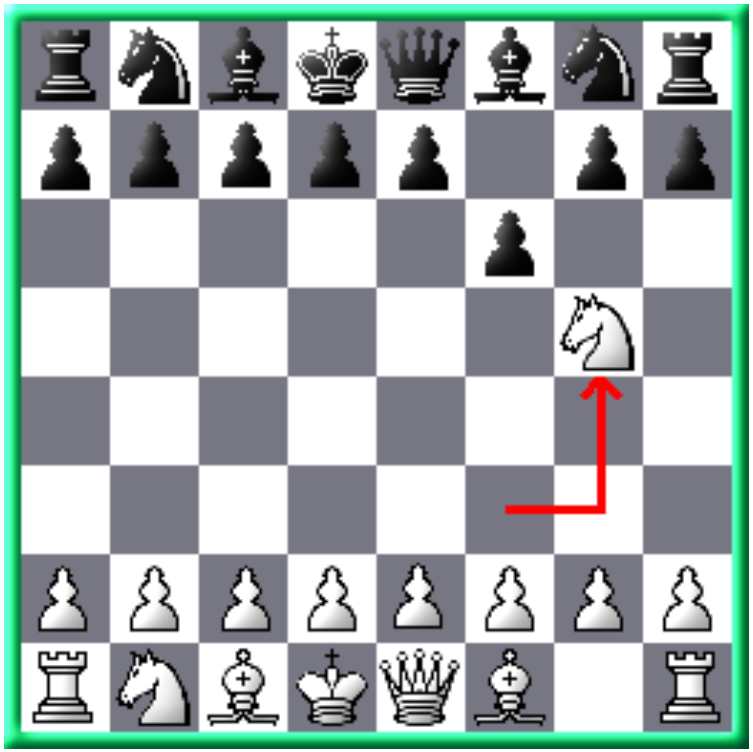
GAMES WITH OPPONENTS AS SEARCH

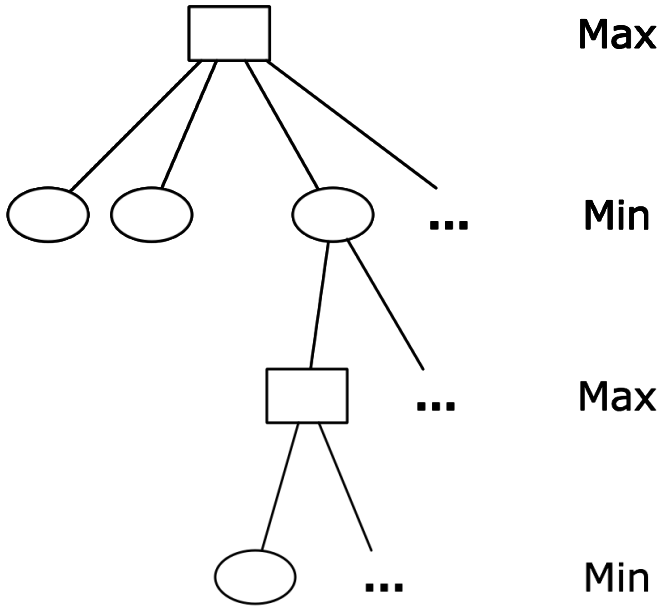
- Multi-agent environment that has to account for the presence of an opponent
- Game Theory→ branch of the economy
- Currently computers have outperformed humans in many games such as Othello, Checkers, Chess, Backgammon and GO.
- The game of Go is extremely complex and the world champion has been recently beated (March 2016).

GAMES

- We consider games with the following properties:
  - two-player games (we call them MIN and MAX) in which the moves are alternated and players have complementary objective functions (win and lose);
  - games with perfect knowledge in which players have the same information (not true in card games such as poker, bridge, etc).
- The development of a match can be interpreted as a tree in which the root is the starting position and leaves the final positions.

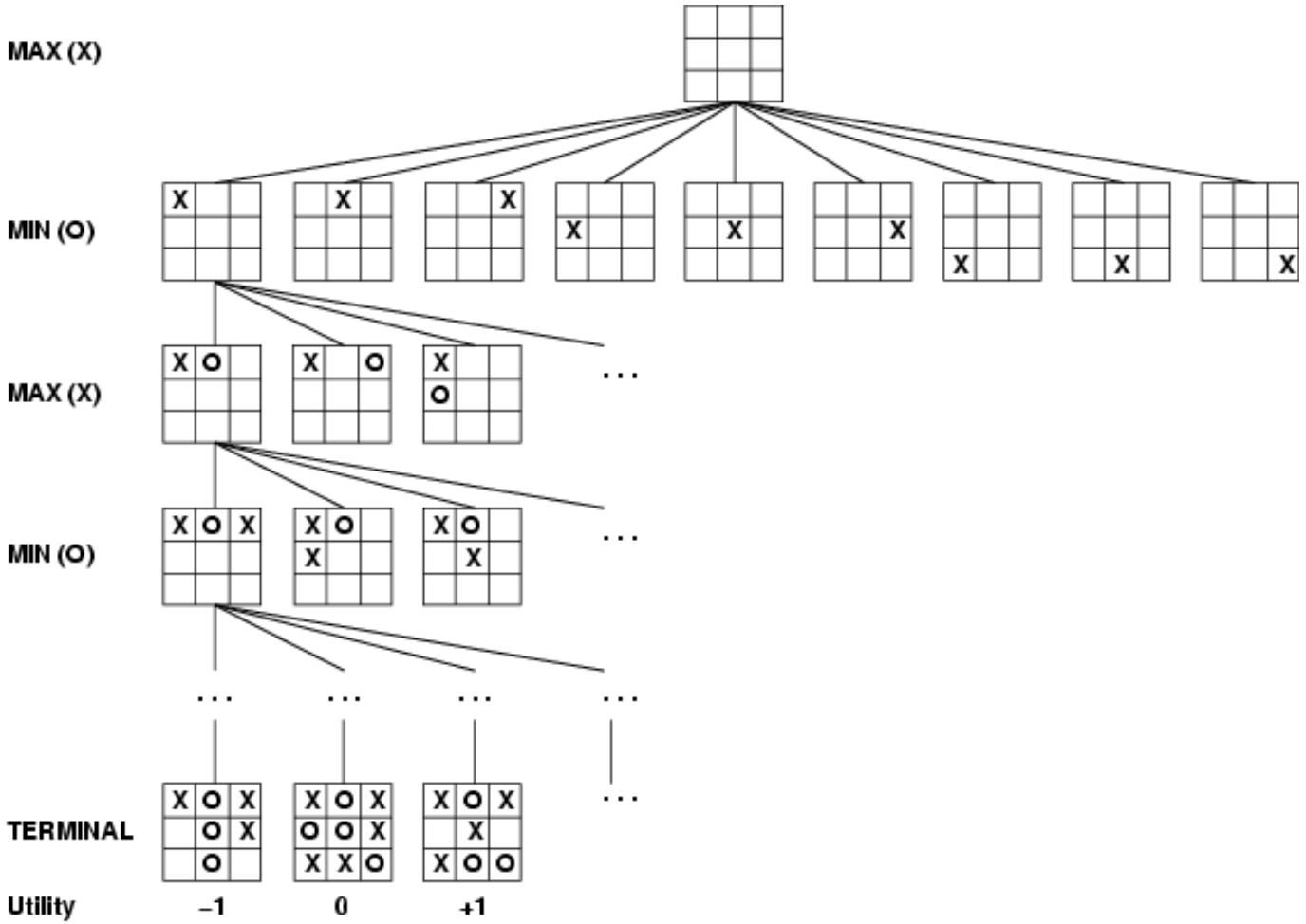
GAMES IN IA





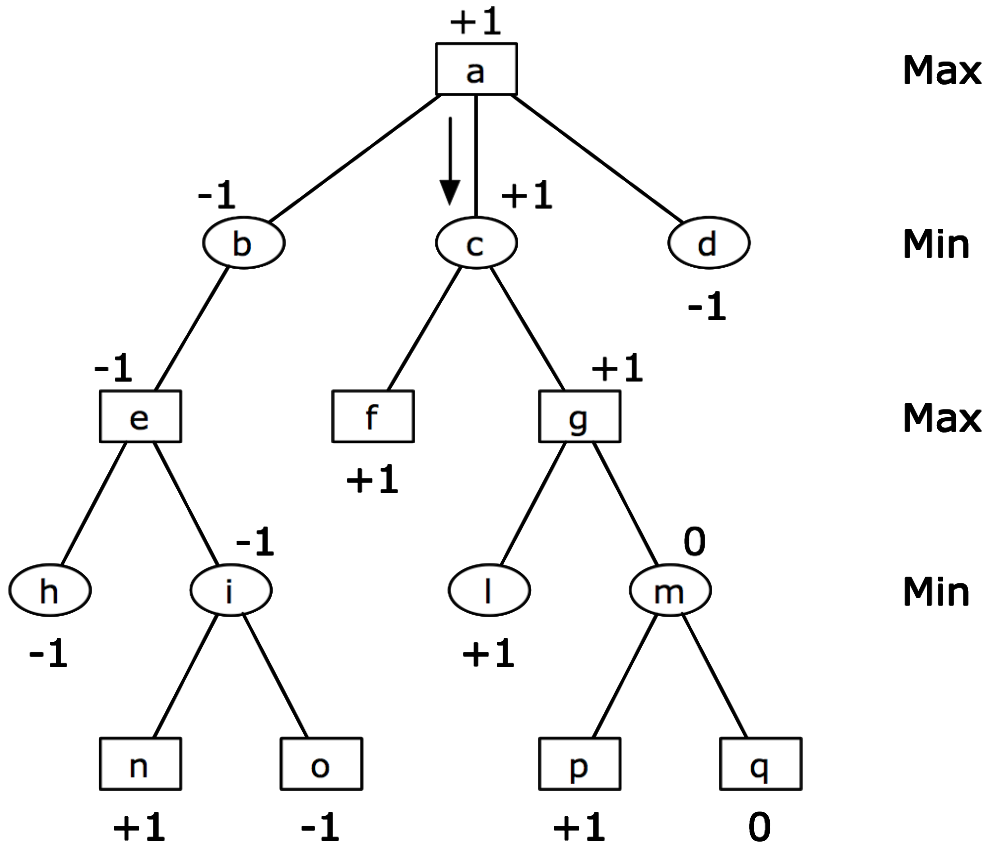
Game Tree

GAME TREE



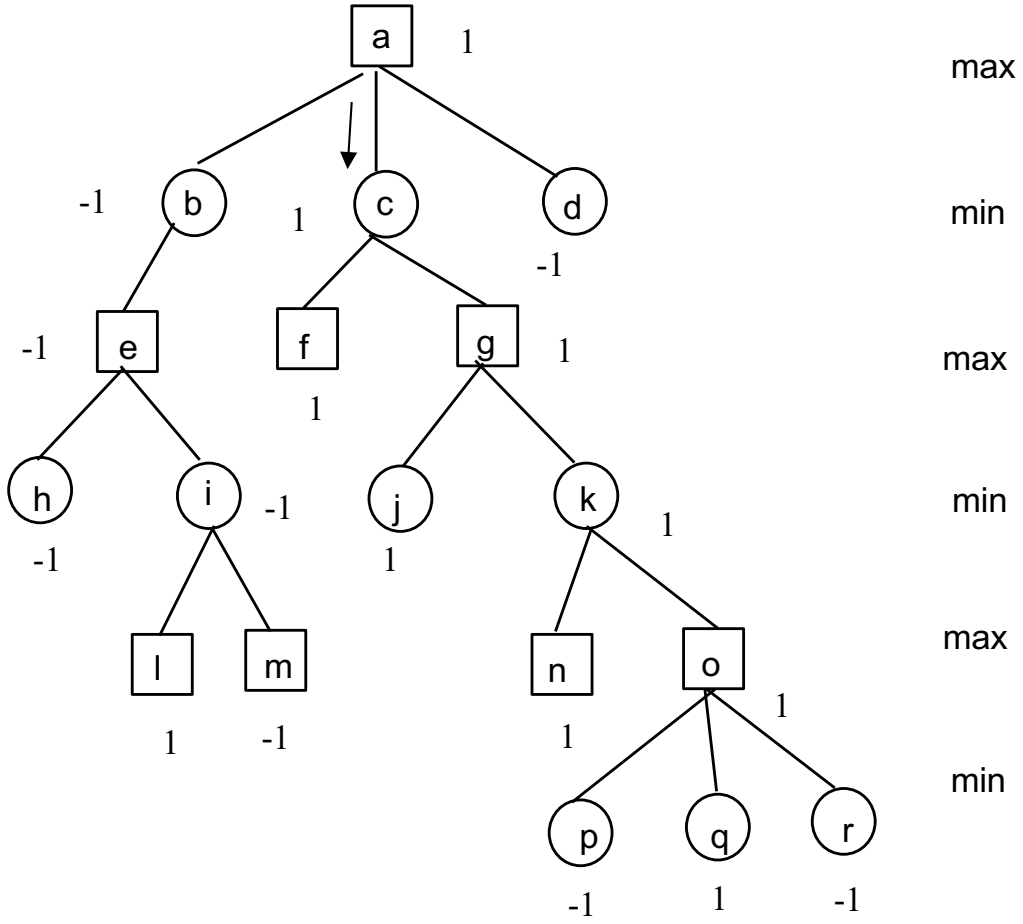
MIN-MAX ALGORITHM

- The minmax algorithm is designed to determine the optimal strategy for MAX and to suggest, therefore, the first best move to be performed;
- MIN is assumed to play at his best
- We are not interested in the path but only in the next move



MIN-MAX

- Leaves are labeled with 1 and -1. 1 is the victory of MAX and -1 is the victory on MIN. MIN tries to get to -1 (minimizer), MAX to +1 (maximizer).



MIN-MAX

- MAX moves for square nodes while MIN for circle nodes
- Consider the node **o**.
  - MAX should move. The game is over in one move. MAX can move to **r** or **p** and lose, or he can move to **q** and win.
  - Therefore **o** is a winning position for MAX and is labelled with a (+1)
- Consider the node **k**.
  - Whatever move MIN does, MIN loses. So the label is (+1).
  - Consider the node **i.** MIN has a winning option and the node is labeled with a (-1).
- Then:
  - A node where MAX has to move has a label equal to the maximum of the children labels. The opposite happens for MIN.

MIN-MAX ALGORITHM

- To evaluate a node n:
  1. Expand the entire tree under n;
  2. Evaluate the leaves as winners for MAX or MIN;
  3. Select n' as an unlabeled node whose children are labeled. If it does not exist then return the value assigned to n;
  4. If n' is a node in which MIN has to move, then label it with the minimum value of the children label. If n' is a node in which MAX has to move, then label it with the maximum value of the children label. Return to 3.
- In case of parity the label is 0.
- Indeed we can assign temporary values to nodes and update them when children have a value
- Complexity in time and space = b^d

MIN-MAX ALGORITHM in detail

- To evaluate a node n in a game tree:

1. Put n in L, a list of open nodes (not yet expanded nodes)
2. Let x be the first node in L. If x = n and there is a value assigned to it, then return this value.
3. Otherwise, if x has an assigned value Vx, p is the father of x and Vp a provisional value assigned to it.
   - If p is a MIN node, Vp= min (Vp, Vx),
   - otherwise Vp = max (Vp, Vx).
   - Remove x from L and return to step 2.
4. If x is not assigned any value and is a leaf node, assign it either a 1, or a -1, or a 0. Put x in L because we need to update its ancestors and return to step 2.
5. If x is not assigned any value, and is not a terminal node, assign Vx = - ∞ if x is a MAX and Vx = + ∞ if it is a min. Add the children of x to L and return to step 2.

– Bd complexity in space.

MIN-MAX PROPERTIES

- Complete? Yes (if the tree is finite)
- Optimal? Yes (against an opponent who plays at his/her best)
- Temporal complexity? $O(b^m)$
- Space complexity? O(bm) (depth-first)
- For chess, b ≈ 35, m ≈100 for "reasonable" games
  → this solution is out of reach!!!

WE NEED TO PRUNE THE TREE

MIN-MAX ALGORITHM REVISED

- If we have to develop the whole tree, the procedure is very inefficient (exponential).
- If b is the branching factor and d is the depth, then the number of nodes becomes b^d .
- Solution (Shannon, 1949): look forward few levels and assess the configuration of a non-terminal node. In practice we apply the min max algorithm up to a certain depth.
- Use an evaluation function for estimating the quality of a certain node.

e(n) = -1 if MIN wins;

e(n) = +1 if MAX wins

e(n) = 0 if they have the same probability;

We can use intermediate values for e(n).

EXAMPLE

- In chess, we add the value of all pieces of each player and normalize the result to have a value between -1 and +1.
- For example a weighted sum of values (linear)

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$$

- $w_1 = 9$ with

  $f_1(S)$ = (number of white queens) - (number of black queens), etc.

It could be refined taking into account the relative positions: is the king protected? Do pawns protect any piece? etc.

- Trade-off between search and evaluation function.
- In any case we need to select an evaluation function, e(n).

MIN-MAX ALGORITHM II

To evaluate a node n in a game tree:

1. Put n in L, a list of open nodes (not yet expanded nodes)
2. Let x be the first node in L. If x = n and there is a value assigned to it, then return this value
3. Otherwise, if x has an assigned value Vx, p is the father of x and Vp a provisional value assigned to it.
   - If p is a MIN node, Vp= min (Vp, Vx),
   - otherwise Vp = max (Vp, Vx).
   - Remove x from L and return to step 2.
4. If x is not assigned any value and is a leaf node or you **decide not to expand the tree further**, **assign it the value using the evaluation function e(x).** Put x in L because you will have to update the ancestors and return to step 2.
5. If x is not assigned any value, and is not a terminal node, assign Vx = - ∞ if x is a MAX and Vx = + ∞ if it is a min. Add the children of x to L and return to step 2.

MIN-MAX

```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← ∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s))
    return v
```

Note: eval replaced by TERMINAL-TEST with:
if CUTOFF-TEST (state, depth) then return EVAL(state)
It also updates depth at each recursive call

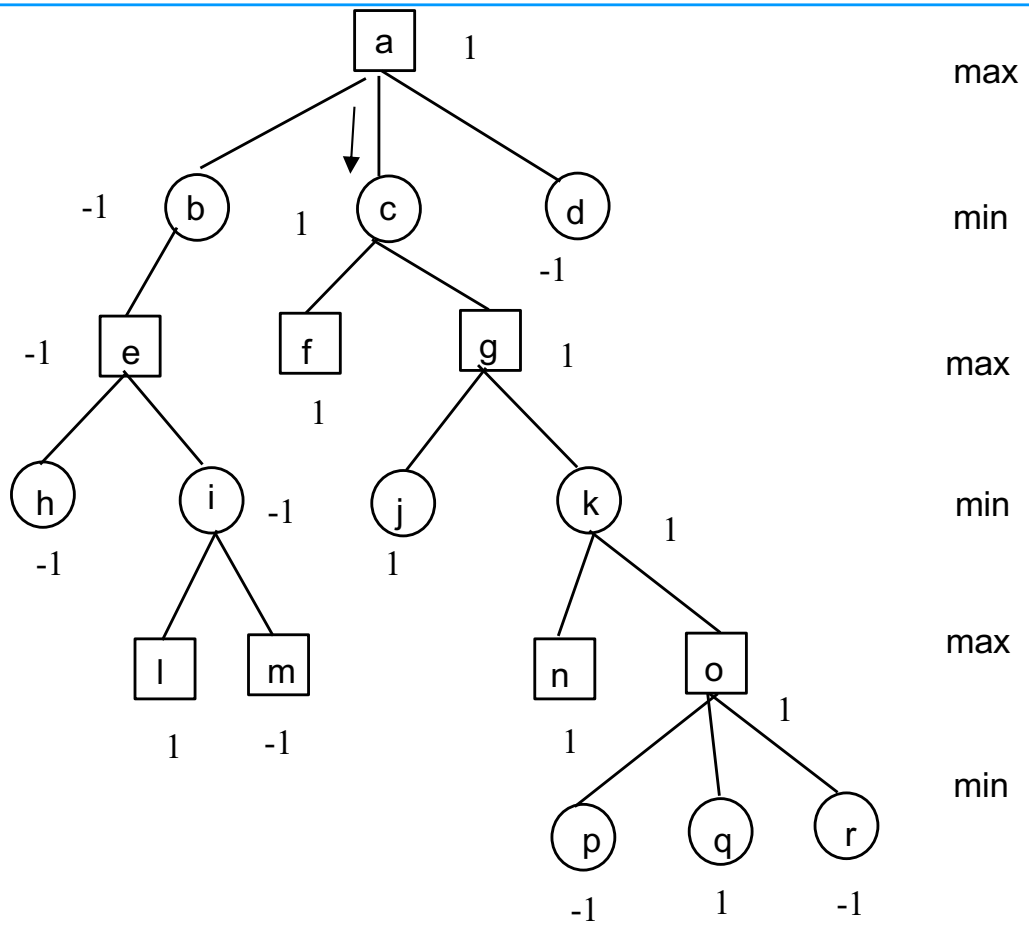
PROBLEM

- How do we decide if to expand a node or not?
- Note: If e(n) was perfect I would not have this problem. I could stop after the first level and evaluate the children of the root node.
- Simple solution from the computational point of view: always expand nodes up to a certain depth p.
- Problems:
  - More tactically complicated moves (with higher variance for e(n)) should be evaluated with higher depth up to quiescence when e(n) values change more slowly.
- Horizon effect:
  - With useless moves, extend depth to values larger then p, so the basic moves are not really taken into account.
- Solution: Sometimes it pays to do a secondary search, focused on the best move choice.

ALFA BETA CUTS

- From what we have seen so far computers simply play all possible matches up to a certain depth, evaluate leaves and propagate back the evaluation.
- So they also consider moves and nodes that will never occur.
- You should try to reduce the search space.
- The best-known technique is that of the alpha-beta cut.

EXAMPLE

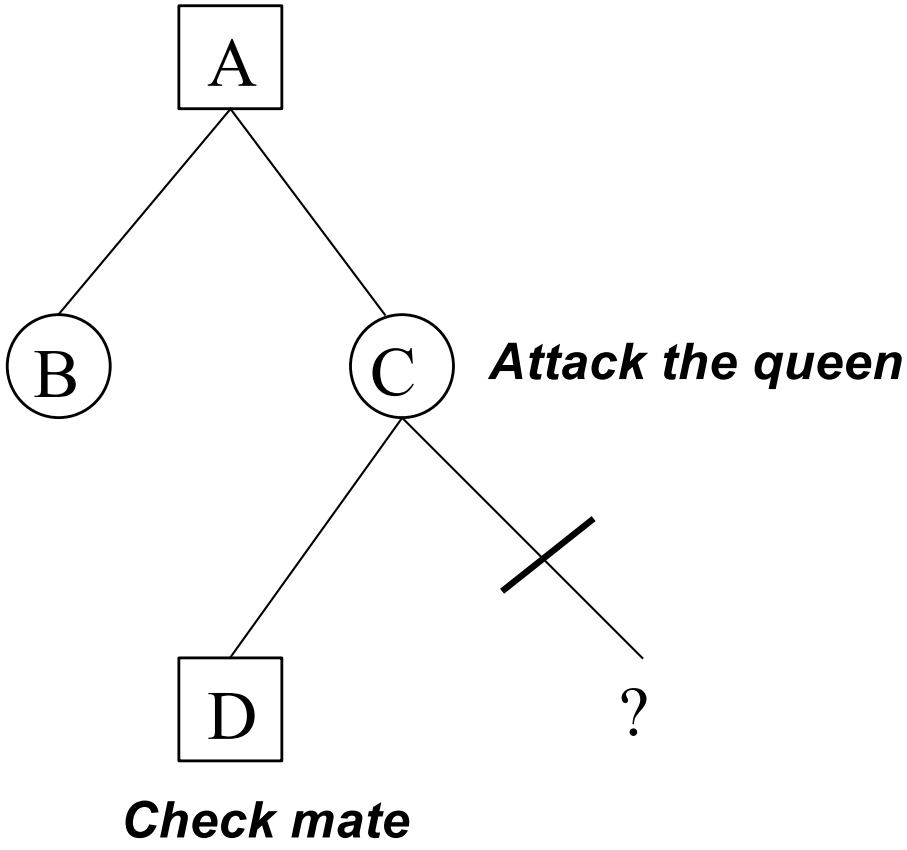


- When we find out that the move to **c** is a winning move, we do not need to expand the nodes b and d.
- The nodes under b will never influence the choice.

ALFA-BETA CUTS: GENERAL PRINCIPLES

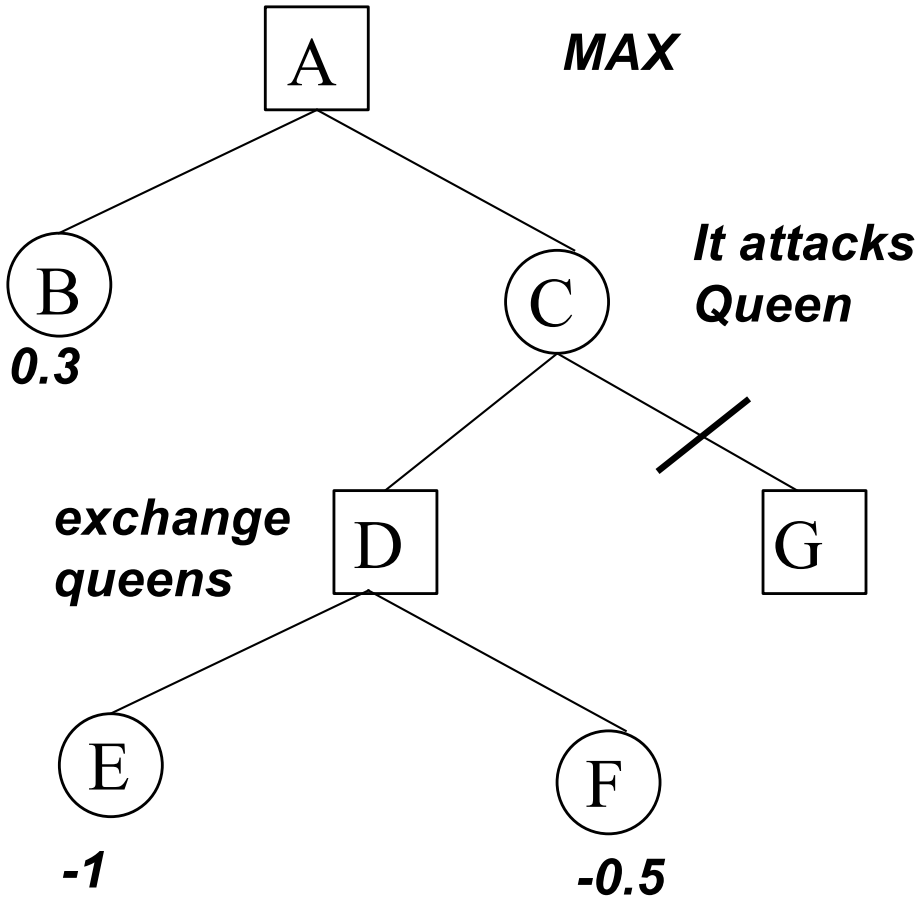
- Consider a node N in the tree. Will the player move to that node?
- If the player had a better choice (we call it M) in the parent node level or at any point along the path, then N will never be selected. If we reach this conclusion we can eliminate N.
- Call ALFA the value of the best choice found on the path for MAX (the highest) and BETA the value of the best choice found on the path for MIN (the lowest).
- The algorithm updates ALPHA and BETA and cuts branches when their choice the worst.

EXAMPLE



- No matter which is the evaluation of the other children of C (I realize that I should never move to C).

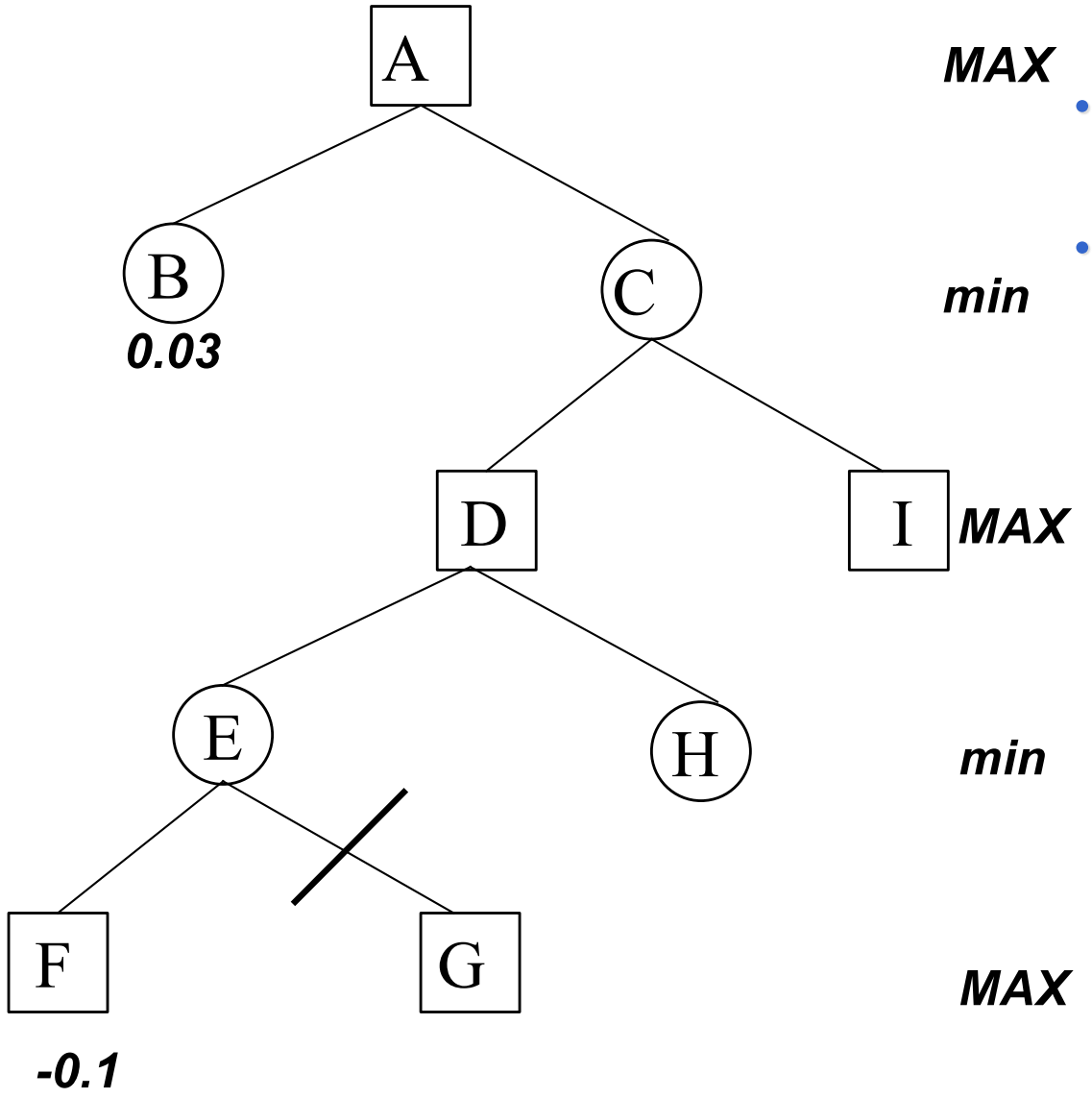
ANOTHER EXAMPLE



- Max in A avoids C because B is better. At most max gets from C a - 0.5 so 0.3 is better
- The subtree in G can be cut as soon as I receive the value of D.
  Indeed: C = min (-0.5, G);
  A = max (0.3, min (-0.5, G)) = 0.3

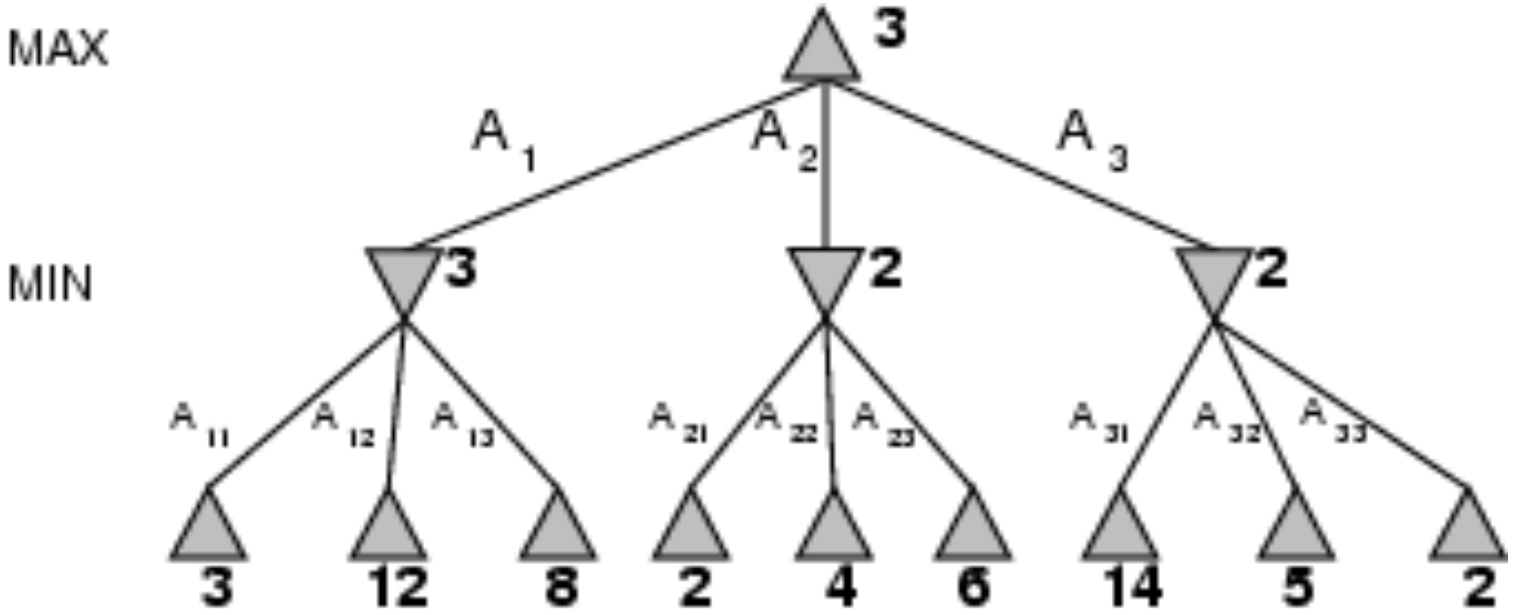
Since A is independent of G, the tree under G can be cut.

ANOTHER EXAMPLE

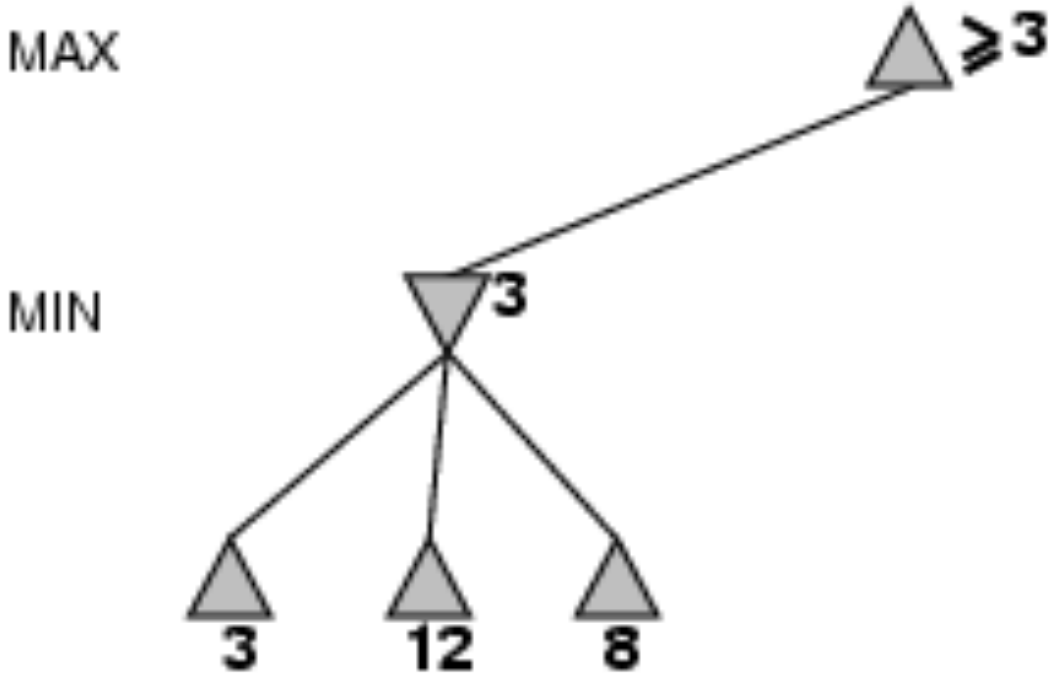


- **G is on the search path that will be developed?**
- If G is on the search path, then also E is. From min E can always obtain either -0.1 or worst which is always worst than 0.03 for max. Then G cannot be in the current search path.

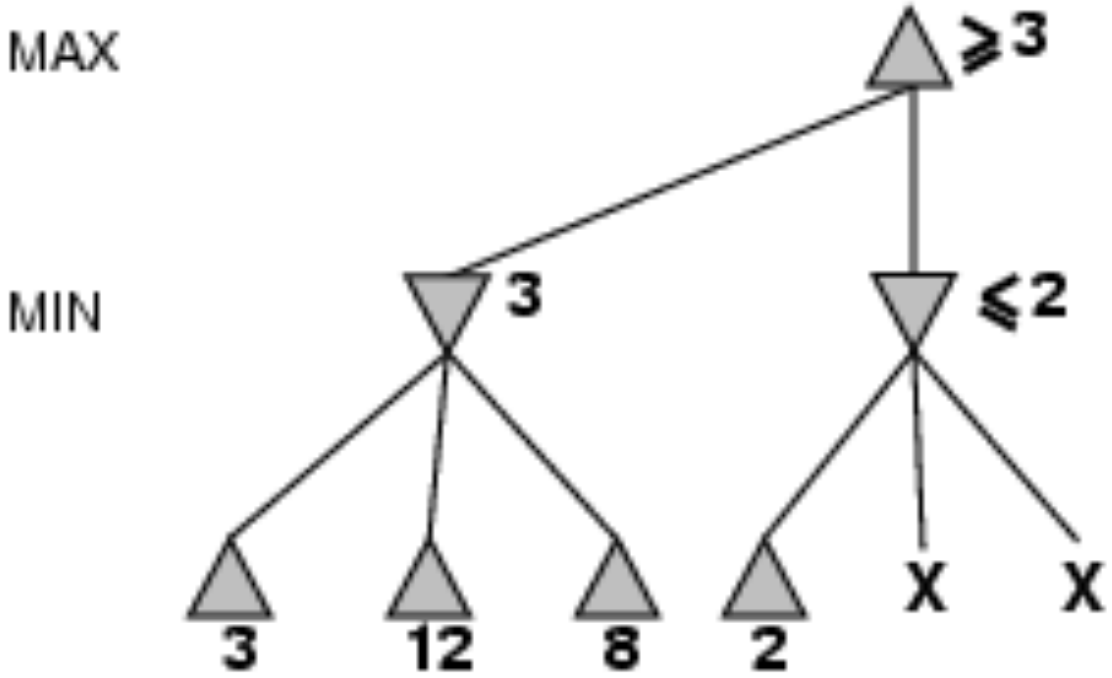
MIN MAX


MAX
MIN
3
A 1
A 2
A 3
3
2
2
A 11
A 12
A 13
A 21
A 22
A 23
A 31
A 32
A 33
3
12
8
2
4
6
14
5
2

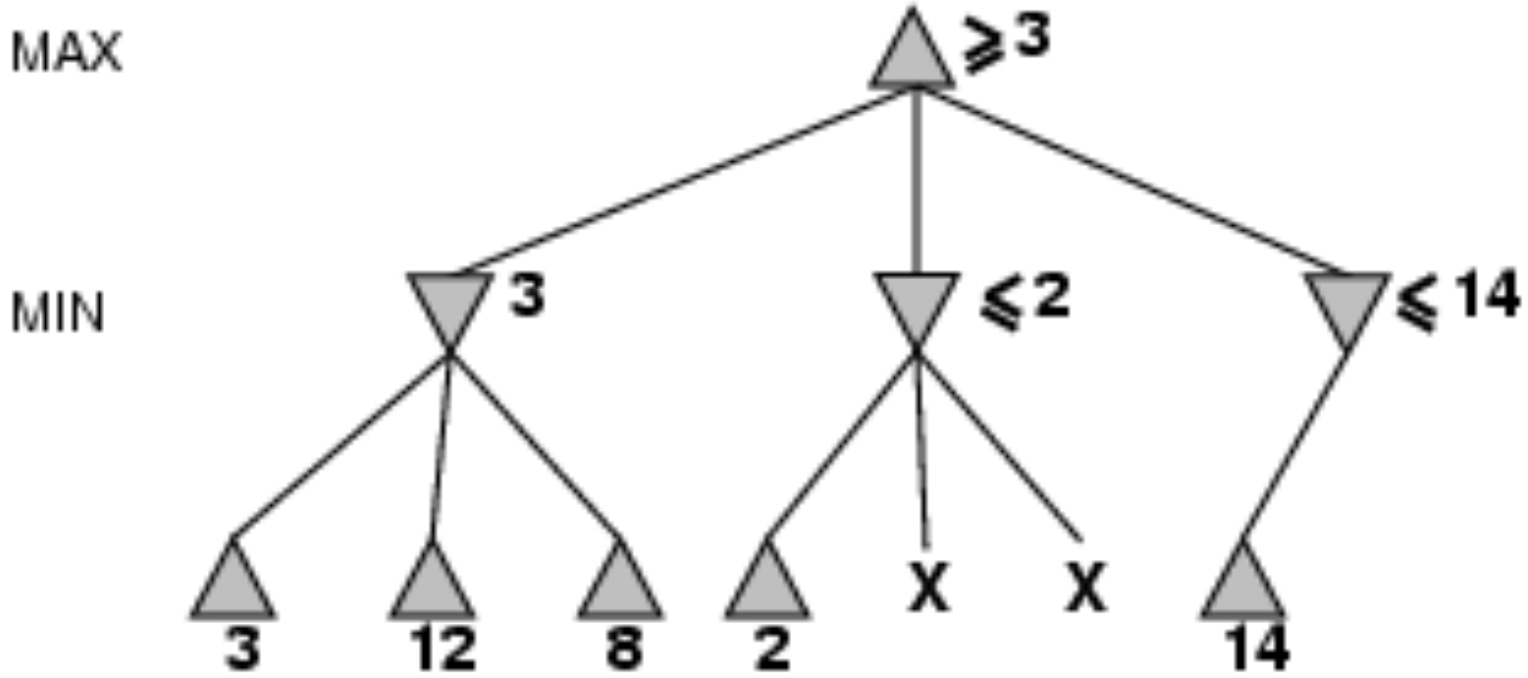
ALPHA-BETA CUTS



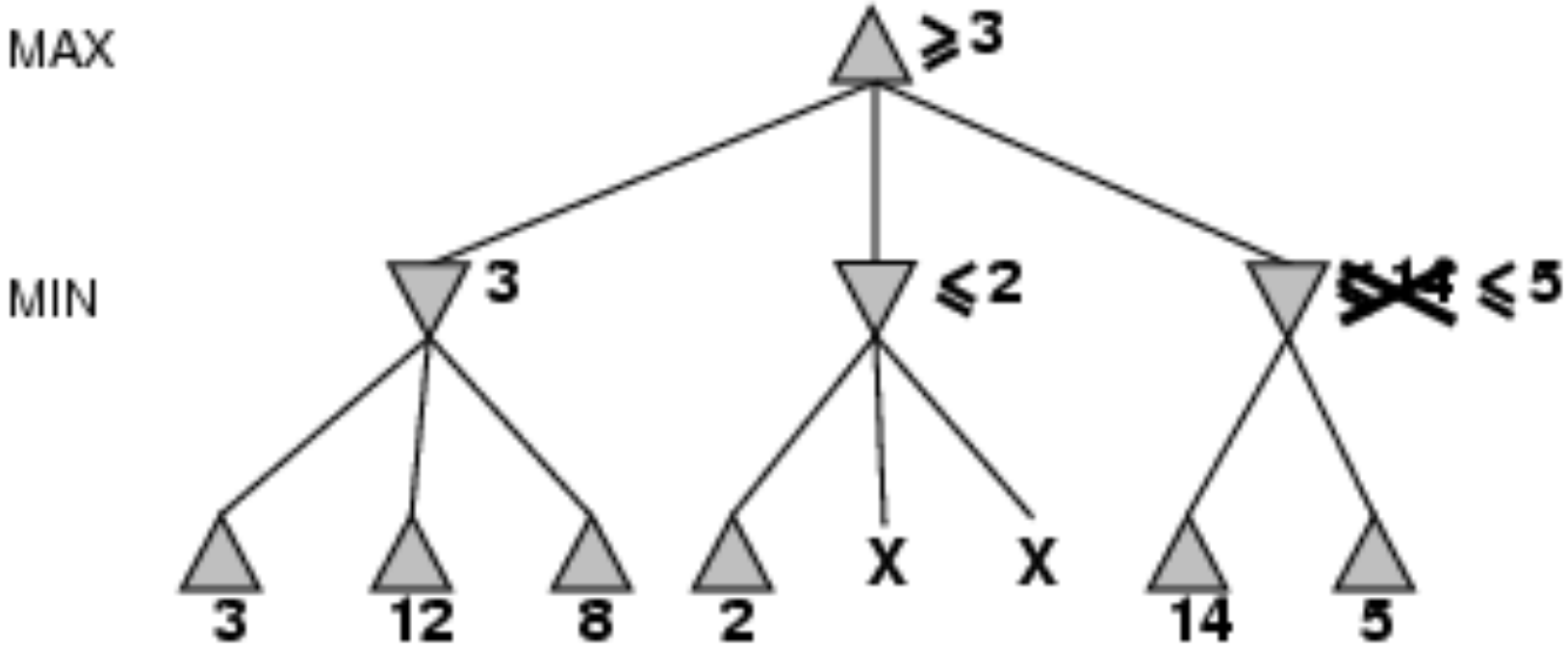
ALPHA-BETA CUTS



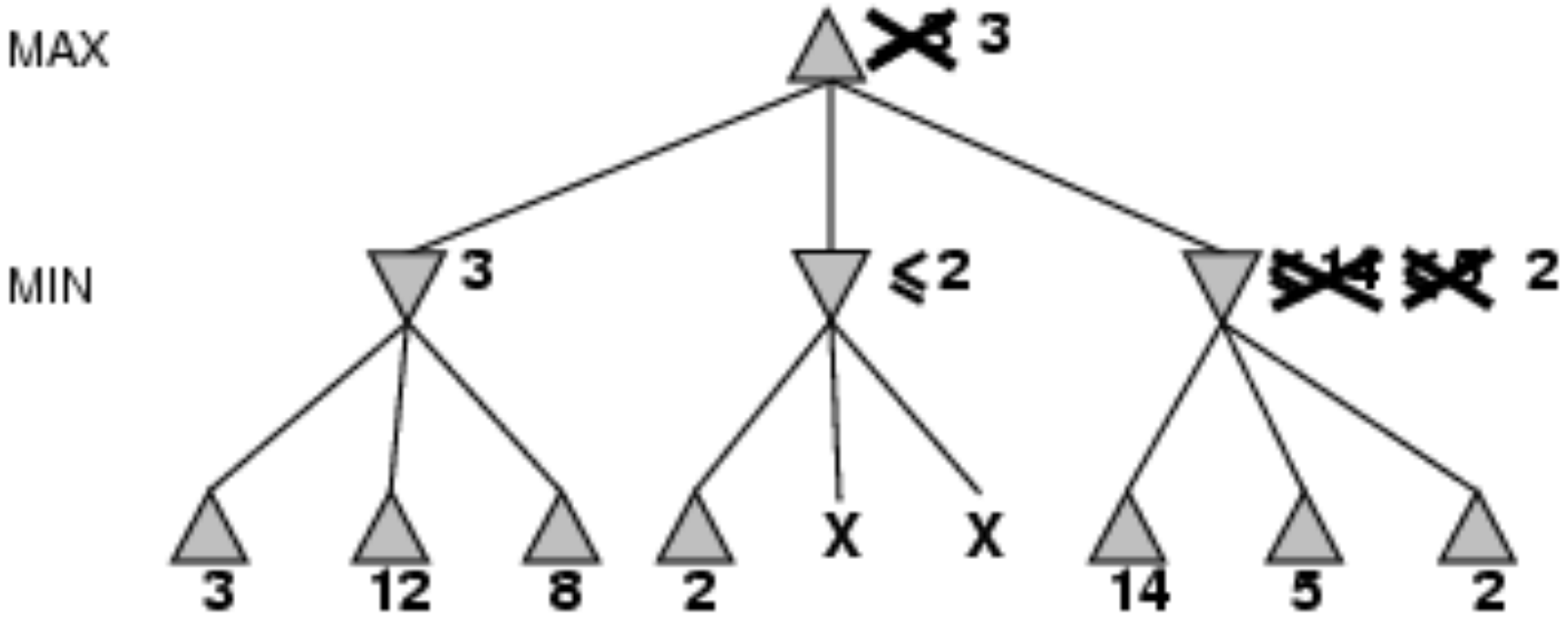
ALPHA-BETA CUTS



ALPHA-BETA CUTS



ALPHA-BETA CUTS

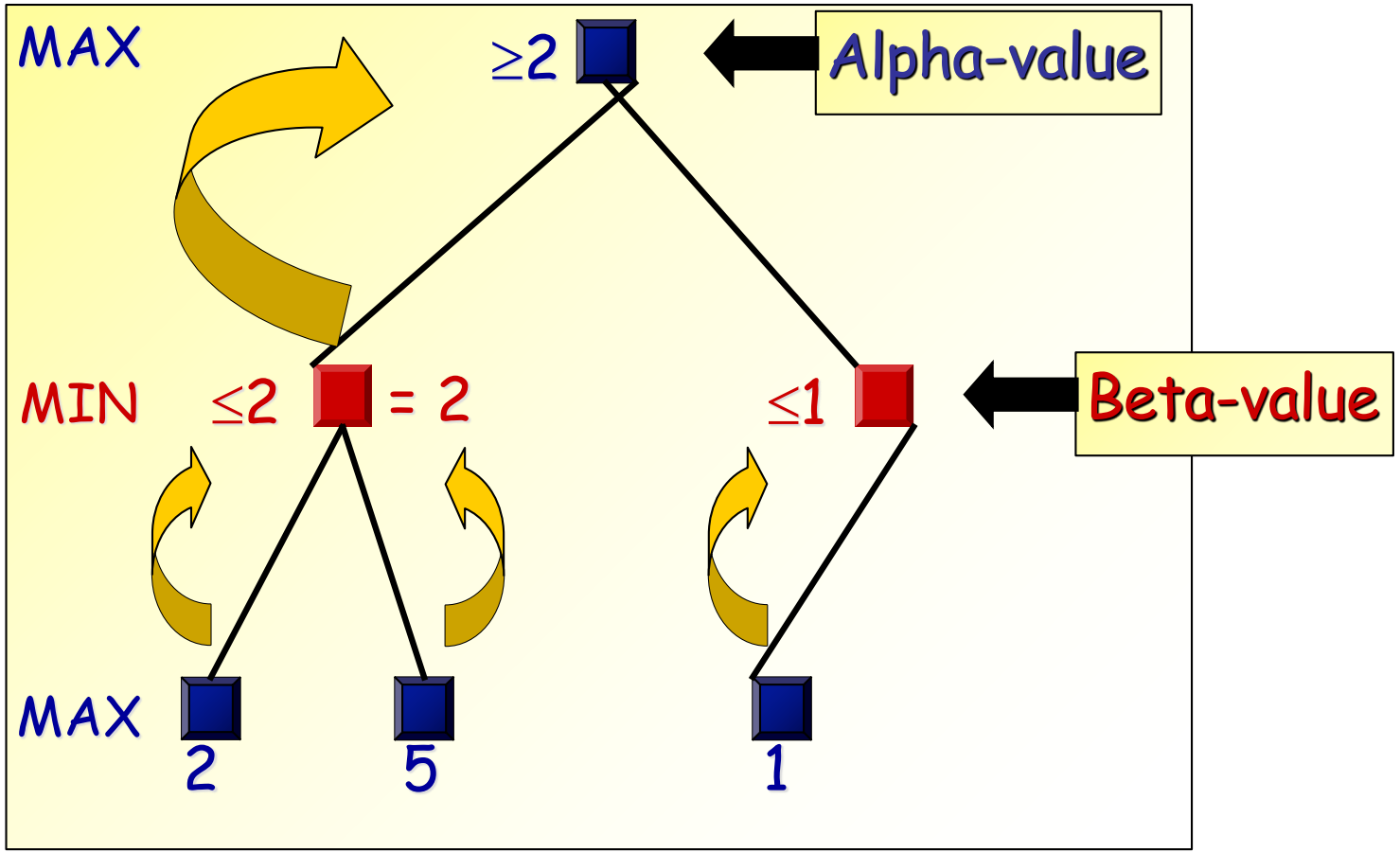


ALPHA-BETA CUTS

- **Principles:**
  - **we generate depth-first search tree, left-to-right**
  - propagate the (estimated) values from the leaves

TERMINOLOGY

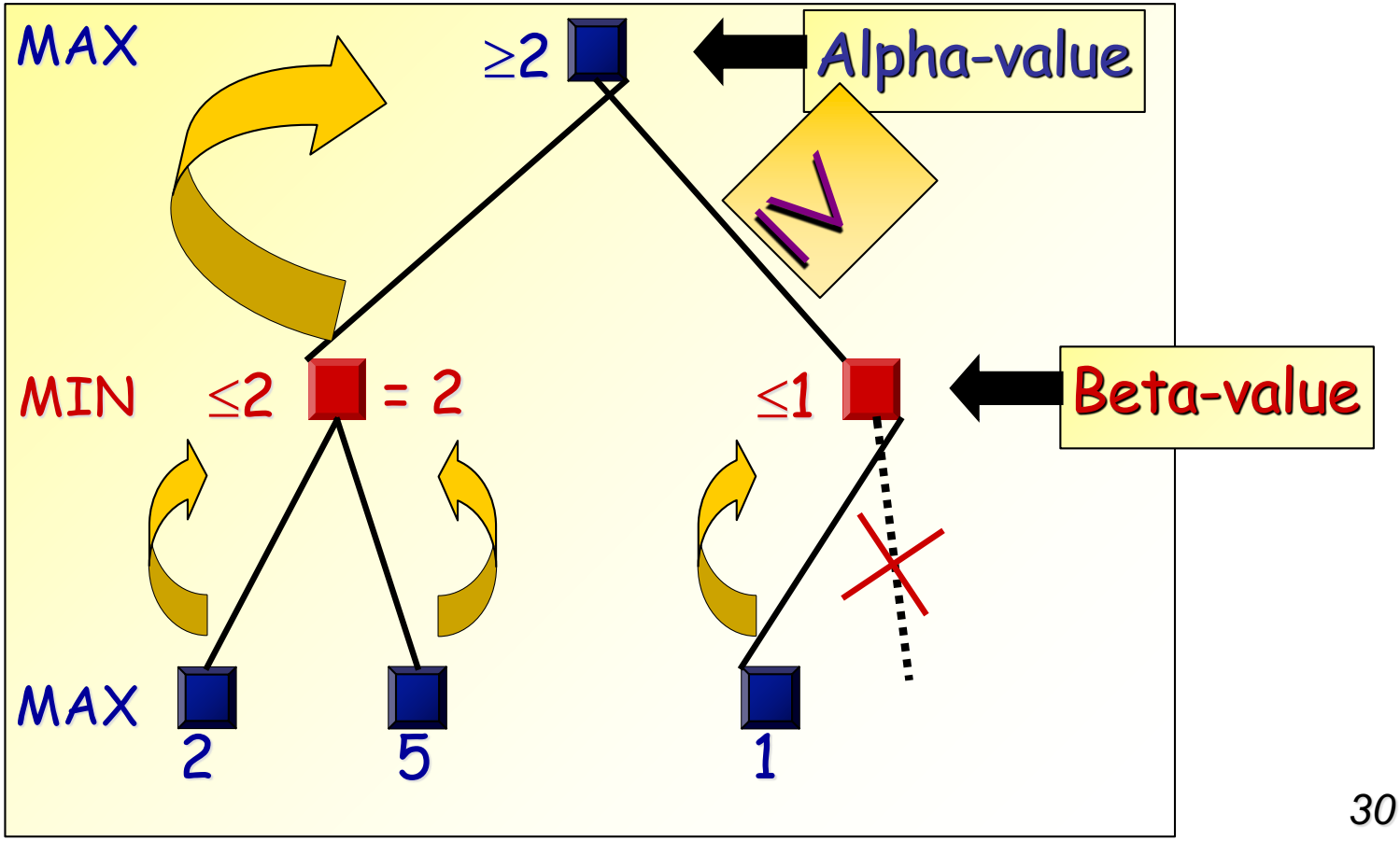
- The (temporary) values in MAX-nodes are ALPHA-values
- The (temporary) values in MIN-nodes are BETA-values



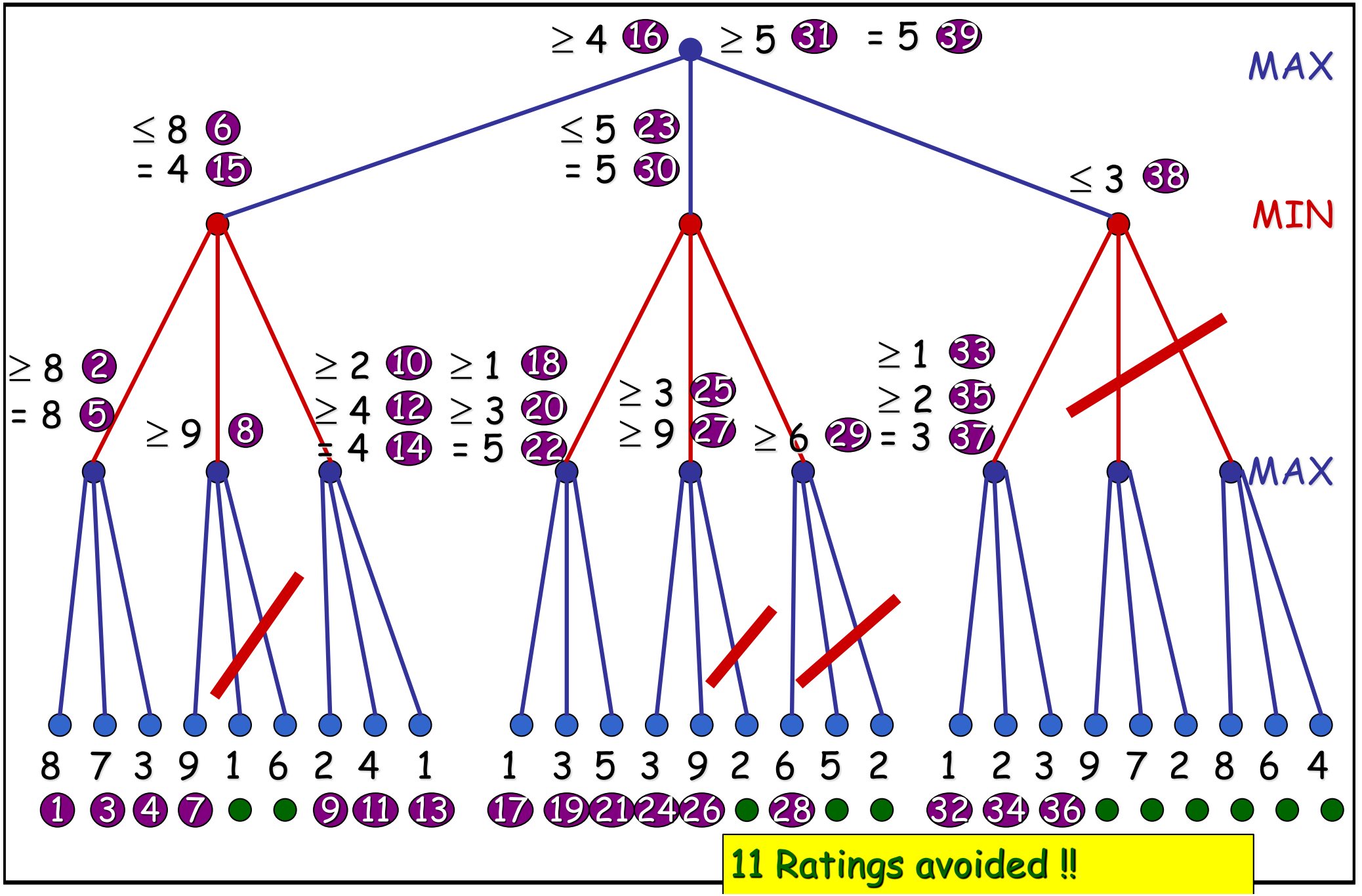
THE ALPHA-BETA PRINCIPLE

- If a ALPHA-value is greater than or equal than a Beta-value of a descending node: stop the generation of children of the descending node!

- If a Beta-value is smaller than or equal than a Alpha-value of a descending node: stop the generation of children of the descending node!



MIN MAX WITH ALPHA-BETA CUTS



ALGORITHM ALFA-BETA

To evaluate a node n in a game tree:

1. Put n in L, a list of open nodes (not yet expanded nodes)
2. Let x be the first node in L. If x = n and there is a value assigned to it, return this value
3. Otherwise, if x has a value assigned Vx, let p be the father of x. If x has not an assigned value, go to step 5.

    - We need to determine if p and its children can be removed from the tree. If p is a min node, alpha is the maximum of all current values assigned to the brothers of p and of nodes that are ancestors of p.
    - If there are no such values alpha = - infinity.
    - If Vx <= alpha, then remove p and all his descendants from L (dually if p is a max node).

ALFA-BETA ALGORITHM

4. If p cannot be eliminated, let Vp be its current value. If p is a min node, Vp= min (Vp, Vx), otherwise Vp = max (Vp, Vx). Remove x from L and return to step 2.
5. If x is not assigned to any value and is a terminal node, or we decide not to further expand the tree, give it a value using the evaluation function e(x). Leave x in L because you have to update the ancestors and return to step 2.
6. If x is not assigned to any value, and is not a terminal node, assign Vx = -infinity if x is a max and Vx = + infinity if it is a min. Add the children of x to L and return to step 2.

ALFA-BETA ALGORITHM

```
function ALPHA-BETA-SEARCH(state) returns an action
    inputs: state, current state in game

    v ← MAX-VALUE(state, −∞, +∞)
    return the action in SUCCESSORS(state) with value v

---

function MAX-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← −∞
    for a, s in SUCCESSORS(state) do
        v ← MAX(v, MIN-VALUE(s, α, β))
        if v ≥ β then return v
        α ← MAX(α, v)
    return v
```

ALFA-BETA ALGORITHM

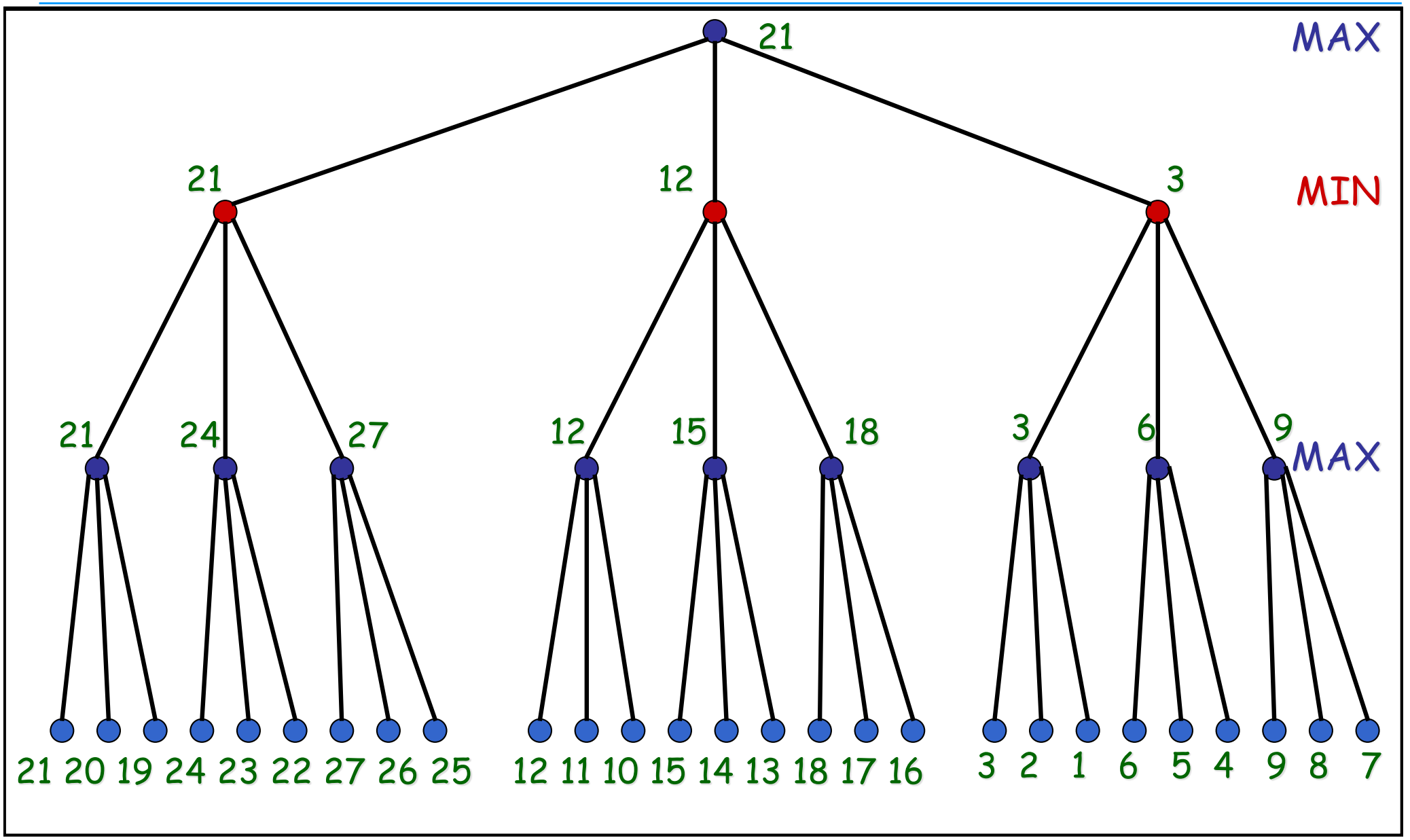
```
function MIN-VALUE(state, α, β) returns a utility value
    inputs: state, current state in game
            α, the value of the best alternative for MAX along the path to state
            β, the value of the best alternative for MIN along the path to state

    if TERMINAL-TEST(state) then return UTILITY(state)
    v ← +∞
    for a, s in SUCCESSORS(state) do
        v ← MIN(v, MAX-VALUE(s, α, β))
        if v ≤ α then return v
        β ← MIN(β, v)
    return v
```

EFFECTIVENESS OF CUTS

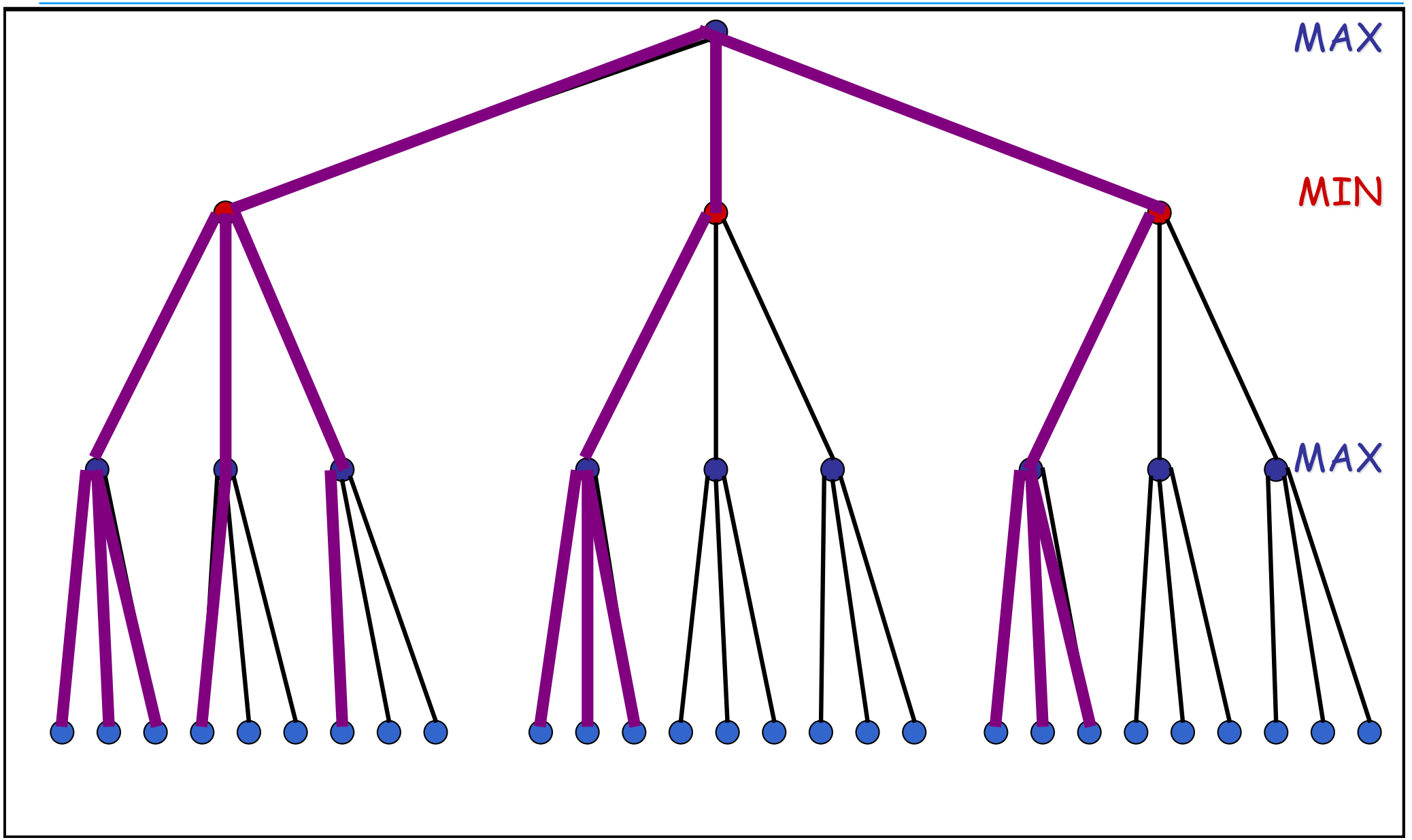
- Obviously if you always evaluate the worst nodes, then best nodes are always in the search path and there is no cut.
- The best case is when the best nodes are evaluated first. The other are always cut (of course it is entirely theoretical).
- In this case we can reduce the number of nodes (which is b^d) to about $b^{d/2}$. (In practice, it is reduced by the square root branching factor, or you can look forward twice more at the same time).
- In the average case with random distribution of the node values, the number of nodes becomes about $b^{3d/4}$.
- So it is important to choose a good order of the children of a node.
- Also note that all the results mentioned herein are for an "ideal" game tree with depth and branching fixed for all the branches and nodes.
- Repeated States, the list of closed nodes (see graph search).

PERFECT EXAMPLE



PERFECT EXAMPLE

- At each level the best node is on the left.



THE GAME OF CHESS

- The problem size is enormous. Only at the beginning of the game we have 400 possible moves, they become more than 144,000 for the second move.....
- In particular, chess has a branching factor of ~ 35 and ~ 50 moves for each player. So we have 35^{100} nodes. (Actually lawful moves are 10^{40}).
- There must be an evaluation function. In particular, it will give a weight to each piece, but this is not enough.
- It must also take account of the relative position of the pieces (two towers in a column sometimes are more important than a Queen) and the moment of the game (opening, intermediate stages, end)

EXAMPLE: EVALUATION OF A KNIGHT

The moment of the game is also important. For example, knights are important in the middle of the game but they are not important in the end of the game with just a few pieces.

But even giving a weight to all these components is not enough, we also need a function that better links all these parameters.

The other choice is how much to go deep into the solution tree. The machine is expected to respond in a time comparable to that of a human player.

A computer processes around 1000 positions per second (but can even reach 2,500).

Each move takes a maximum of 150 seconds. So a computer processes around 150,000 possible moves that correspond to about 3-4 levels playing at beginner level).

ALFA BETA CUTS : become essential

- Current programs explore about 7 levels and process about 250,000 configurations for each move but under certain circumstances it can explore up to 20 levels and 700,000 configurations.
- Almost all programs use the time that the human player uses to choose her move.
- It has been estimated that a human player, even if expert, actually never explores for more than 5 levels, with substantial cuts. It then uses a heuristic evaluation function.
- All chess programs consult the library of openings (there are openings that have been previously fully explored and that can affect the entire game).
- While a computer is very strong in the middle part of the game, the human player is smarter at the end, when exploring many moves is less important. But today's chess programs avoid this and tend to use libraries and specialized algorithms for the final.

DEEPBLUE

- On 10/05/1997, in New York, DeepBlue beated the world champion in a match of six games (match DeepBlue - Kasparov - 2-1 and three draws).
- There is a system of evaluation of the playing strength (Elo) capable of measuring the progress of human and artificial players.

- Points are earned in official tournaments:
  - beginner player: 500 Elo points
  - Master: 2,200
  - World Champion: 2,800
  - Deep Blue: 3,000
- Artificial players can be classified into two categories according to whether they use generic hardware (PC) or special hardware (the case of Deep Blue).
- In particular, Deep Blue used a general-purpose parallel machine with specialized chip processors for generating moves and evaluations.

DEEP MIND APPROACH

- DeepMind is a small company in London co-founded, in 2011, by Demis Hassabis, child chess prodigy, video game designers and computational neuroscientist.
- In 2014 DeepMind was bought for hundreds of millions of dollars by Google.
- Algorithm based on deep learning (networks Neural) and Learning that teaches itself to play (even at video games), and often much better than human players.
- The algorithm has been trained on 49 different games for the Atari 2600, all developed for generations of teenagers.
- It managed to beat at the game of GO the European champion in October 2015.

PRESS RELEASE March 9, 2016

Supercomputer Google beats the world champion Go

The first round of the historic match between artificial intelligence and man went to AlphaGo program DeepMind Technologies, which has vanquished the South Korean Lee Sedol, considered the best player in the ancient Chinese board game.

Go is a board game originated in China over 2500 years ago and very popular in East Asia. Played over 40 million people worldwide, it has simple rules: players, in turn, must place the white or black stones on a table, trying to capture the opponent's stones or dominate the empty spaces to conquer the territory. But the simplicity of the rules hides a profound complexity of the game: the possible positions are higher than the number of atoms in the universe.

GO BOARD