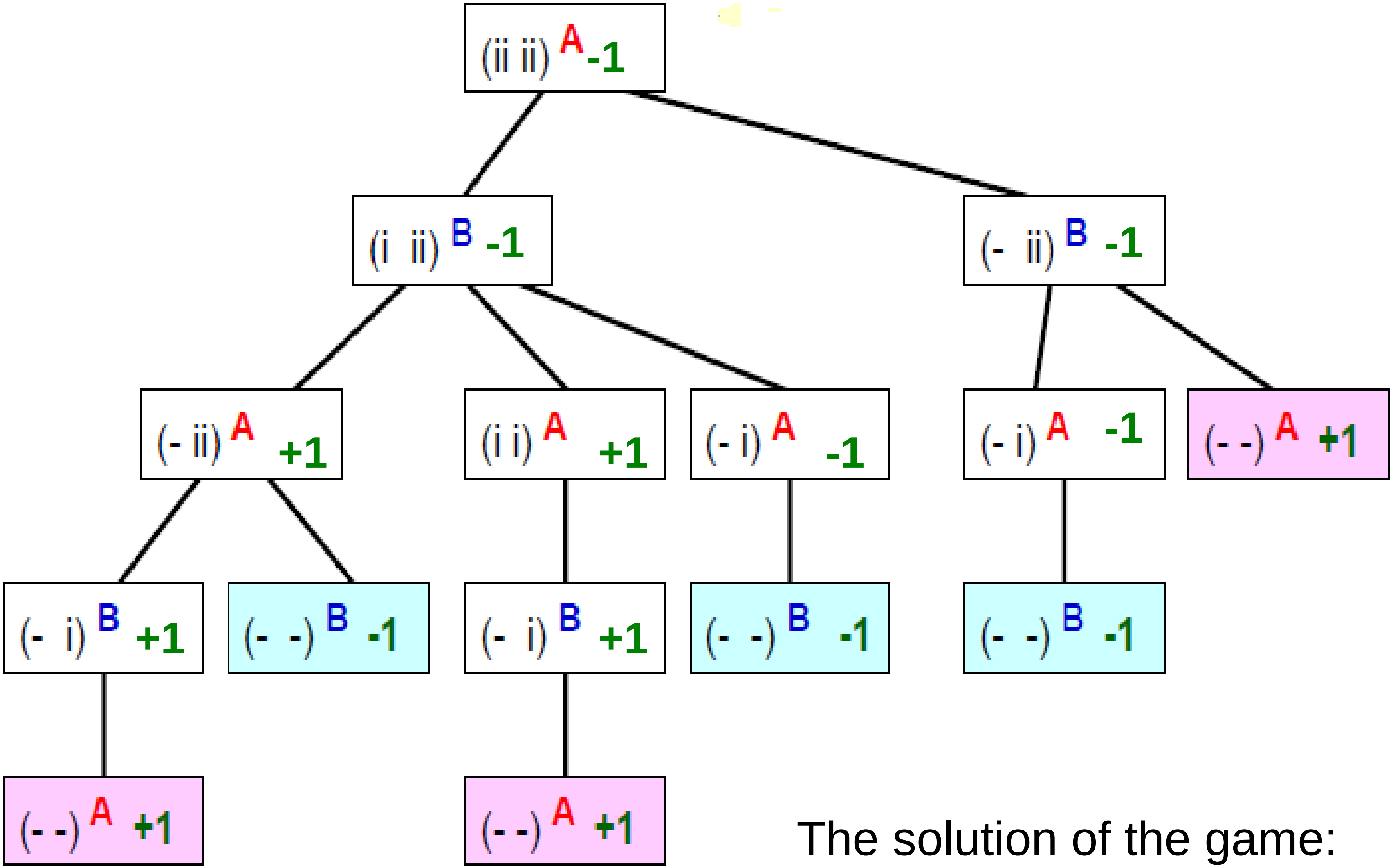
Nim



Nim

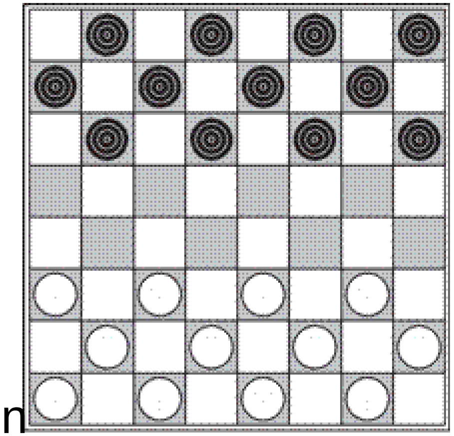


Nim



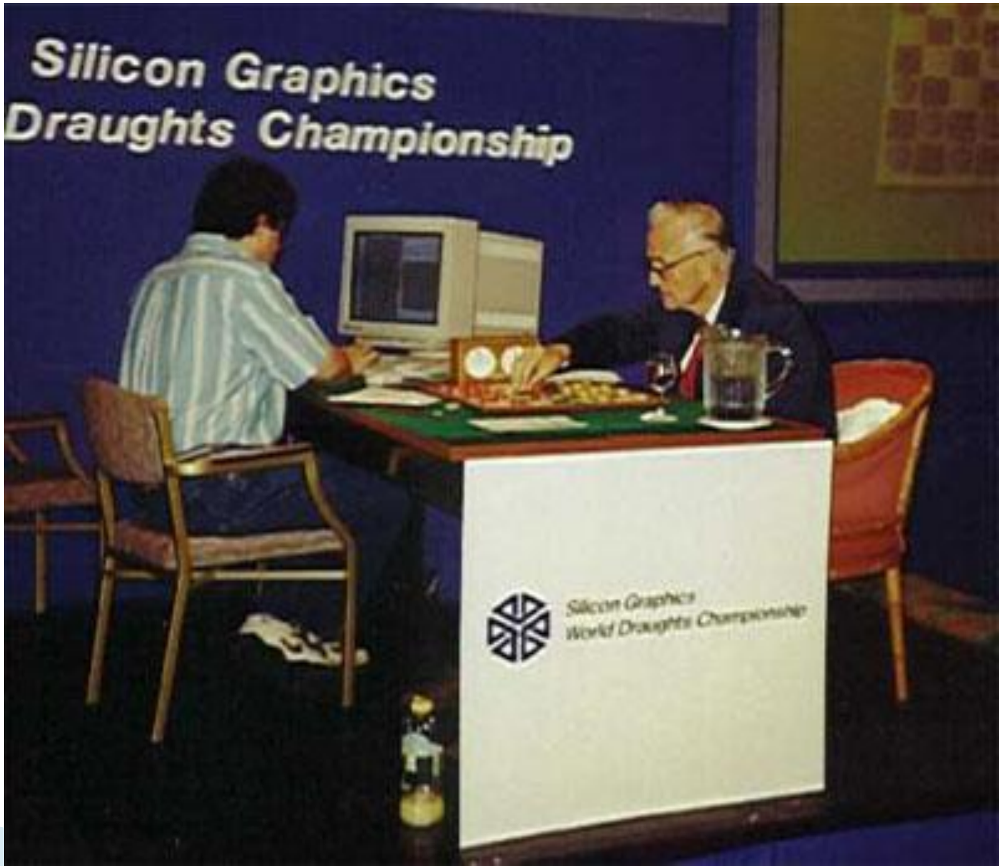
The solution of the game: beginner loses, if adversary plays optimally.

Checkers solved, **optimal game is a draw**!
Search space of checkers ca. 5 x 10^{20} board position



History

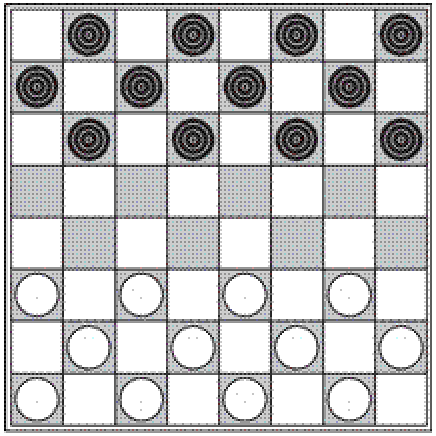
1950 – self-learning program of Arthur Samuel
1963 – one win against a talented human
1989 - Chinook project, to win against the human world champion
1990 - Chinook aproved to take part in the World Championship
1992 - Marion Tinsley, world champion, has difficulties in the match for the title, but finally wins
1994 – return match, Tinsley stands down due to health conditions, dies shorty after
1996 - Chinook stronger, than every human player



History of computing

1989 – 2007
1992 peak: more than 200 processors
2006-2007 50 processor, on average

The longest distributed, continuously run computation until recently

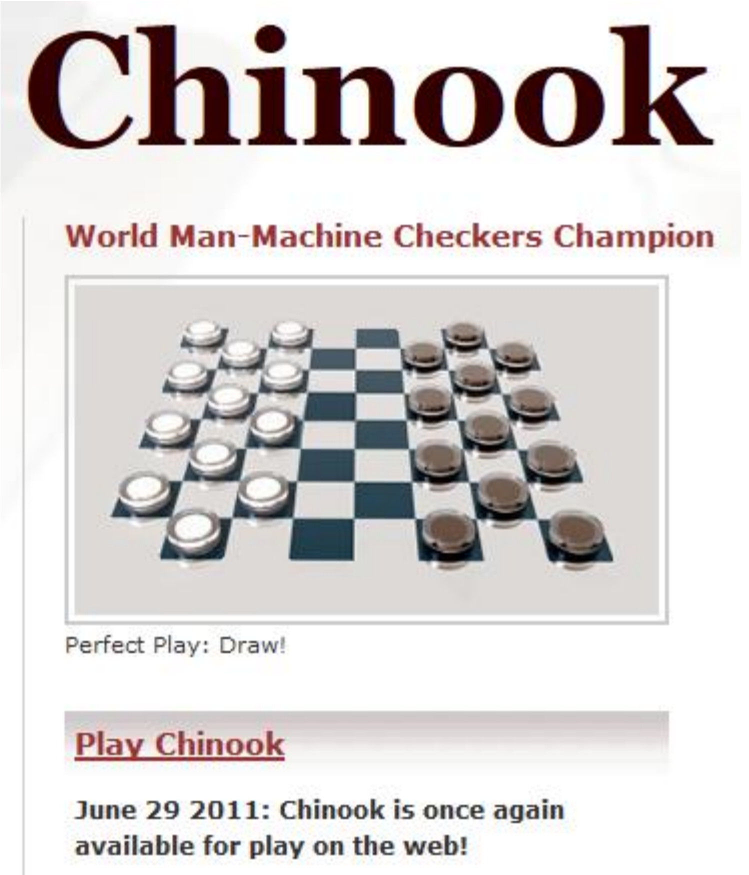


Fault tolerance of the computations and computer systems

Errors due to frequent moving of very large datafiles to local disks, to other computers on the network (duplication, control)

Periodic control of the already computed data bases due to the data loss (“bit rot”), (copy, safe, recomputing)

Disk manufacturer warranty – 10^{-13} error rate
The computations were more complicated

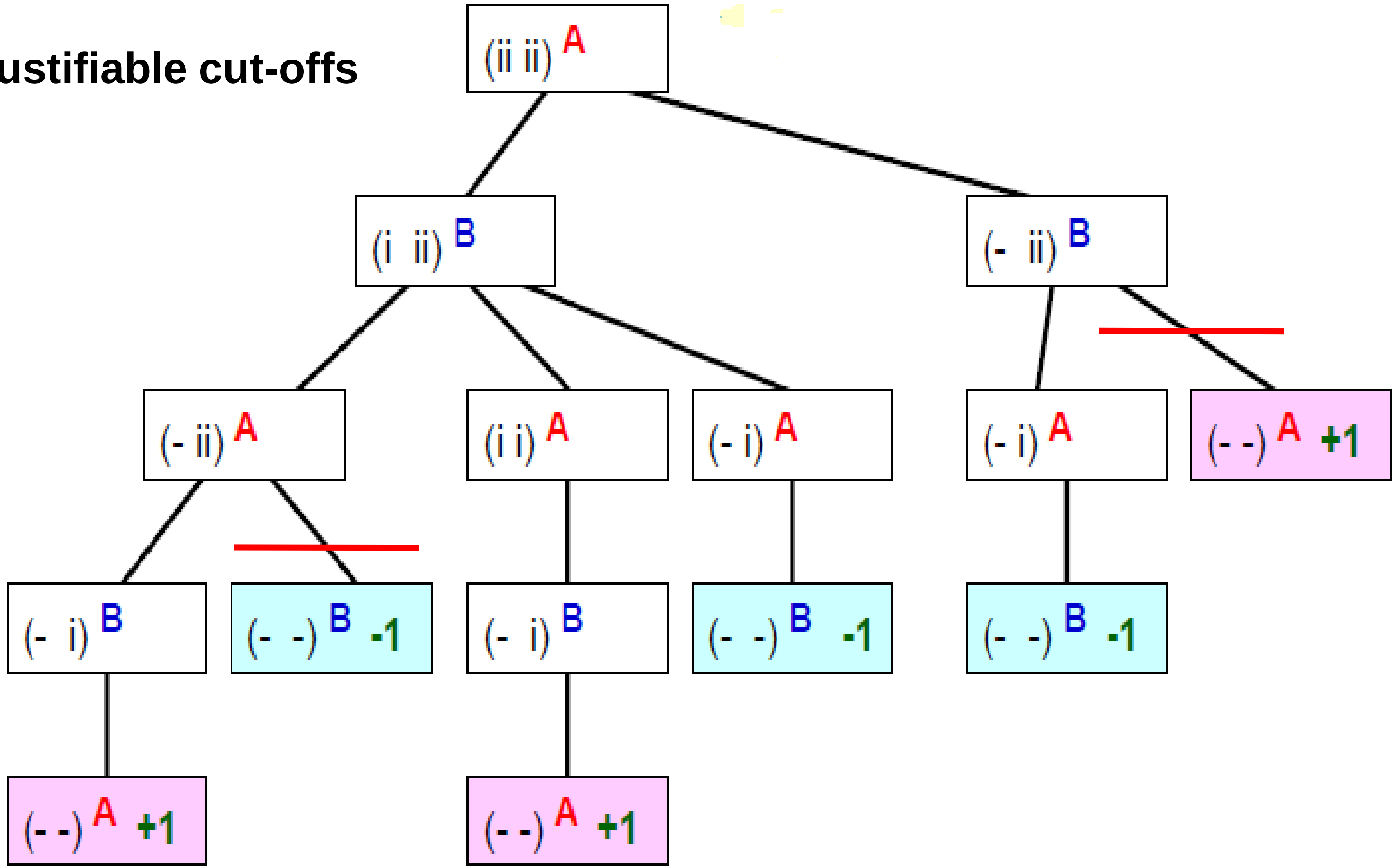


http://webdocs.cs.ualberta.ca/~chinook/

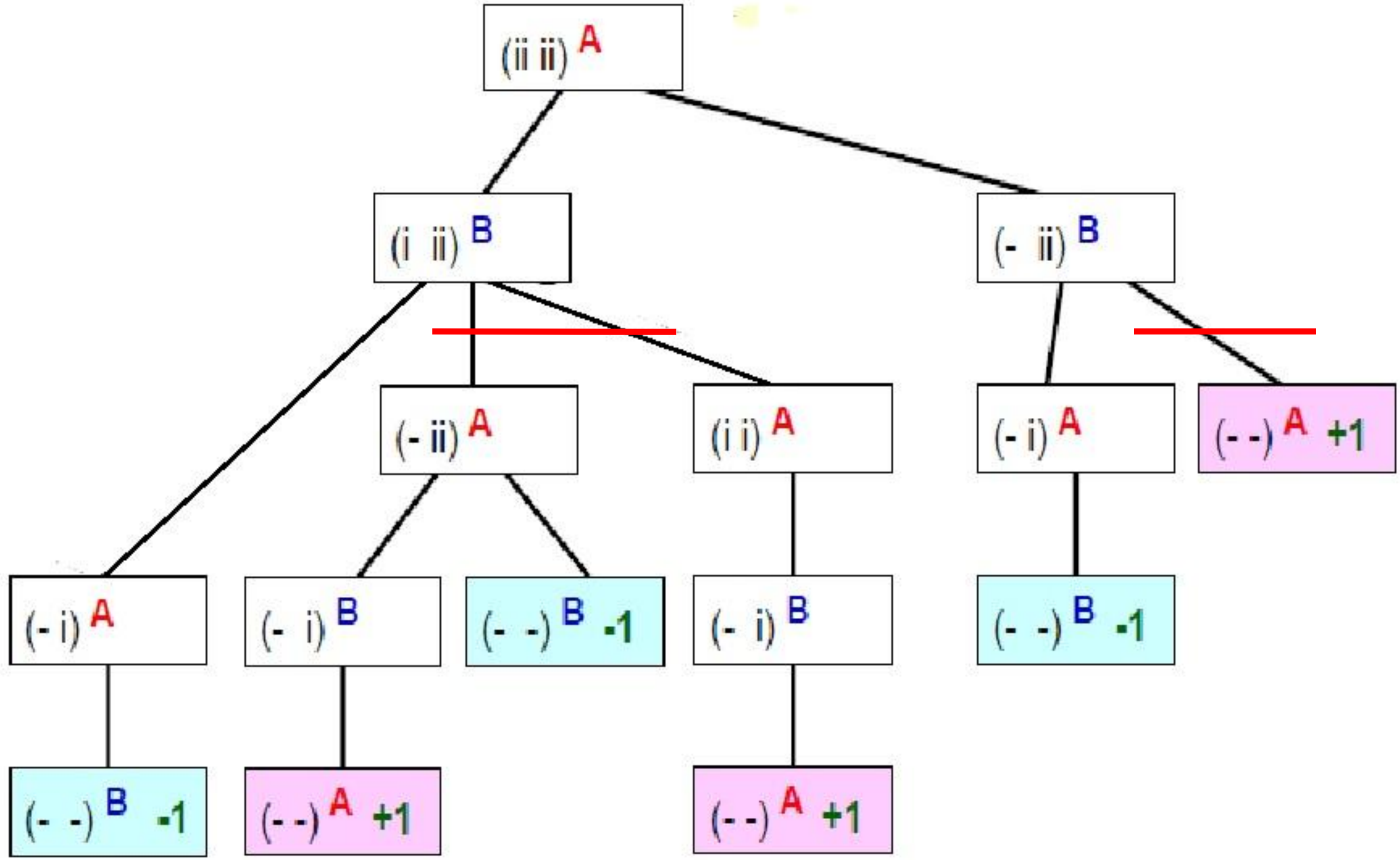
Minimax reconsidered

Assume, that the outcome of the game is only -1 és 1.
Can we spare some computations? And if real?

Justifiable cut-offs



Minimax reconsidered



Minimax reconsidered

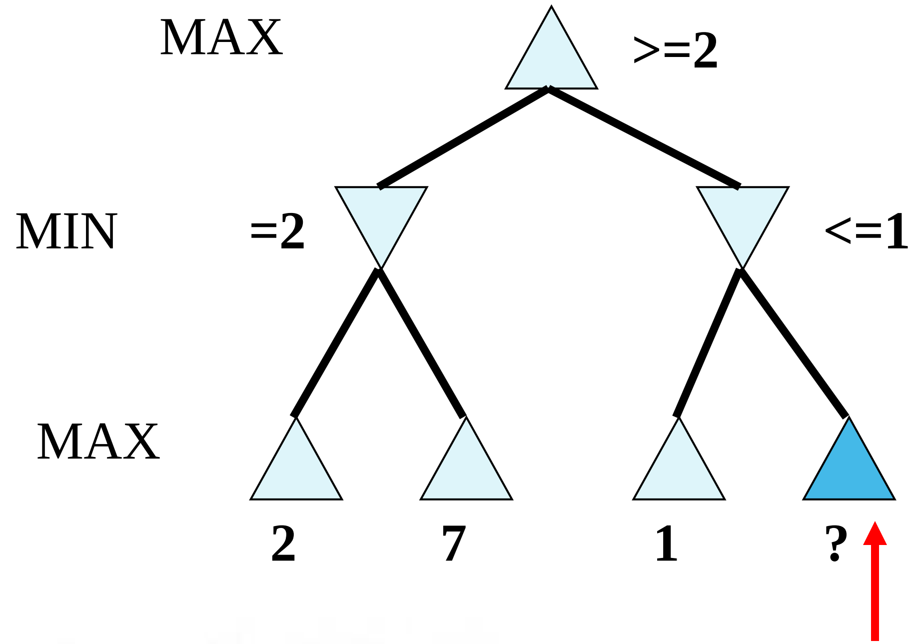
Number of game states is exponential in the number of moves.
Solution: Do not examine every node
=> pruning:

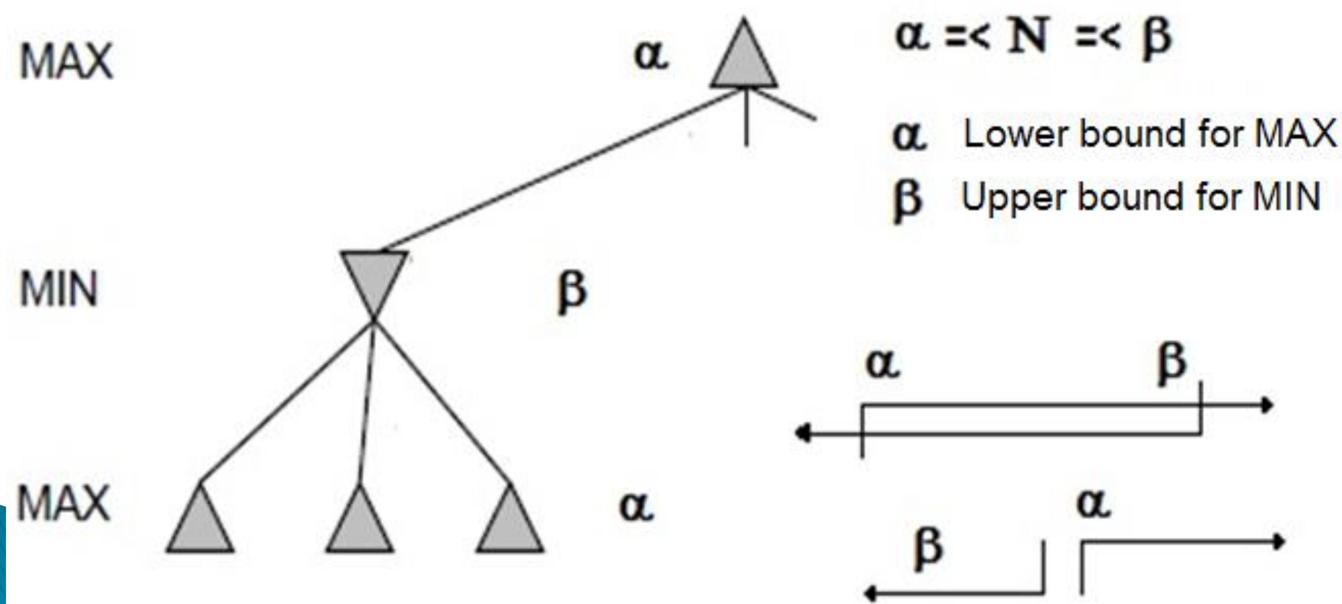
remove branches that do not influence final decision

Alpha-beta prunning

Depth first search – only nodes along a single path at any time
α = highest-value choice that we can guarantee for MAX so far in the current subtree.
β = lowest-value choice that we can guarantee for MIN so far in the current subtree.
Update values of α and β during search and prune remaining branches as soon as the value is known to be worse than the current α or β value for MAX or MIN.

Alpha-beta prunning





Alpha-beta prunning

Recall:

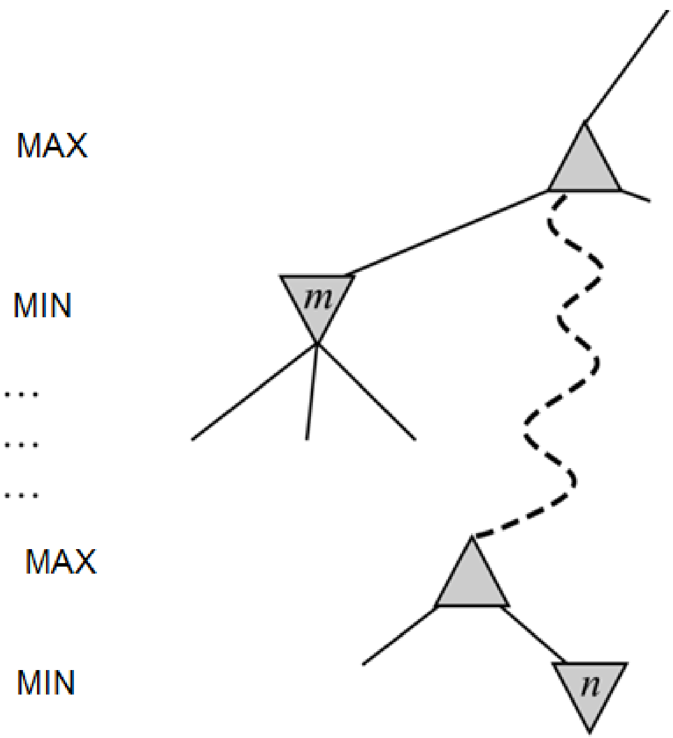
– **α:** value of best move for us seen so far in current search path

– **β:** best move for opponent (worst move for us) seen so far in current search path

If **α ≥ β,** prune

Initial α: −∞

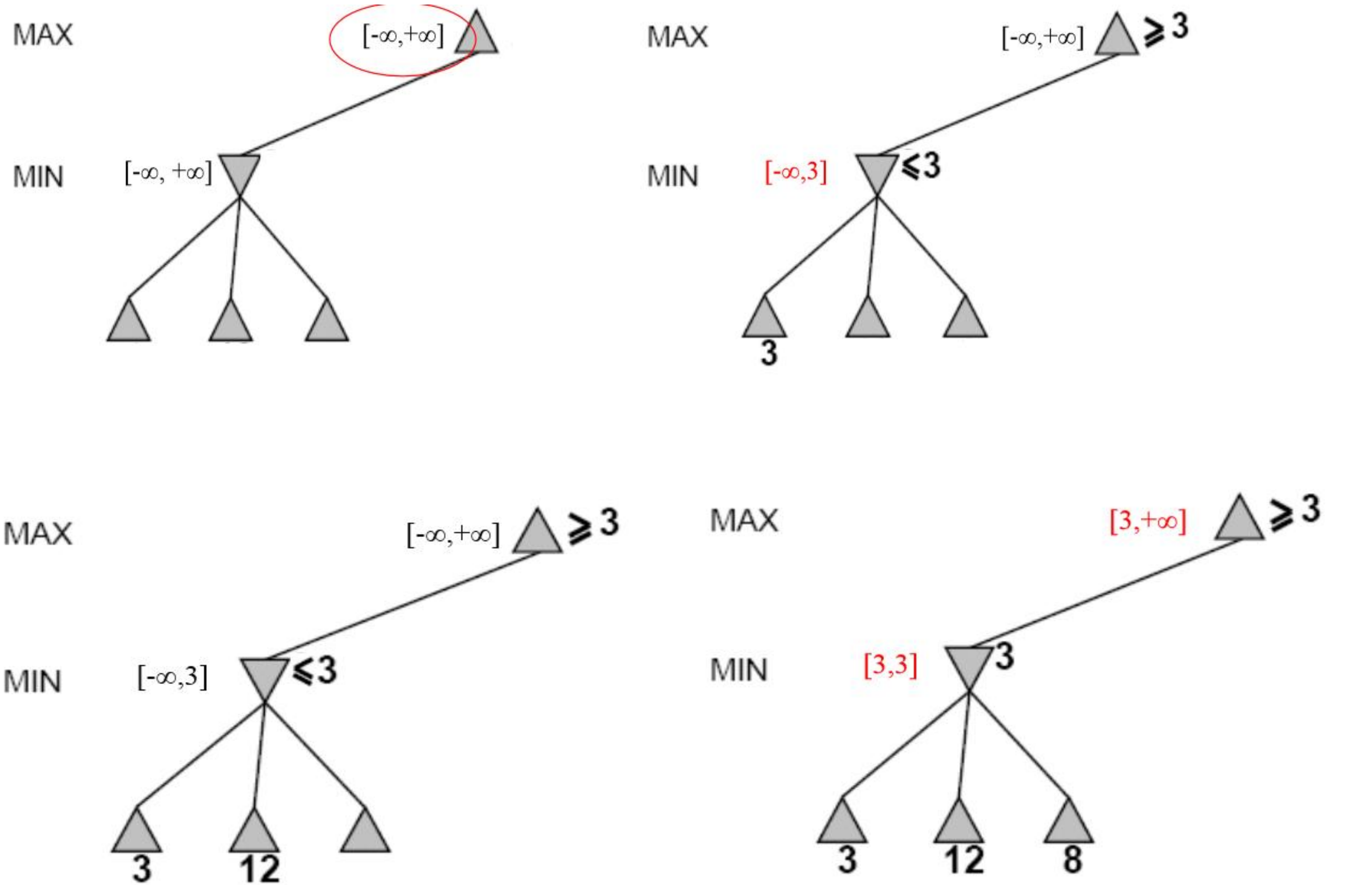
Initial β: ∞

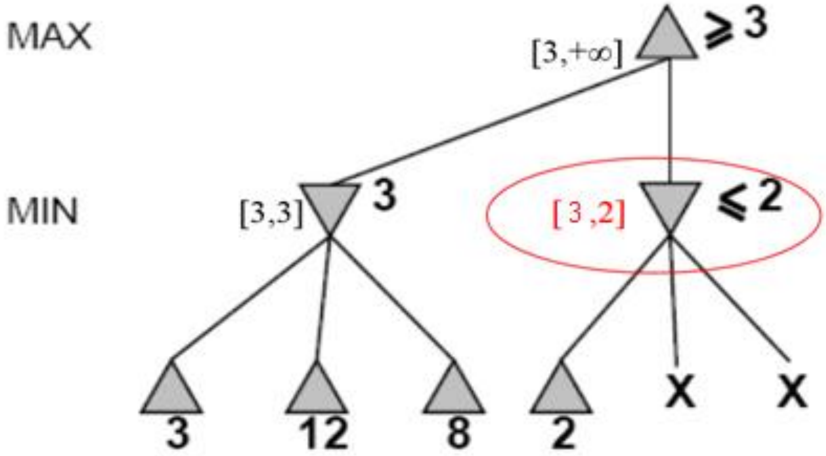


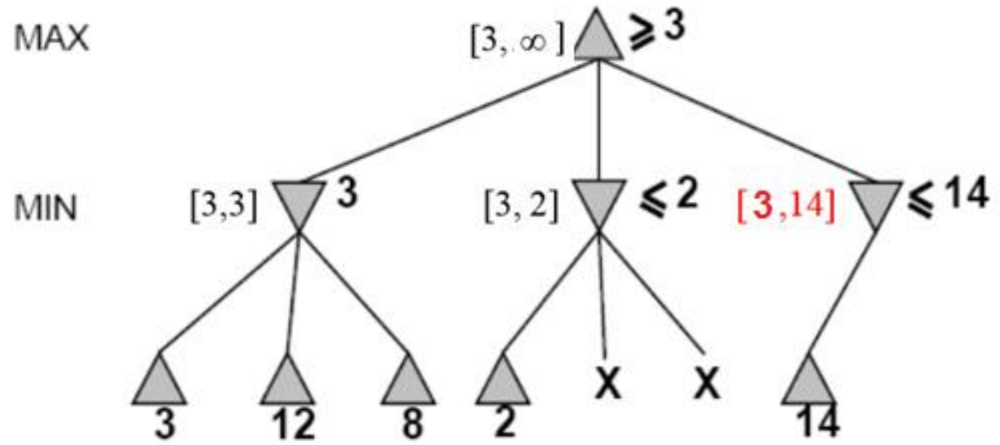
If *m* is better for MAX, than *n*, then we never get to *n* in the game.

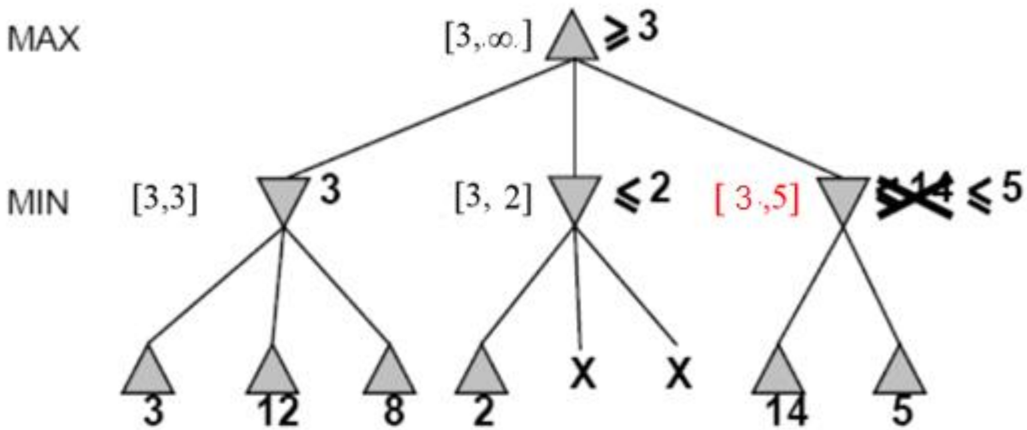
Prunning does not influence the outcome. Good step ordering increases the prunning efficiency.

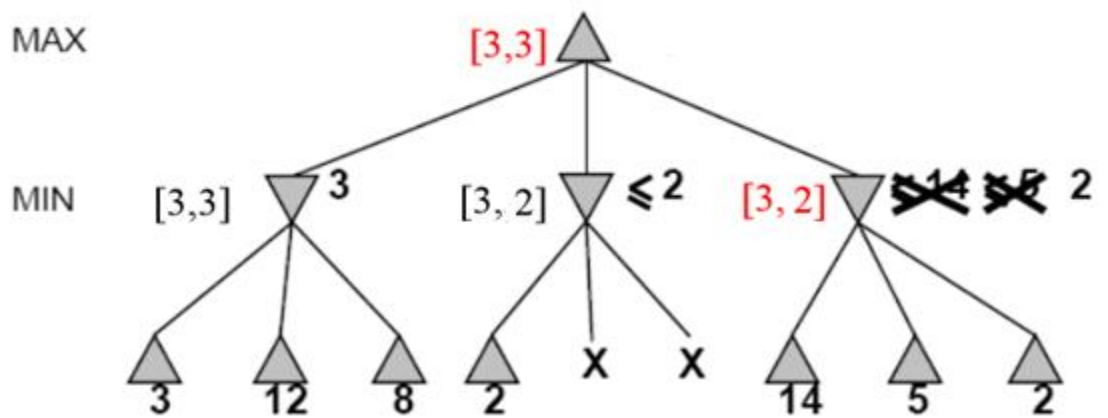
With „ideal ordering" the time complexity is $O(b^{d/2})$, $O(b^{3d/4})$ on the average (in Chess $O(b^{m/2}) = O((\sqrt{b})^m)$, the effective b is 6 instead of 35: $b = \sqrt{35} \approx 6$)


MAX
[-∞,+∞]
MIN
[-∞, +∞]
MAX
[-∞,+∞]
≥3
MIN
[-∞,3]
≤3
3
MAX
[-∞,+∞]
≥3
MIN
[-∞,3]
≤3
3
12
MAX
[3,+∞]
≥3
MIN
[3,3]
3
3
12
8



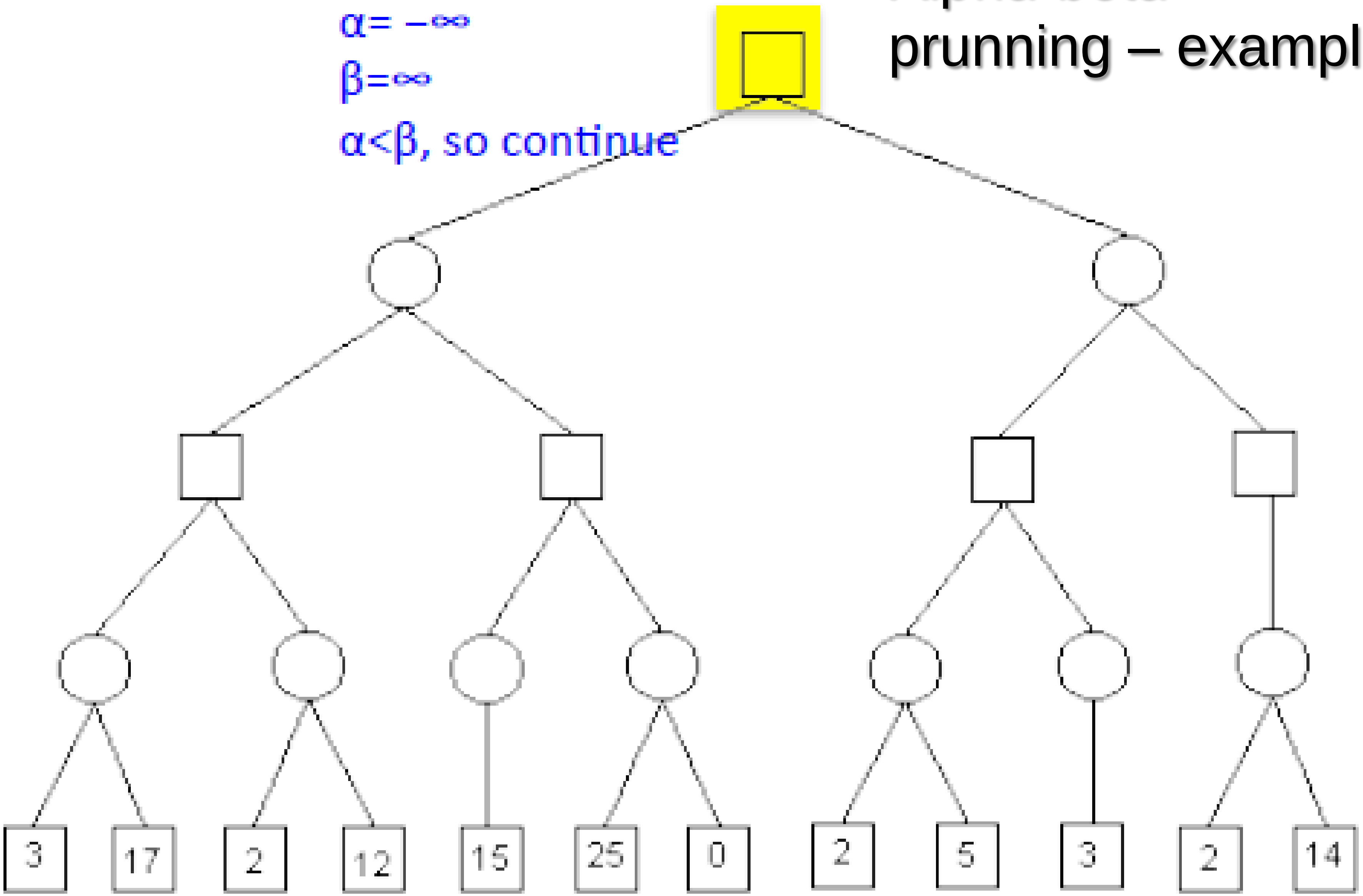






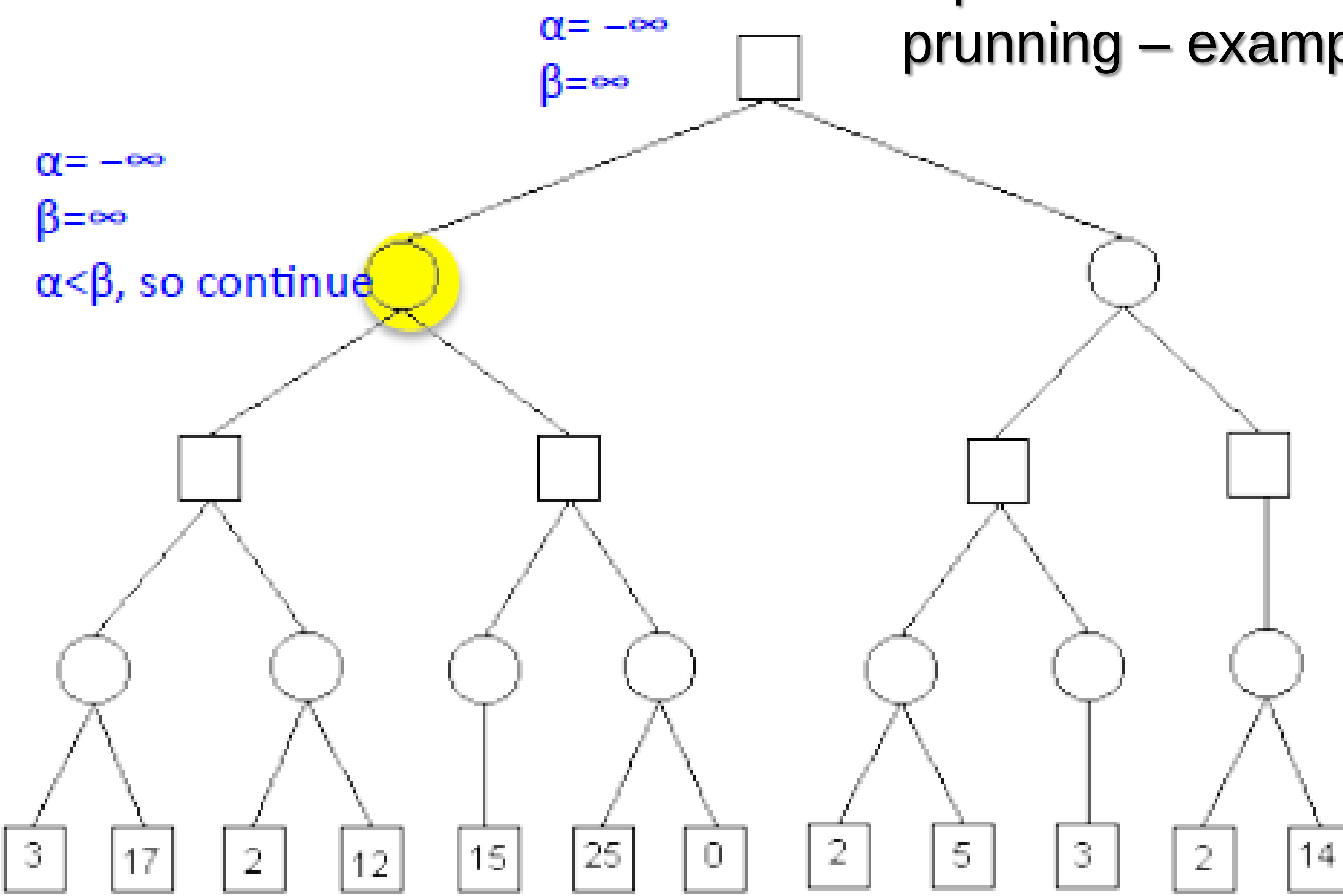
http://inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice/

Alpha-beta prunning – example

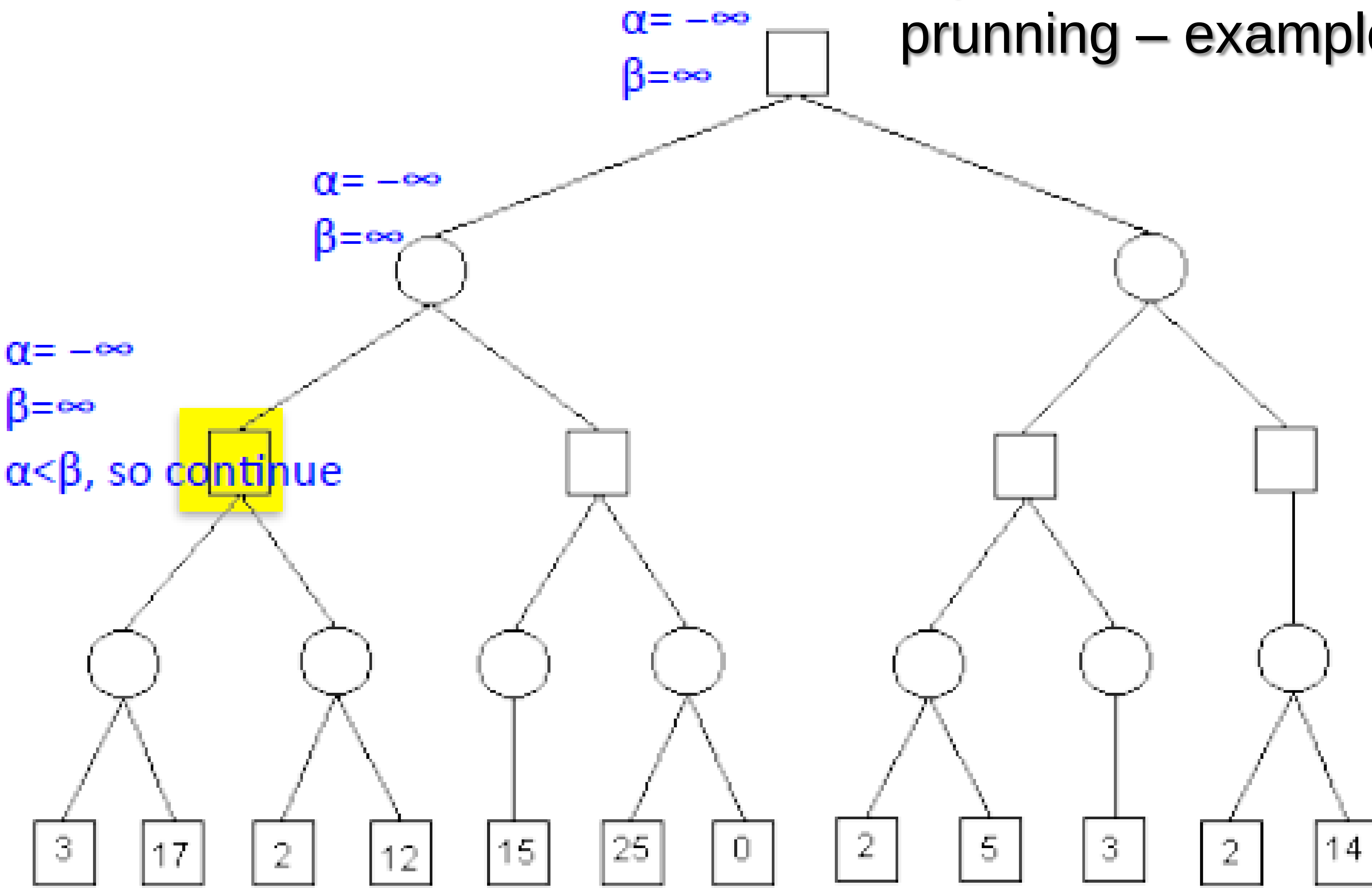


From http://www.cs.ucla.edu/~rosen/161/notes/alphabeta.html

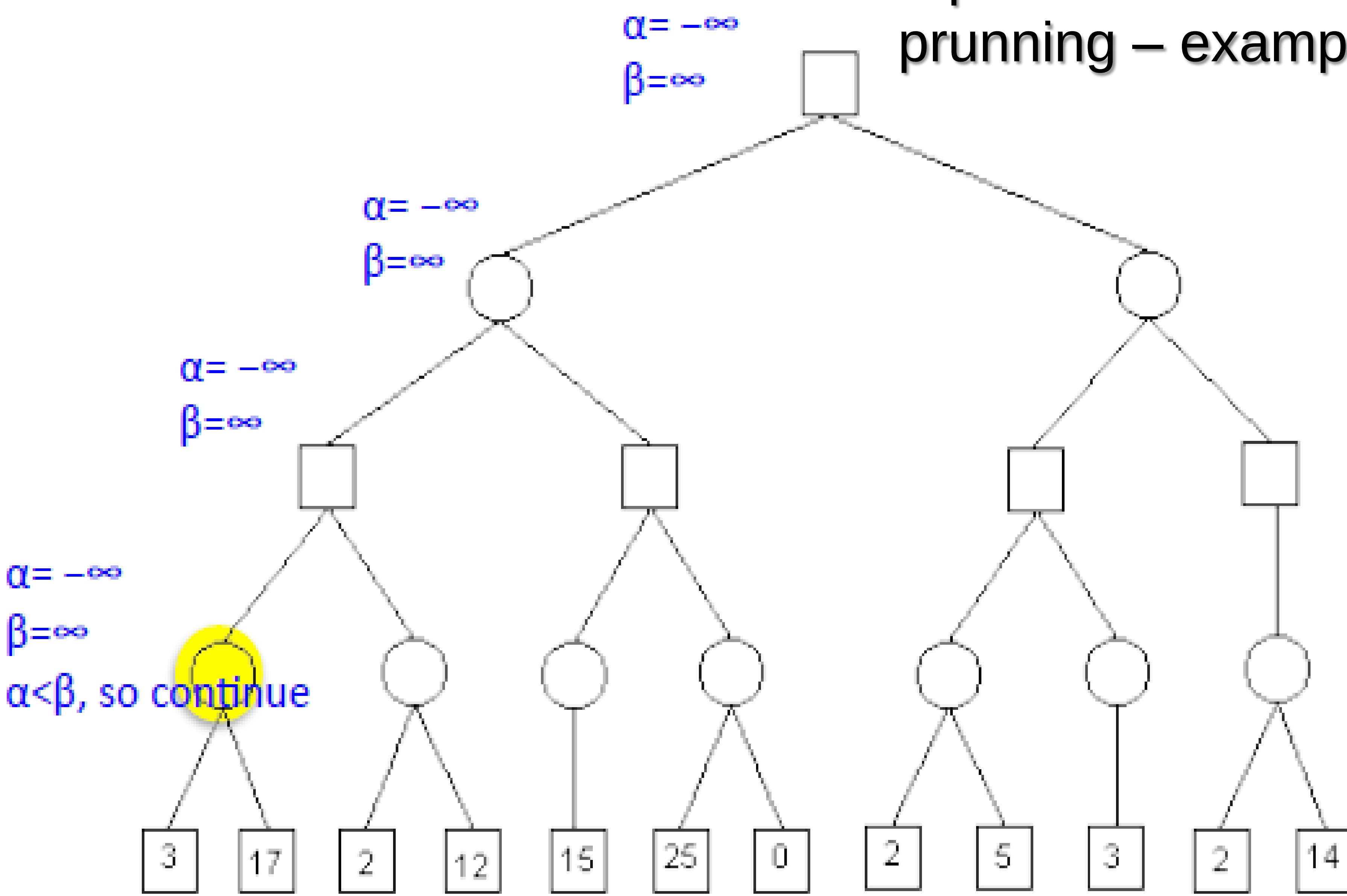
Alpha-beta prunning – example



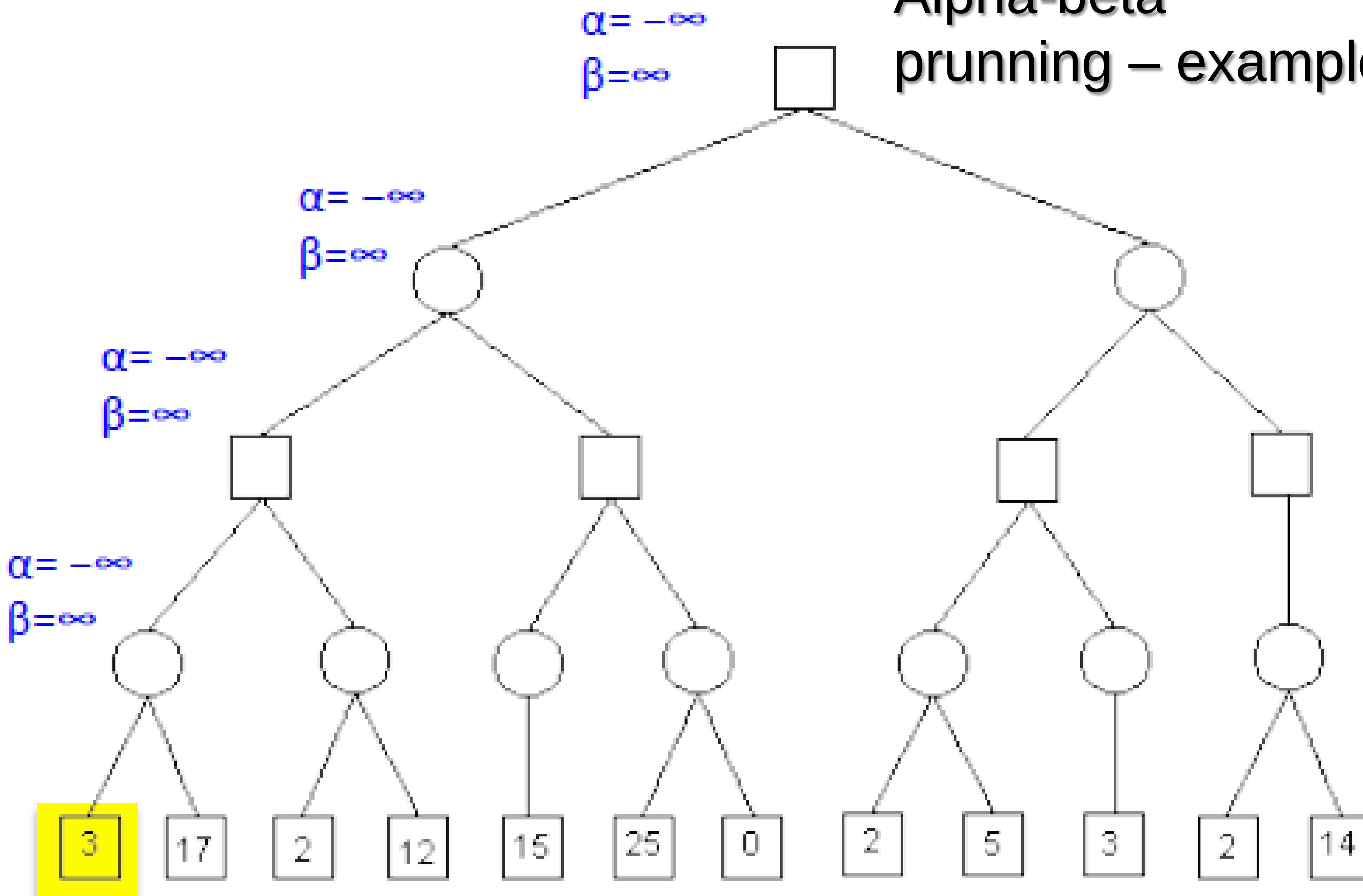
Alpha-beta prunning – example



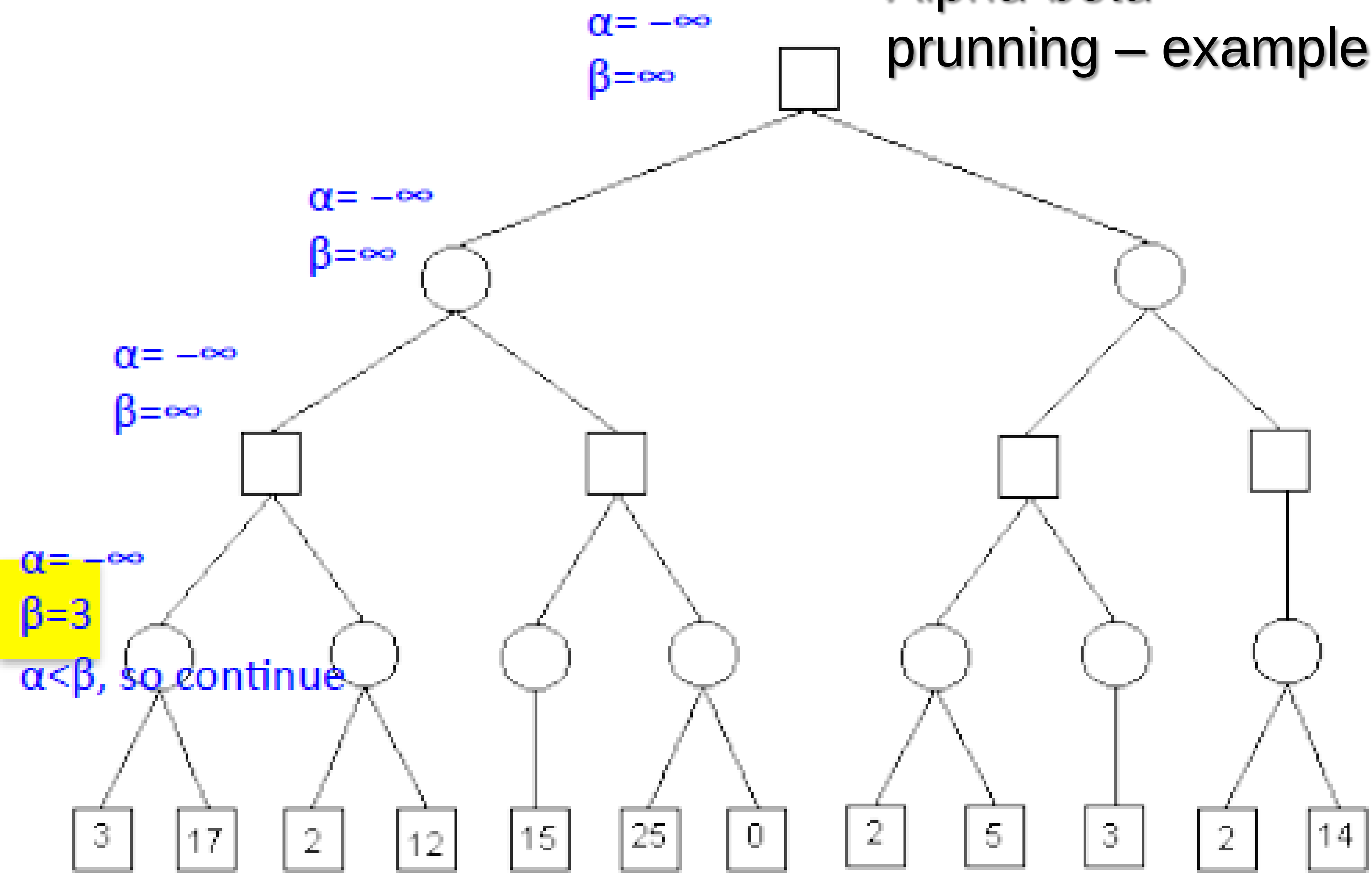
Alpha-beta prunning – example



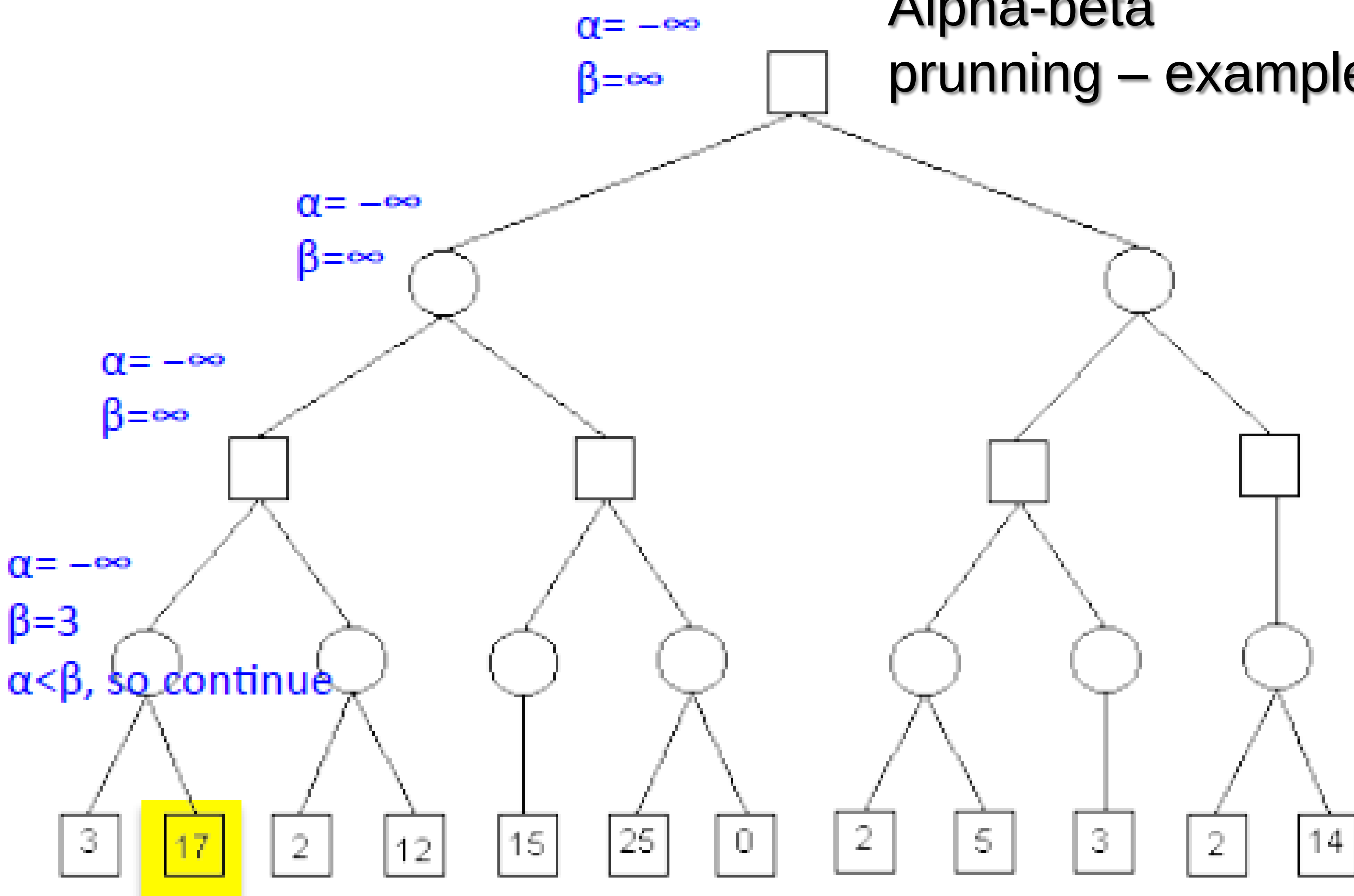
Alpha-beta prunning – example



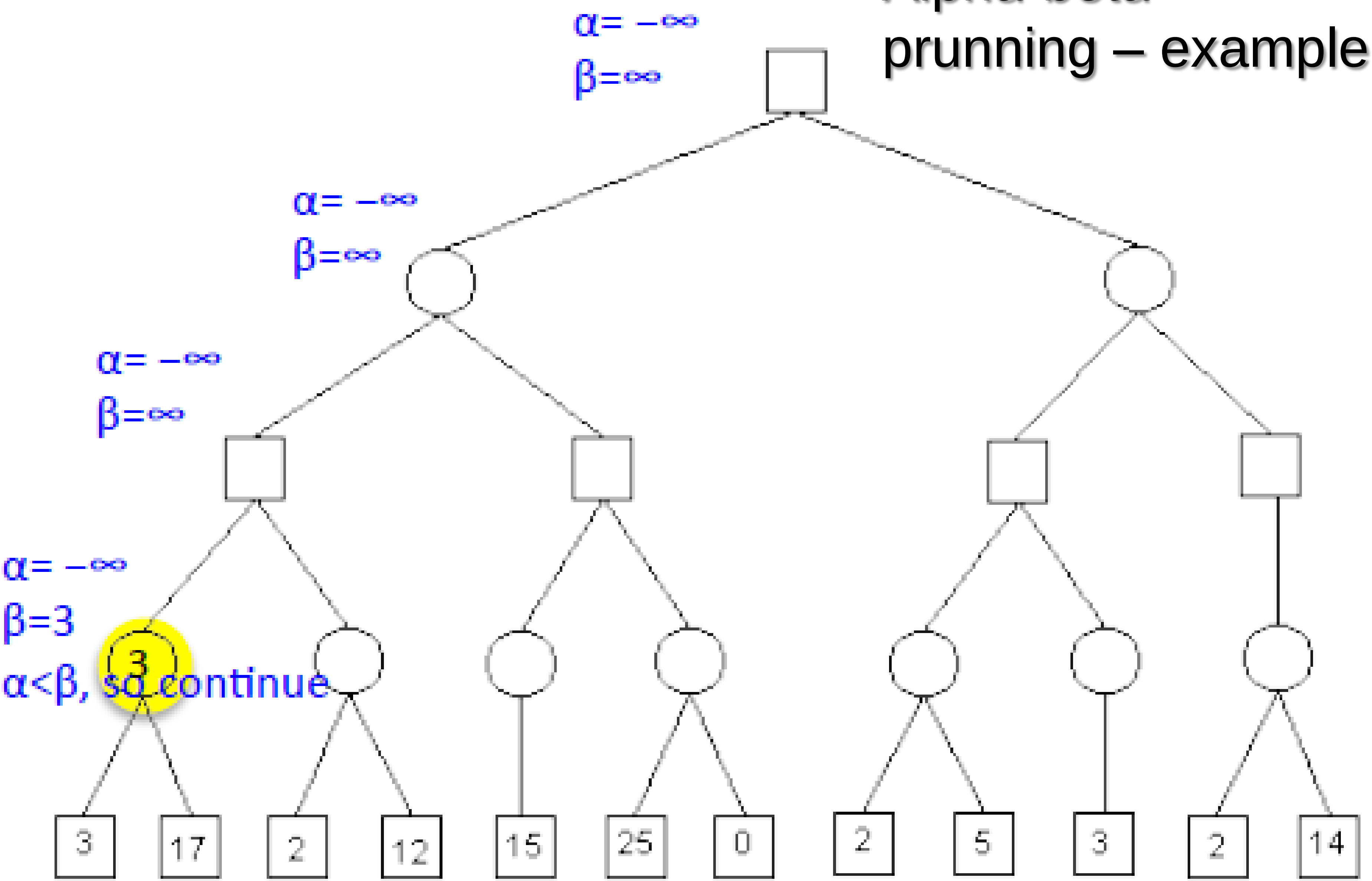
Alpha-beta prunning – example



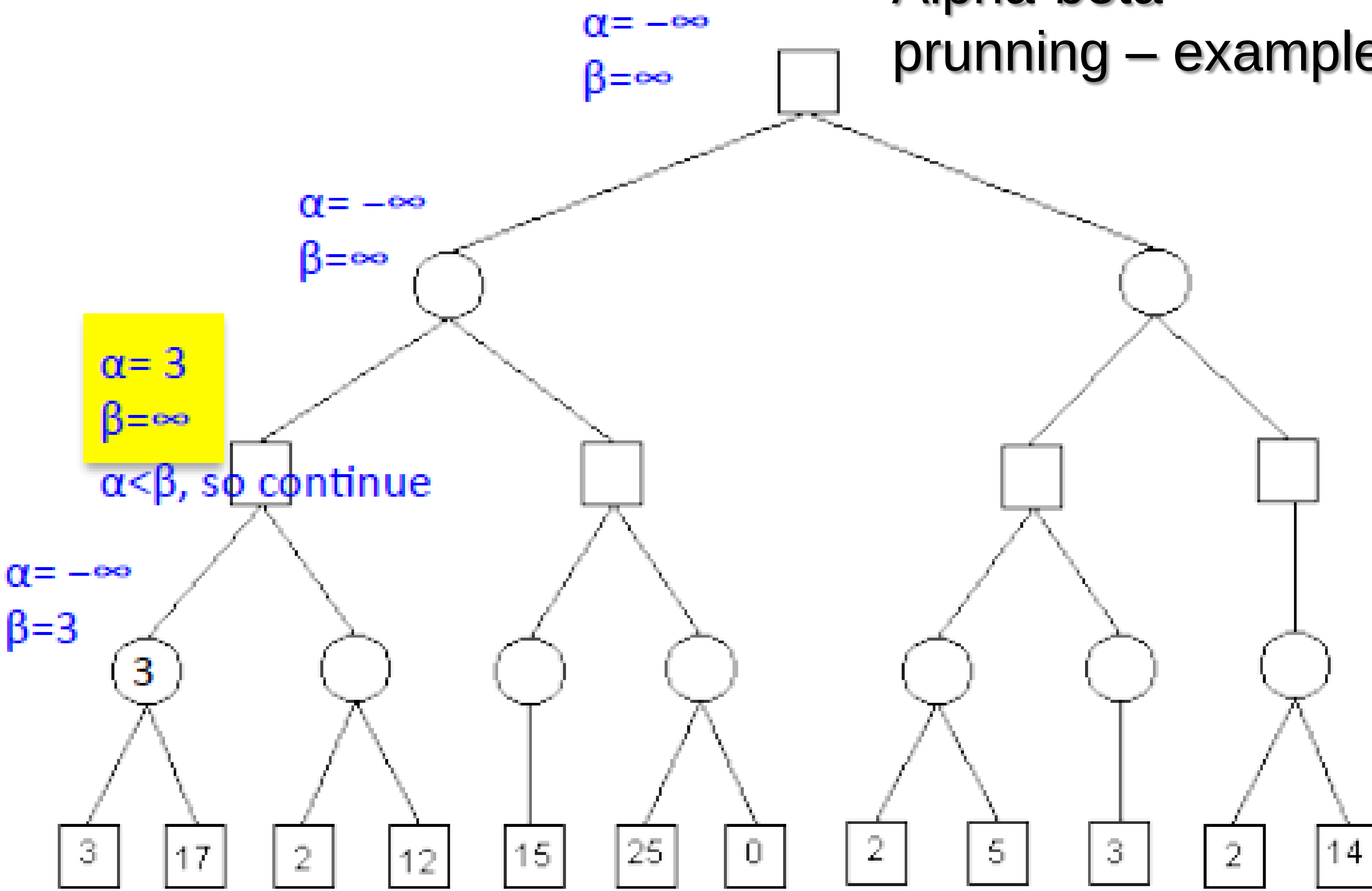
Alpha-beta prunning – example



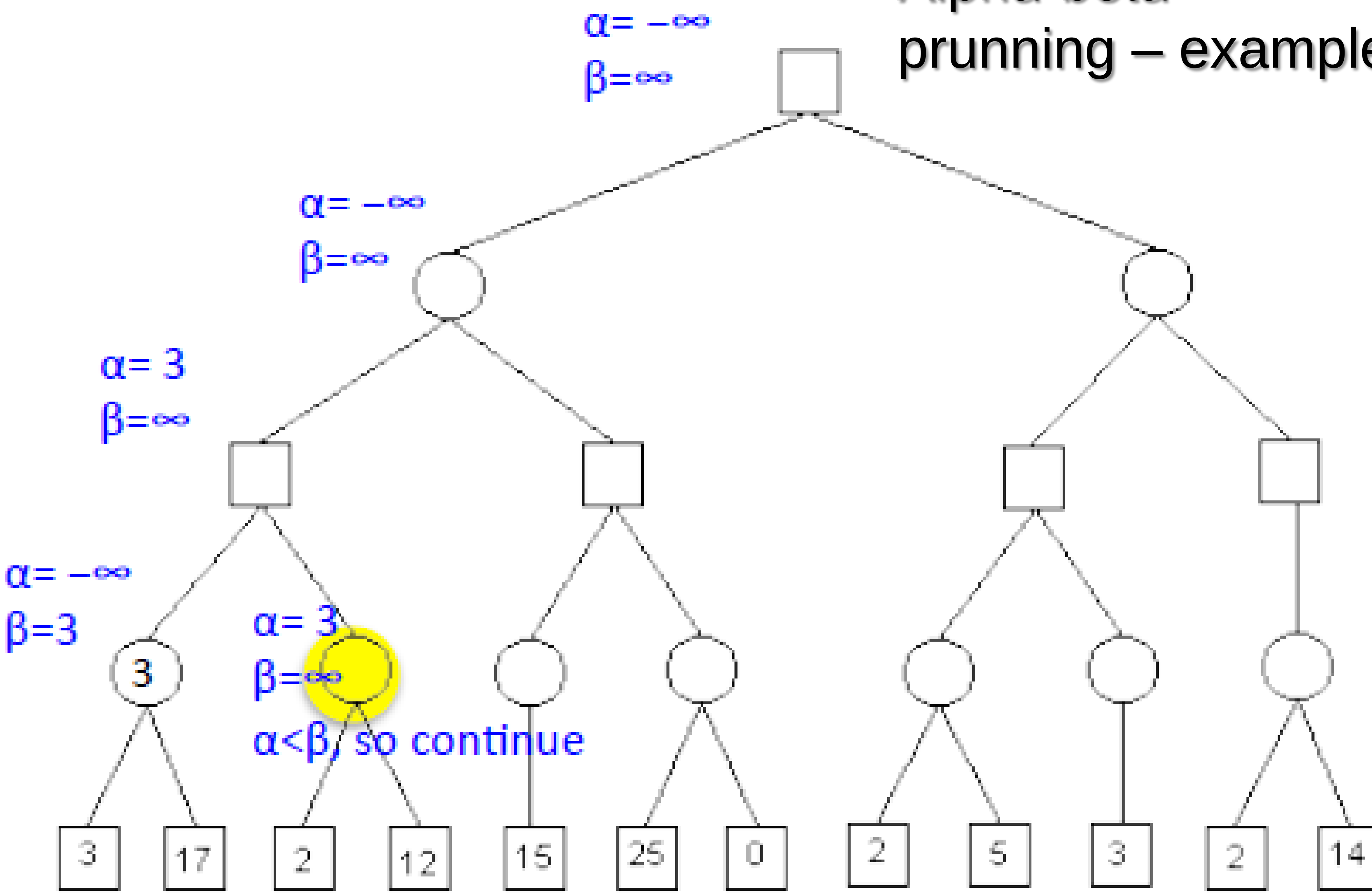
Alpha-beta prunning – example



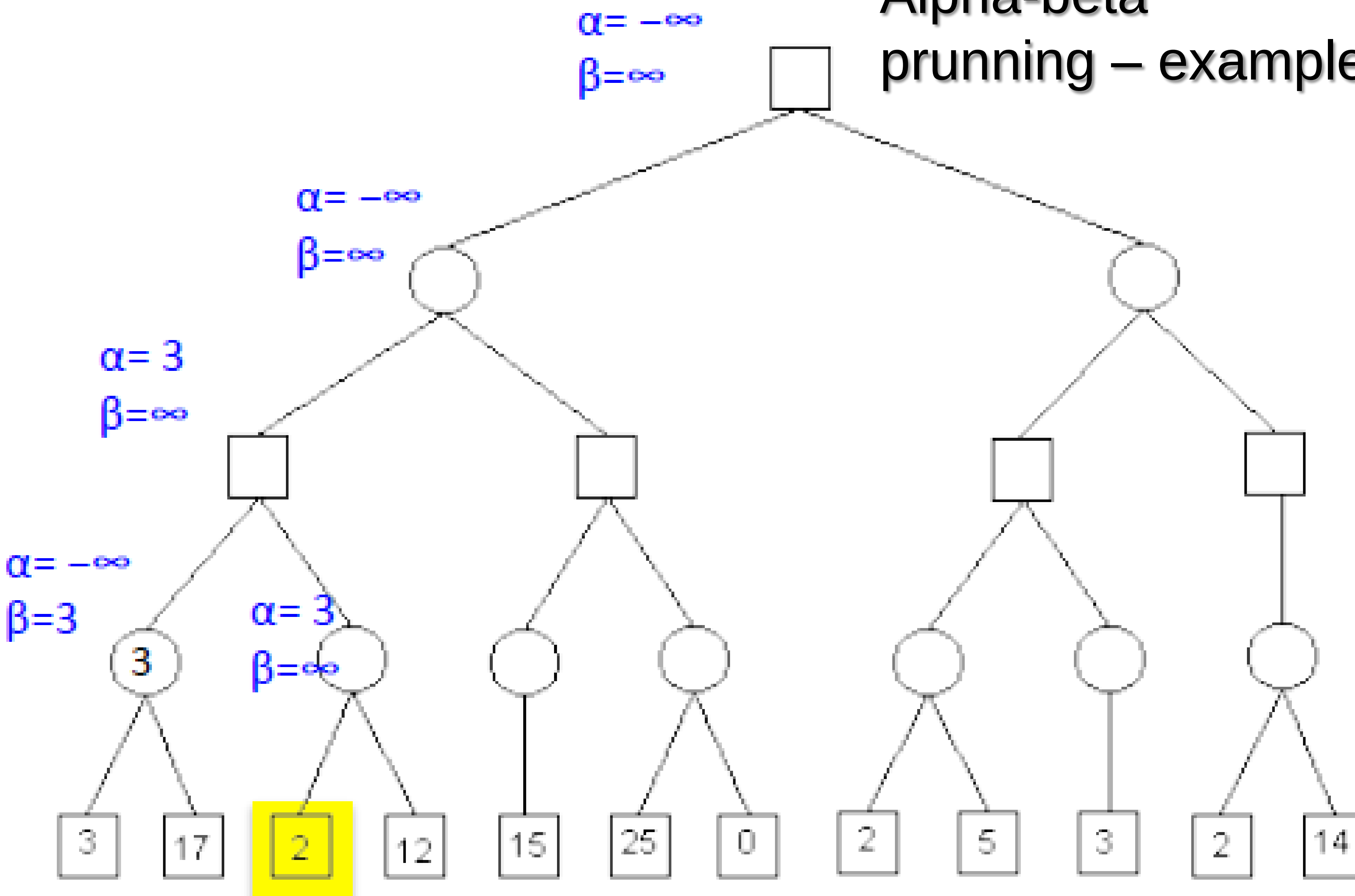
Alpha-beta prunning – example



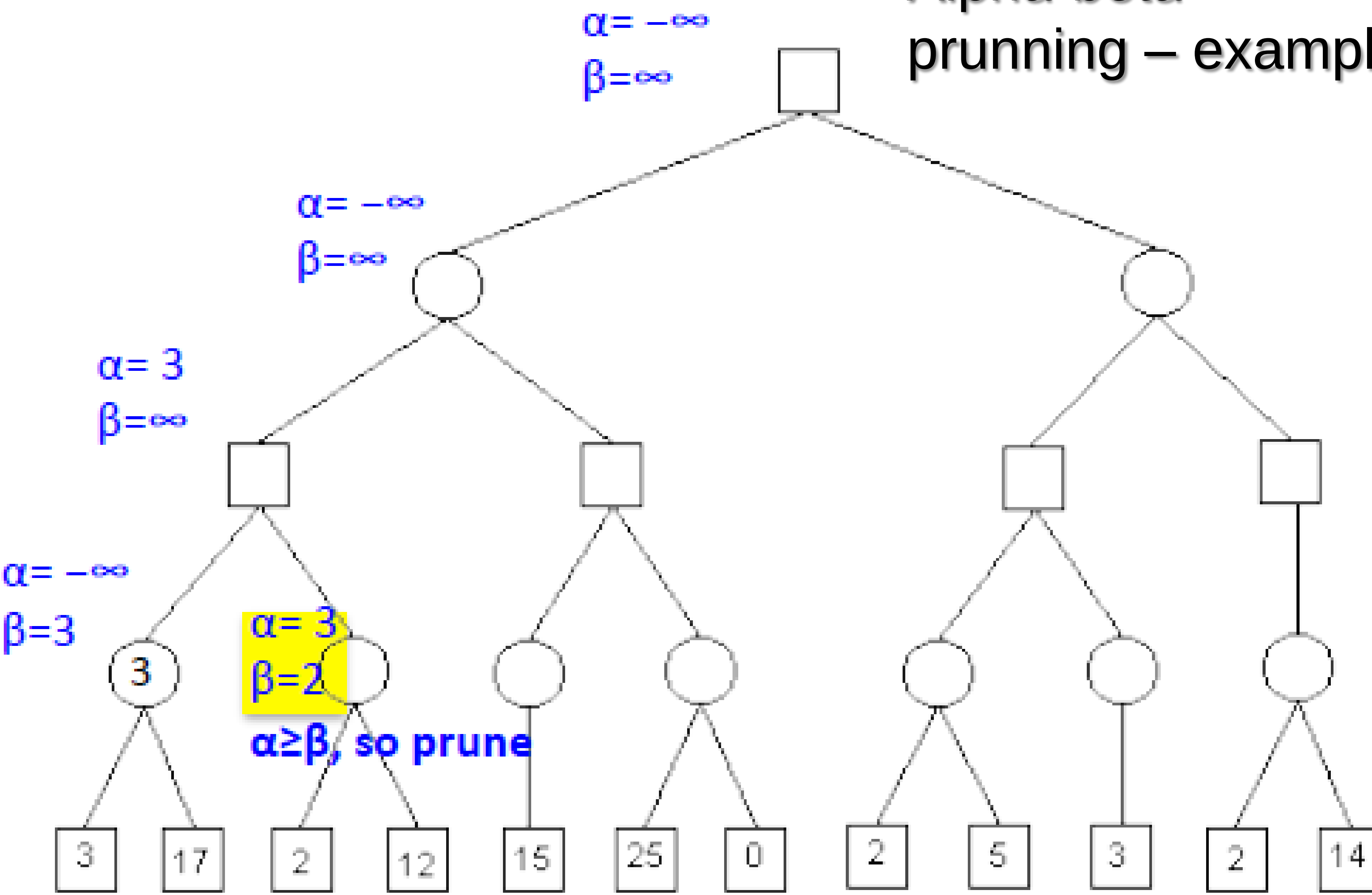
Alpha-beta prunning – example



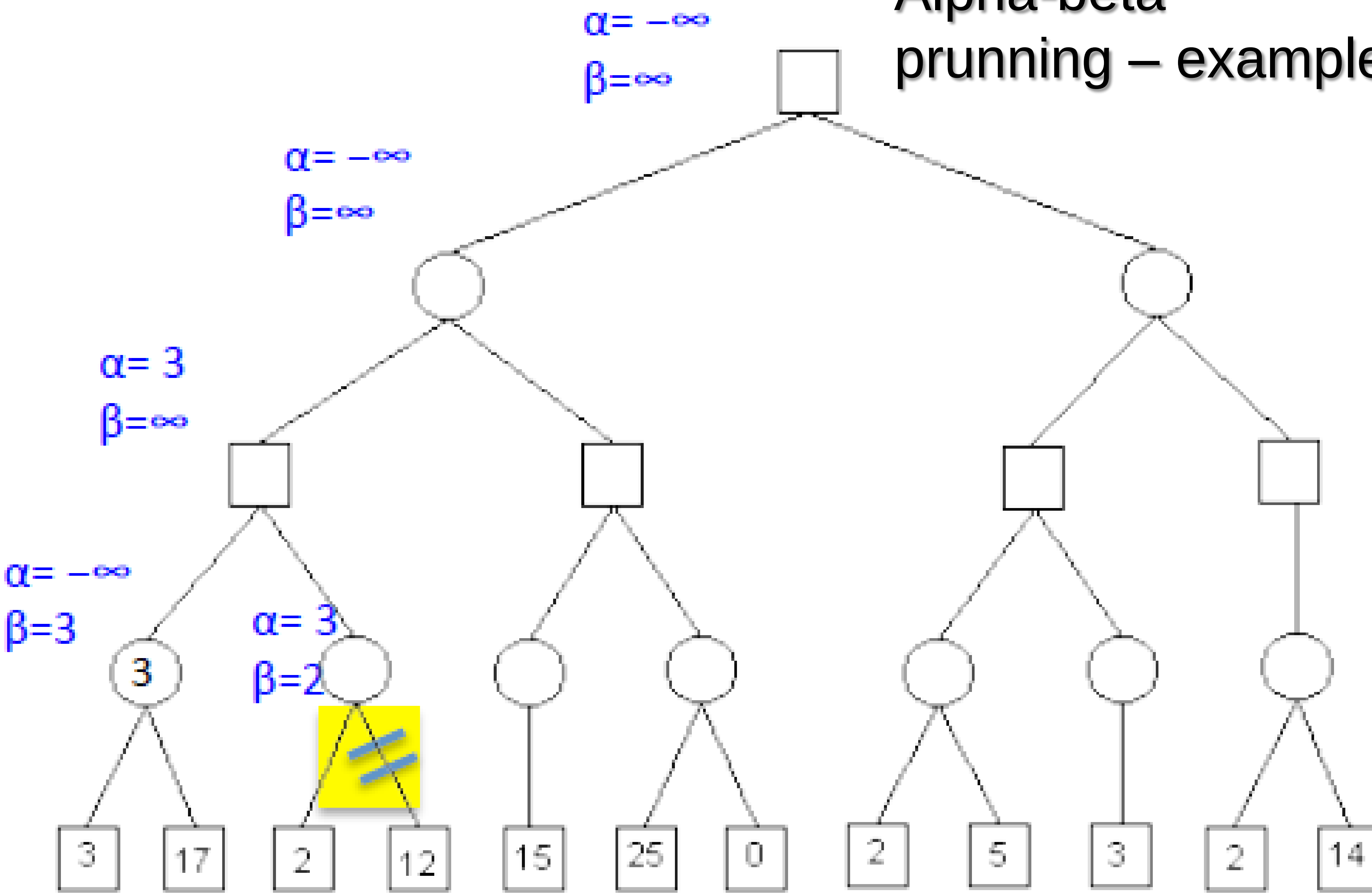
Alpha-beta prunning – example



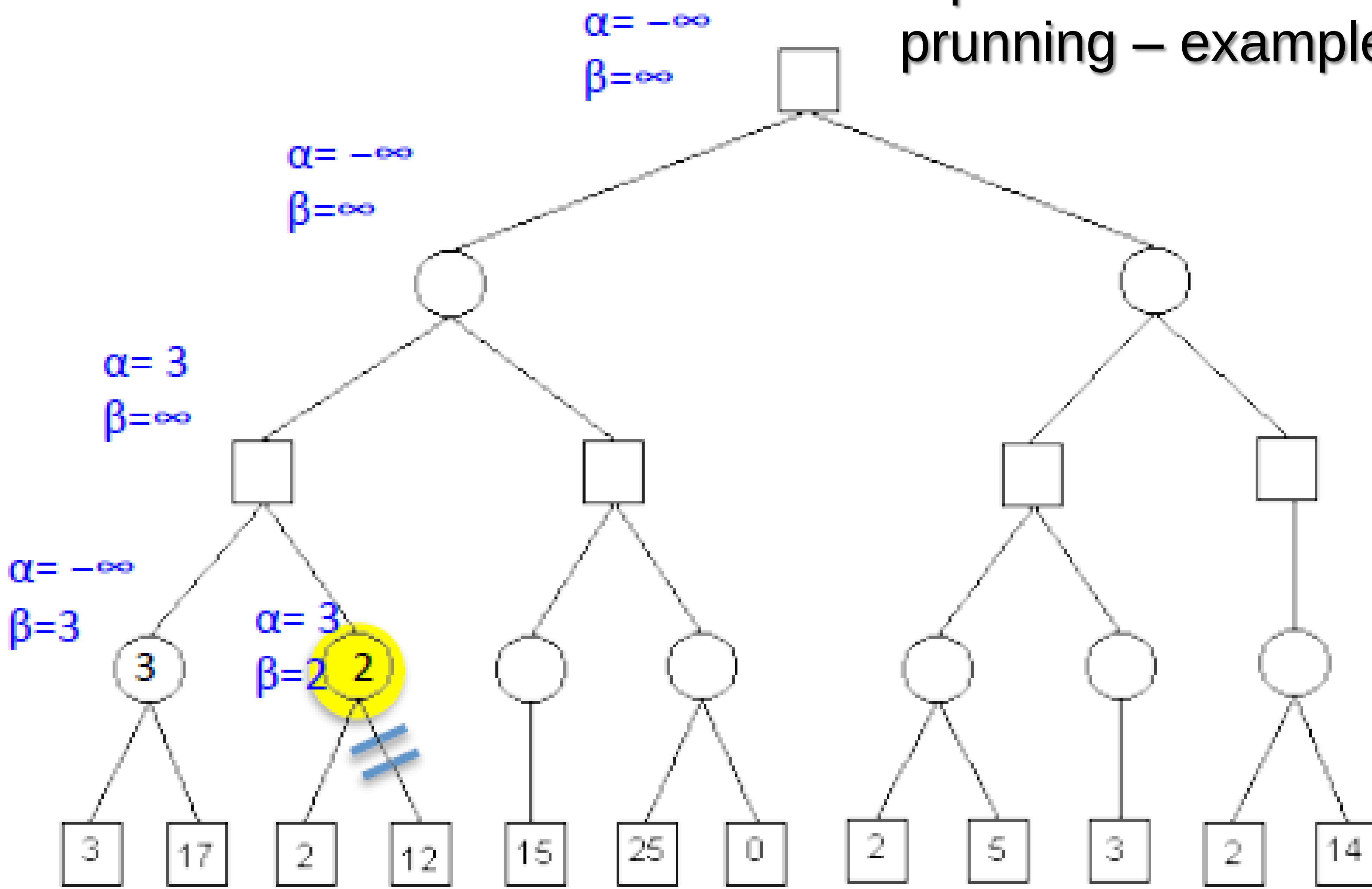
Alpha-beta prunning – example



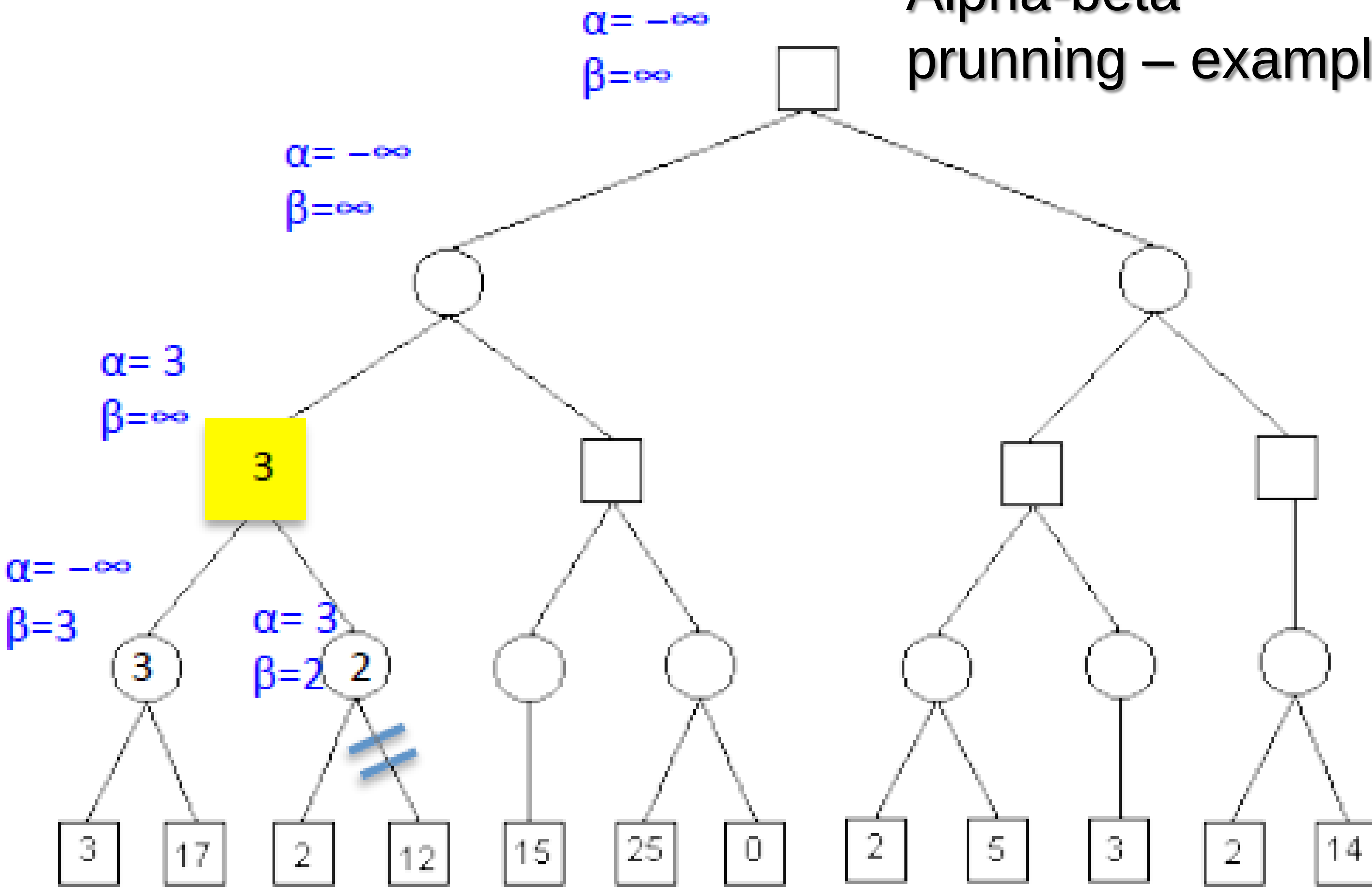
Alpha-beta prunning – example



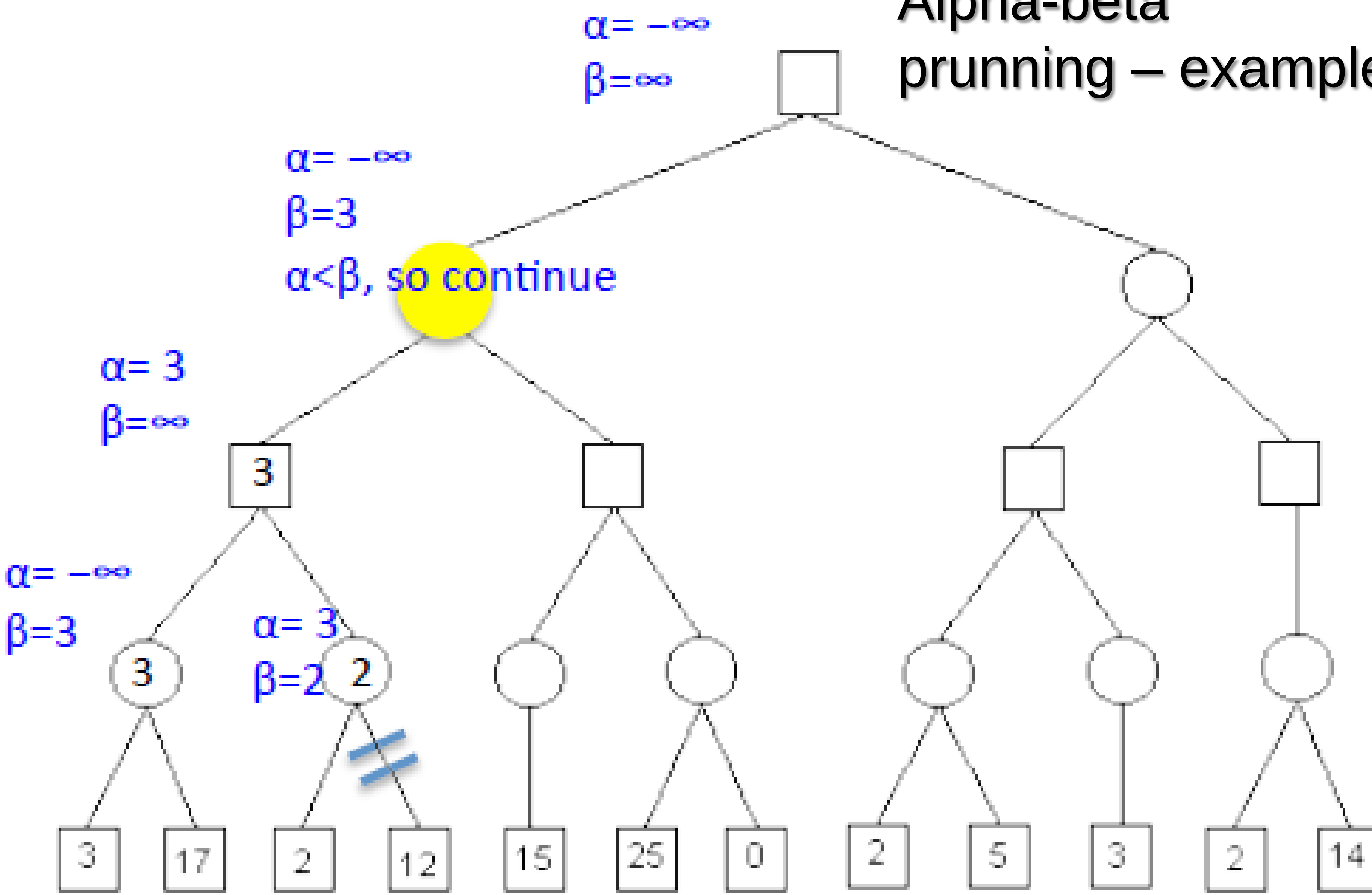
Alpha-beta prunning – example



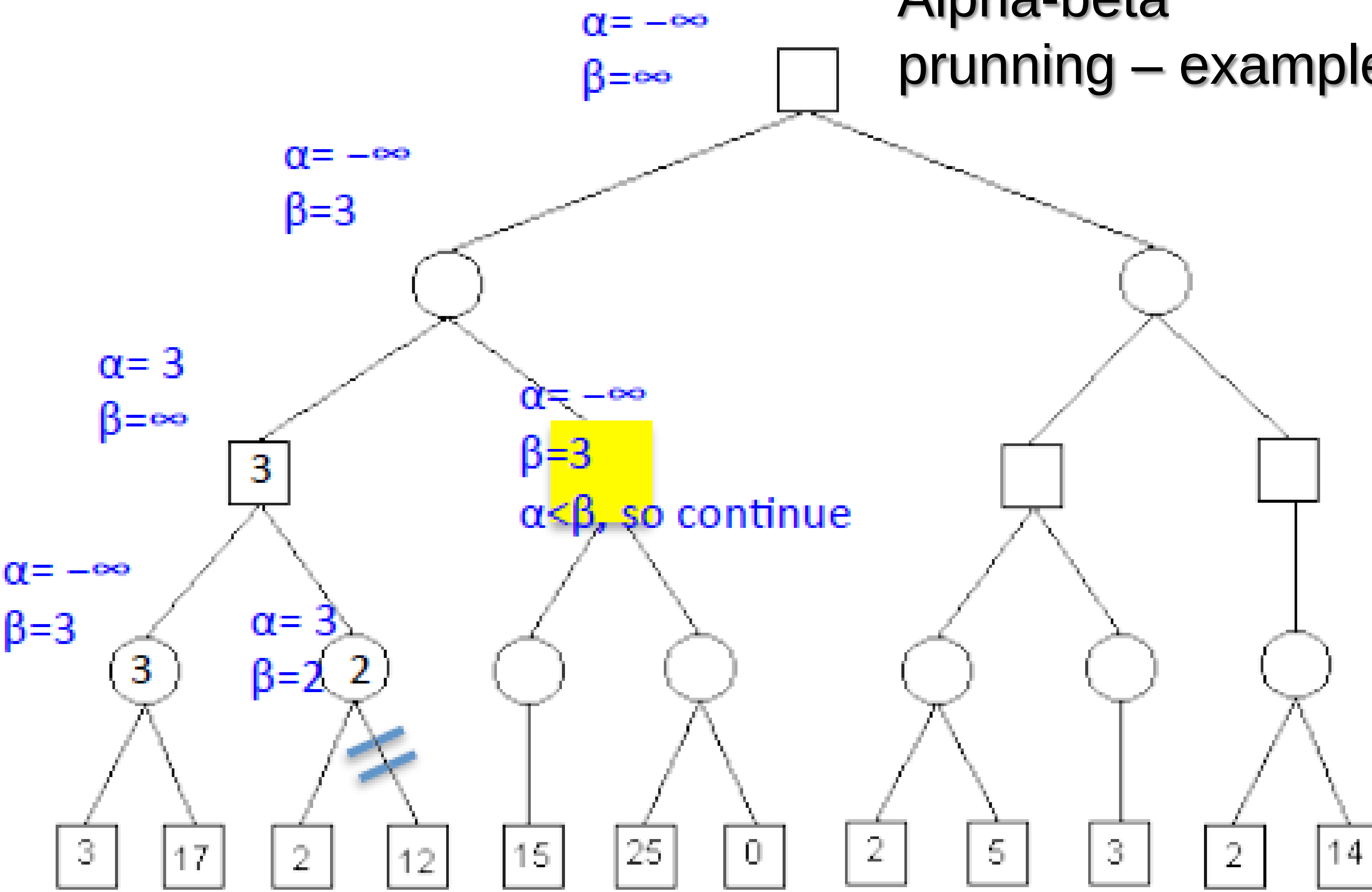
Alpha-beta prunning – example



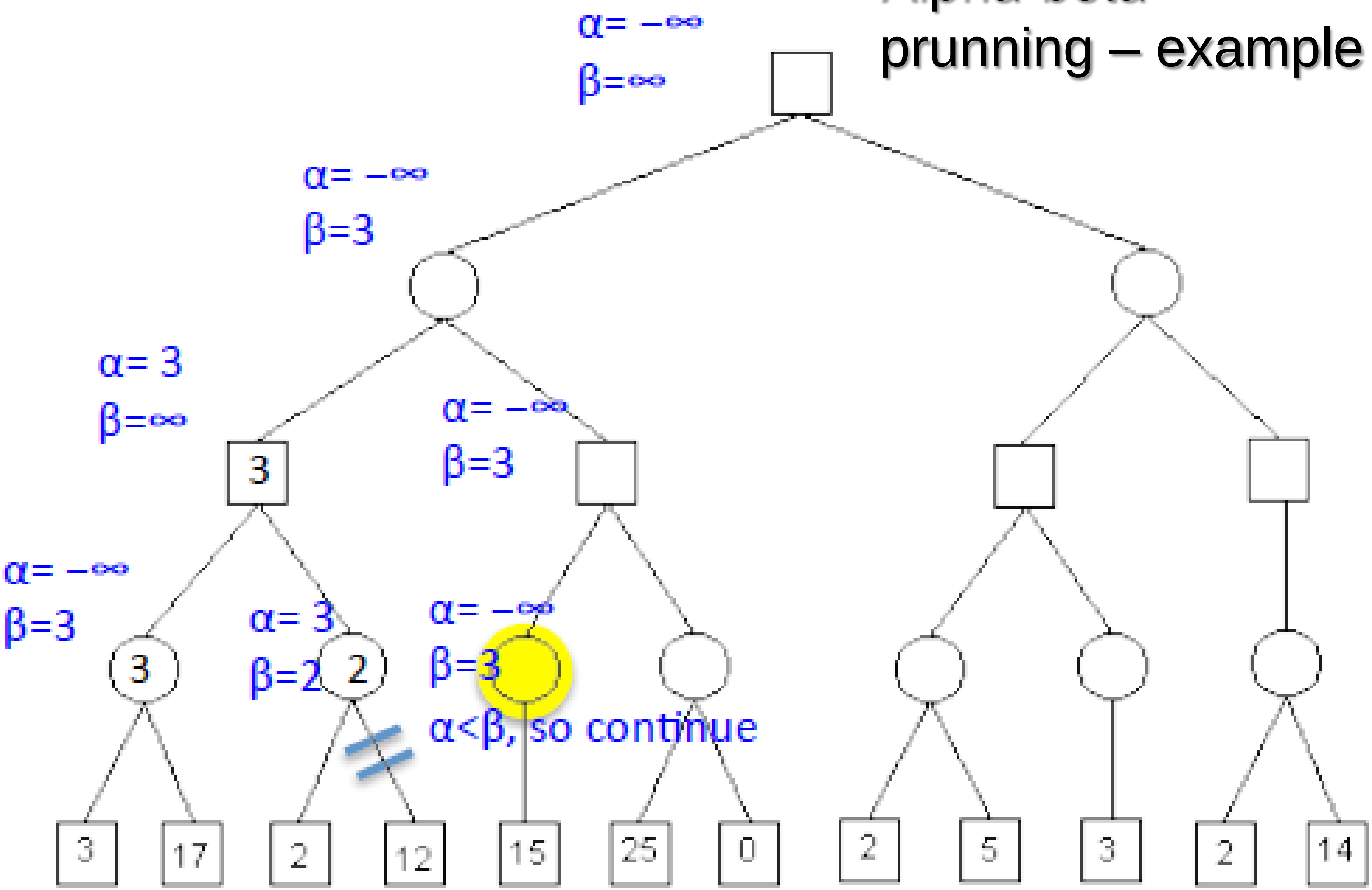
Alpha-beta prunning – example



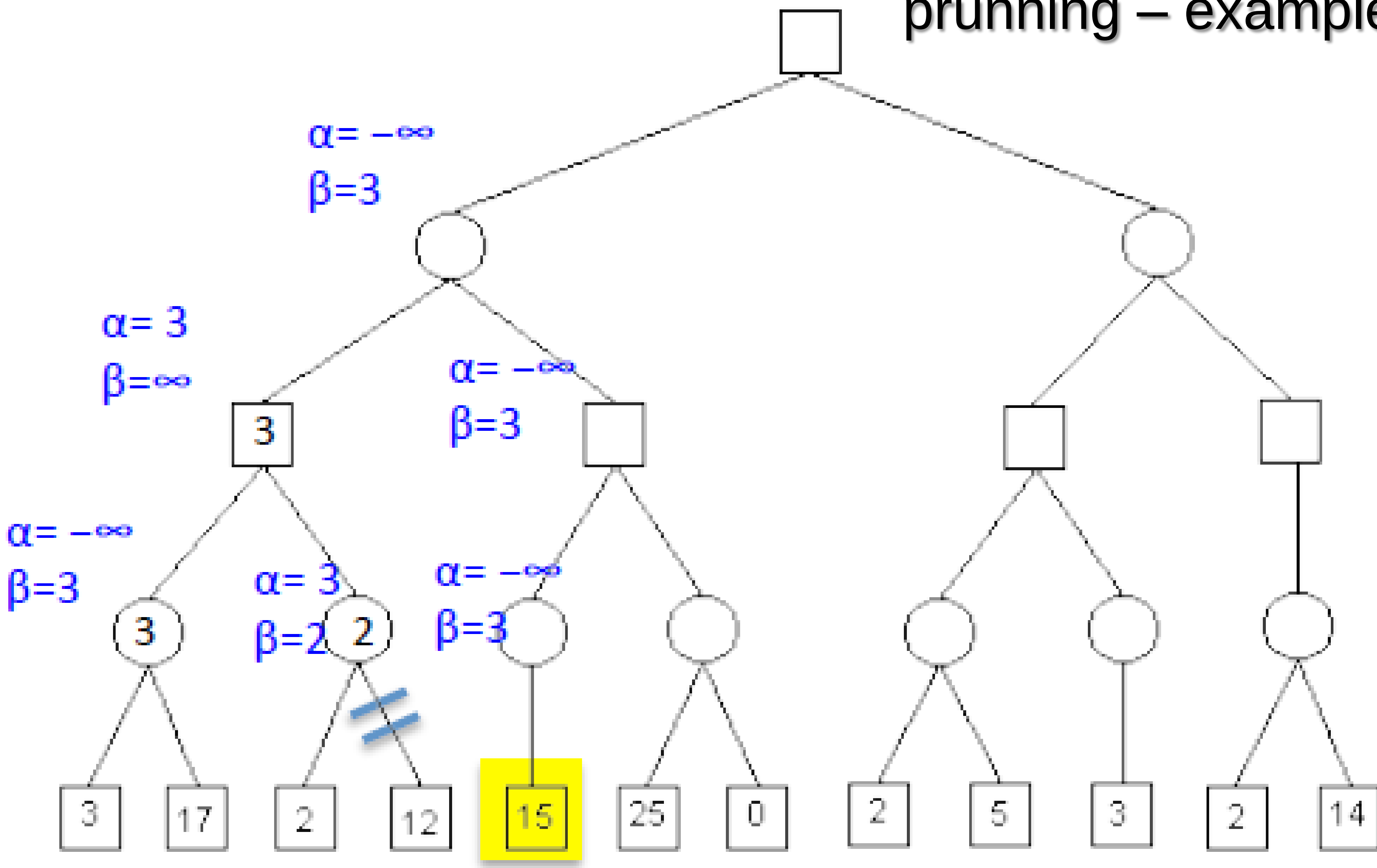
Alpha-beta prunning – example



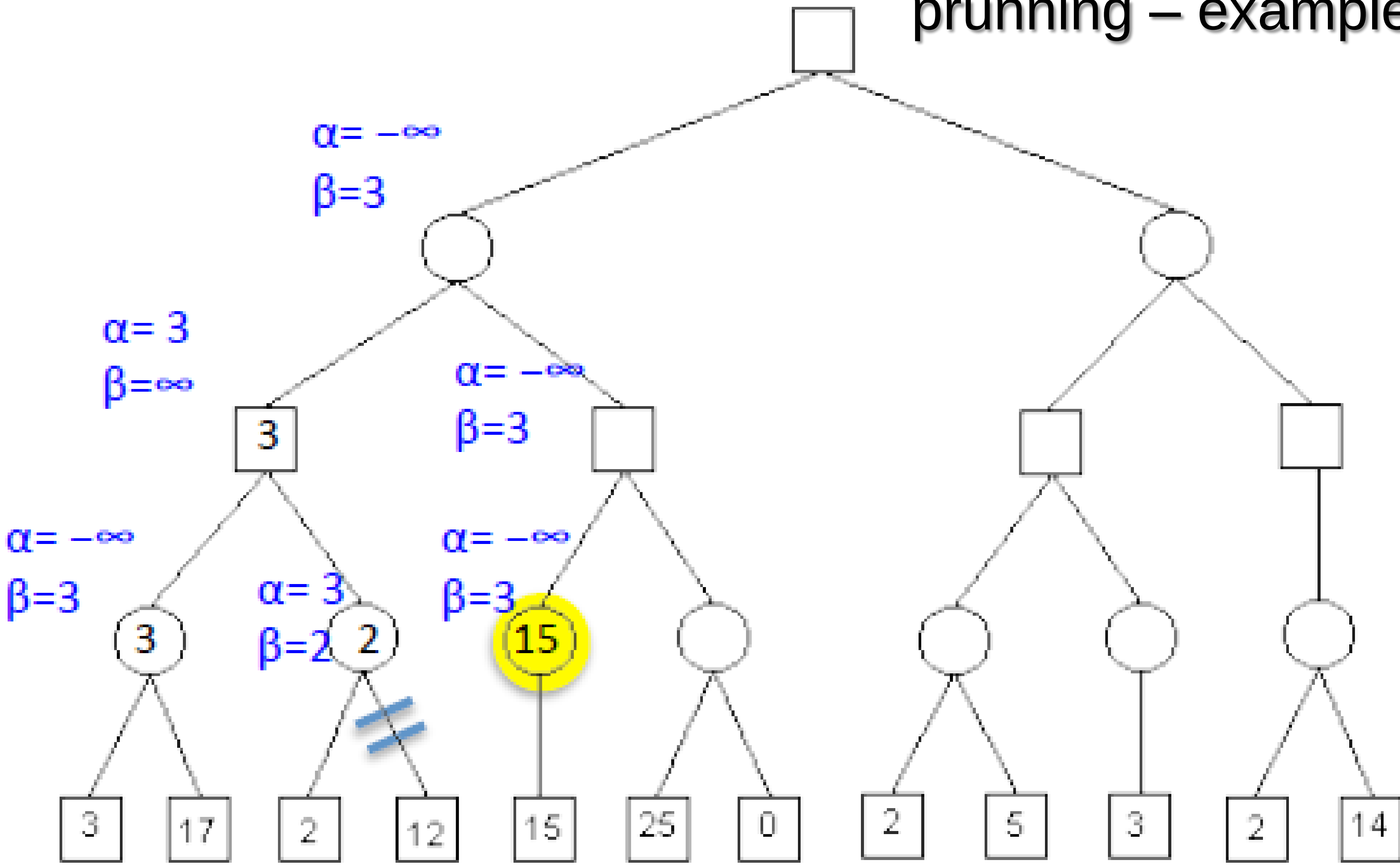
Alpha-beta prunning – example



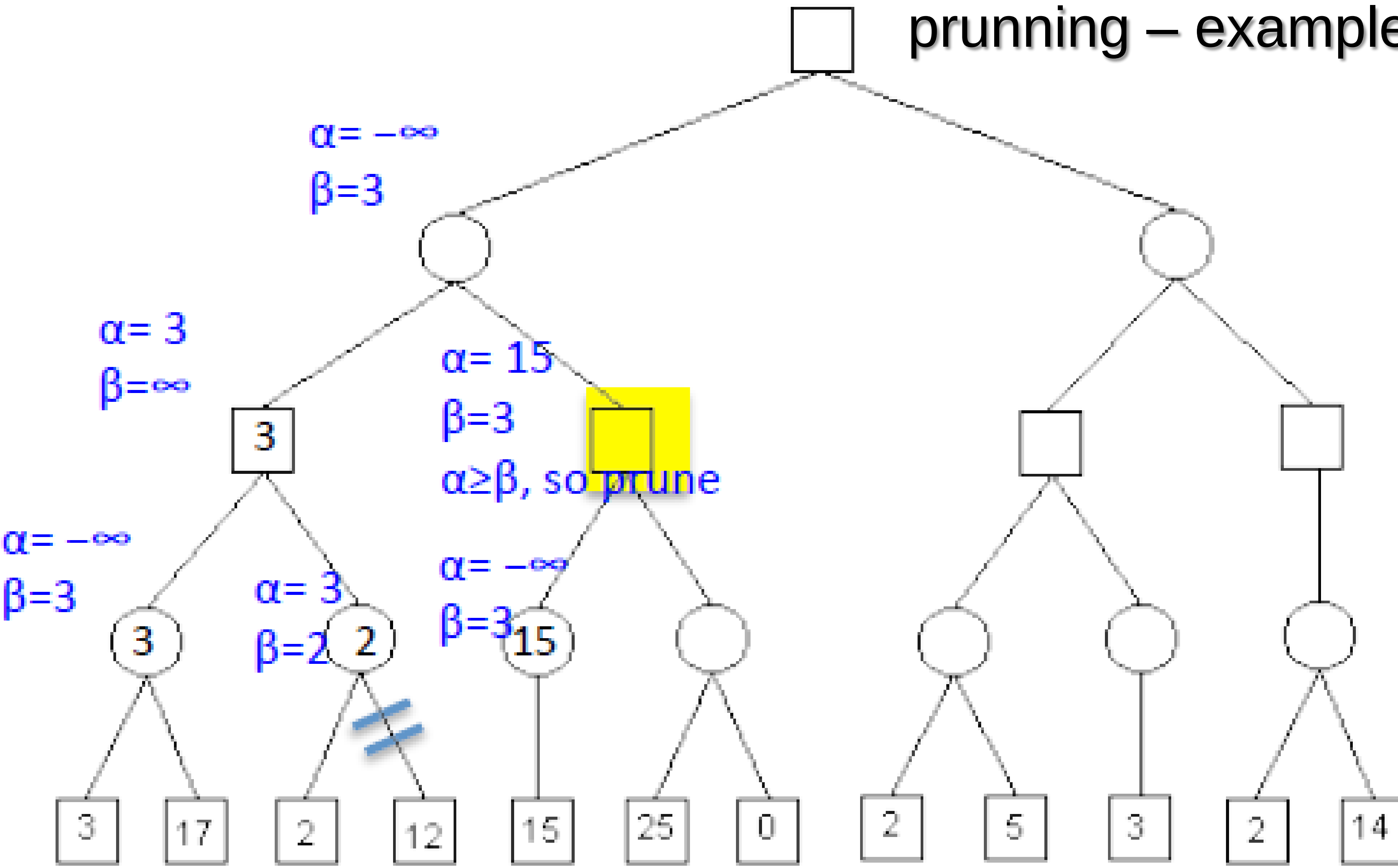
Alpha-beta prunning – example



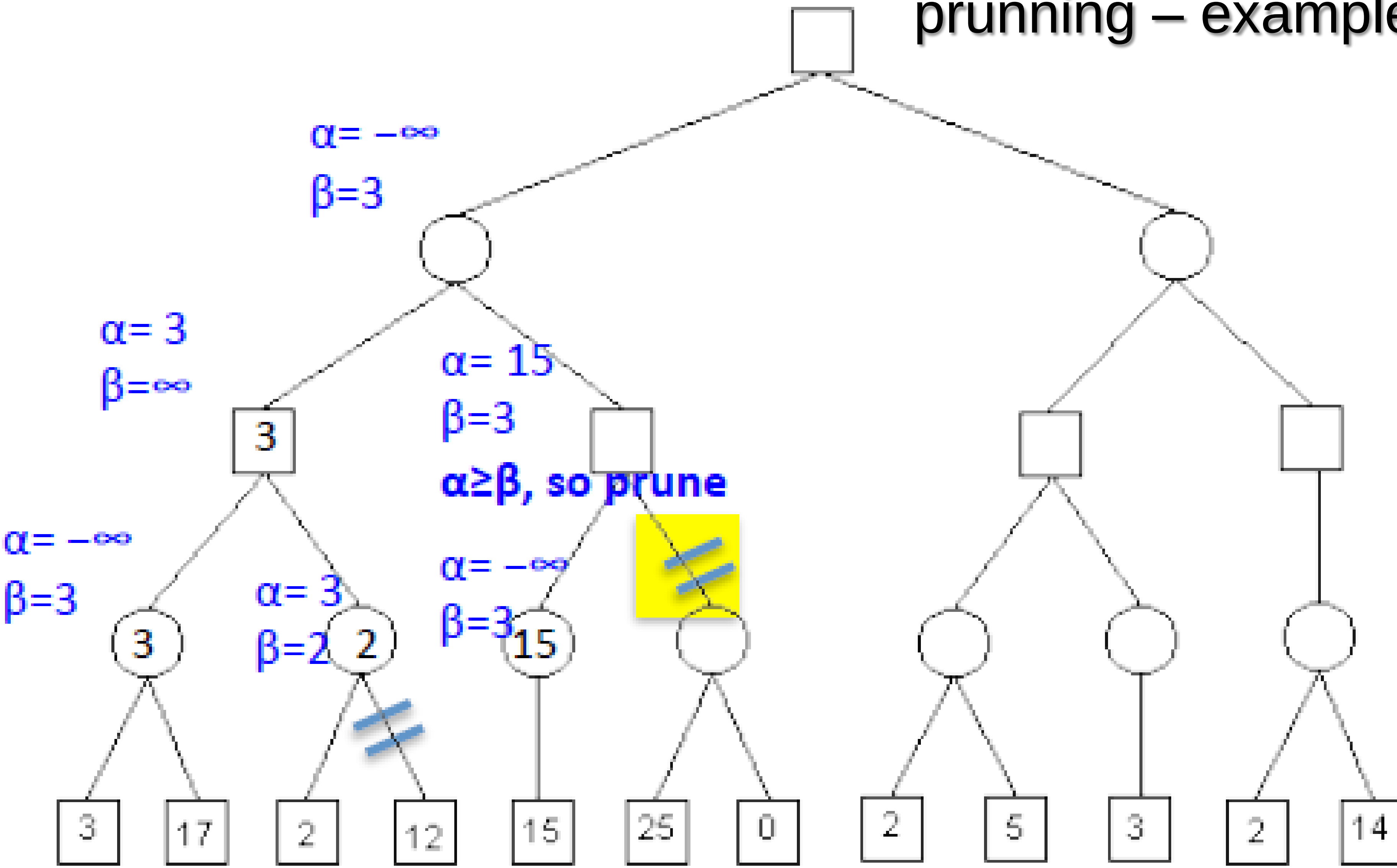
Alpha-beta prunning – example



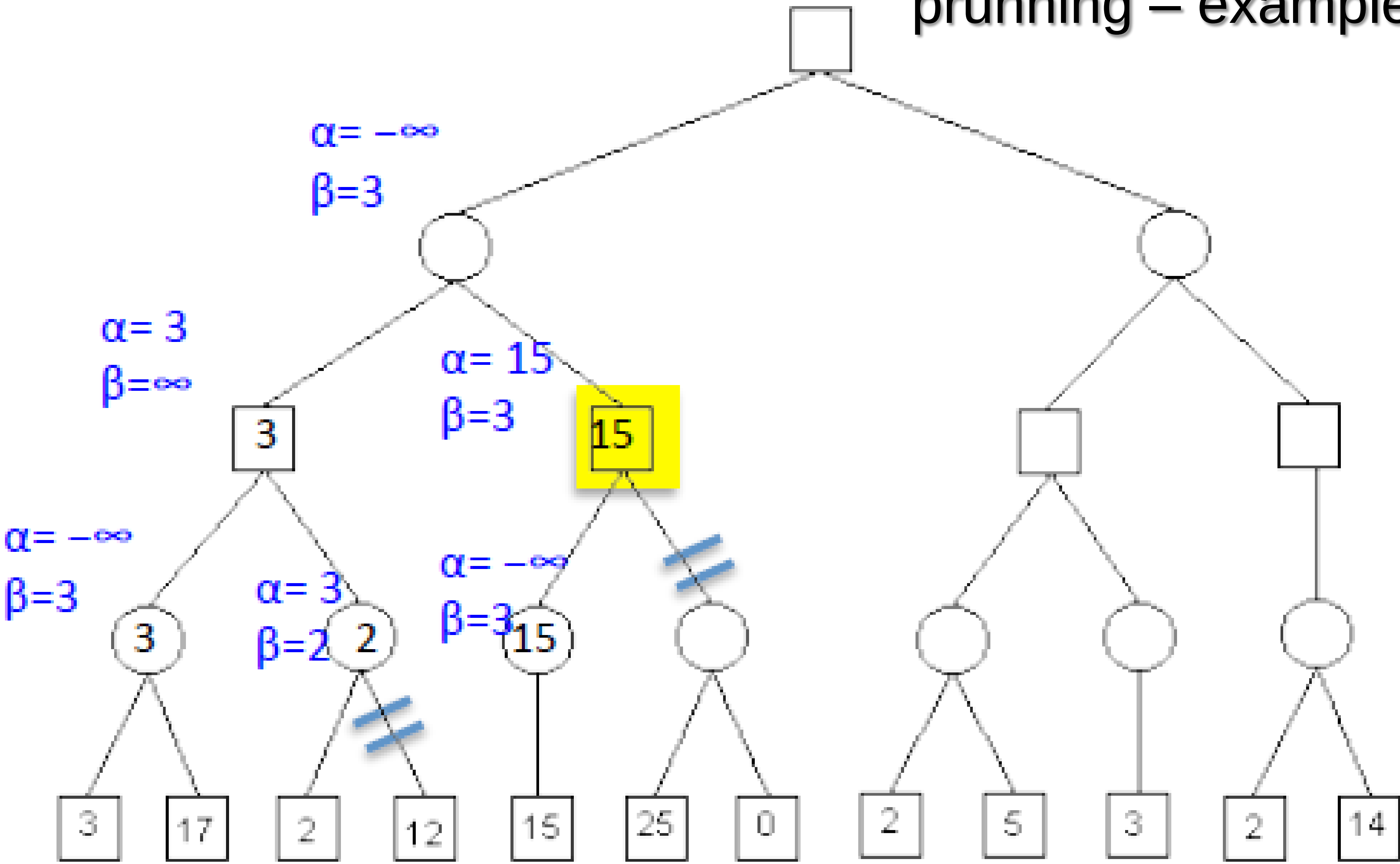
Alpha-beta prunning – example



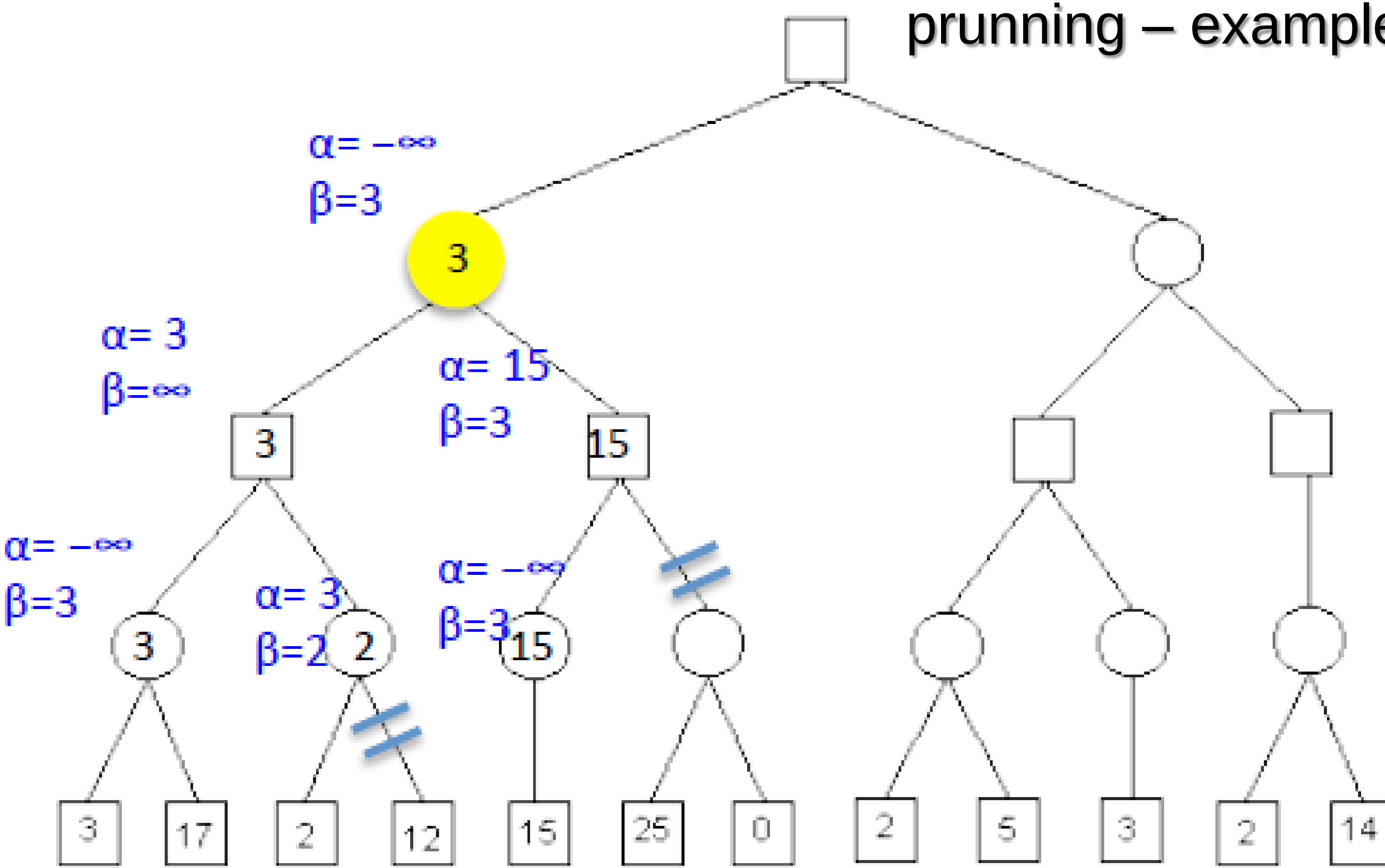
Alpha-beta prunning – example



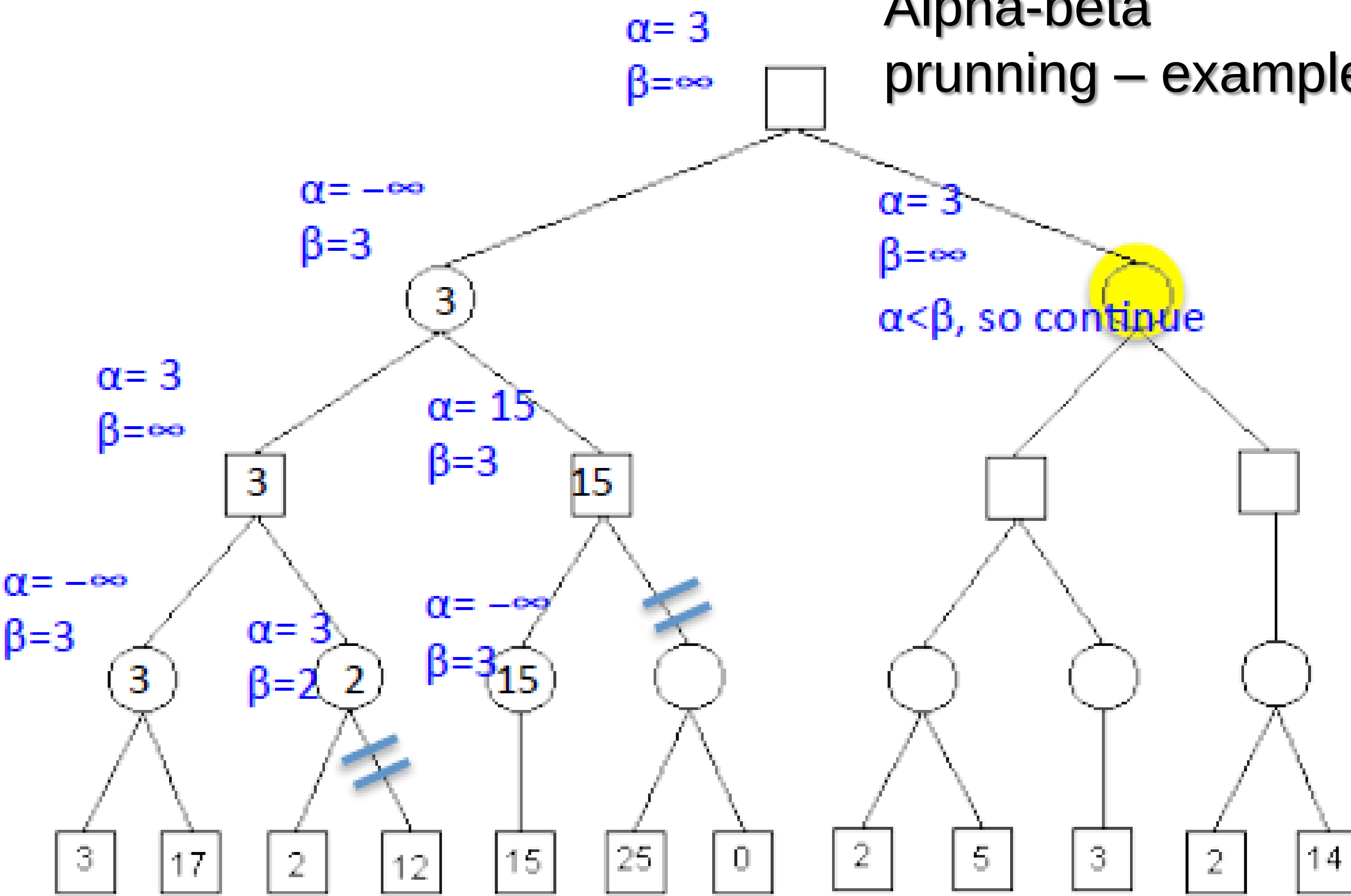
Alpha-beta prunning – example



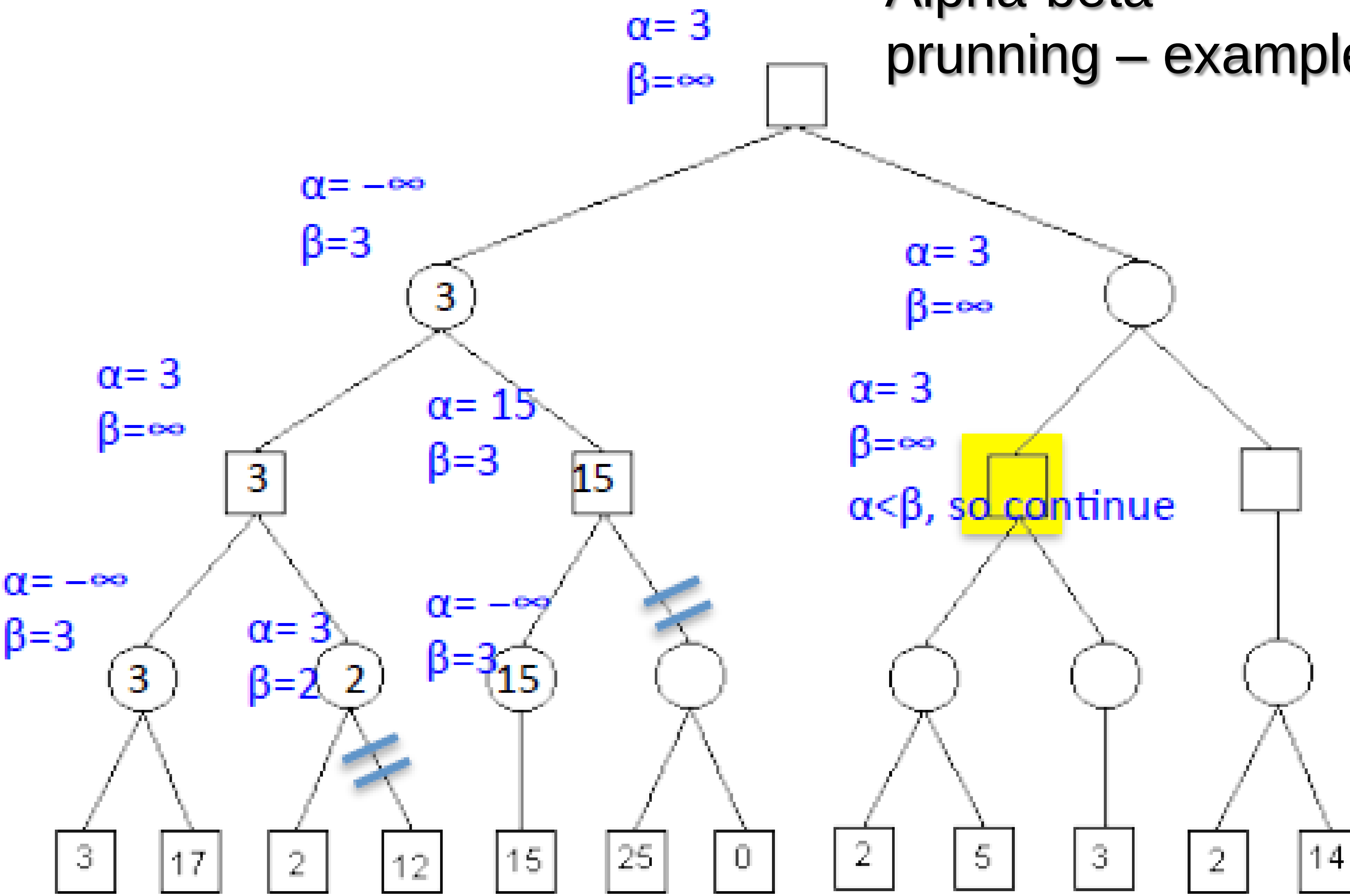
Alpha-beta prunning – example



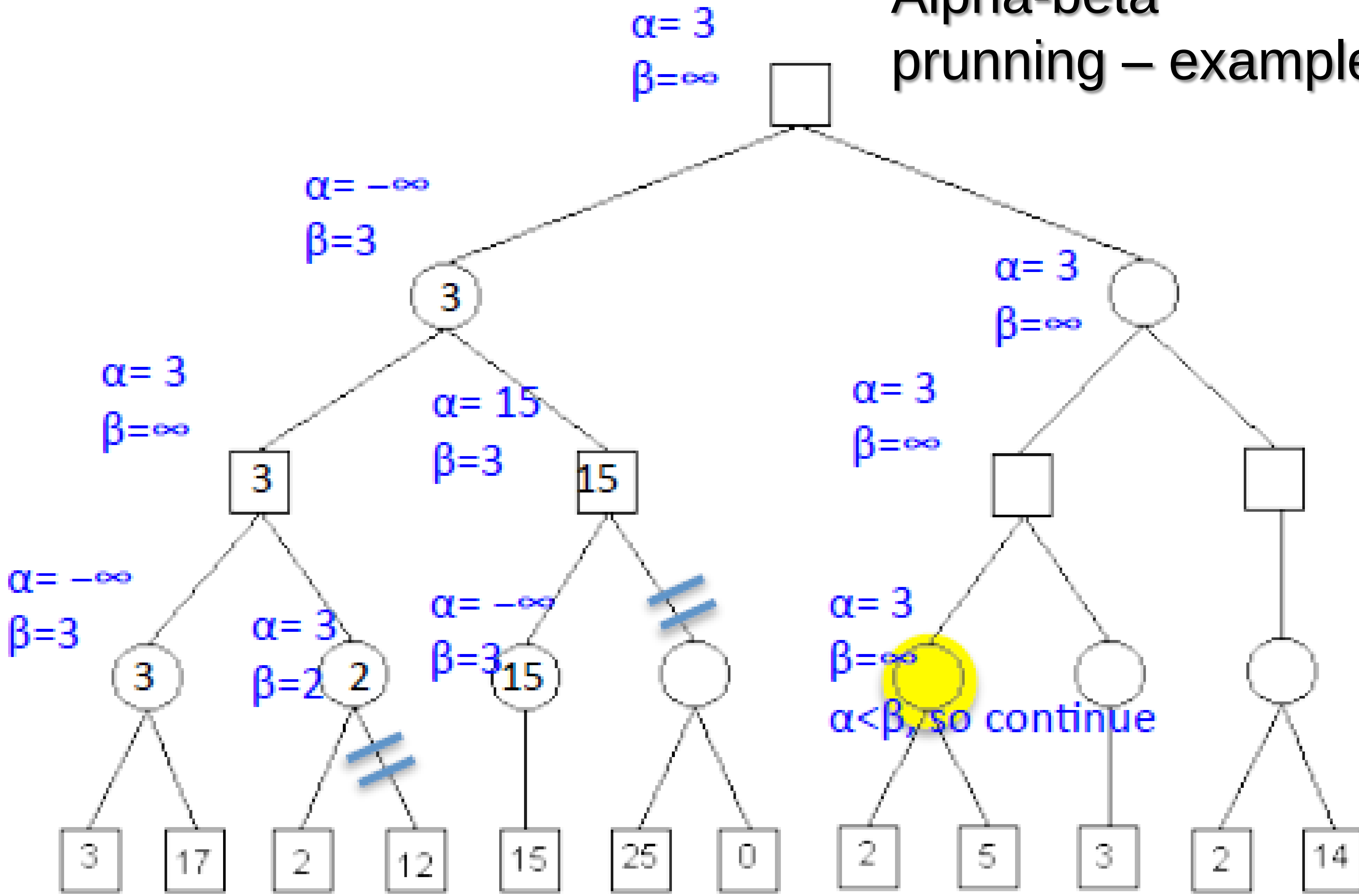
Alpha-beta prunning – example



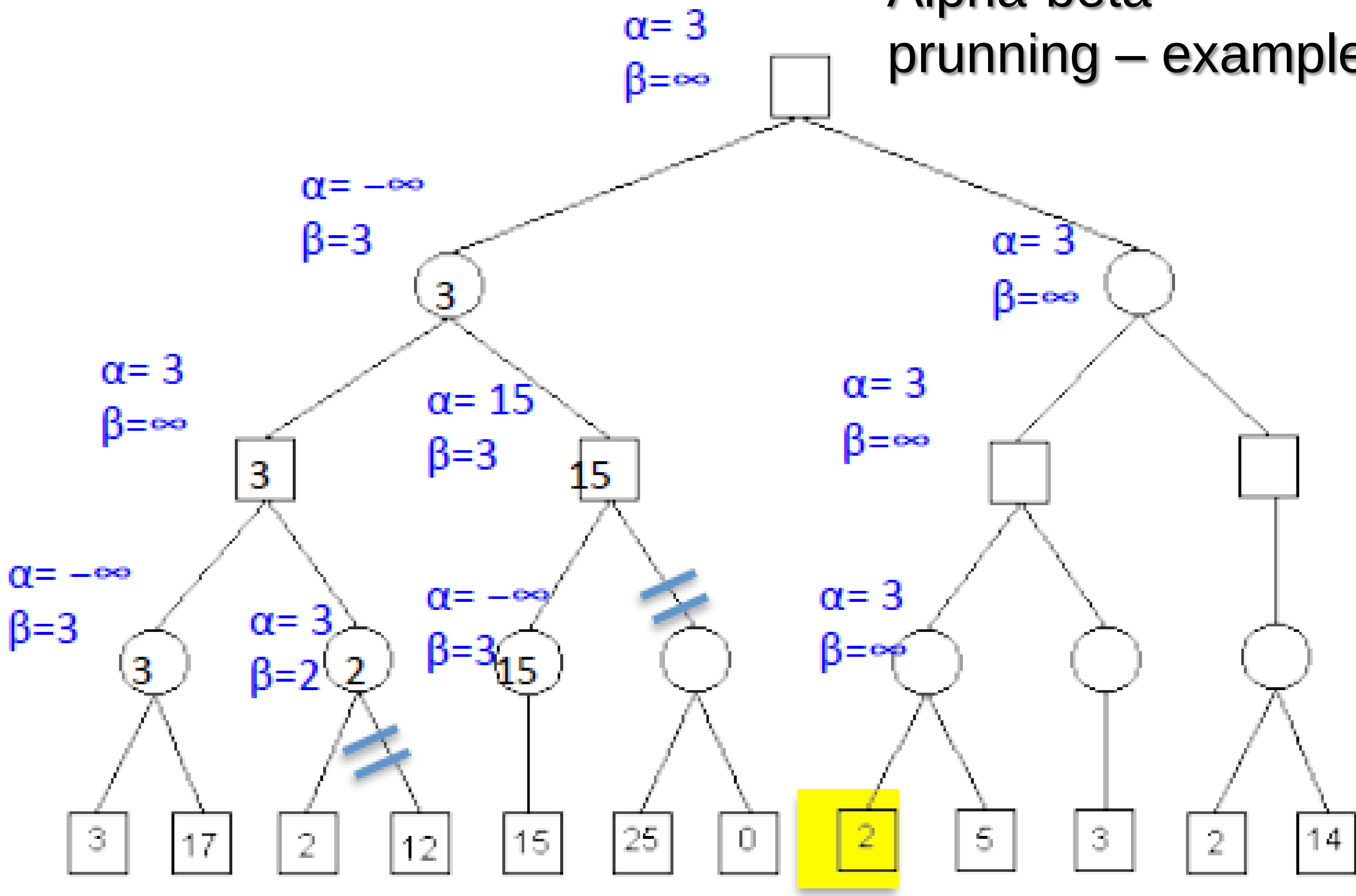
Alpha-beta prunning – example



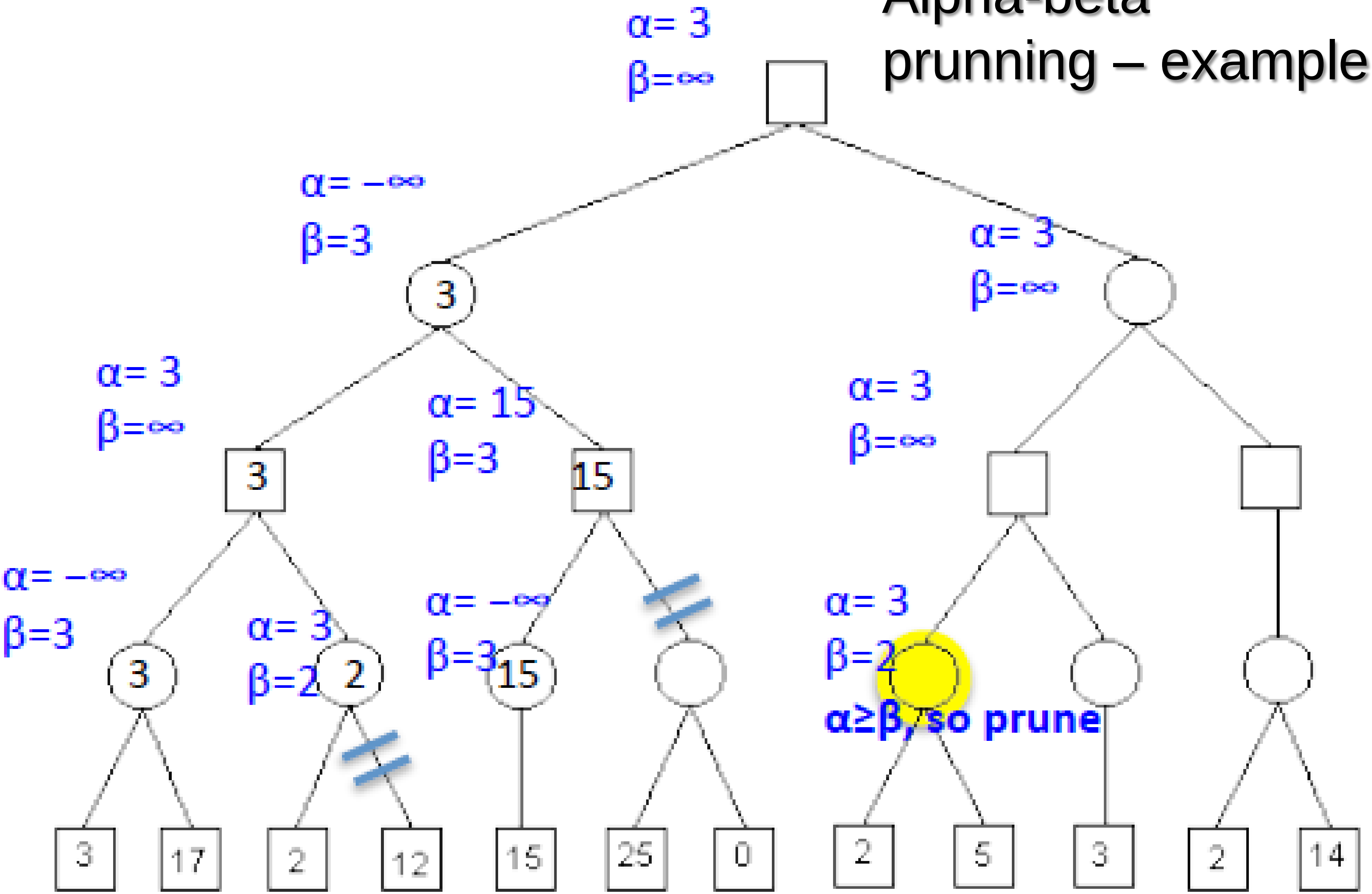
Alpha-beta prunning – example



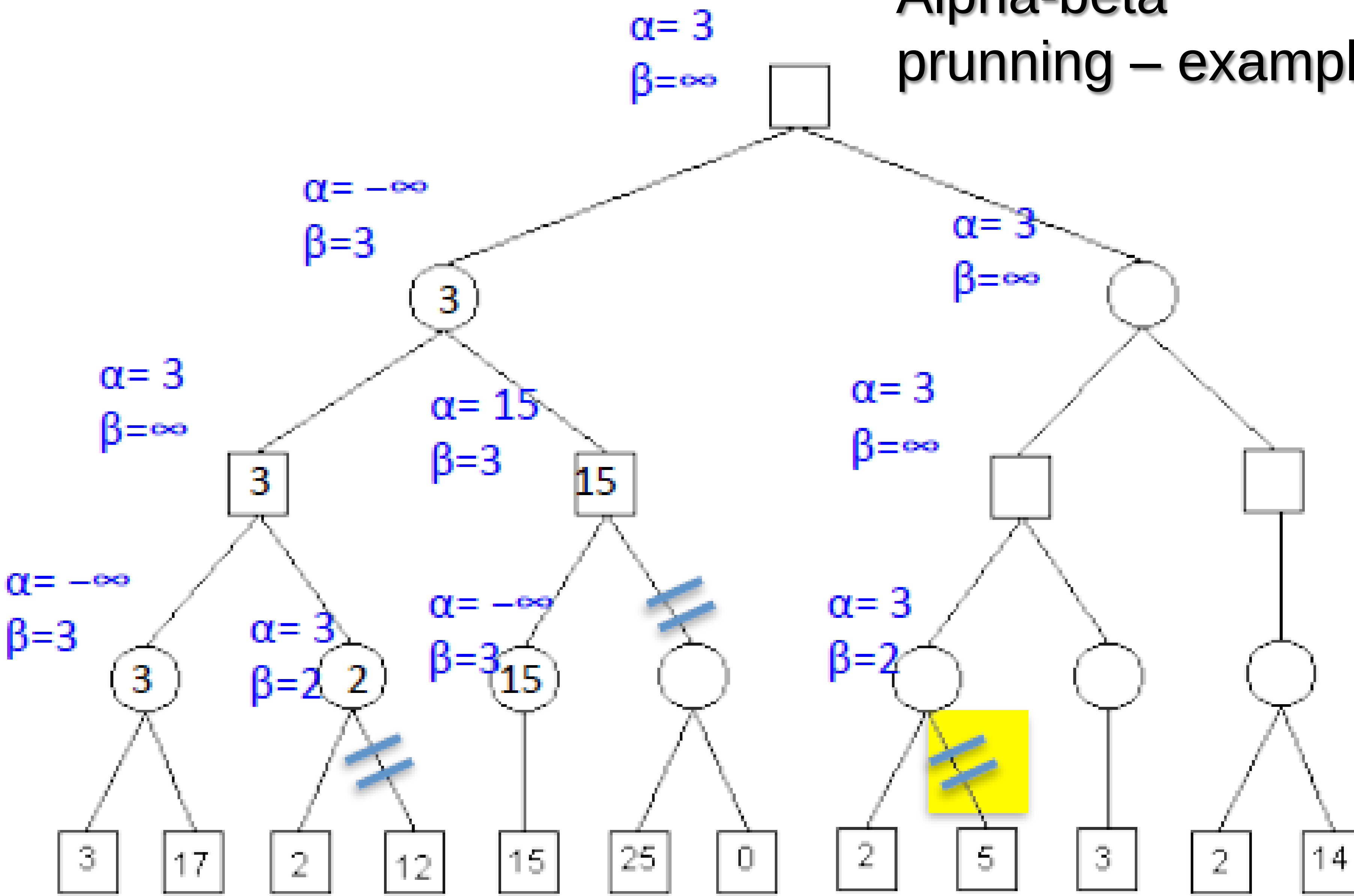
Alpha-beta prunning – example



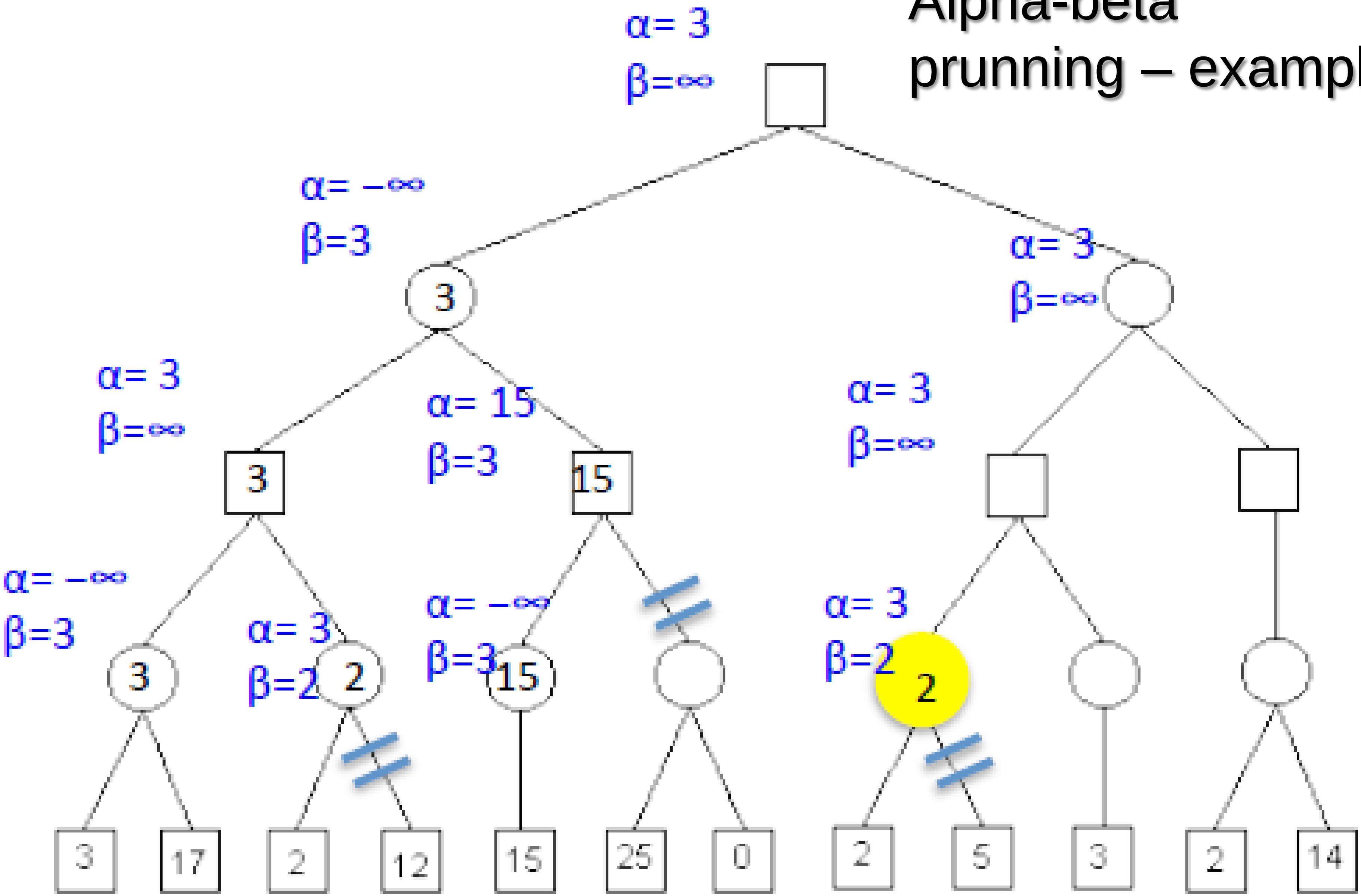
Alpha-beta prunning – example



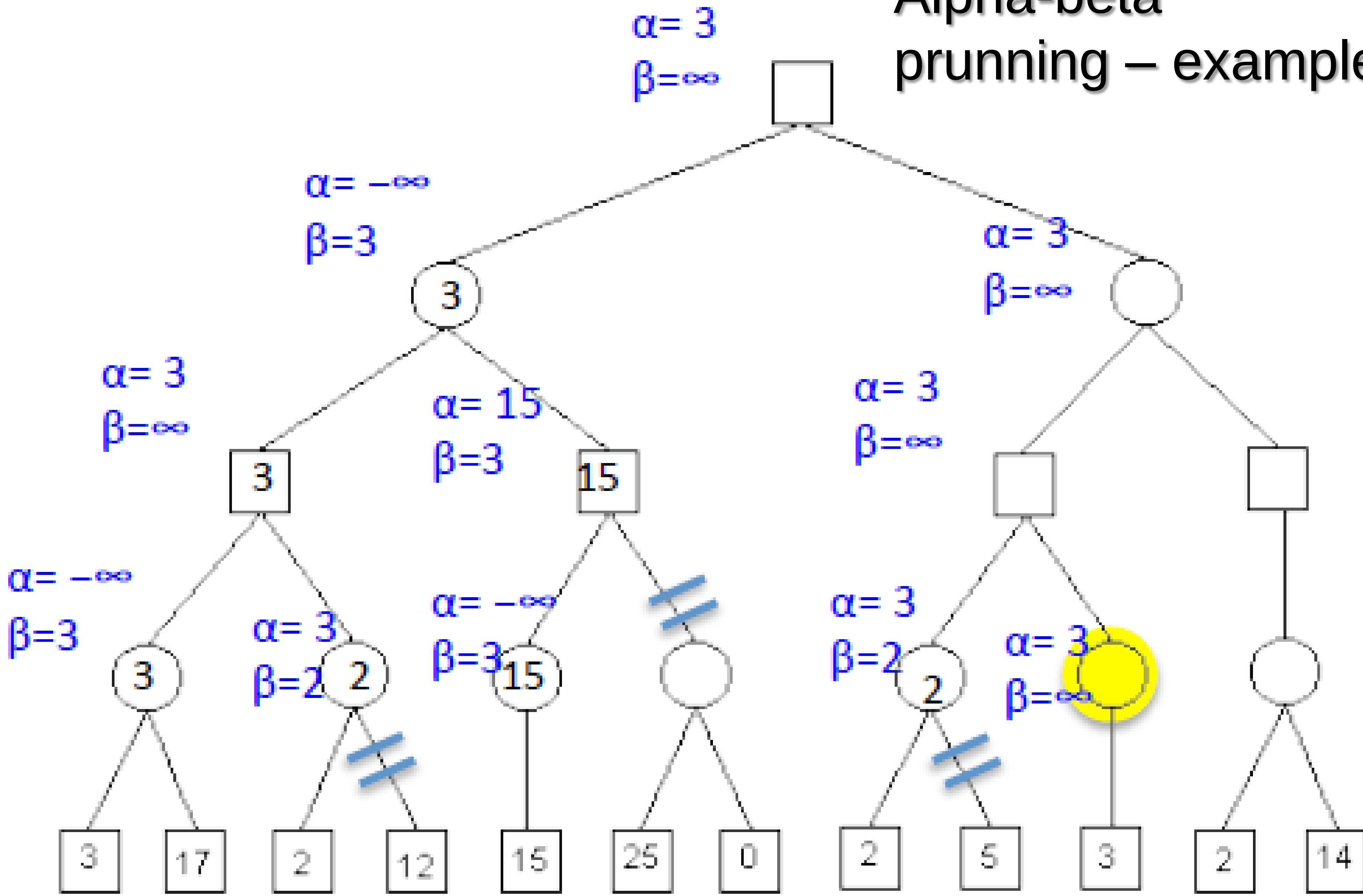
Alpha-beta prunning – example



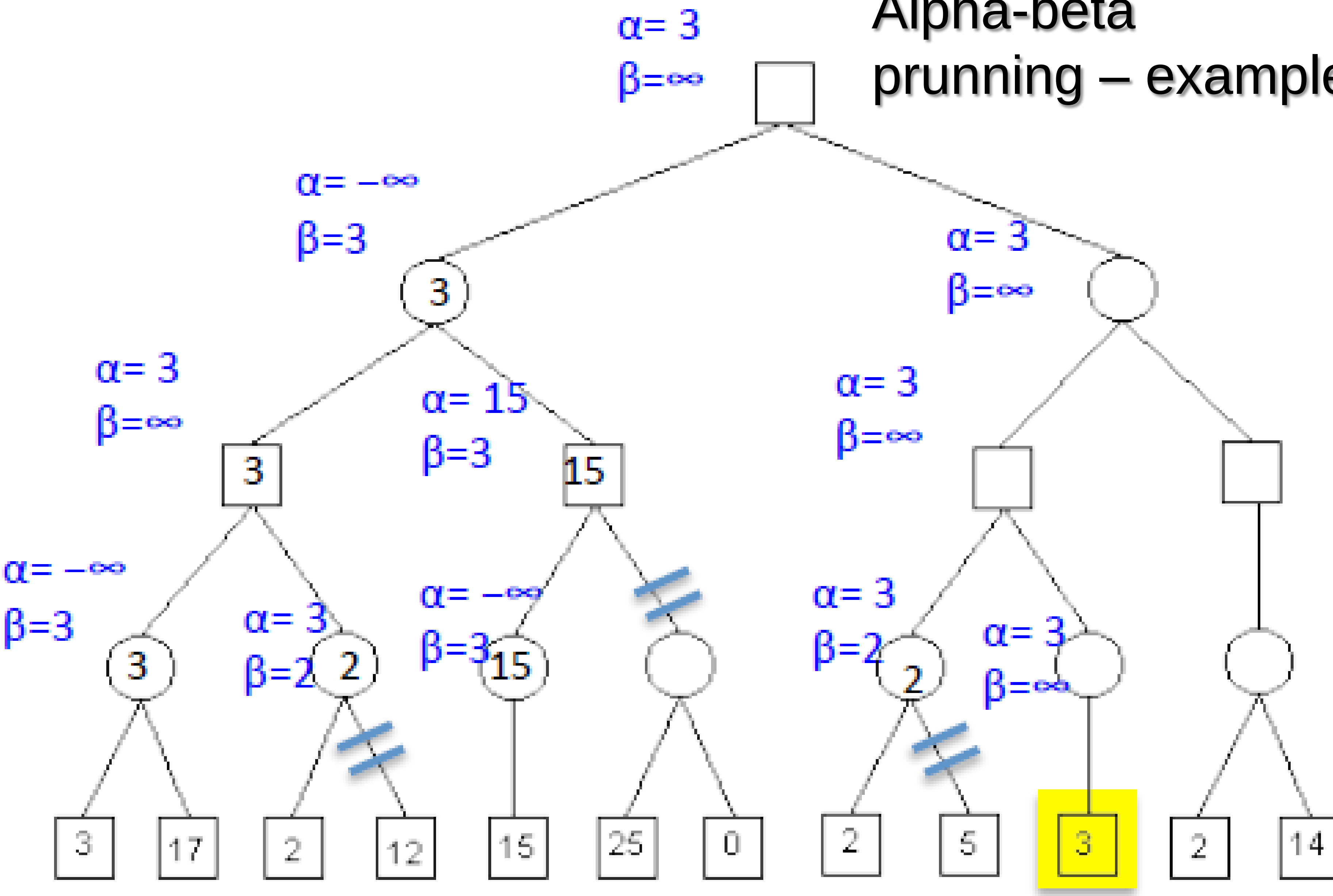
Alpha-beta prunning – example



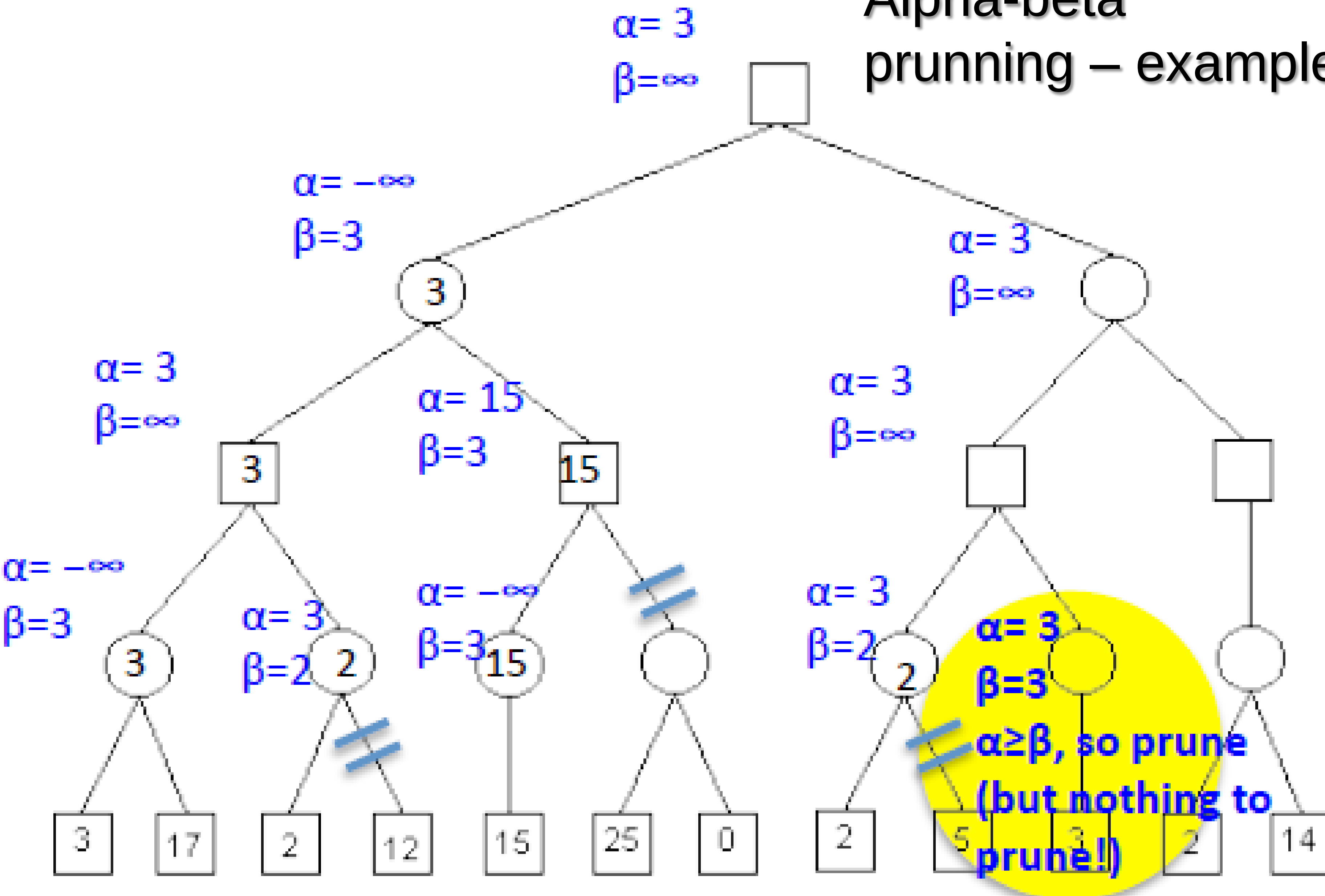
Alpha-beta prunning – example



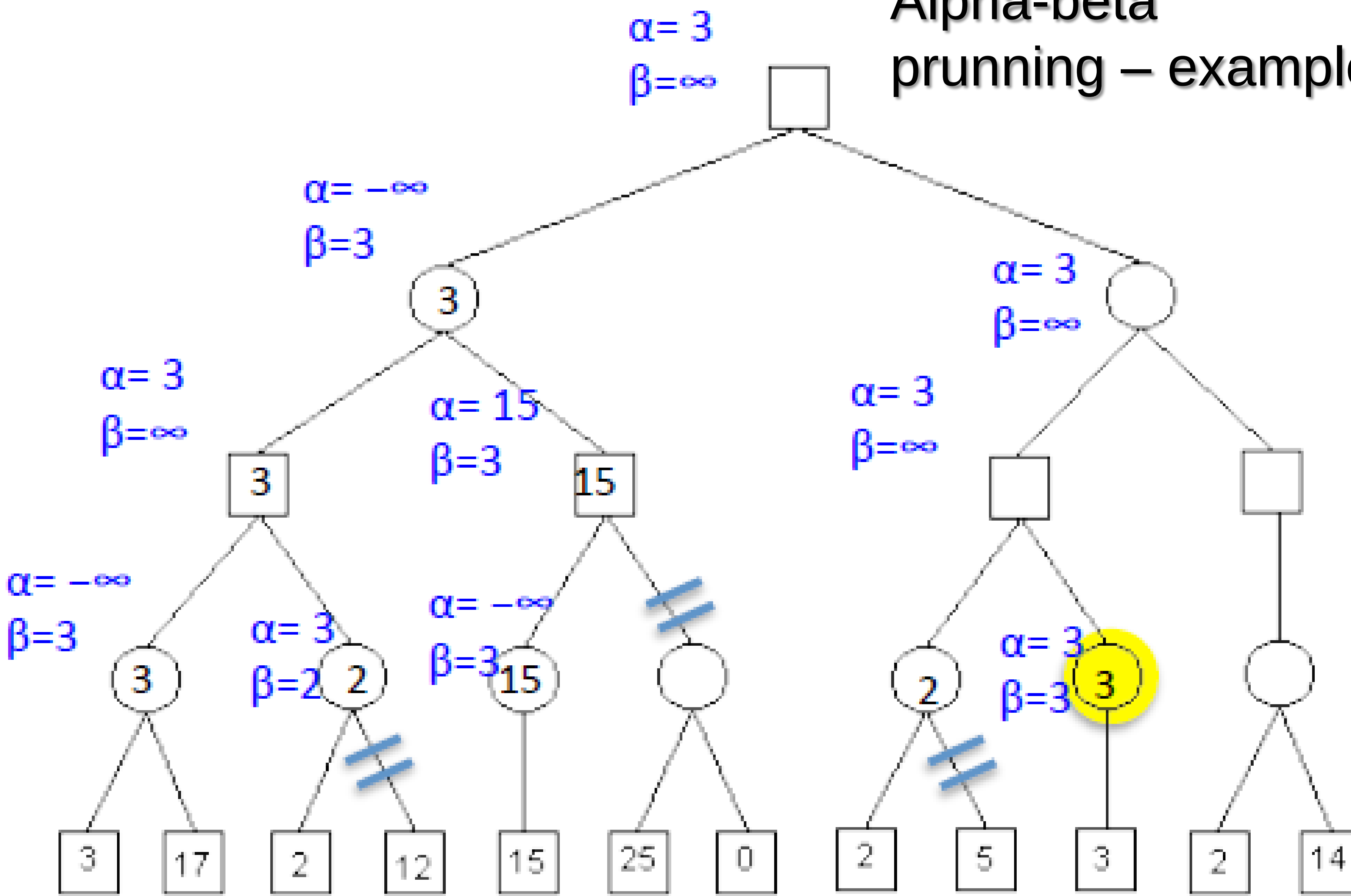
Alpha-beta prunning – example



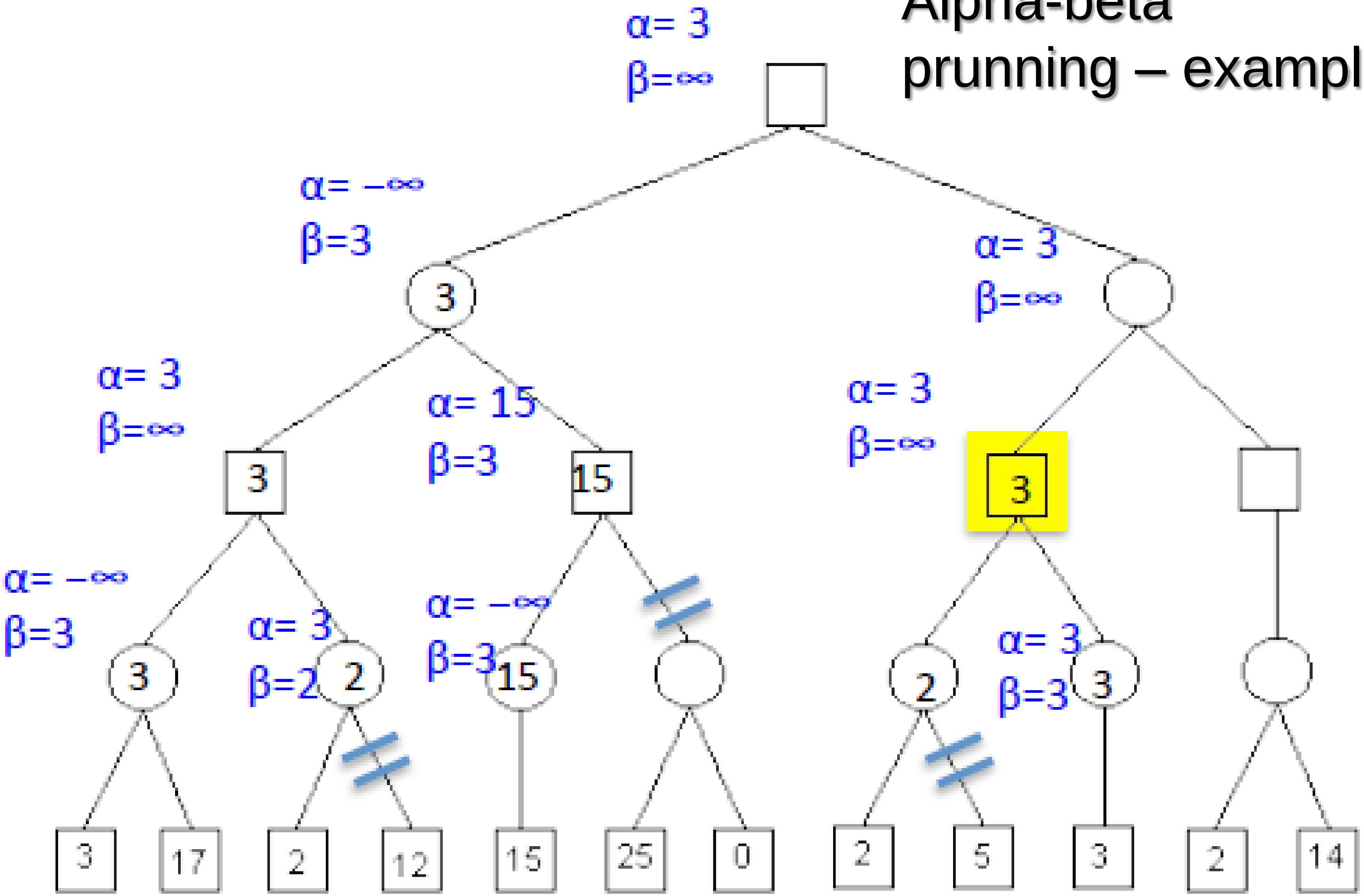
Alpha-beta prunning – example



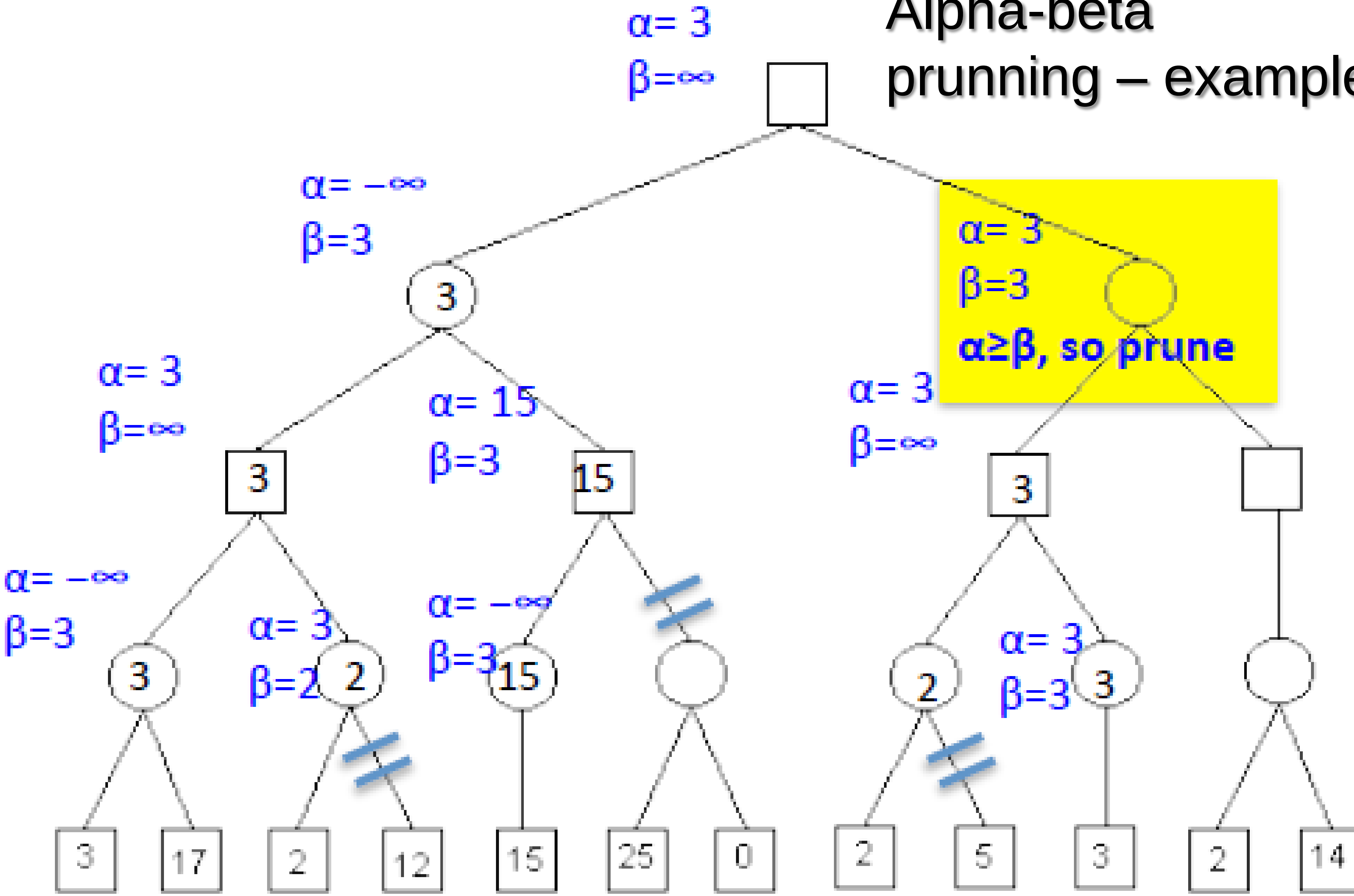
Alpha-beta prunning – example



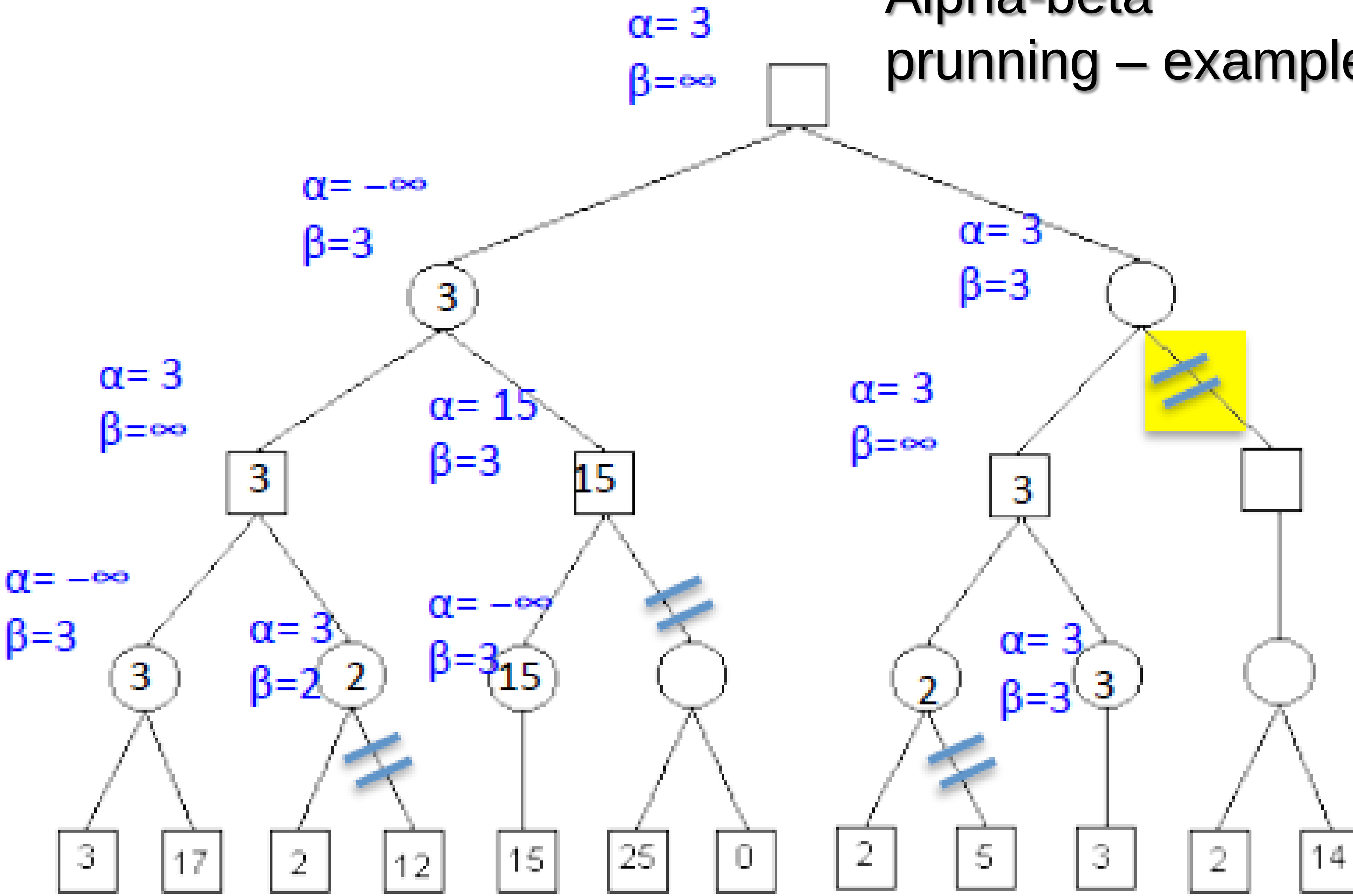
Alpha-beta prunning – example



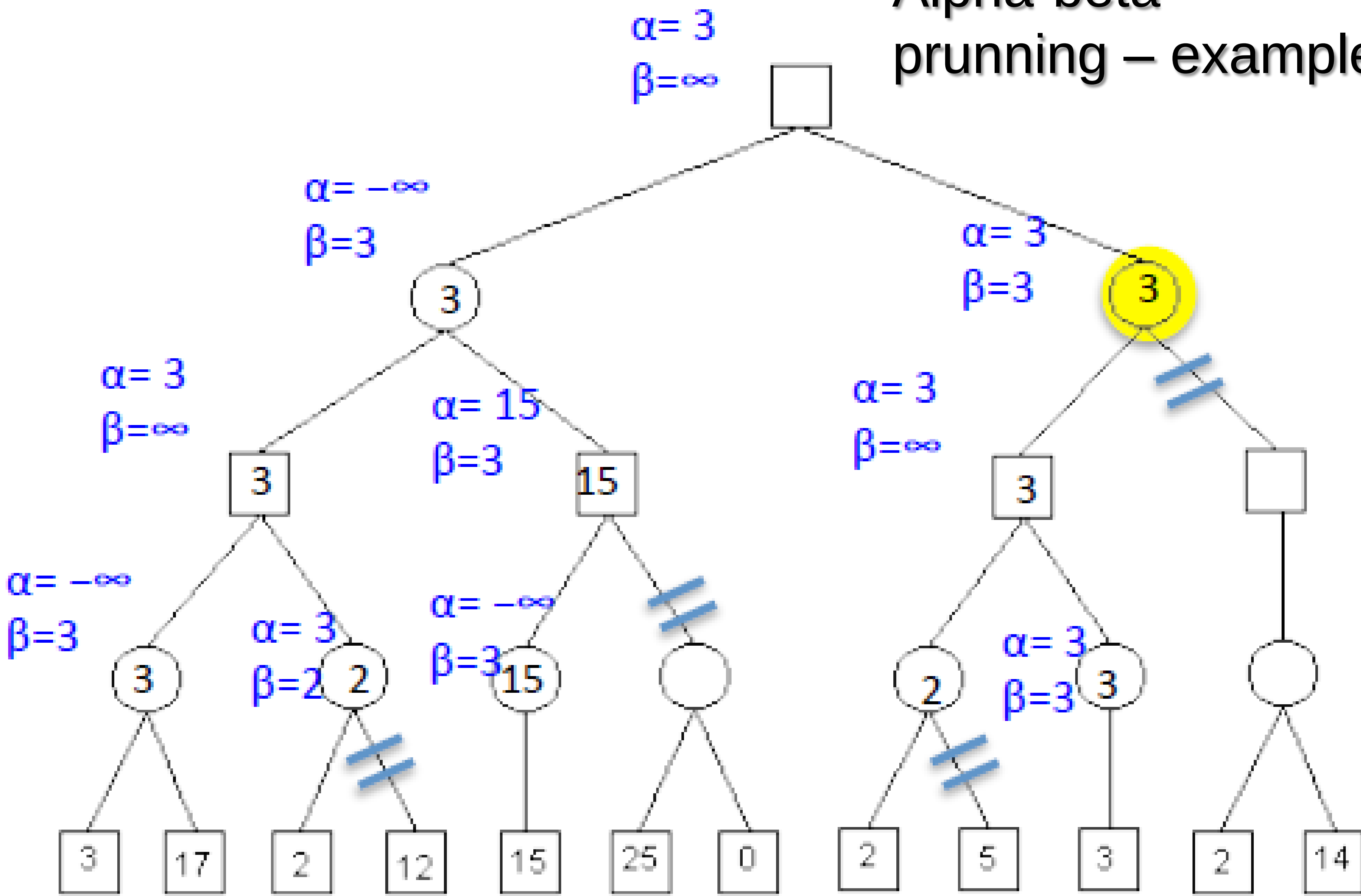
Alpha-beta prunning – example



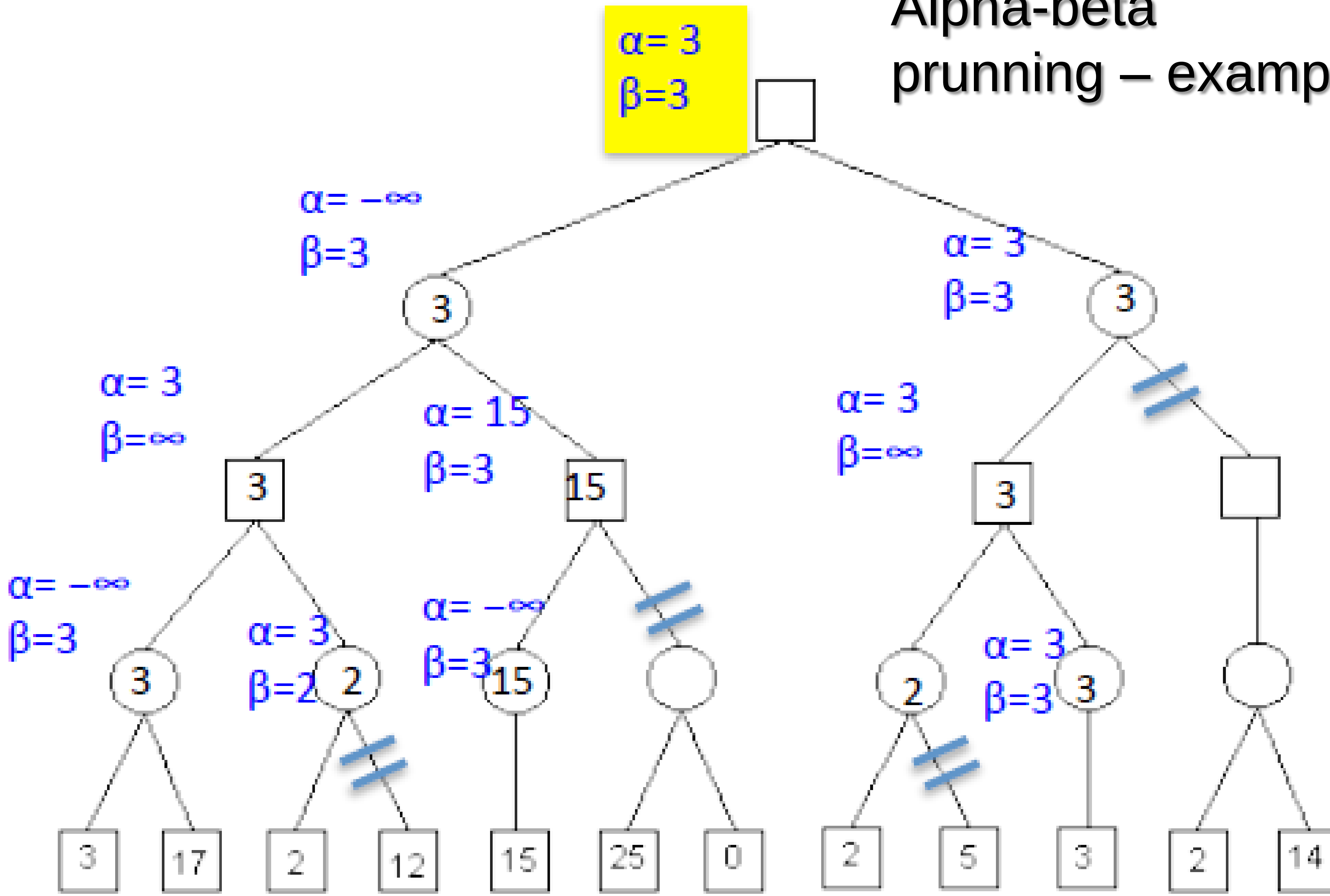
Alpha-beta prunning – example



Alpha-beta prunning – example



Alpha-beta prunning – example



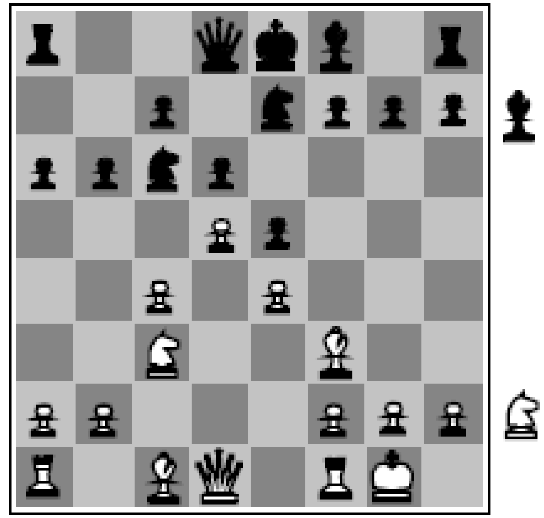
Evaluation functions

State → number mapping. The larger the number, the more valuable the position.
Qualifying a given state – checking the search tree up do a given depth, where the leaves are not the final states of the game, but some intermediate game states, qualified by the evaluating function.

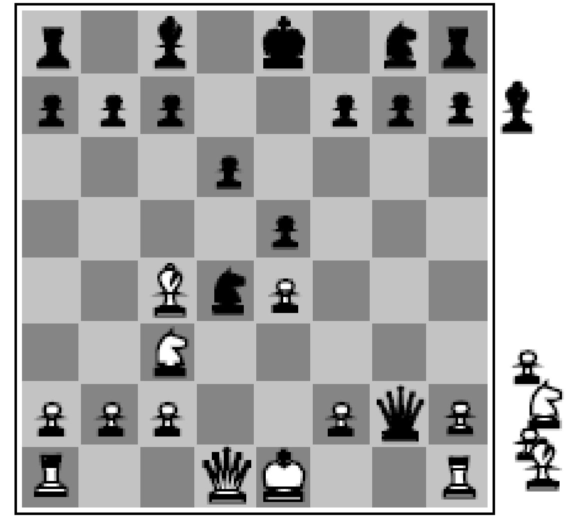
At the beginning of the game (far from the final state) the evaluating function is inaccurate, must be coupled with a search to a certain depth.
At the end of the game (close to the final state) the evaluating function can be fairly accurate, enough to qualify the state without any search. alkalmazható.

$$\mathbf{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s) = \sum_{i-1}^{n} w_i f_i(s)$$

Evaluation functions



Black to move
White slightly better



White to move
Black winning

1957: Herbert Simon

- “within 10 years a computer will beat the world chess champion”
- 1997: Deep Blue beats Kasparov

  Parallel machine with 30 processors for “software” and 480 VLSI processors for “hardware search”

  Searched 126 million nodes (chess positions) per second on average

  Generated up to 30 billion positions per move

  Reached depth 14 routinely

  Uses iterative-deepening alpha-beta search with transpositioning

  Can explore beyond depth-limit for interesting moves

Evaluation functions

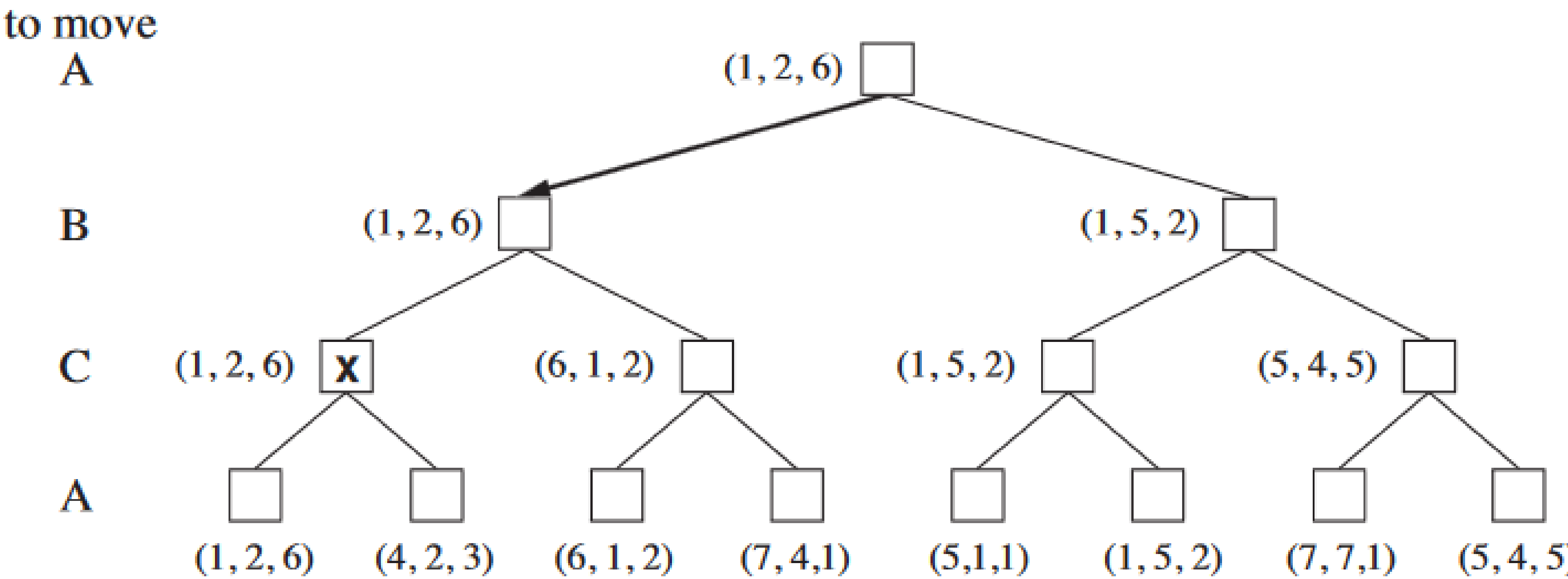
Deep Blue I – Deep Blue II, 6400 – 8000 features (chess chip)

Deep Blue
Murray Campbell, A. Joseph Hoane Jr., Feng-hsiung Hsuc
IBM T.J. Watson Research Center
Sandbridge Technologies
Compaq Computer Corporation
Artificial Intelligence 134 (2002) 57–83

Appendix A. Evaluation tables and registers

http://www.sciencedirect.com/science/article/pii/S0004370201001291

Optimal decisions in multiplayer games

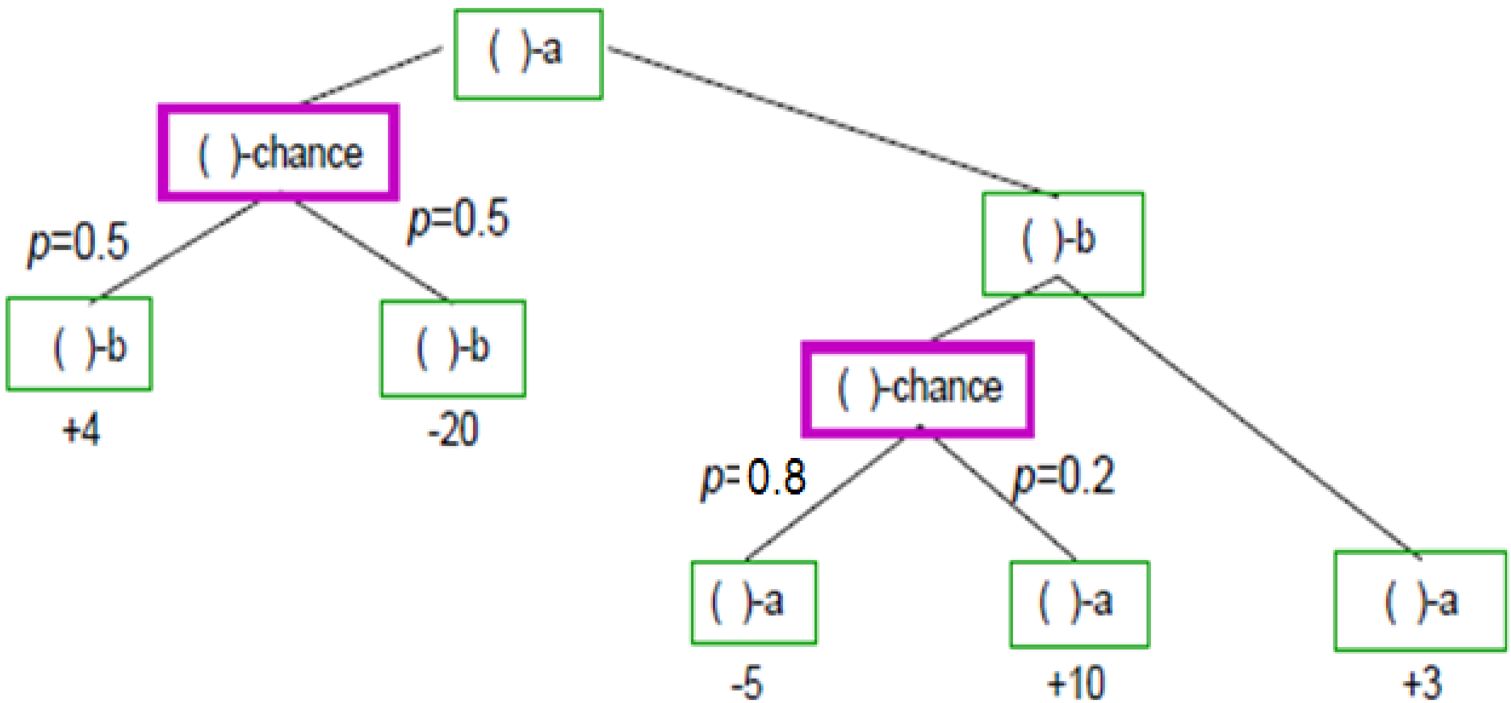


Aliances, collaboration?

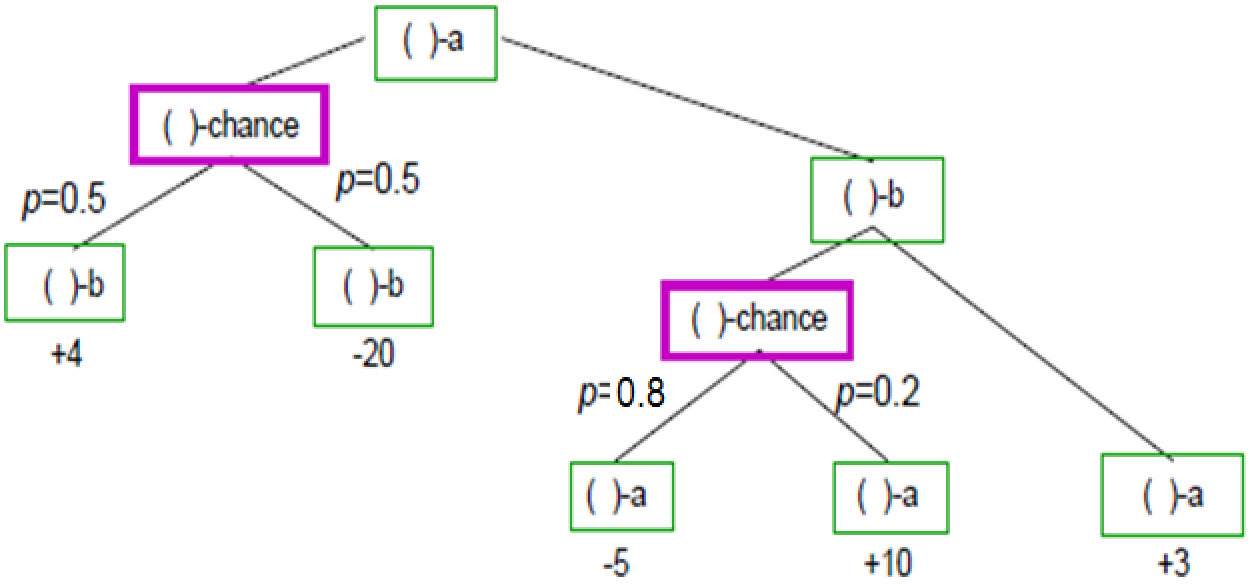
Element of Chance - Expectiminimax

Decision states of the players + the random state of the „nature" = chance nodes, with the probabilities characteristic to the problem.

Evaluation of a state = final expected value. ...

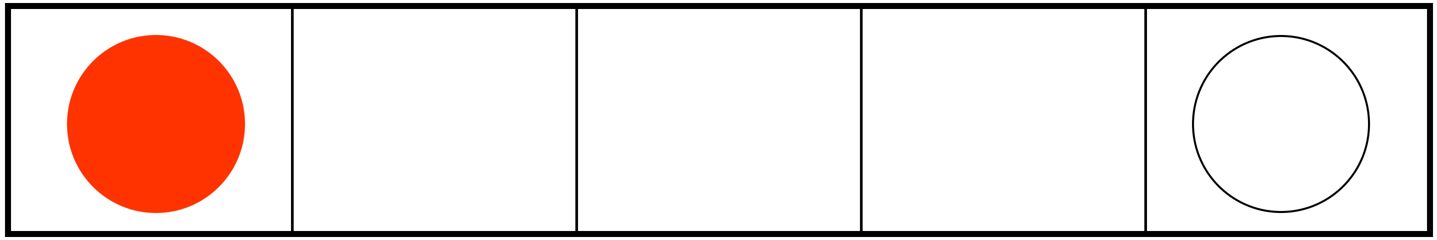


Element of Chance - Expectiminimax



$$\text{EXPECTIMINIMAX}(s) = \begin{cases} \text{UTILITY}(s) & \text{if TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s,a)) & \text{if PLAYER}(s) = \text{MIN} \\ \sum_r P(r)\text{EXPECTIMINIMAX}(\text{RESULT}(s,r)) & \text{if PLAYER}(s) = \text{CHANCE} \end{cases}$$

A small excercise



Rules of the game:

1. Red moves first.
2. Everyone is in zugzwang.
3. A player can move to a neighbouring place, if empty, or can jump over the adversary, if the immediate place behid it is empty.
4. The winner is that who reaches the starting place of the adversary first.
5. Let the red win cost +1, and the white win -1. Compute with the minimax algorithm the value of the root (i.e. who is surely the winner)! Take care of the loops.

Summary

- Game playing can be effectively modeled as a search problem
- Game trees represent alternate computer/opponent moves
- Evaluation functions estimate the quality of a given board configuration for the Max player.
- Minimax is a procedure which chooses moves by assuming that the opponent will always choose the move which is best for them
- Alpha-Beta is a procedure which can prune large parts of the search tree and allow search to go deeper
- For many well-known games, computer algorithms based on heuristic search match or out-perform human world experts.