TWO PLAYER GAMES

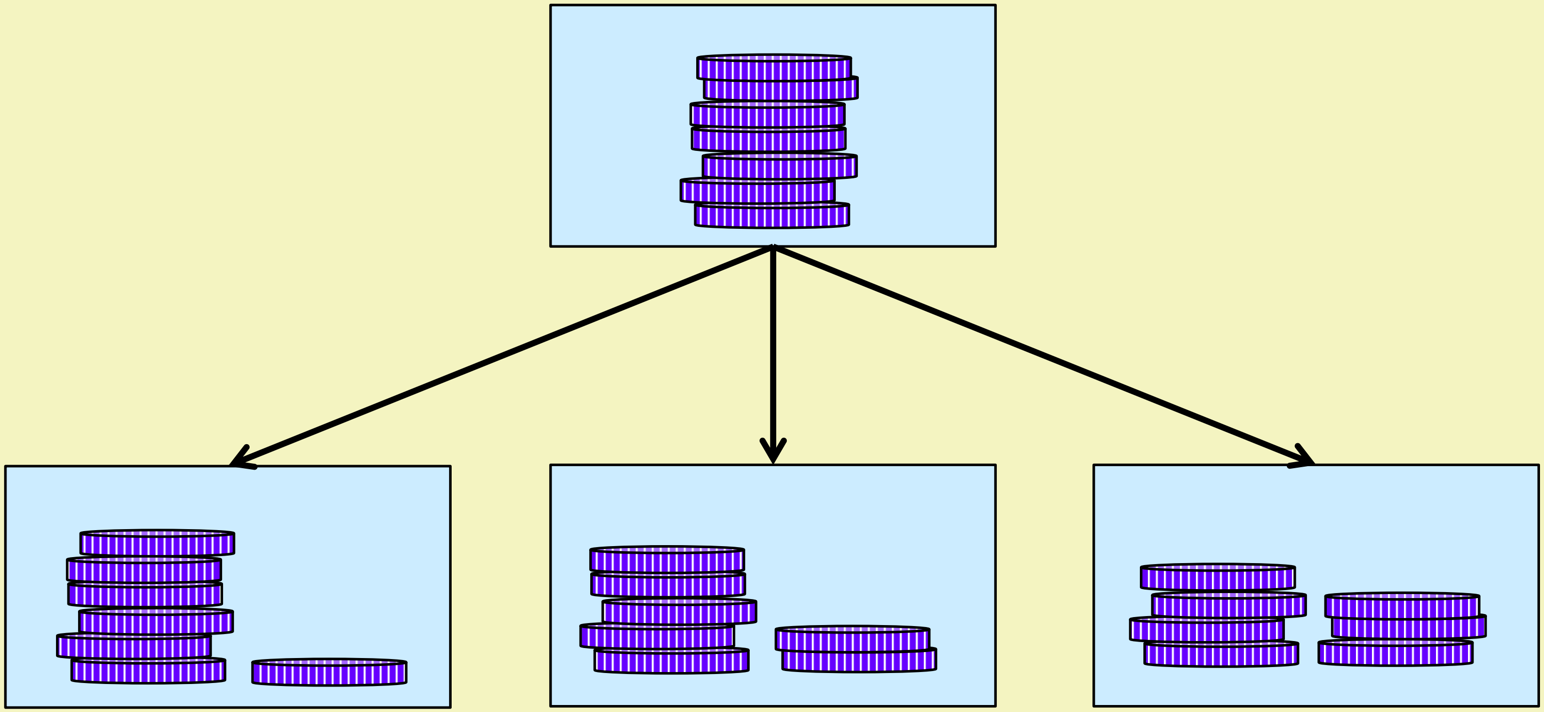
Two-person, turn-taking, perfect-informed, finite and deterministic, zero-sum games

- Two players take turns according to given rules until the game is over.
- The game is in a fully observable environment, i.e., the players know completely what both players have done and can do.
- Either the number of the possible steps in a current state or the length of the plays of the game are finite.
- Each step is unequivocal, its effect is predictable. The plays of the game do not depend on chance at all.
- The sum of the payoff values of the players at the end of the game is always zero. (In special case players can only win or lose. Sometimes the draw is also possible.)

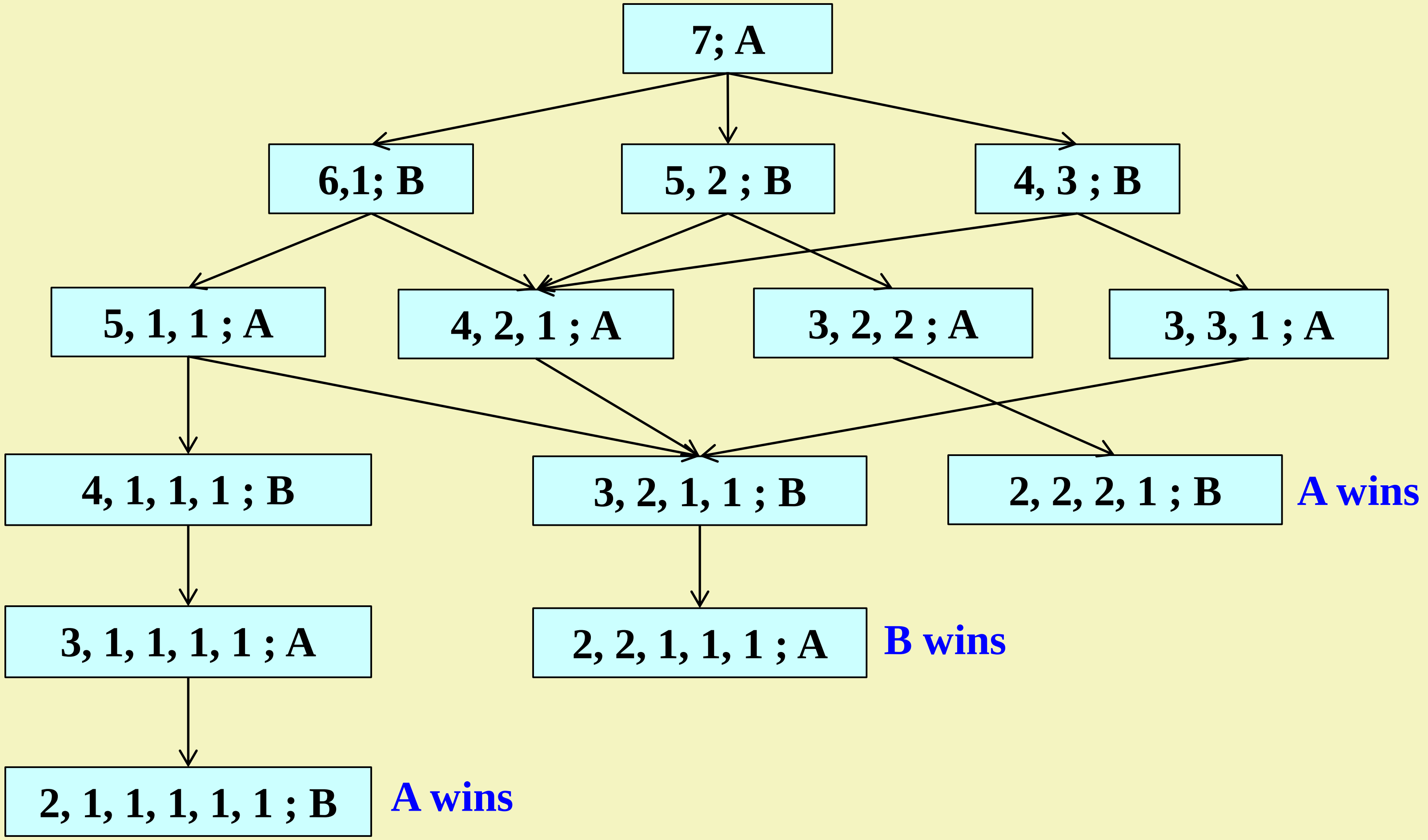
State space model

- state – configuration + player next to move
- operator – legal step
- initial state – initial configuration + first player
- final state – terminal configuration + next player
- two payoff functions: p_A, p_B : final states $\longrightarrow \mathbb{R}$ (players: *A*, *B*)
  - In a zero-sum two-player game: $p_A(t) + p_B(t) = 0$ for all final state *t*
  - In special case the range of these functions: +1, 0, –1
    - +1 if the player wins (winning final state for the very player)
    - –1 if the player loses (losing final state for the very player)
    - 0 if the final state is a draw
- One play of the game is a sequence of operators (moves) driving from the initial state to a terminal state.

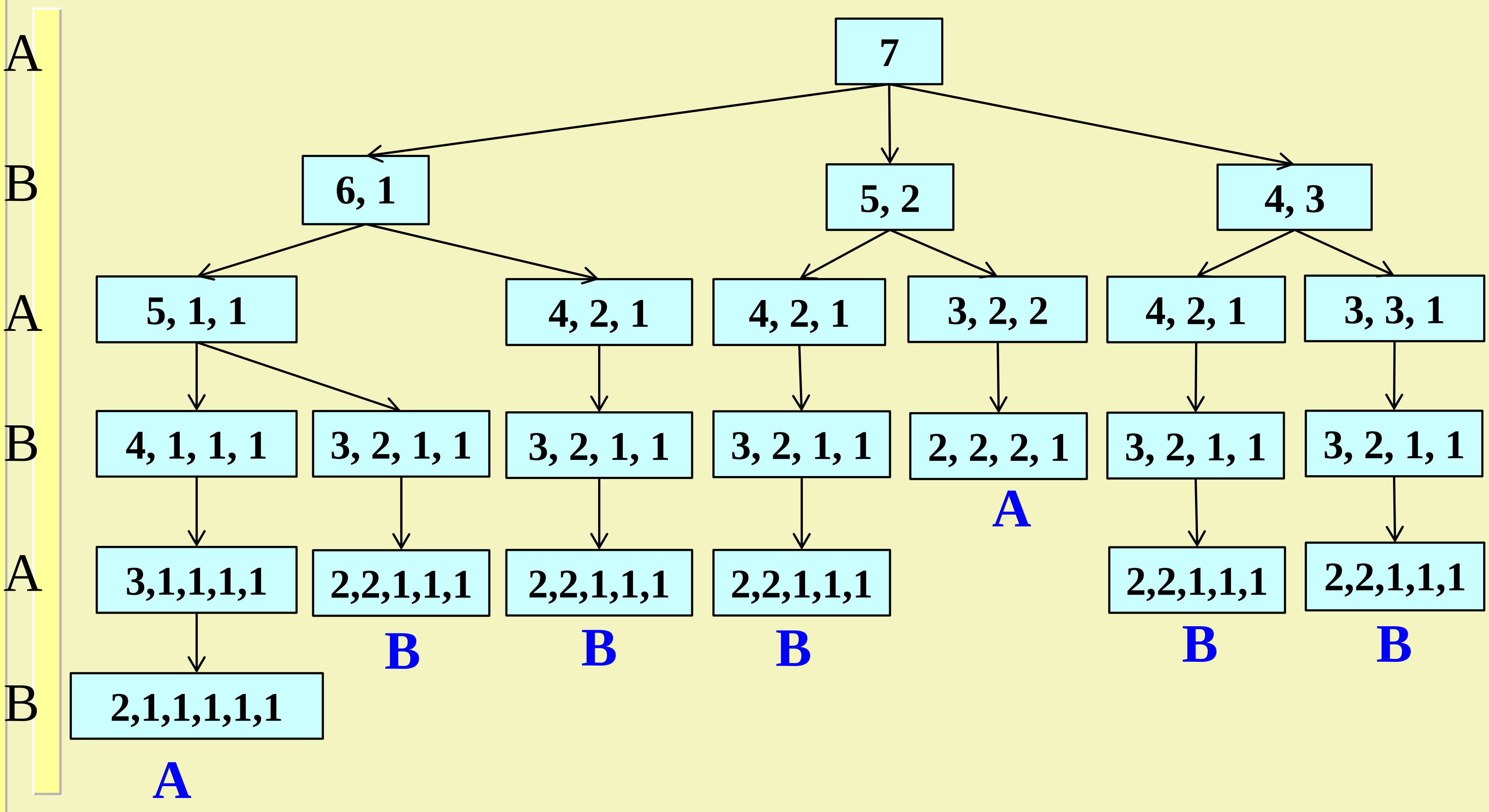
Grundy mum's game



Grundy mum's state graph



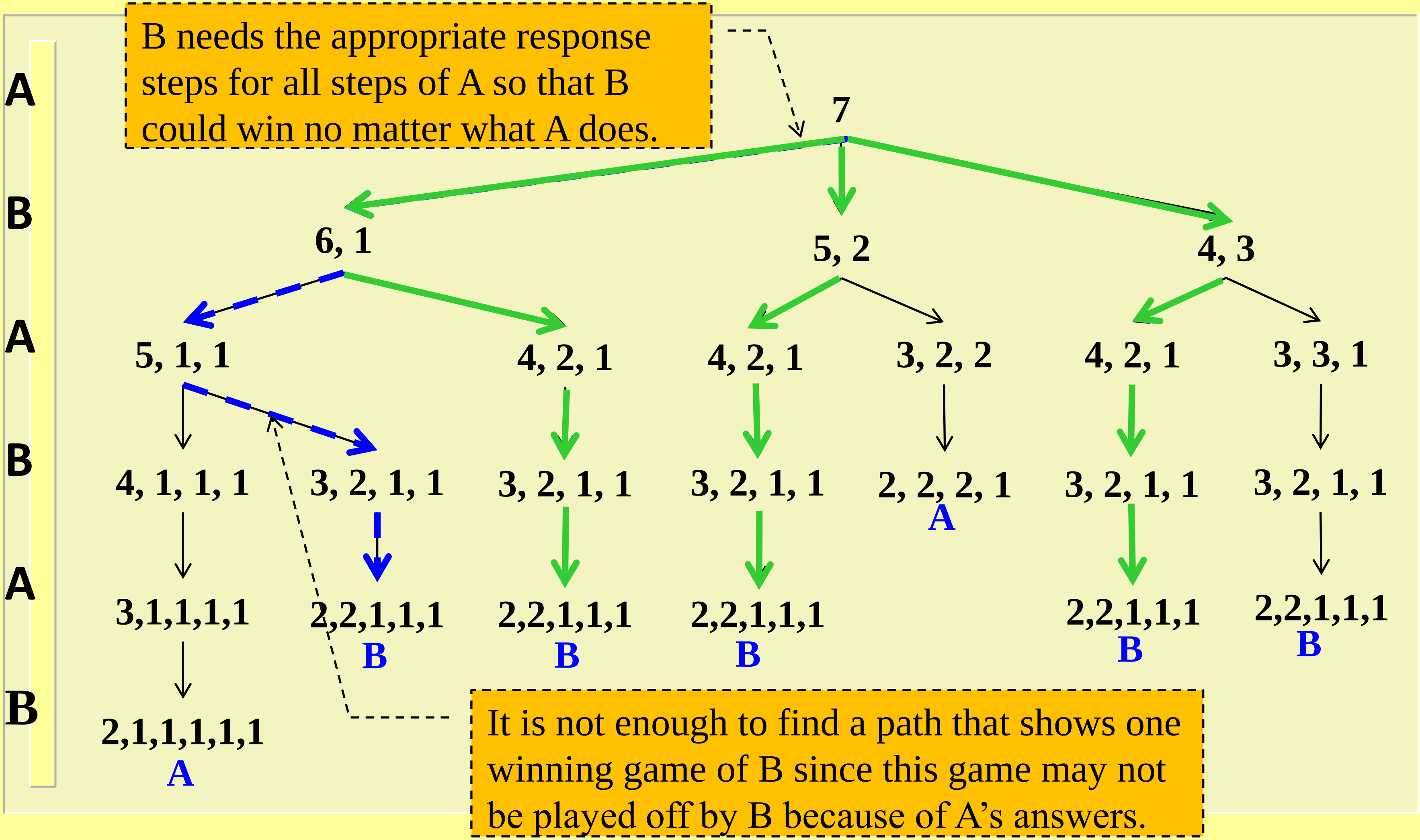
Grundy mum's game tree



Game tree

- node – configuration
  (the same configuration may occur in several nodes)
- level – player (the levels of A and B alternate)
- arc – step (level by level)
- root – initial configuration
- leaf – terminal configuration
- branch – play of the game

How can the player B win?



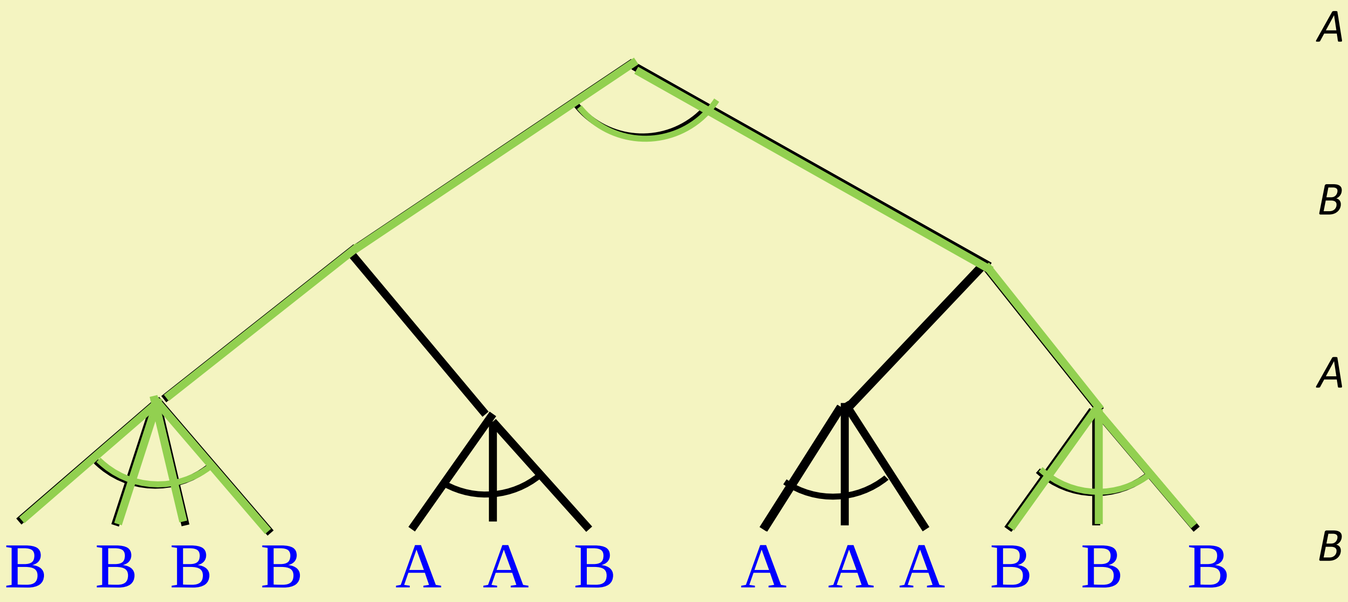
Winning strategy

- The winning strategy shows how a player could win no matter what the opposite player does.
- The winning strategy is not one play of the game but a beam of plays leading to win, and one of these plays can be realized by the player who has got this winning strategy.
- A non-losing strategy (that guarantees at least a tie game) would be useful if draw is possible.

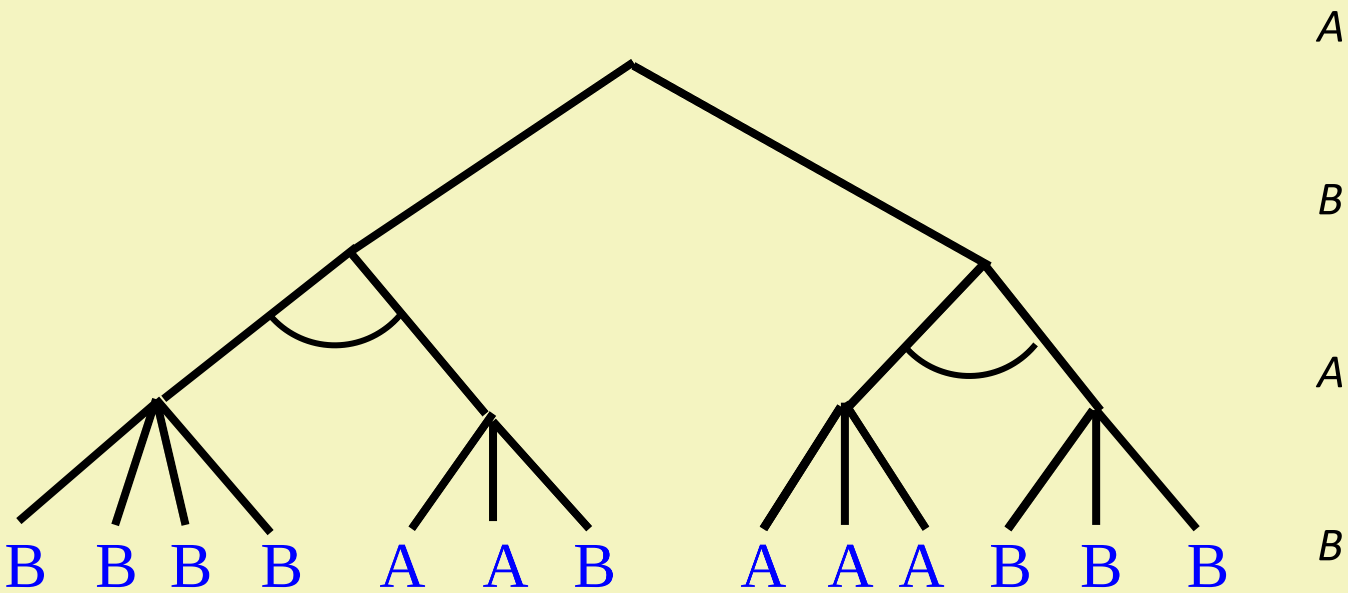
Remarks

- A game tree can be interpreted by the players in different way and these interpretations can be drawn with AND/OR trees.
  - there is OR connection between the arcs going from the nodes on the level of the very player
  - there is AND connection between the arcs going from the nodes on the level of the opponent player
- Both players have got the own AND/OR tree.
- The winning (or non-losing) strategy of one player is a hyper-path of his/her AND/OR tree that is driving from the root to winning goal nodes.
- The search of a winning strategy is a hyper-path-finding problem in an AND/OR tree.

Searching for winning strategy in the AND/OR tree of the player **B**



Searching for winning strategy in the AND/OR tree of the player A

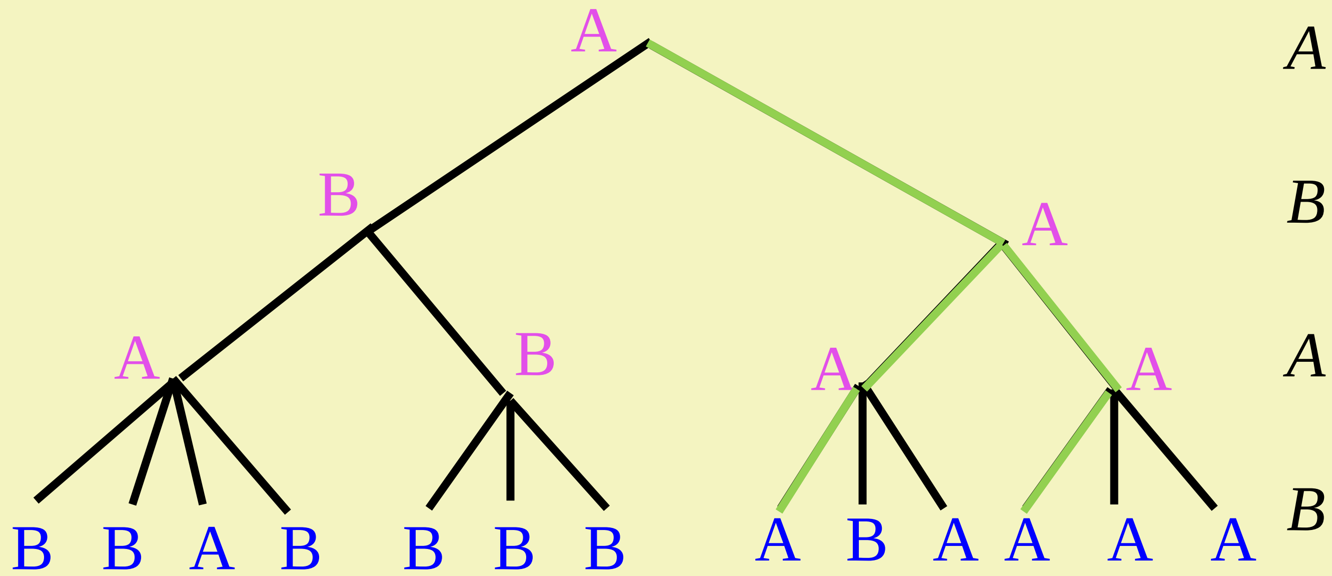


There is no winning strategy for A.

Only one player may have got winning strategy.

Theorem

- In each two-player, perfect-informed, finite and deterministic games where there are two outcomes (victory or defeat) one of the players certainly has a winning strategy.



- If the draw is also possible: non-losing strategy of one the players is guaranteed.

Subtree evaluation

- Finding the winning or non-losing strategy is hopeless in the larger game tree.
- The methods are investigated that can suggest a good next step instead of searching for a winning strategy.
- These methods build up a subtree of the game tree starting from the current state and try to estimate the beneficial of the leaf nodes of this subtree (using a heuristic evaluation function) and thereafter calculate a good next step based on these values.

Evaluation function

- The evaluation function can measure the beneficial of the states from our point of view against our opponent.

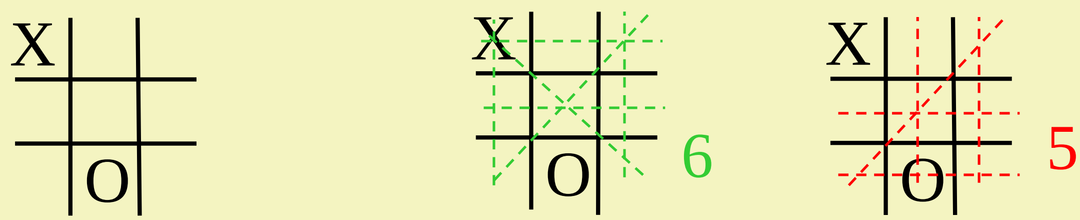
$$f : \text{States} \rightarrow [-1000 \,,\, 1000]$$

- Examples:
  - Chess: (evaluation function for white player)

    $f(s)$ = (number of the white queens) – (number of the black queens)
  - Tic-tac-toe: $f(s) = M(s) - O(s)$

    $M(s)$ = number of My (X) free lines

    $O(s)$ = number of Opponent's (O) free lines

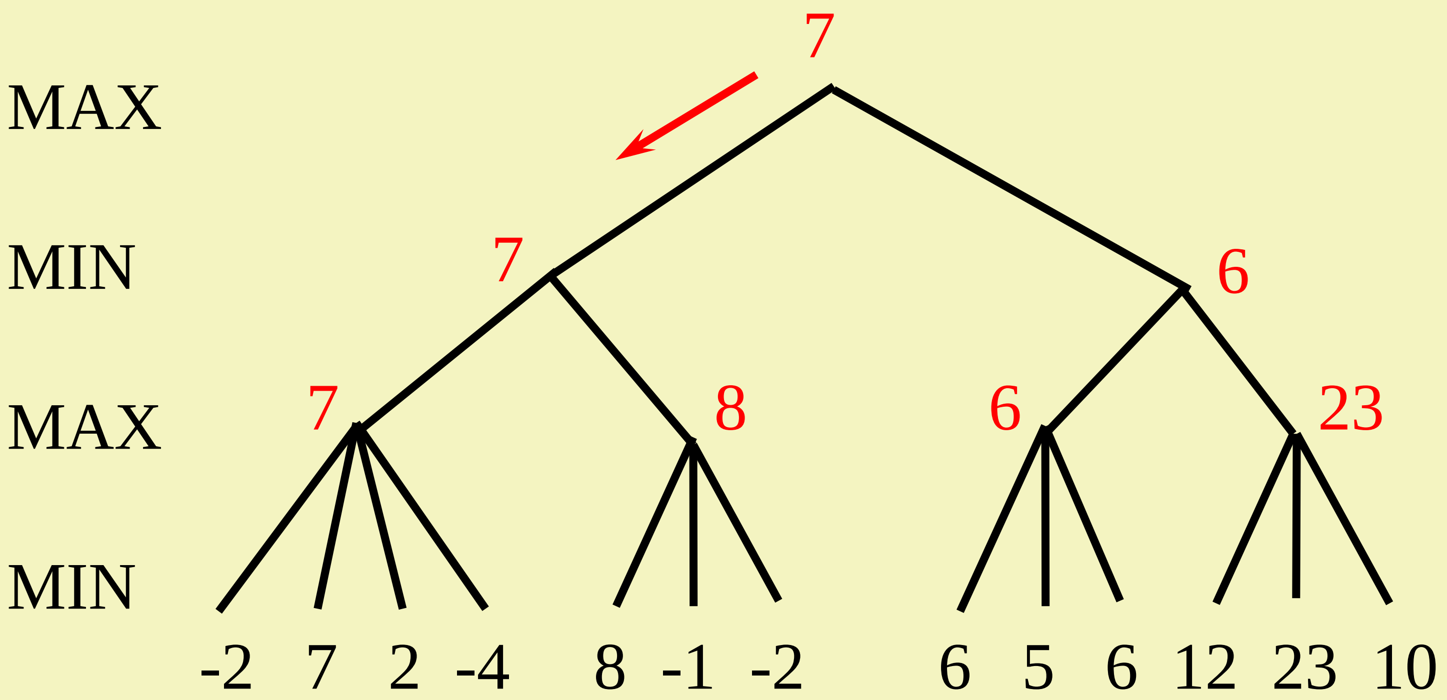


The minimax algorithm

1. Some levels of the game tree are built up starting from the current state (depending on the time or the storage limit).
2. The leaves of this subtree must be evaluated based on the evaluation function.
3. A value can be computed for each inner node
   - this is the maximum of the successors' values if the very node is on our level,
   - this is the minimum of the successors' values if the very node is on the opponent's level.
4. The next step will be towards the successor of the current state which successor has got the largest value.

Example

Let the name of the player representing us be MAX and the name of our opponent be MIN.



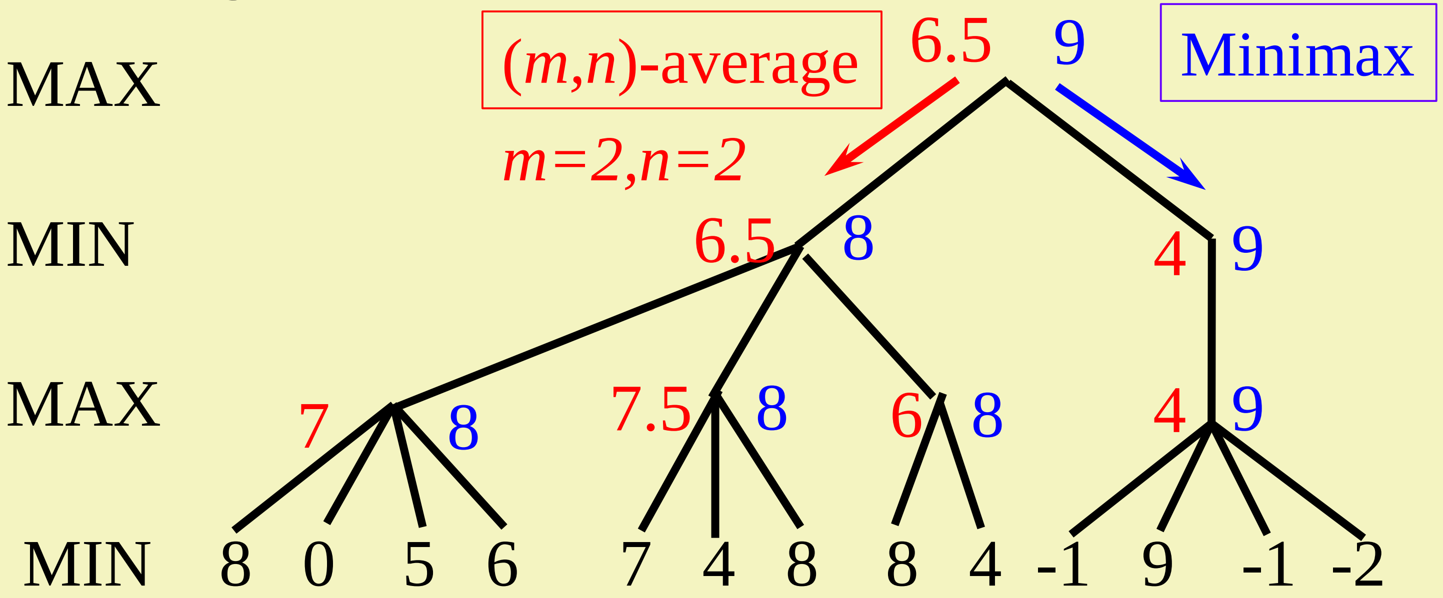
Remark

- The minimax algorithm could not compute our several good steps in advance because the opponent has something to say about it.
- We must repeat this algorithm whenever it is our turn to play since our opponent may not move what we expect.

  - He/she can use other depth bound
  - He/she can use other evaluation function
  - He/she can use other algorithm
  - He/she can miss

(m,n)-average evaluation

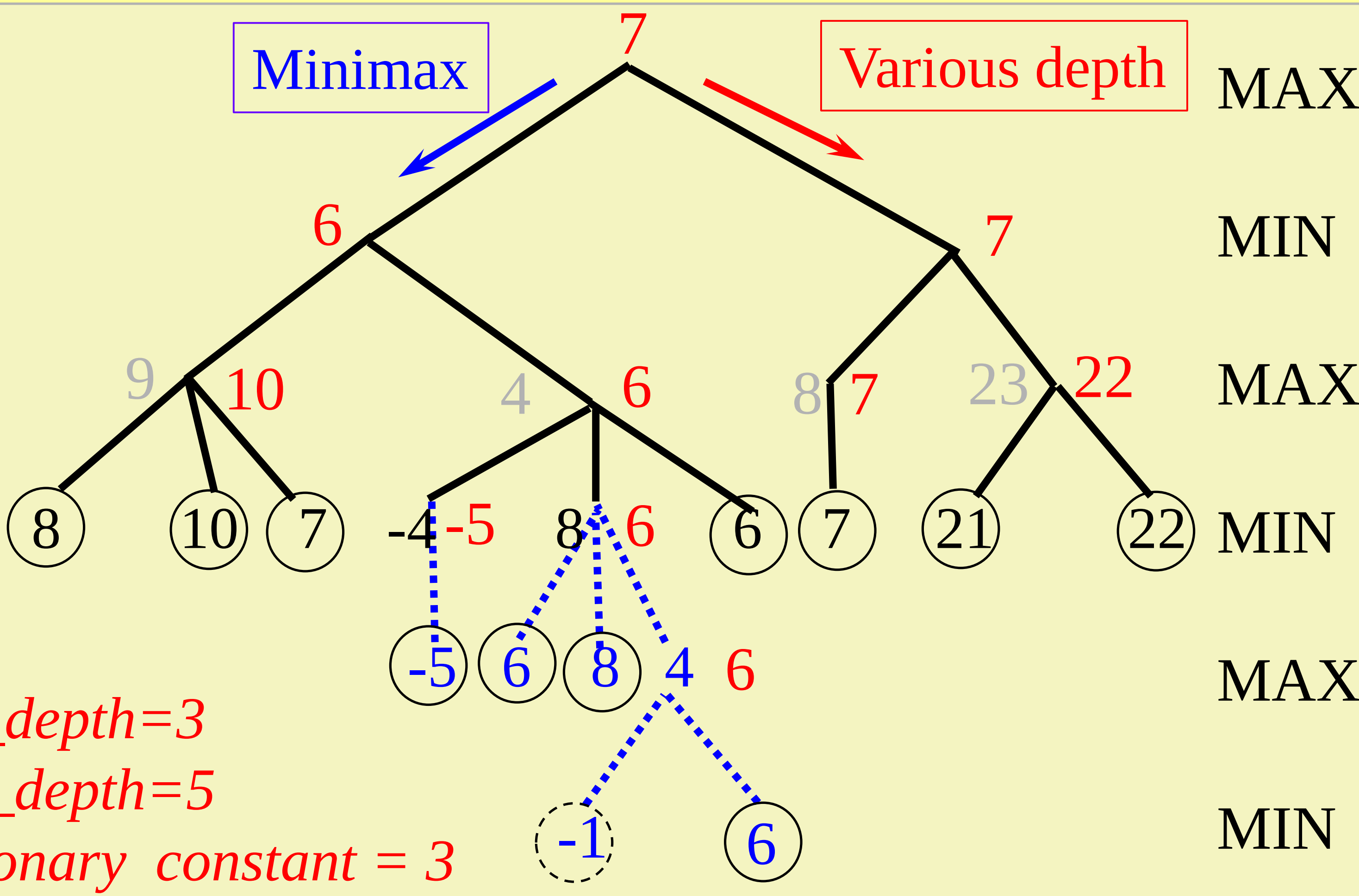
- This method can rectify the miscalculations of the evaluation function. The values of the nodes are
  - the average of the *m* largest child-values on MAX's levels
  - the average of the *n* smallest child-values on MIN's levels



Various depth bound evaluation

- An evaluation value of a node may be misleading if it significantly differs to the value of its parent node :
  $$\left| f(parent) - f(node) \right| > K \quad \text{(stationary test)}$$
- Let us fix two parameters: *min_depth* and *max_depth*
  - all nodes are generated up to the level *min_depth*
  - only the children must be generated between the level *min_depth* and the level *max_depth* which parent is misleading.

Example



min_depth=3
max_depth=5
stationary constant = 3

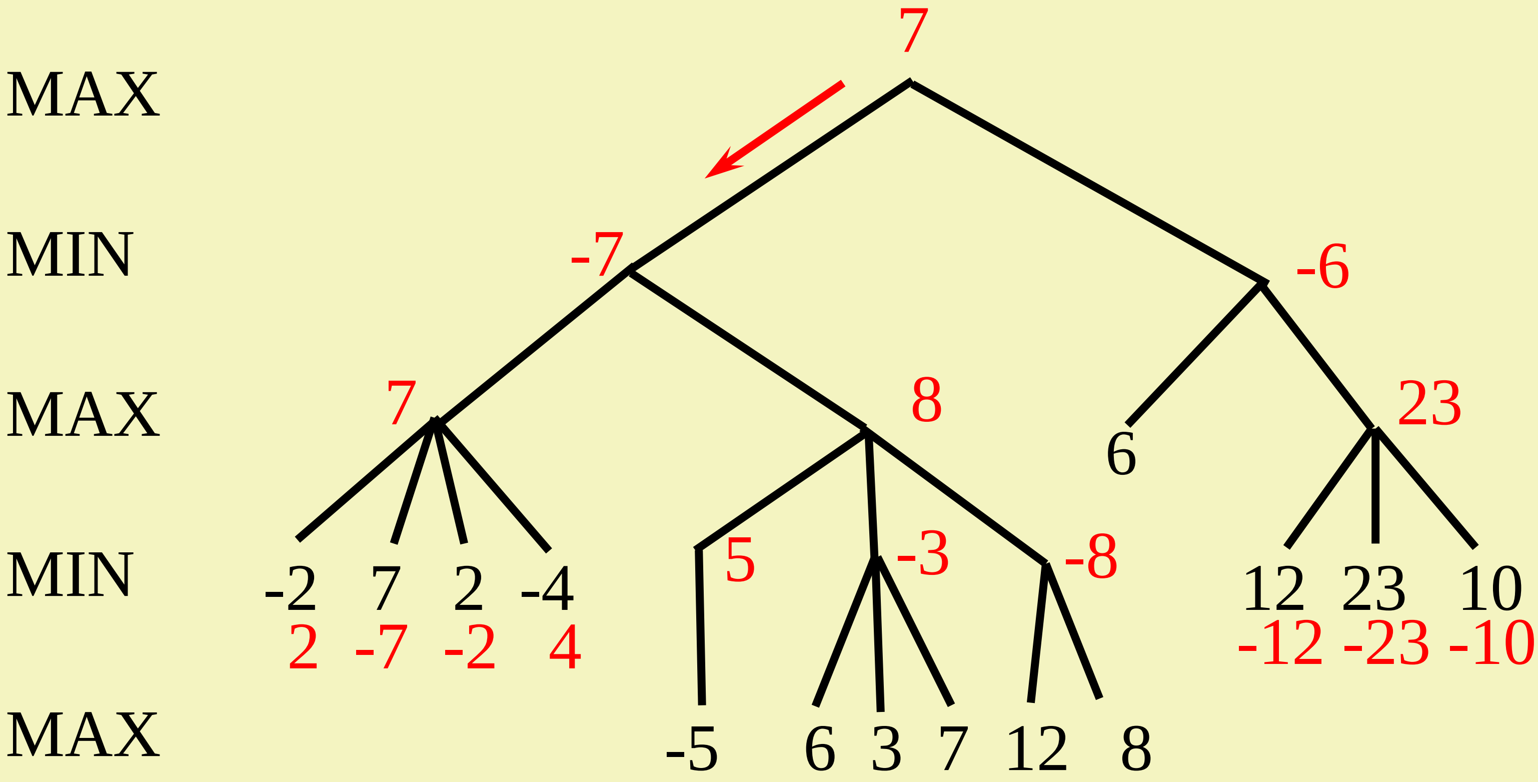
Selecting evaluation

- If the essential steps and the marginal steps can be separated, then it is enough to build up the subtree of the game including only the essential steps.
- This idea reduces the memory space of the evaluation.
- This selection needs some heuristics.

Negamax algorithm

- The implementation of this method is easier than the minimax algorithm.
  - Initially take the negation of the values of the leaf nodes on the opponent's (MIN) levels.
  - Compute the values upwards level by level as $max(-child_1 ,..., -child_n)$

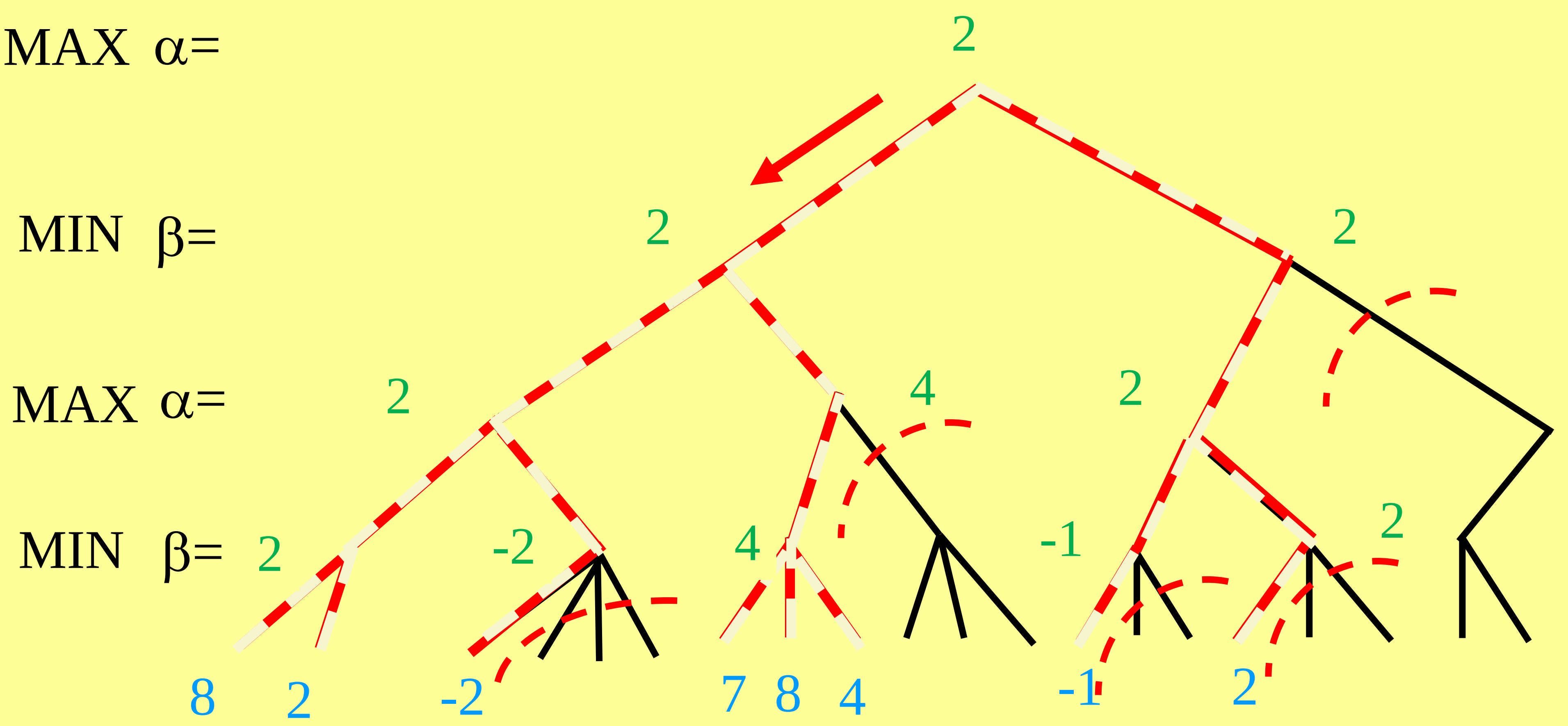
Example


MAX
MIN
MAX
MIN
MAX
7
-7
-6
7
8
6
23
5
-3
-8
-2 7 2 -4
2 -7 -2 4
12 23 10
-12 -23 -10
-5 6 3 7 12 8

Alpha-beta algorithm

- It traverses the subtree according to the backtracking algorithm.
- The nodes of the current path have got temporary values:
  - on MAX's levels: α value (lower limit),
  - on MIN's levels: β value (higher limit)
- Fore step: $\alpha=-\infty$ and $\beta=+\infty$.
- Back step: $\alpha=\max(\alpha, \text{child})$ or $\beta=\min(\beta, \text{child})$
- Cutting rule: if there are an α and β value on the current path so that $\alpha\geq\beta$.

Example



Discussion

- The result of the alpha-beta algorithm is equal to the result of the minimax method. (If several equal values run up to the root, the „left most” direction must be chosen.)
- Memory: only one path.
- Running time: better than minimax because of cutting.
  - Average case: expected value of the number of branches that must be investigated before cutting is only *2*
  - Optimal case: the number of the leaves evaluated: $\sqrt{b^d}$ where the branching factor is b and the depth is d.

Two player game software

- Let us use a negamax algorithm based on selecting the essential steps, various depth bound and average evaluation with alpha-beta pruning.
- A frame program is needed that accepts the steps of the user and generates the steps of the computer.
- Special features must be built in (initial settings, help, hints, saving and reloading games etc.)
- Graphical user interface is very important.
- These are worth nothing without good heuristics (in the evaluation function and the selection of the essential moves).