بازی هایی که در این جا بررسی می کنیم از دسته بازی های ZERO SUM هستند
یعنی بازی هایی که یک کنش هر چه قدر برای یکی از بازیکنان خوب است برای دیگر به همان اندازه بد است

Game playing

Type of games

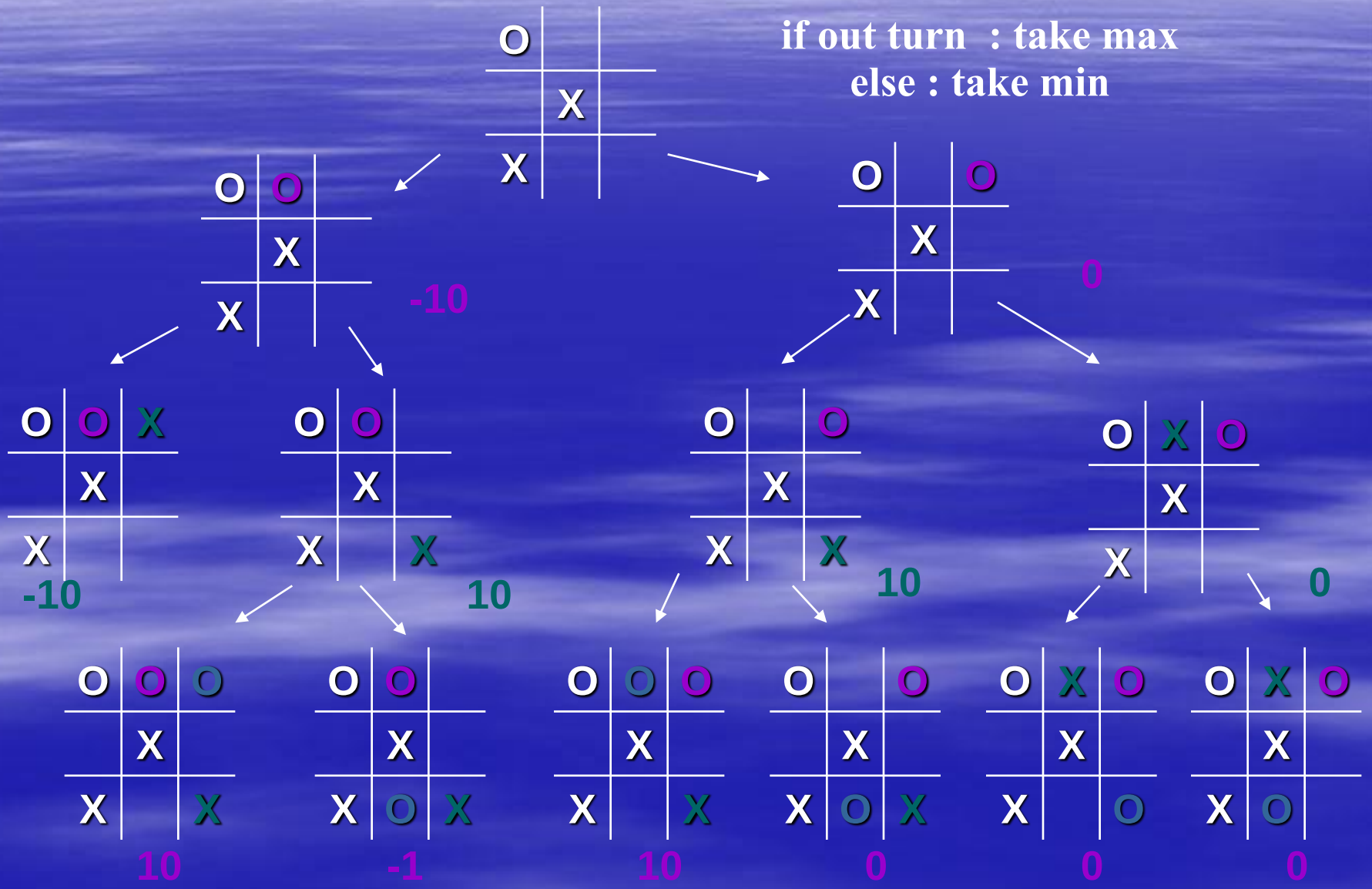
	Deterministic	Chance
Fully Observable Perfect Information	Chess	Backgammon
Partially Observable Imperfect Information	Battle Ship	Bridge

imperfect Information : Randomness in shuffling

Perfect Information :
از روی این information بهترین حرکت قابل تشخیص است .
پس می توان یک func خوش تعریف در نظر بگیریم که information
را بگیرد و بهترین حرکت را مشخص کند .

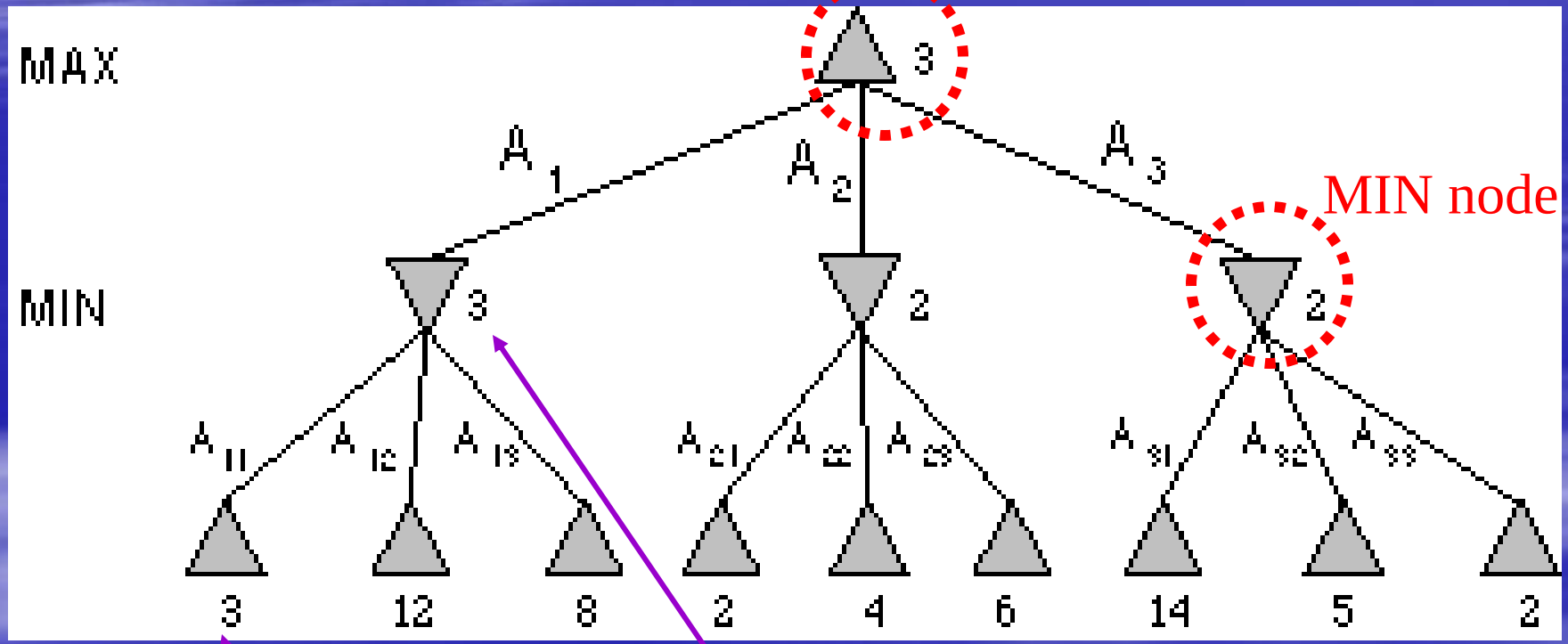
Here we consider Games with Perfect Information

Example - Tic-Tac-Toe



Minimax search

هر حرکتی به غیر از ۳ انجام شود به حریف اجازه داده میشود تا برای ما مقدار ۲ را به وجود آورد



Minimax search

Algorithm: **MINIMAX** (Depth-First Version)

To determine the minimax value $V(J)$, do the following:

1. If J is terminal, return $V(J) = e(J)$; otherwise
2. Generate J's successors $J_1, J_2, \ldots, J_b$.
3. Evaluate $V(J_1), V(J_2), \ldots, V(J_b)$ from left to right.
4. If J is a MAX node, return $V(J) = \max[V(J_1), \ldots, V(J_b)]$.
5. If J is a MIN node, return $V(J) = \min[V(J_1), \ldots, V(J_b)]$.

Minimax algorithm

```
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
```

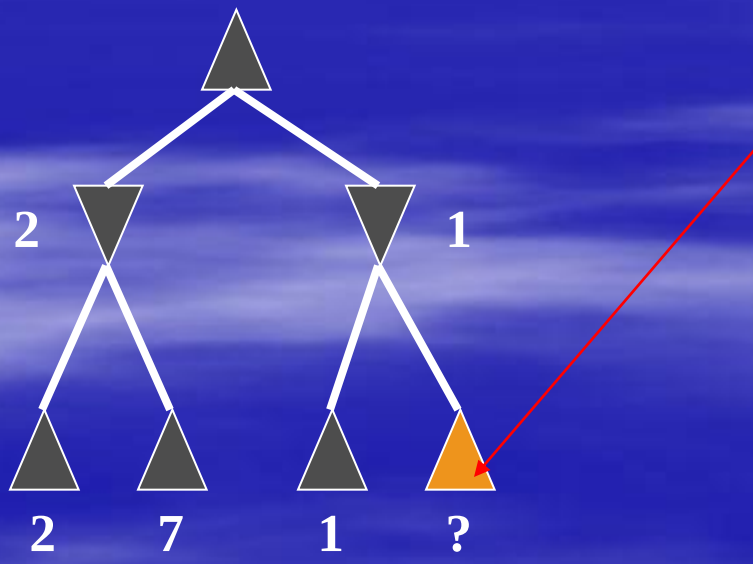
Properties of minimax

- Complete? Yes (if tree is finite)
-
- Optimal? Yes (against an optimal opponent)
-
- Time complexity? $O(b^m)$
-
- Space complexity? O(bm) (depth-first exploration)
-
- For chess, b ≈ 35, m ≈100 for "reasonable" games
  → exact solution completely infeasible

Alpha-beta pruning

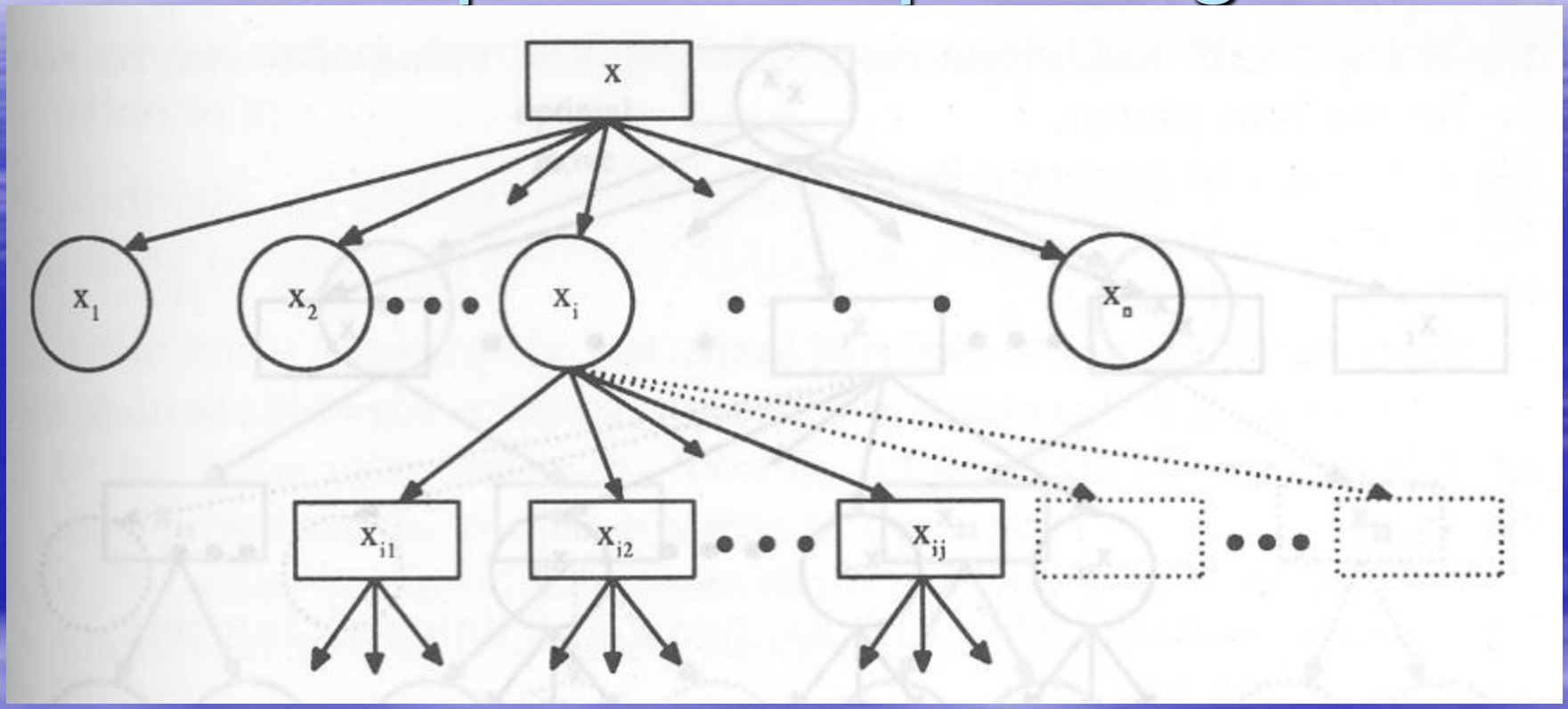
- We can improve on the performance of the minimax algorithm through alpha-beta pruning.

حداکثر چیزی که زیر درخت سمت راست به ما می دهد ۱ می باشد زیرا اگر از یک بیشتر باشد
چون min node اجازه نمی دهد آن مقدار بالا بیاید و اگر از ۱ کمتر باشد قطعا از ۲ نیز کمتر است
بنابراین باز هم ۲ انتخاب می شود (گره قبلی گره maxاست)



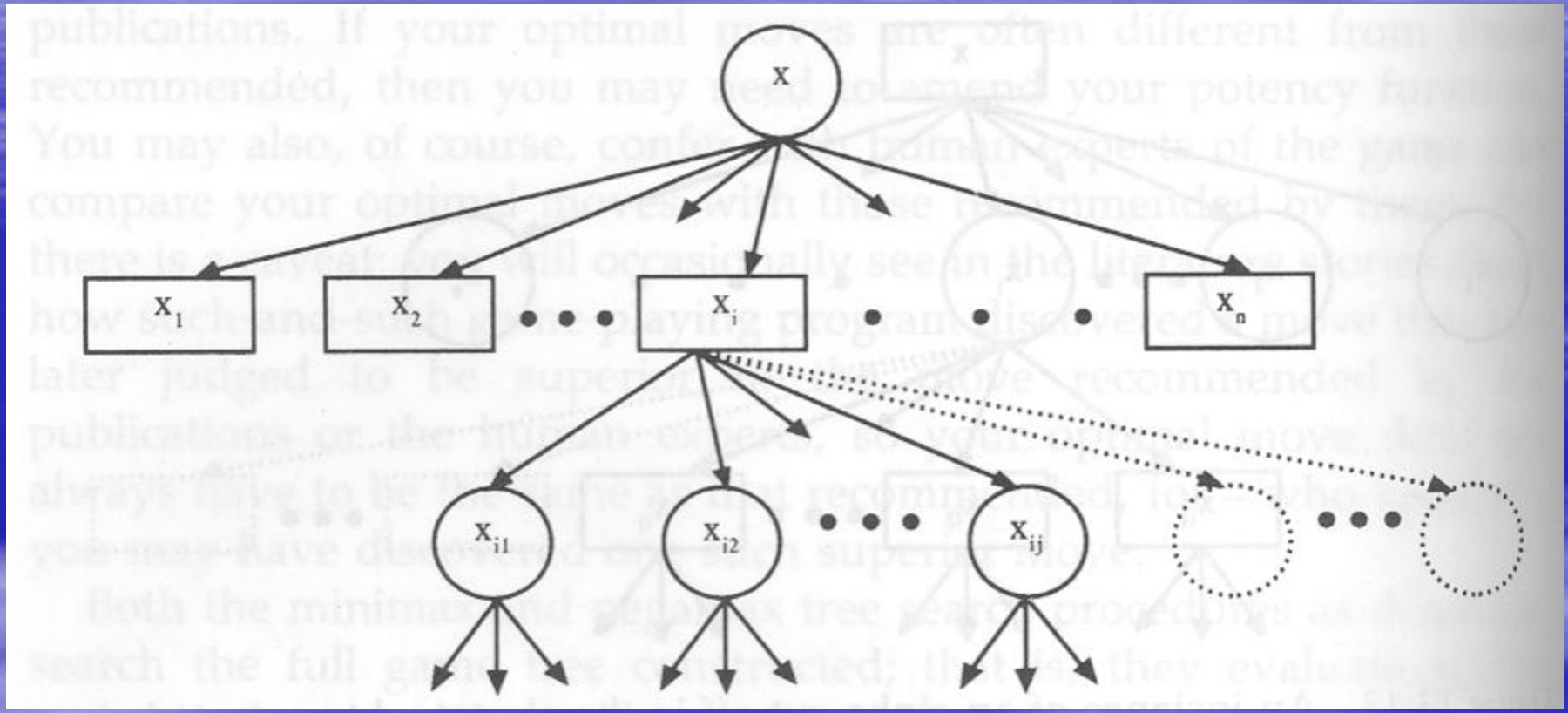
- We don't need to compute the value at this node.
- No matter what it is it can't effect the value of the root node.

Alpha-beta pruning



Max Searching

Alpha-beta pruning



Min Searching

Alpha-beta pruning

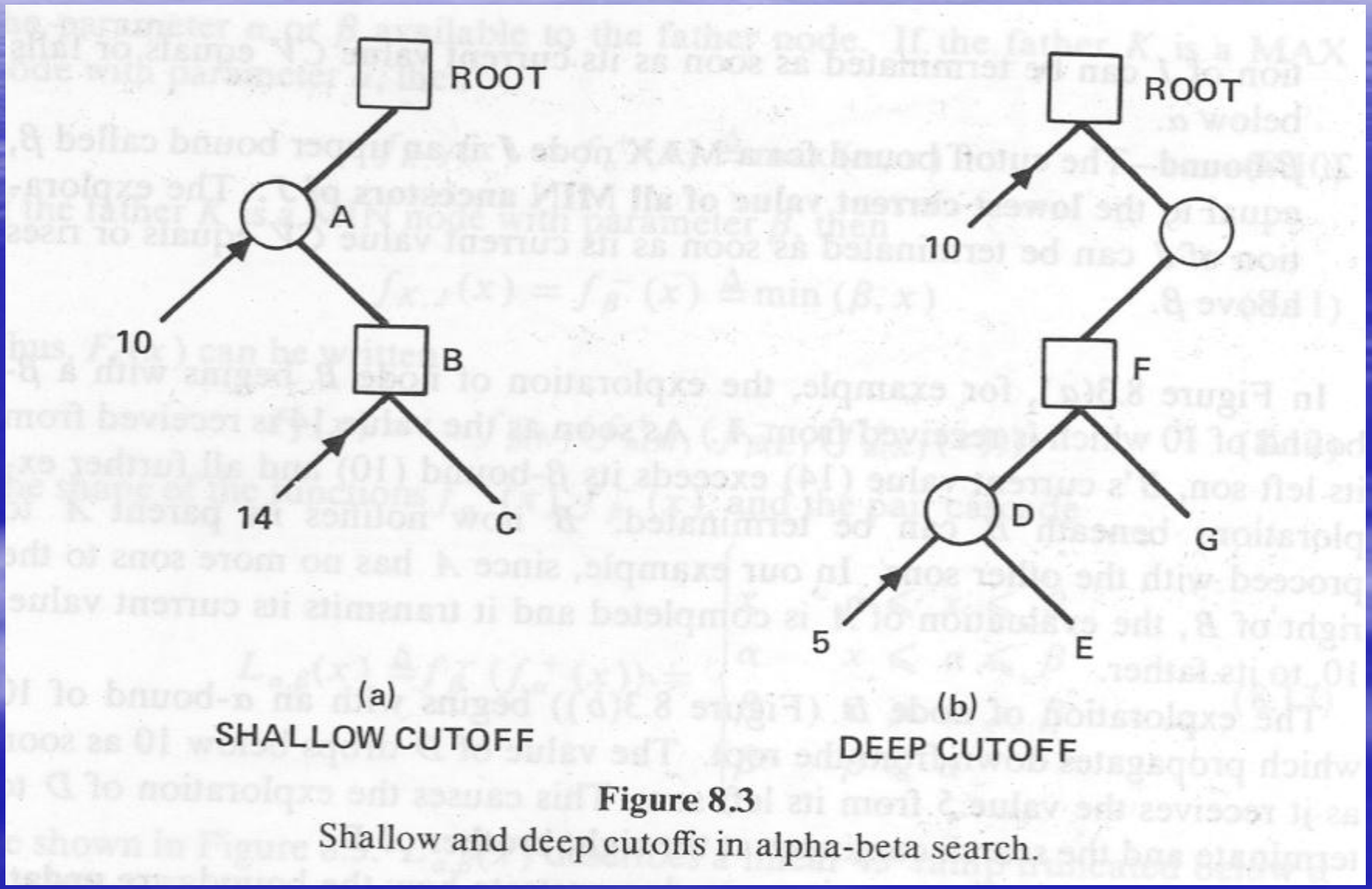
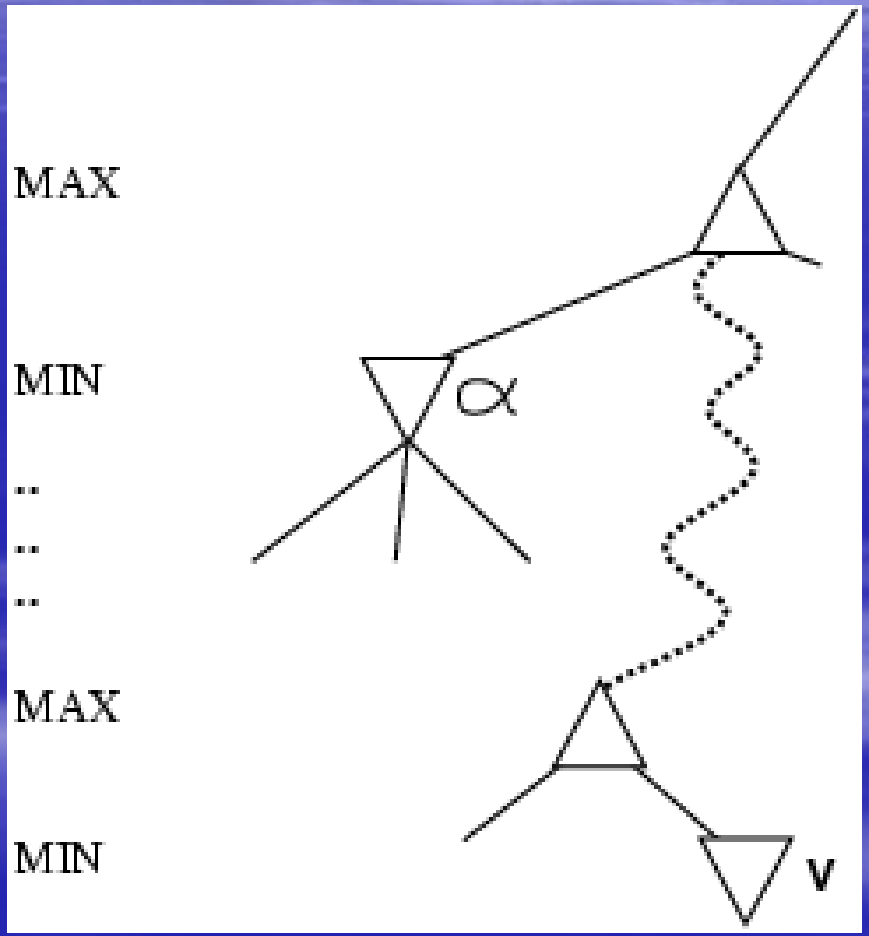


Figure 8.3
Shallow and deep cutoffs in alpha-beta search.

Why is it called α-β?

- α is the value of the best (i.e., highest-value) choice found so far at any choice point along the path for *max*
- If *v* is worse than α, *max* will avoid it
- → prune that branch
- Define β similarly for *min*



The Alpha-Beta Procedure

- There are two rules for terminating search:
  - Search can be stopped below any MIN node having a beta value less than or equal to the alpha value of any of its MAX ancestors.
  - Search can be stopped below any MAX node having an alpha value greater than or equal to the beta value of any of its MIN ancestors.

The α-β algorithm

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

  $v \leftarrow$ MAX-VALUE($state, -\infty, +\infty$)
  return the *action* in SUCCESSORS(*state*) with value v

---

function MAX-VALUE($state, \alpha, \beta$) **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*)
  $v \leftarrow -\infty$
  for a, s in SUCCESSORS(*state*) **do**
    $v \leftarrow$ MAX(v, MIN-VALUE(s, α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$ MAX(α, v)
  return v

The α-β 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
```

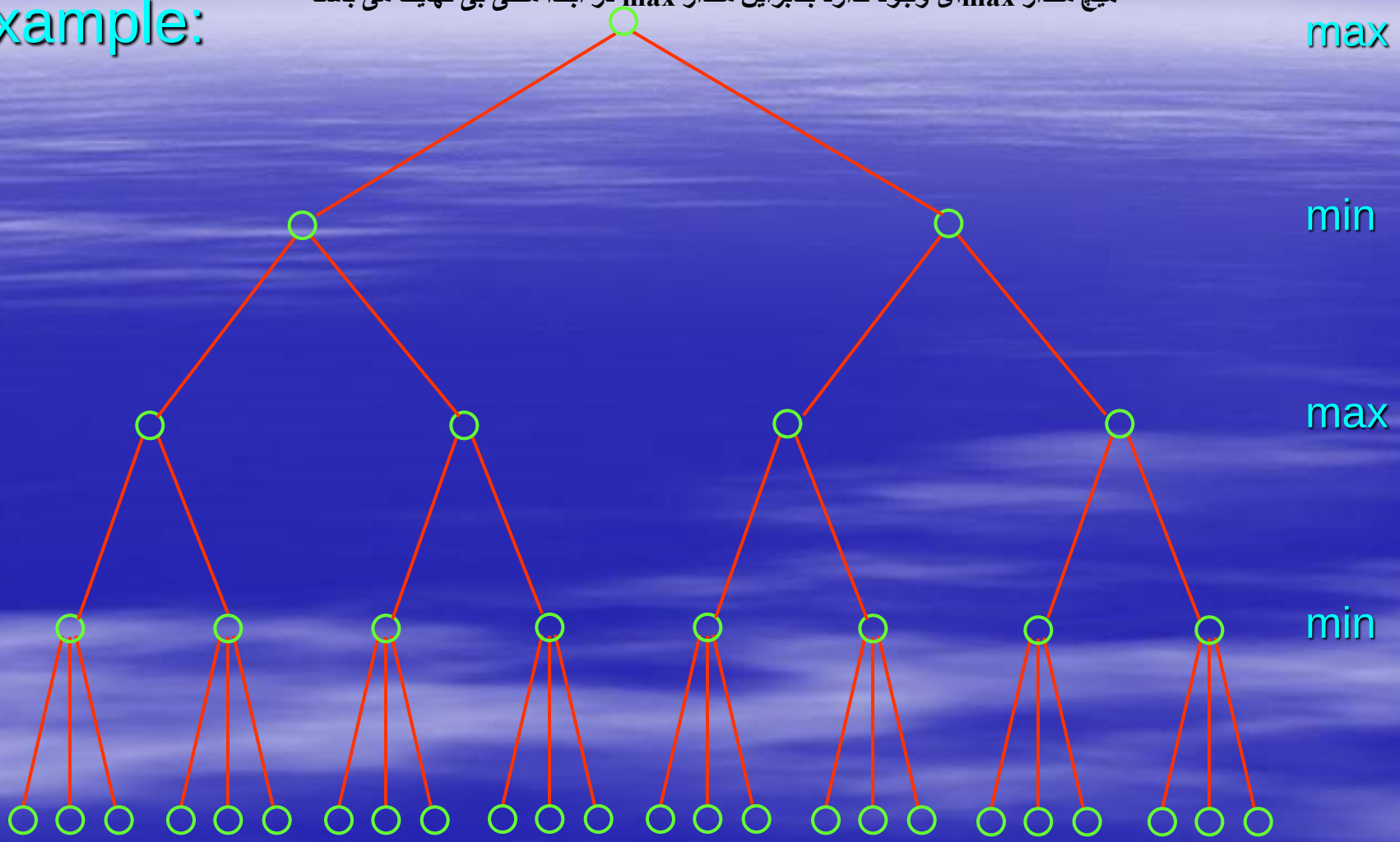
Properties of α-β

- Pruning does not affect final result
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
  → doubles depth of search

The Alpha-Beta Procedure

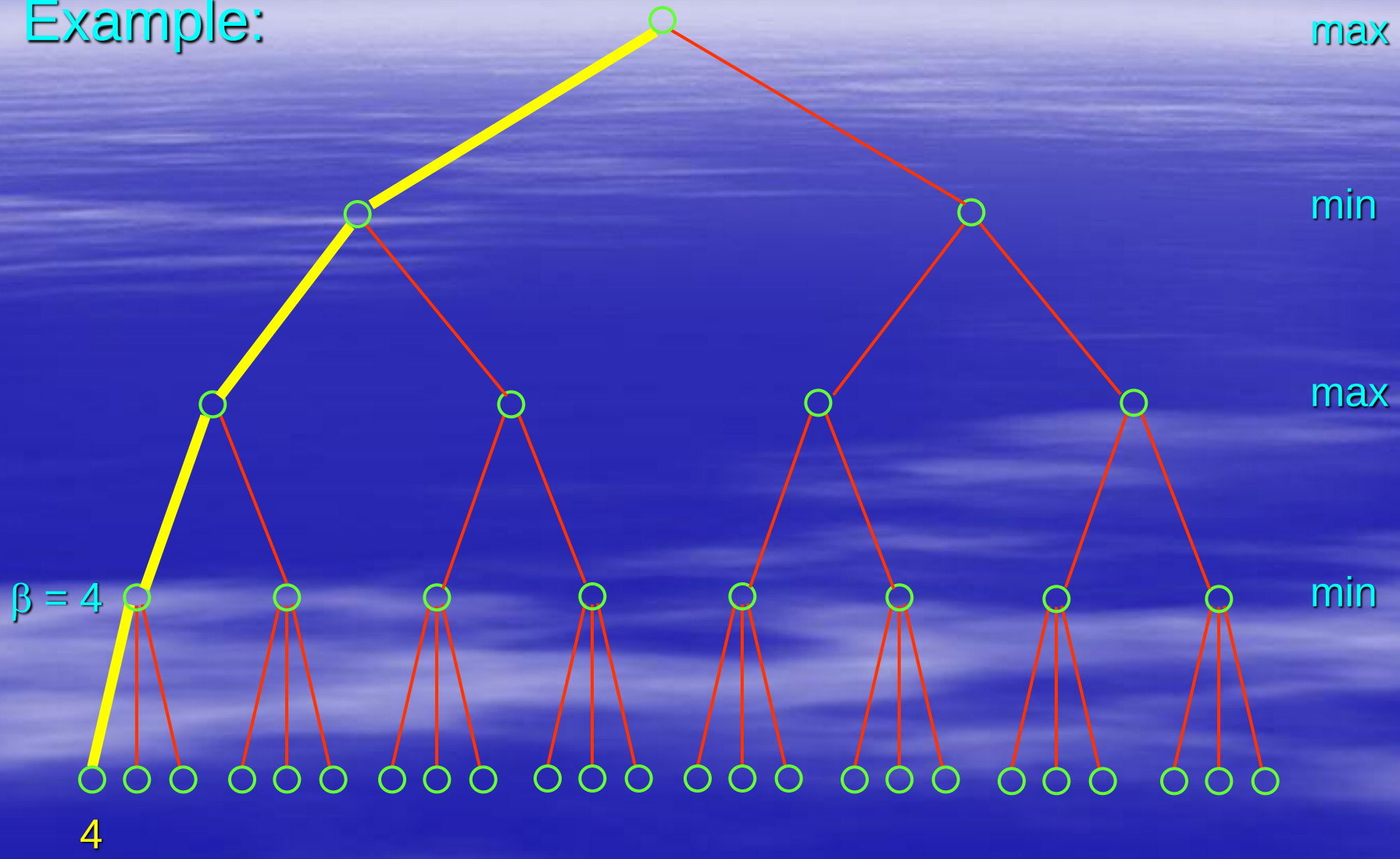
Example:

هیچ مقدار maxای وجود ندارد بنابراین مقدار max در ابتدا منفی بی نهایت می باشد



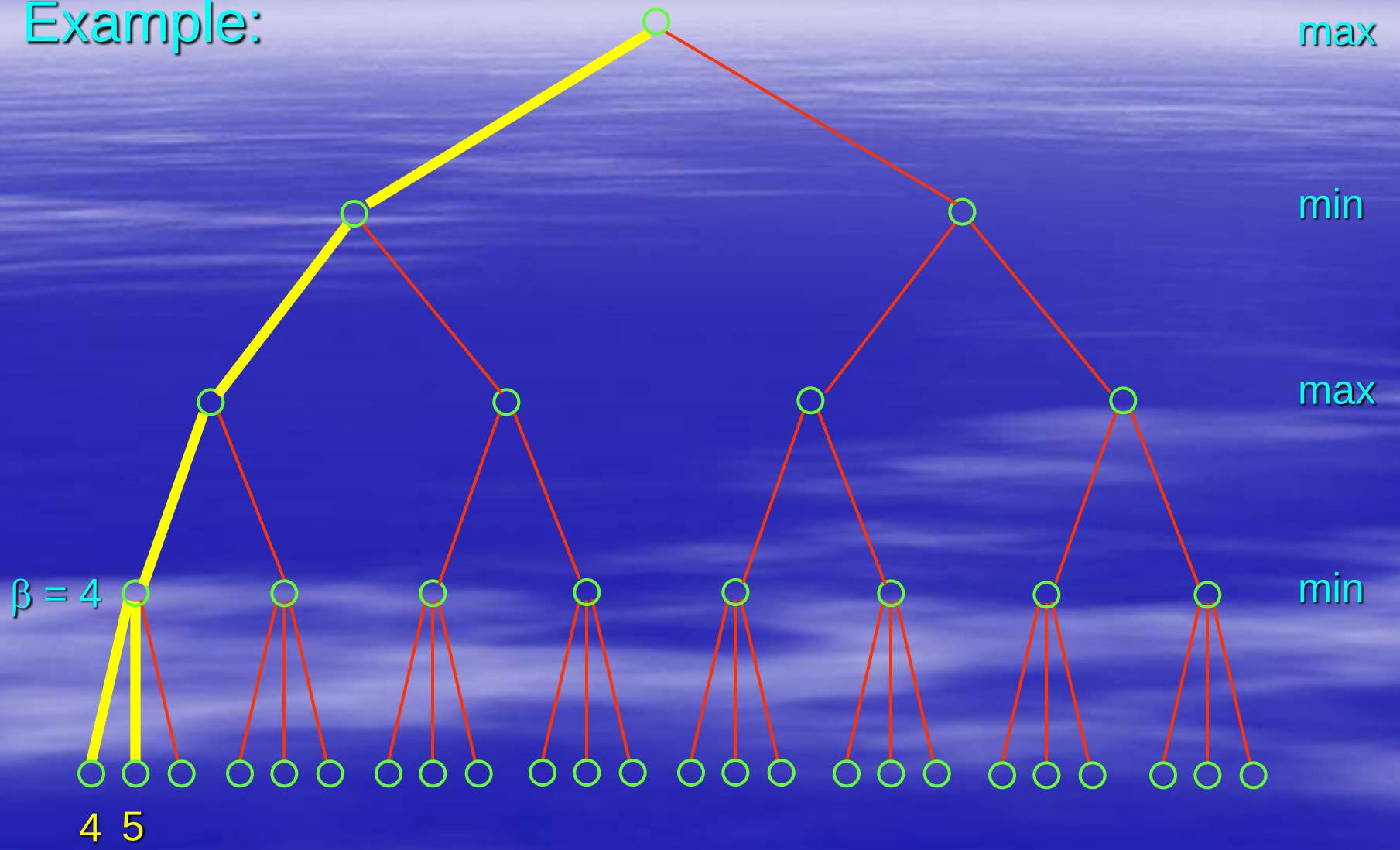
The Alpha-Beta Procedure

Example:



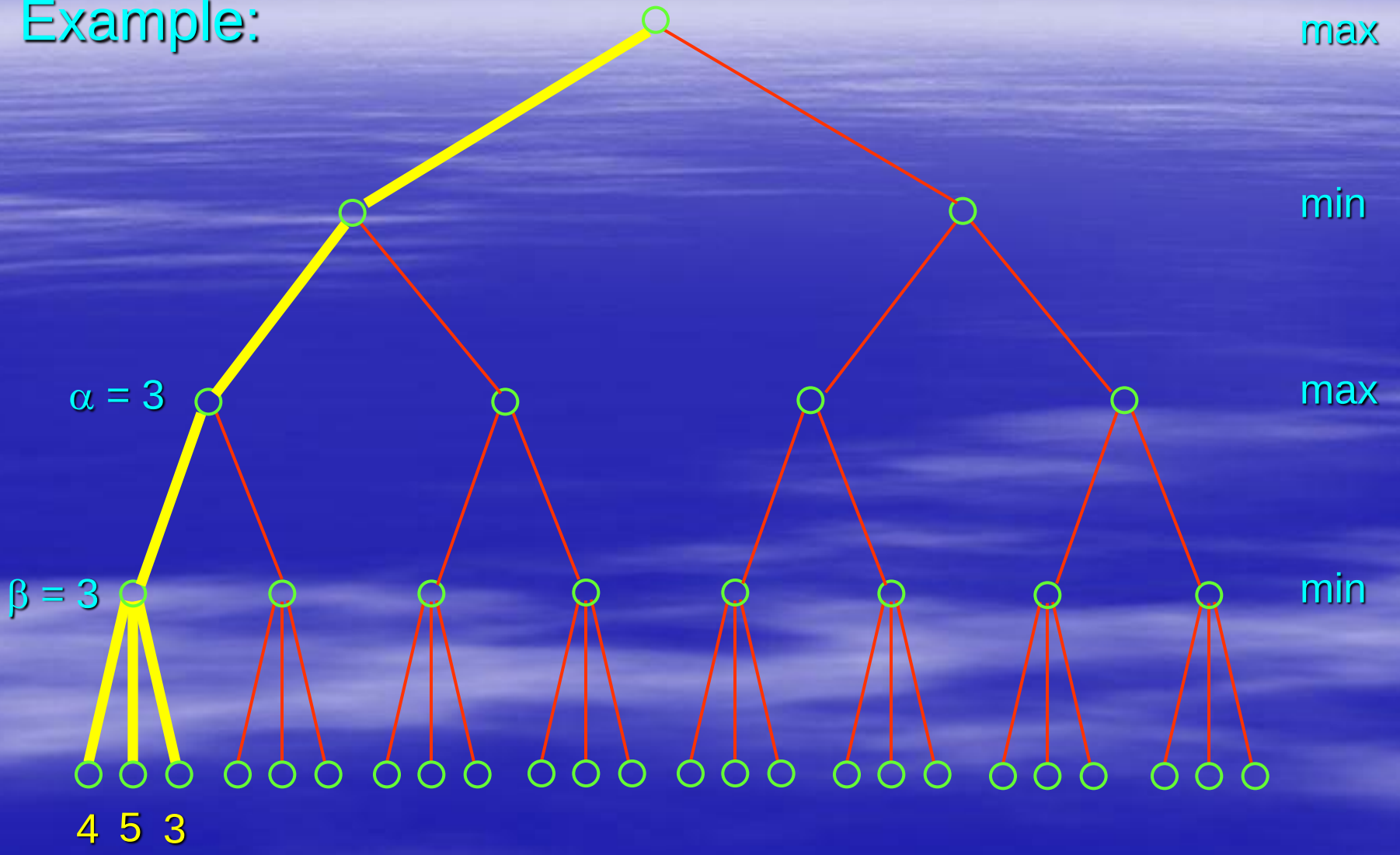
The Alpha-Beta Procedure

Example:



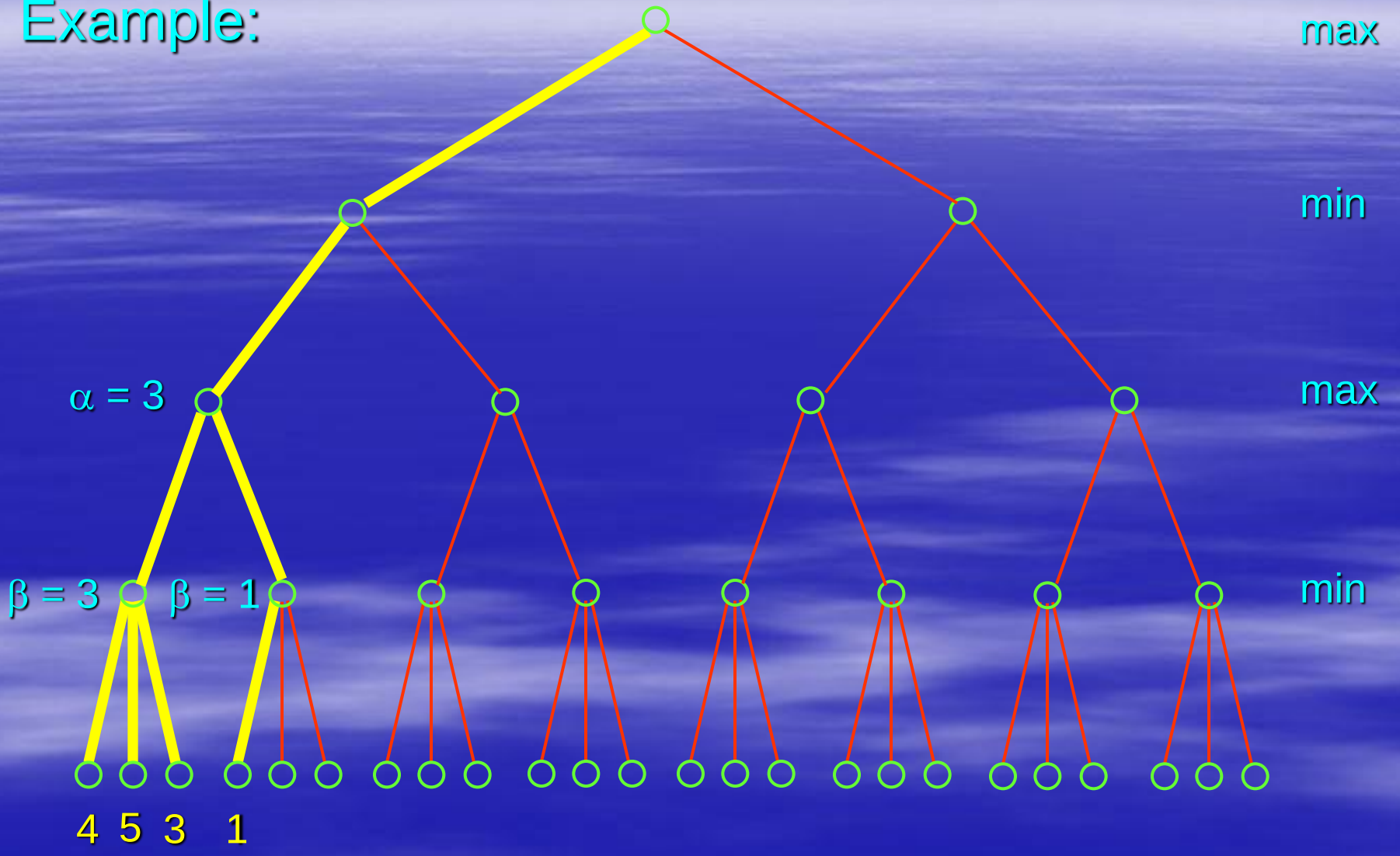
The Alpha-Beta Procedure

Example:



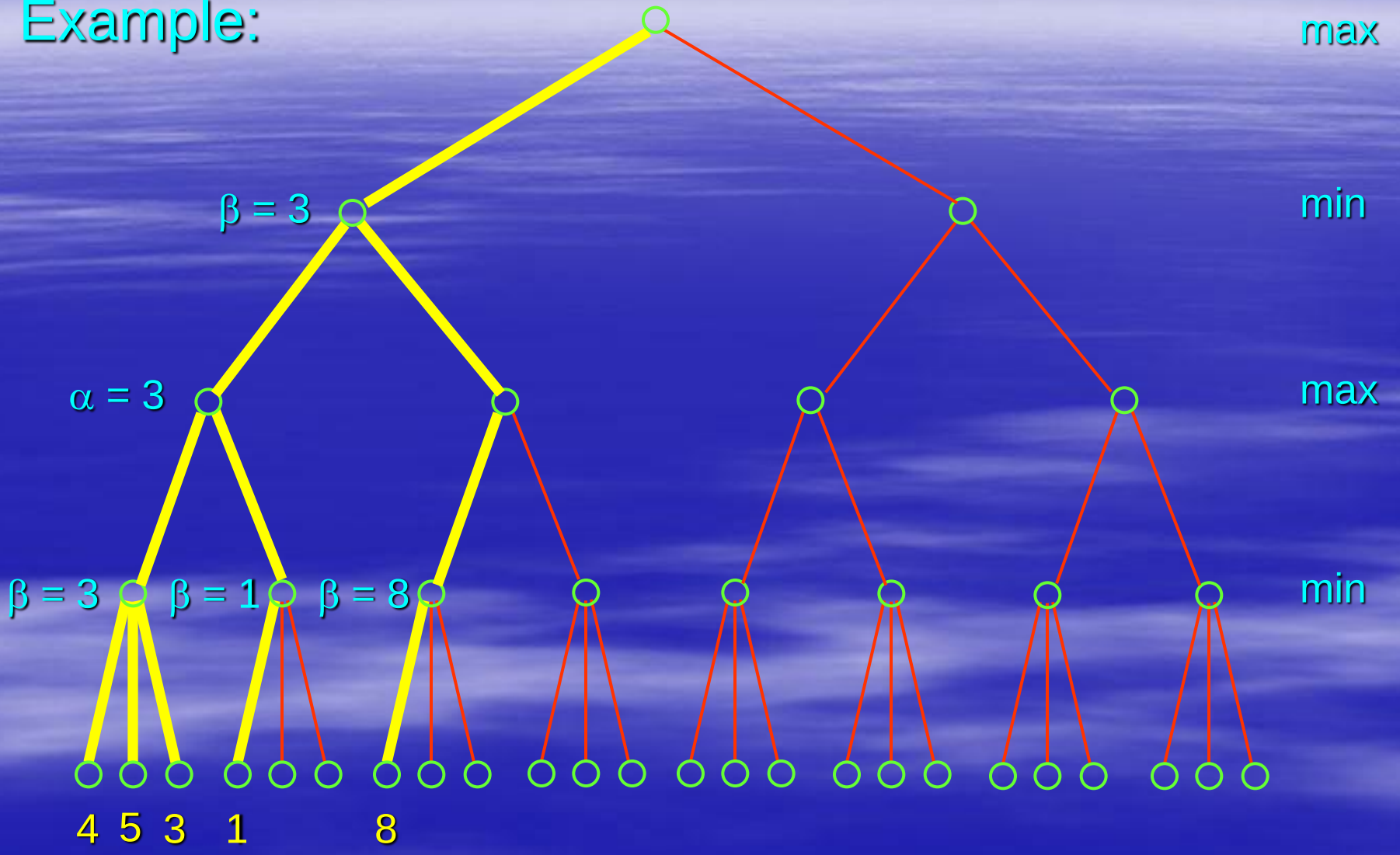
The Alpha-Beta Procedure

Example:



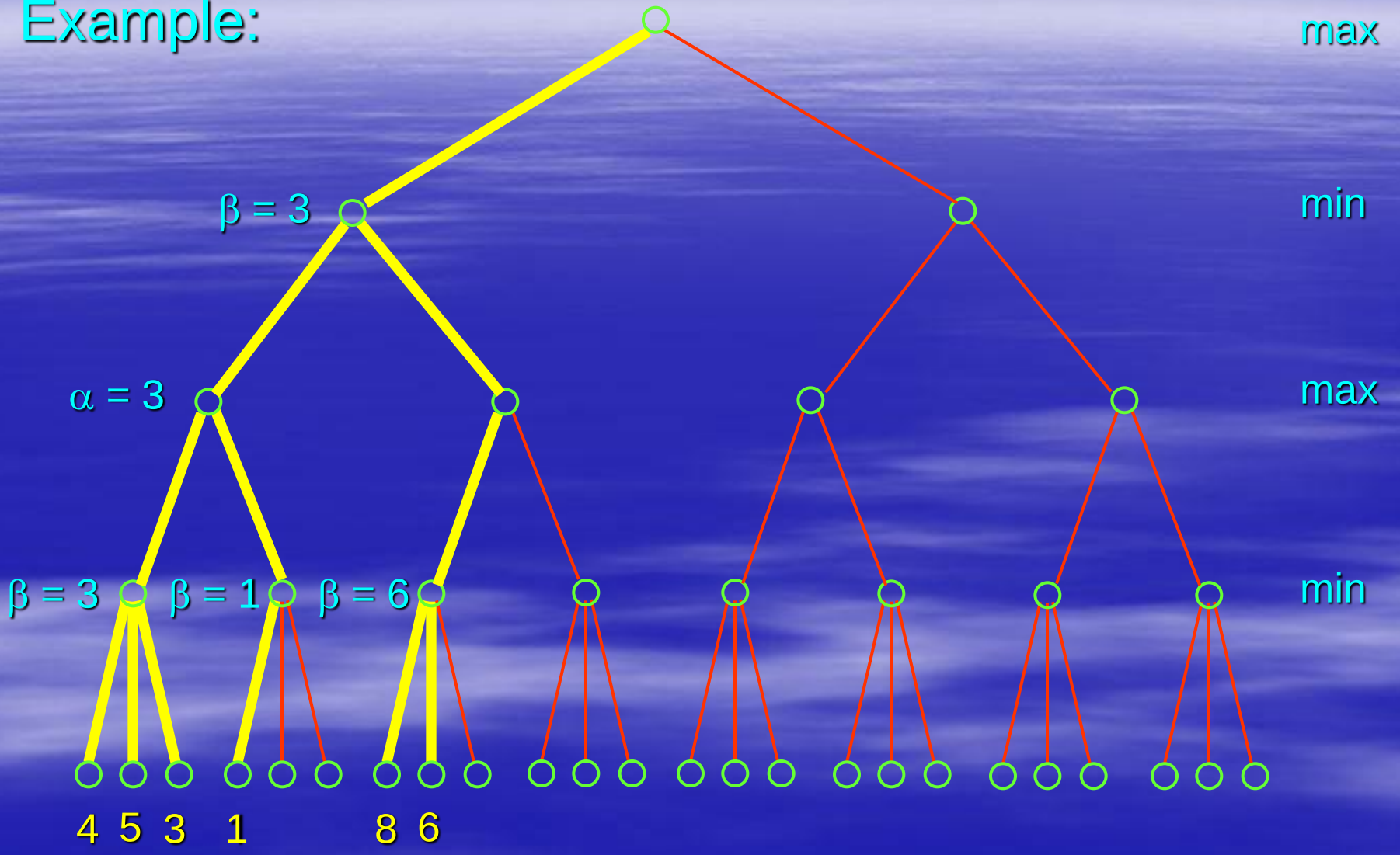
The Alpha-Beta Procedure

Example:



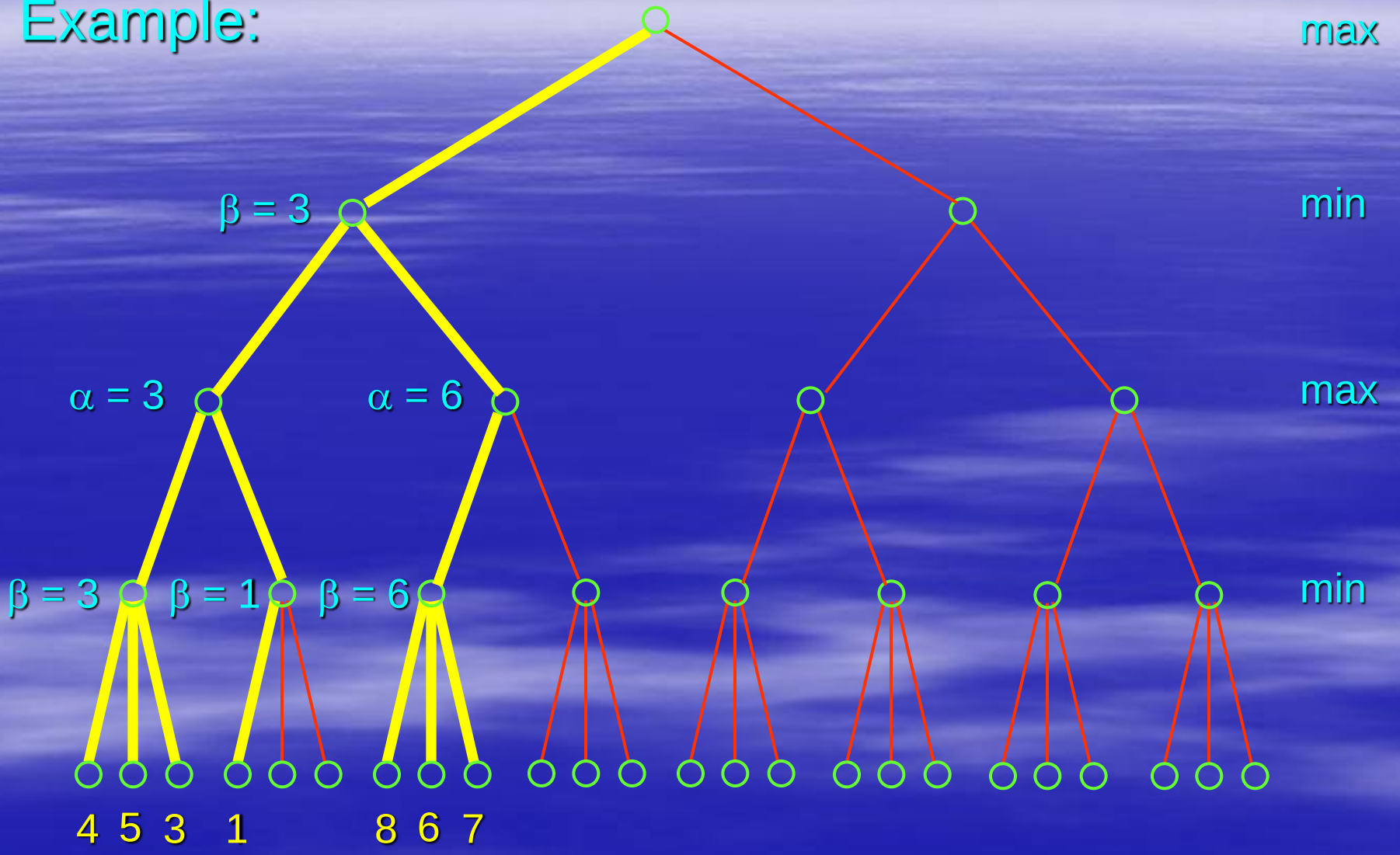
The Alpha-Beta Procedure

Example:



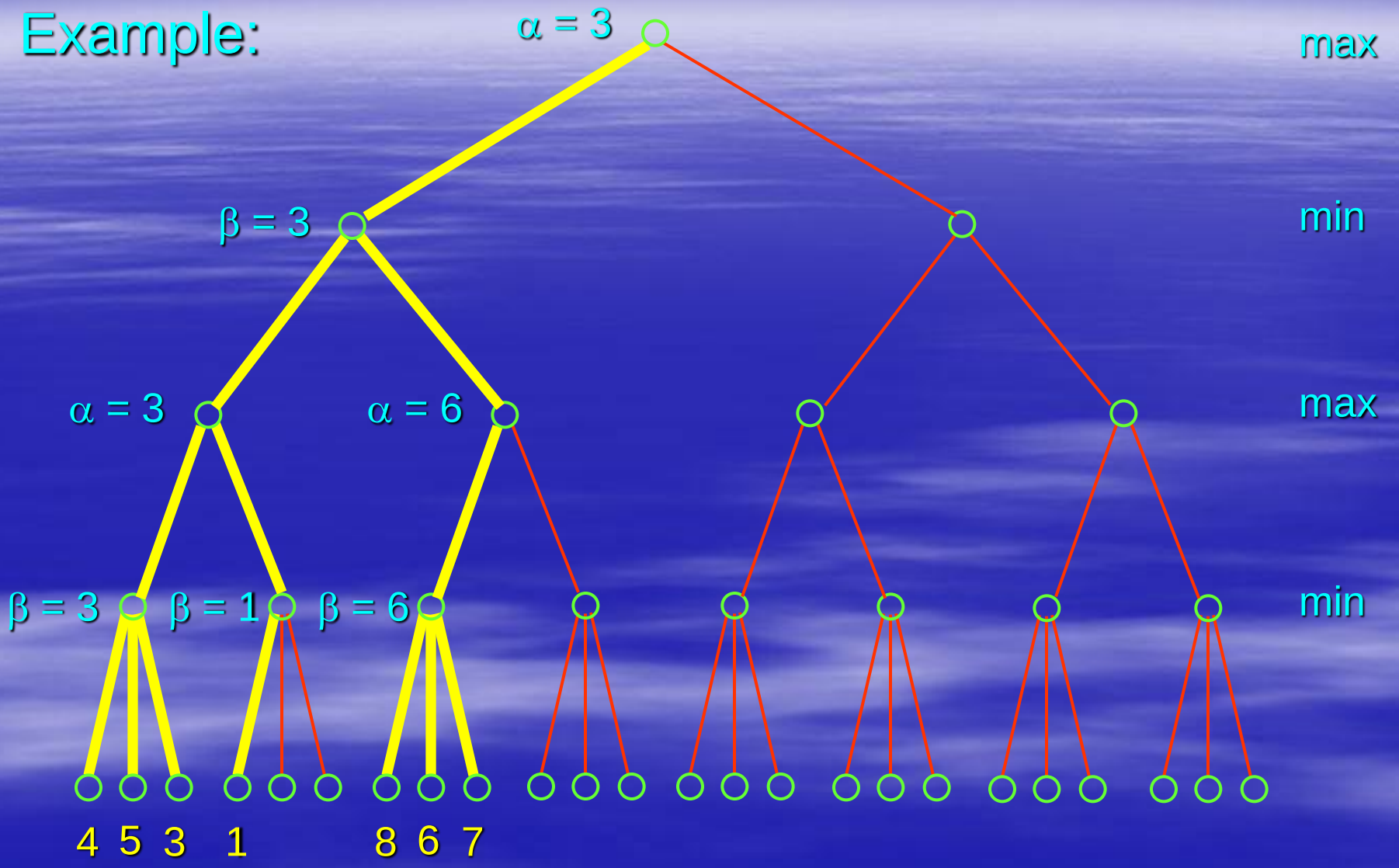
The Alpha-Beta Procedure

Example:



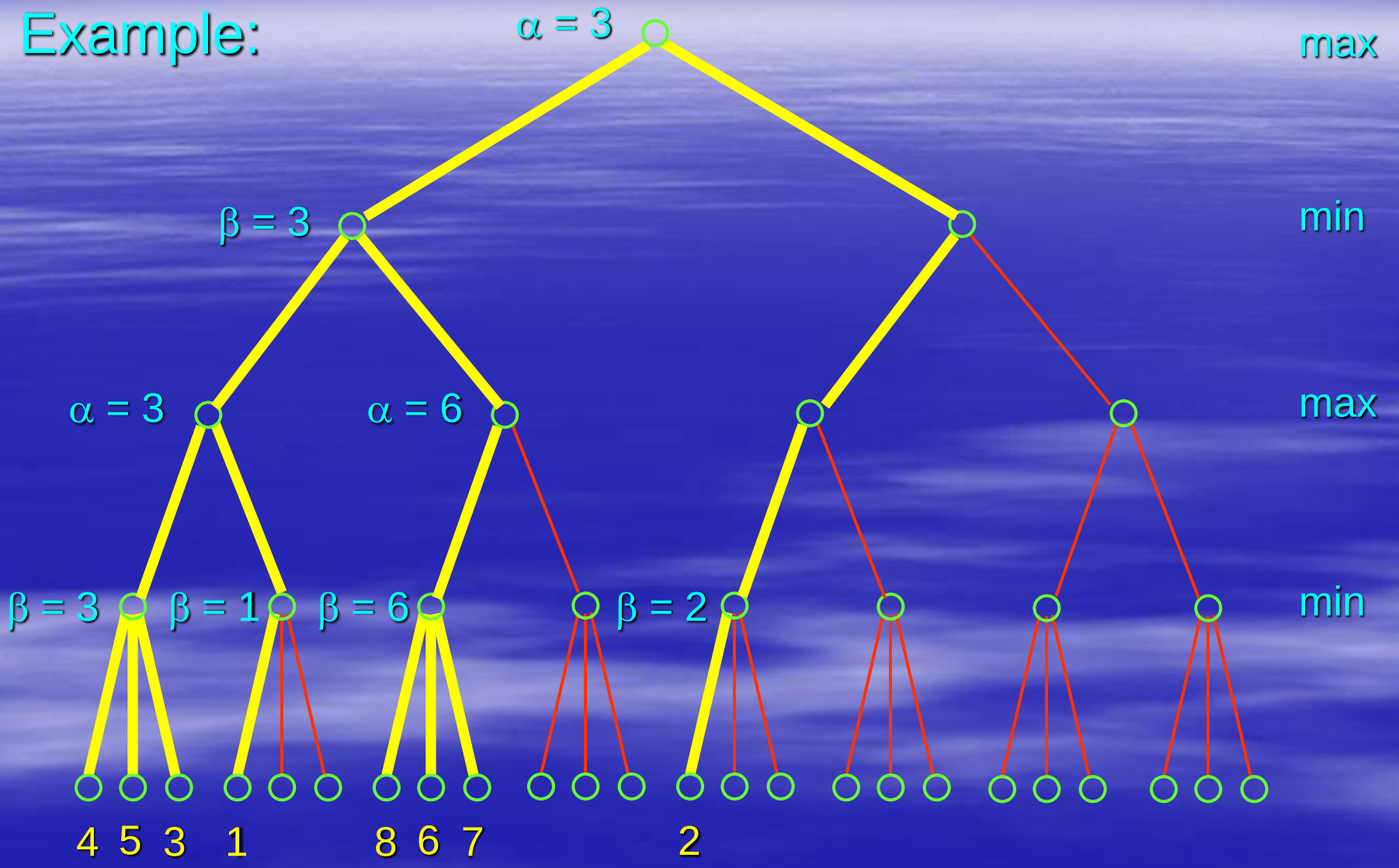
The Alpha-Beta Procedure

Example:



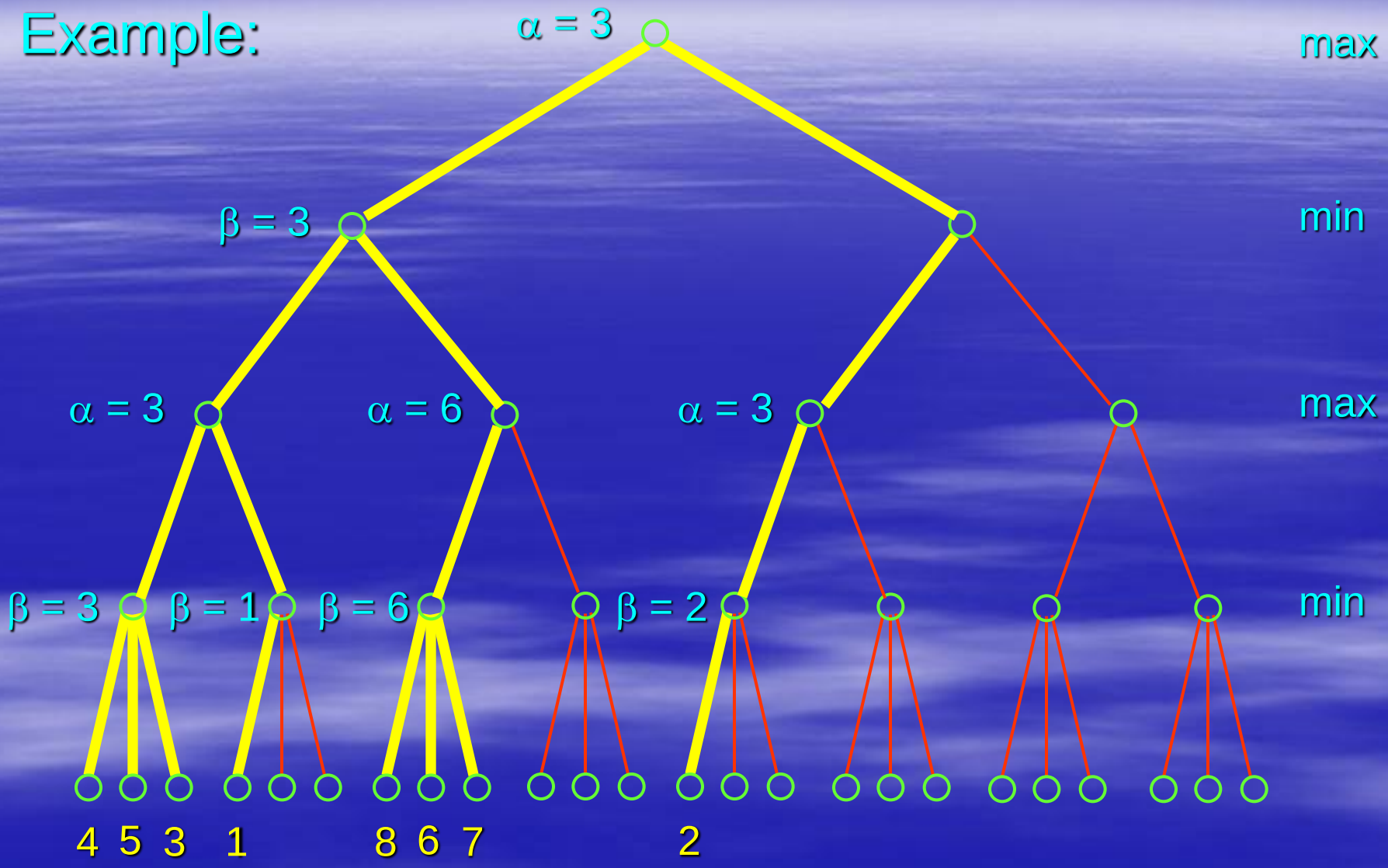
The Alpha-Beta Procedure

Example:



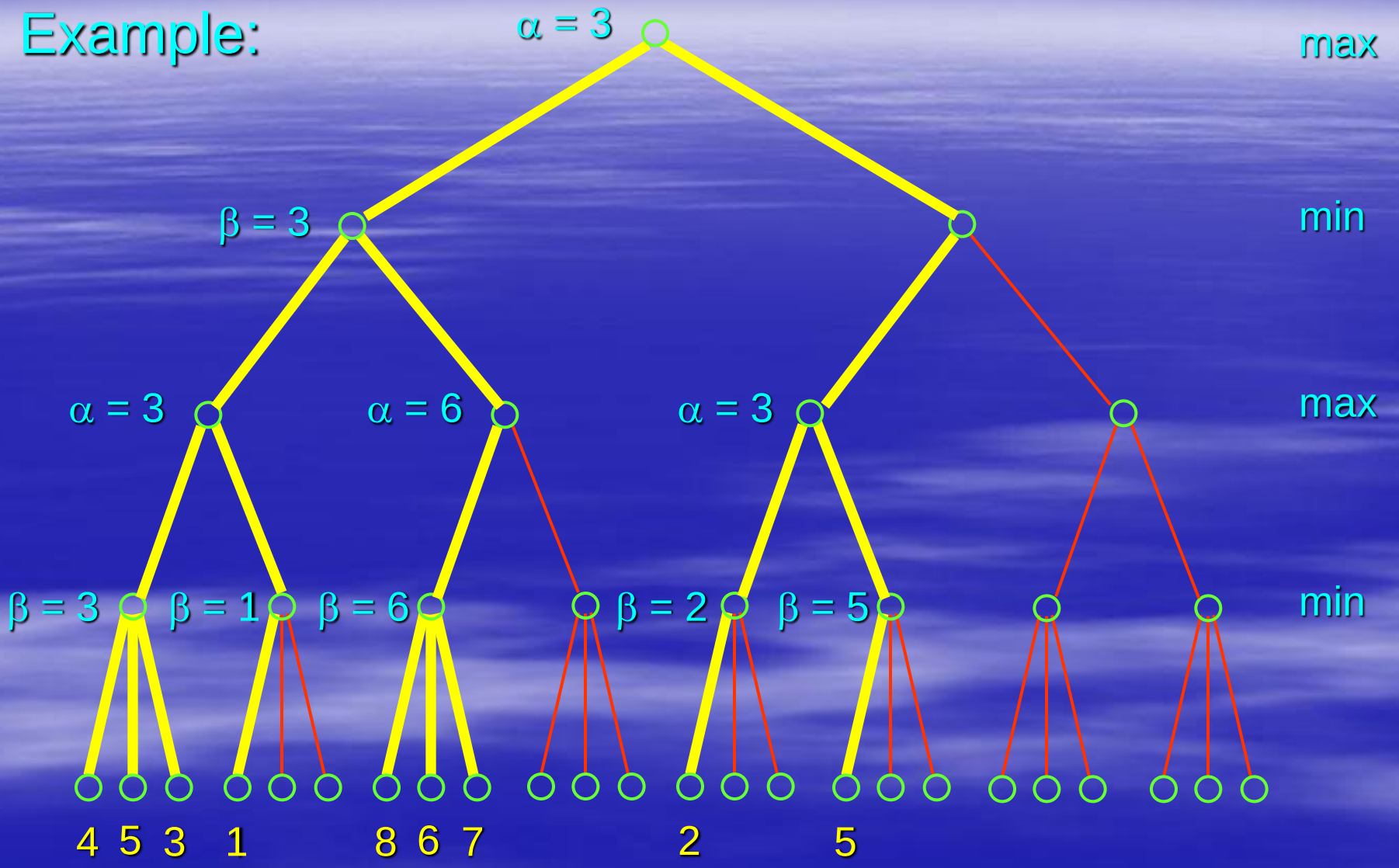
The Alpha-Beta Procedure

Example:



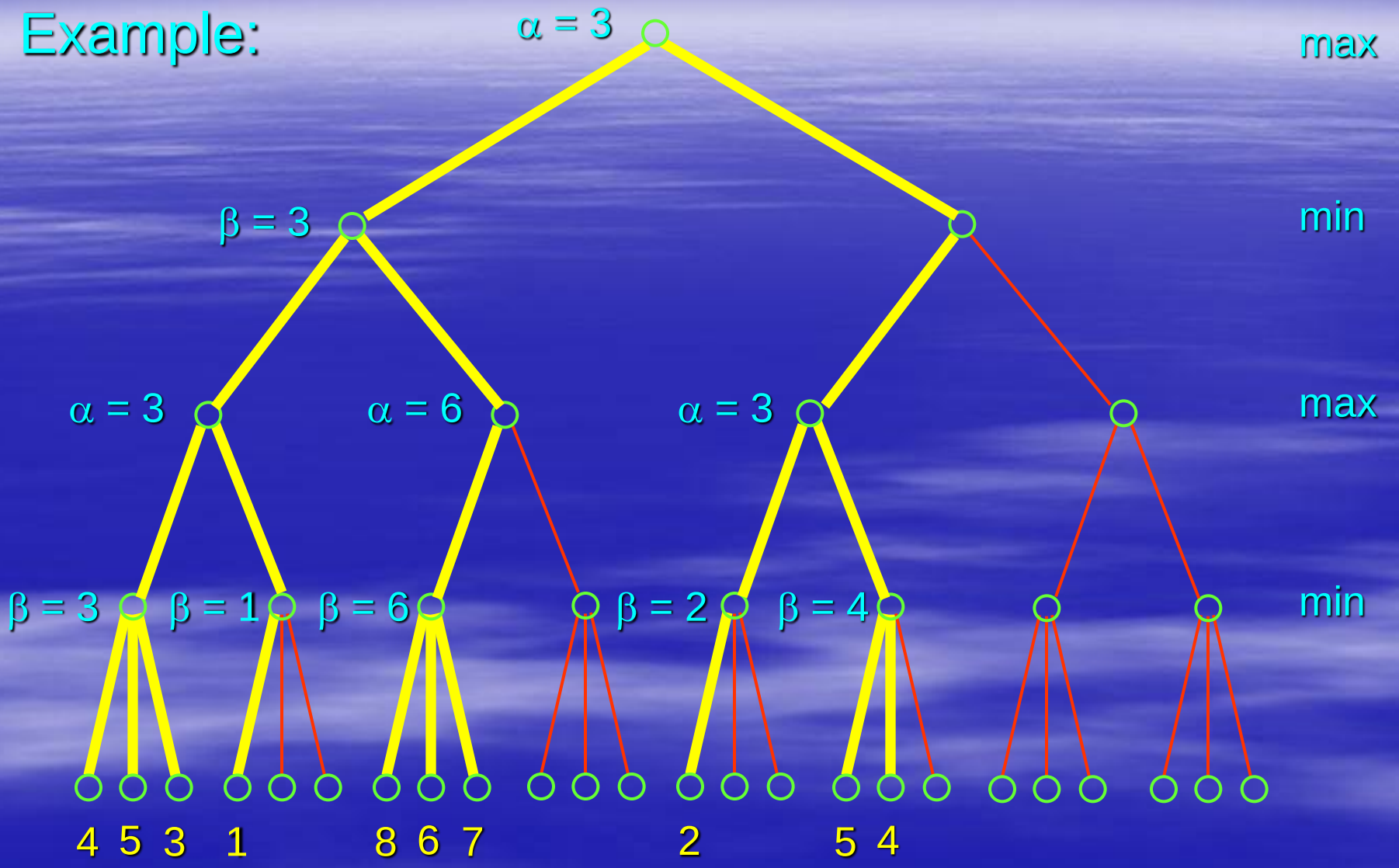
The Alpha-Beta Procedure

Example:



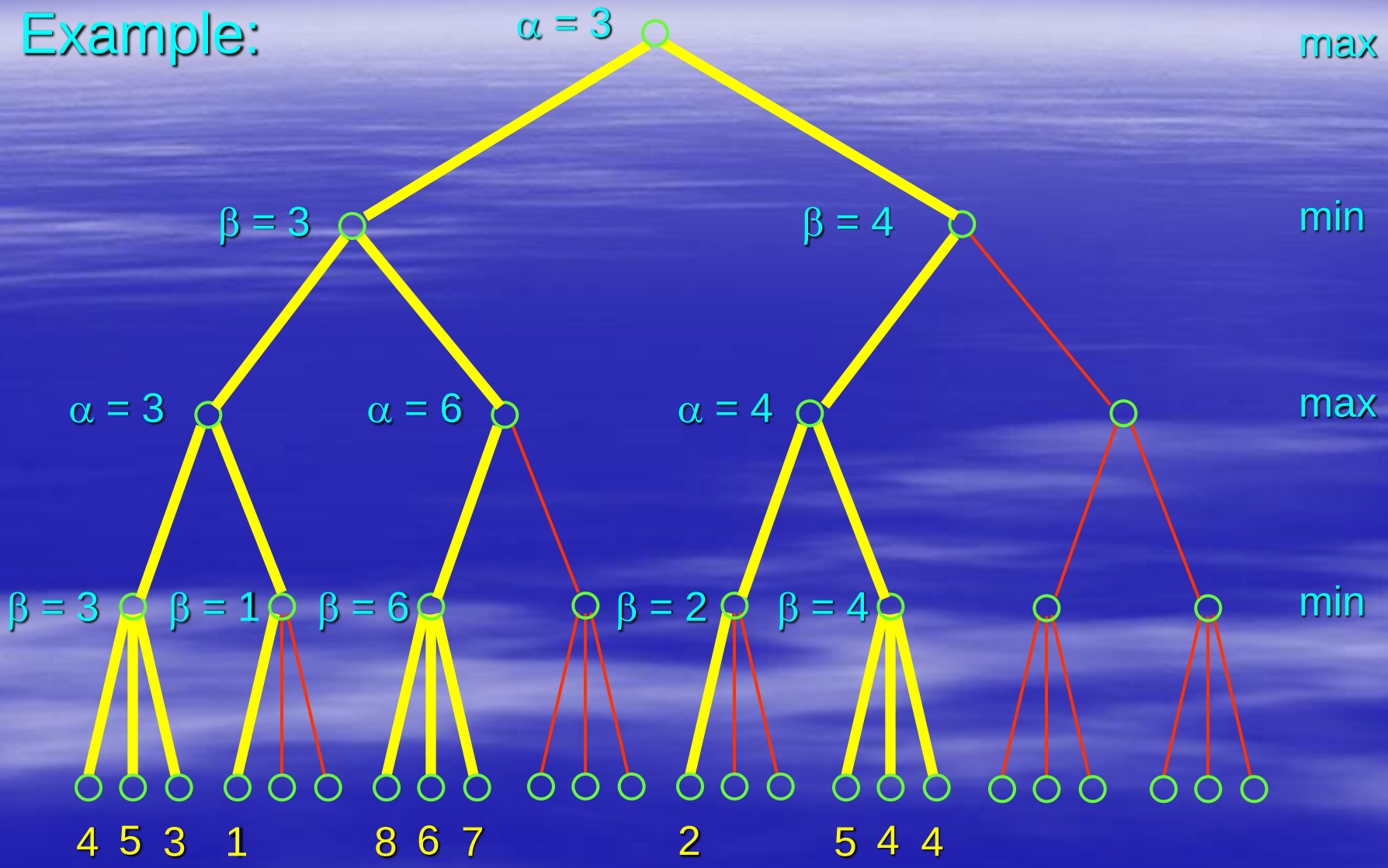
The Alpha-Beta Procedure

Example:



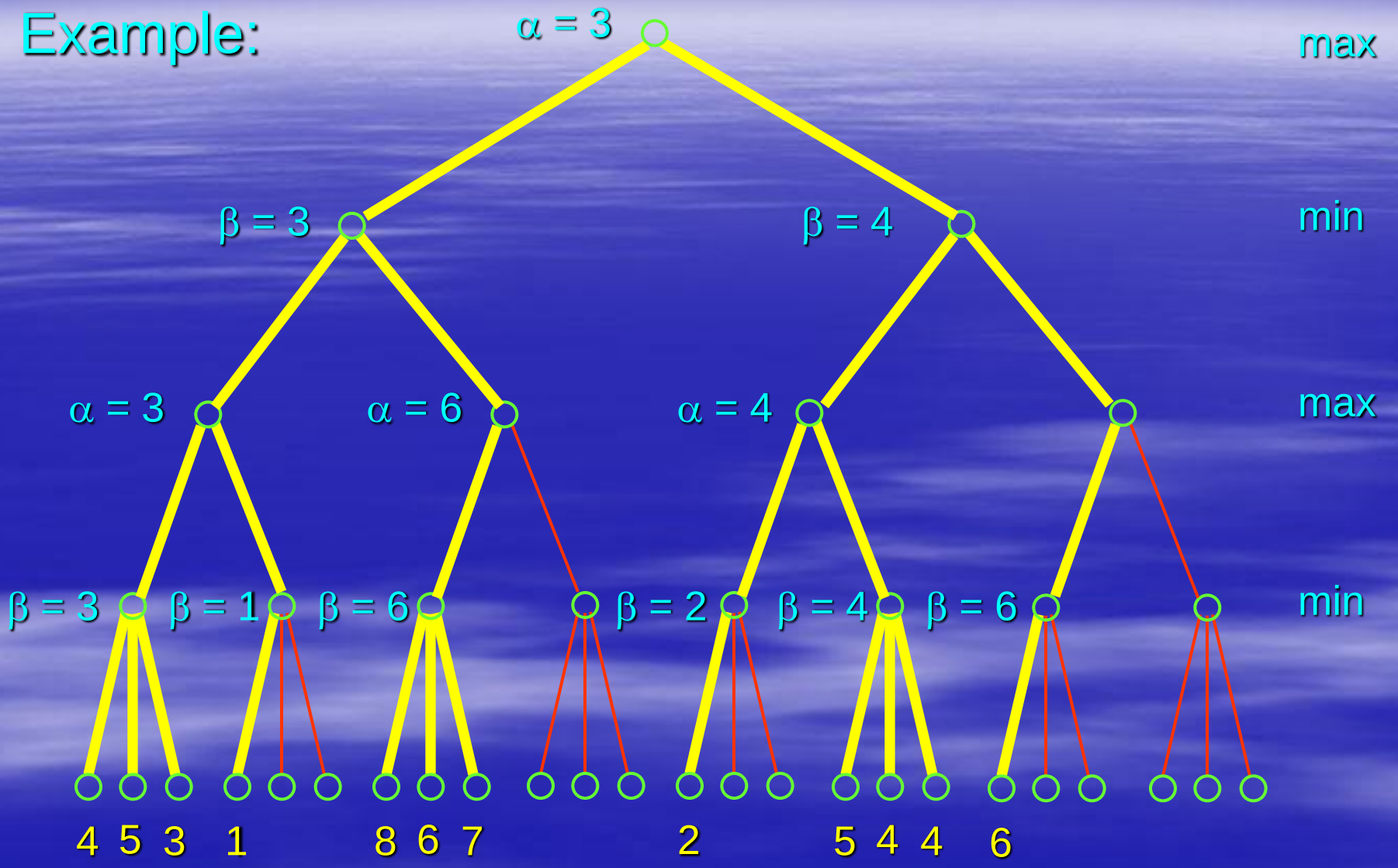
The Alpha-Beta Procedure

Example:



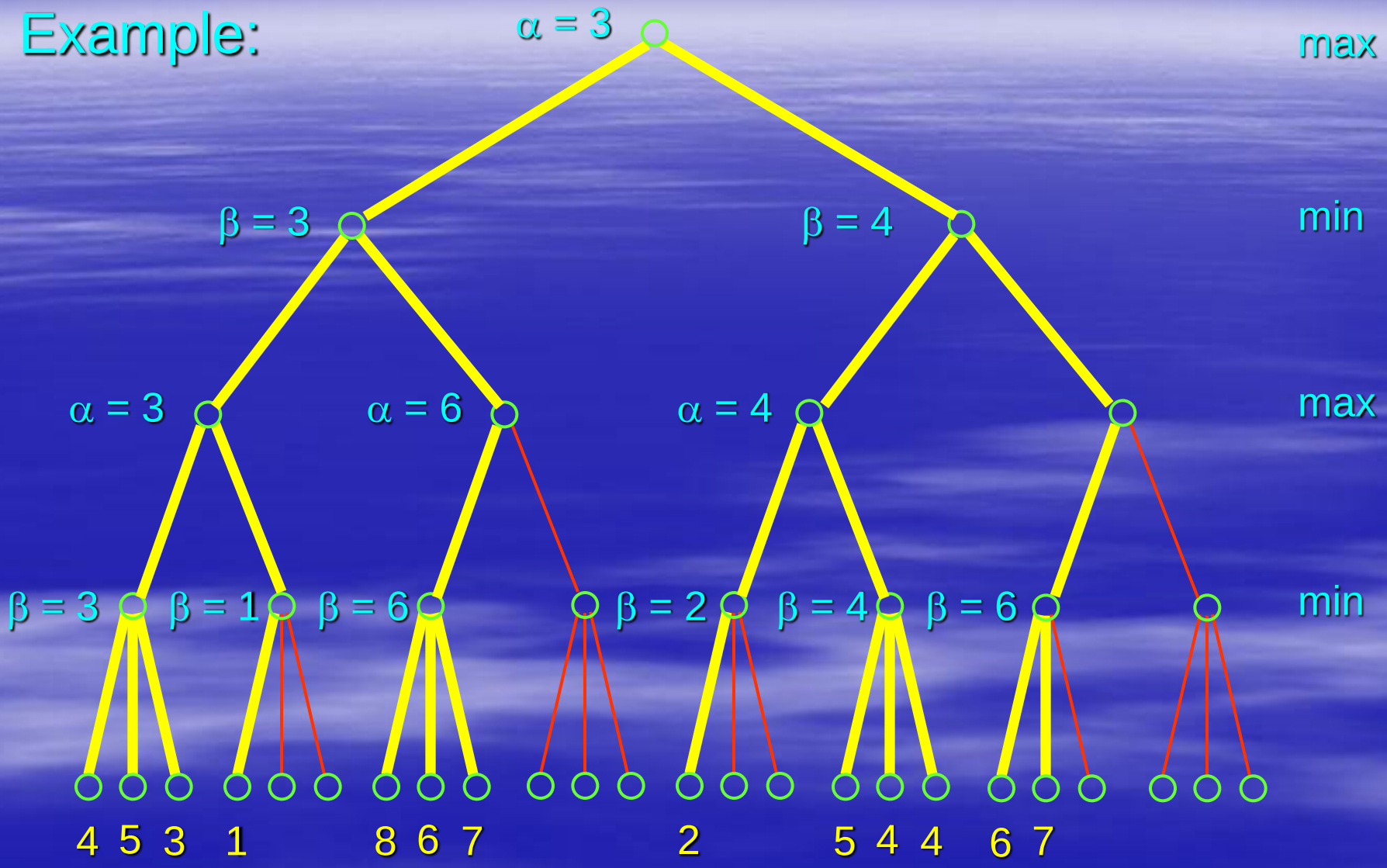
The Alpha-Beta Procedure

Example:



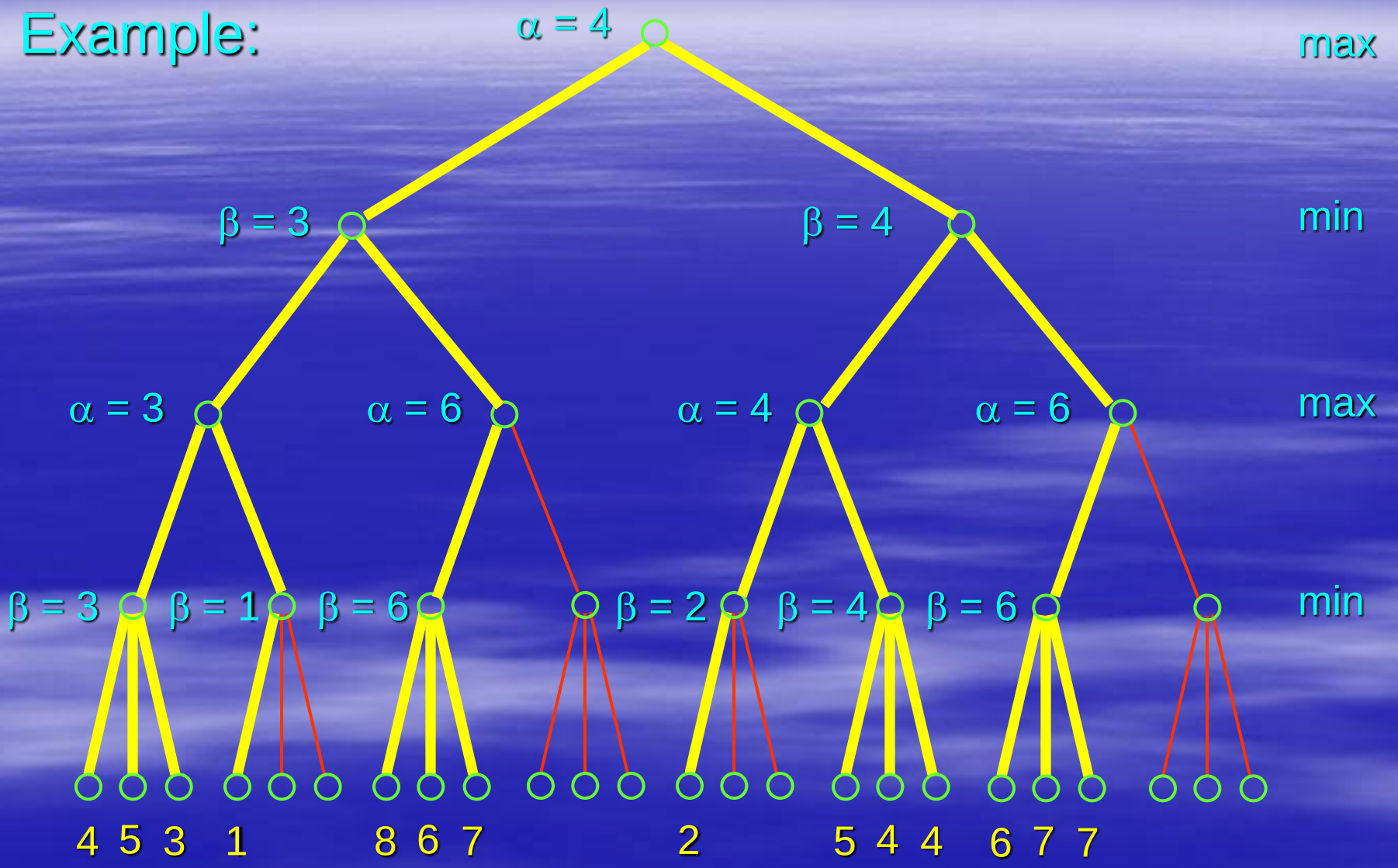
The Alpha-Beta Procedure

Example:



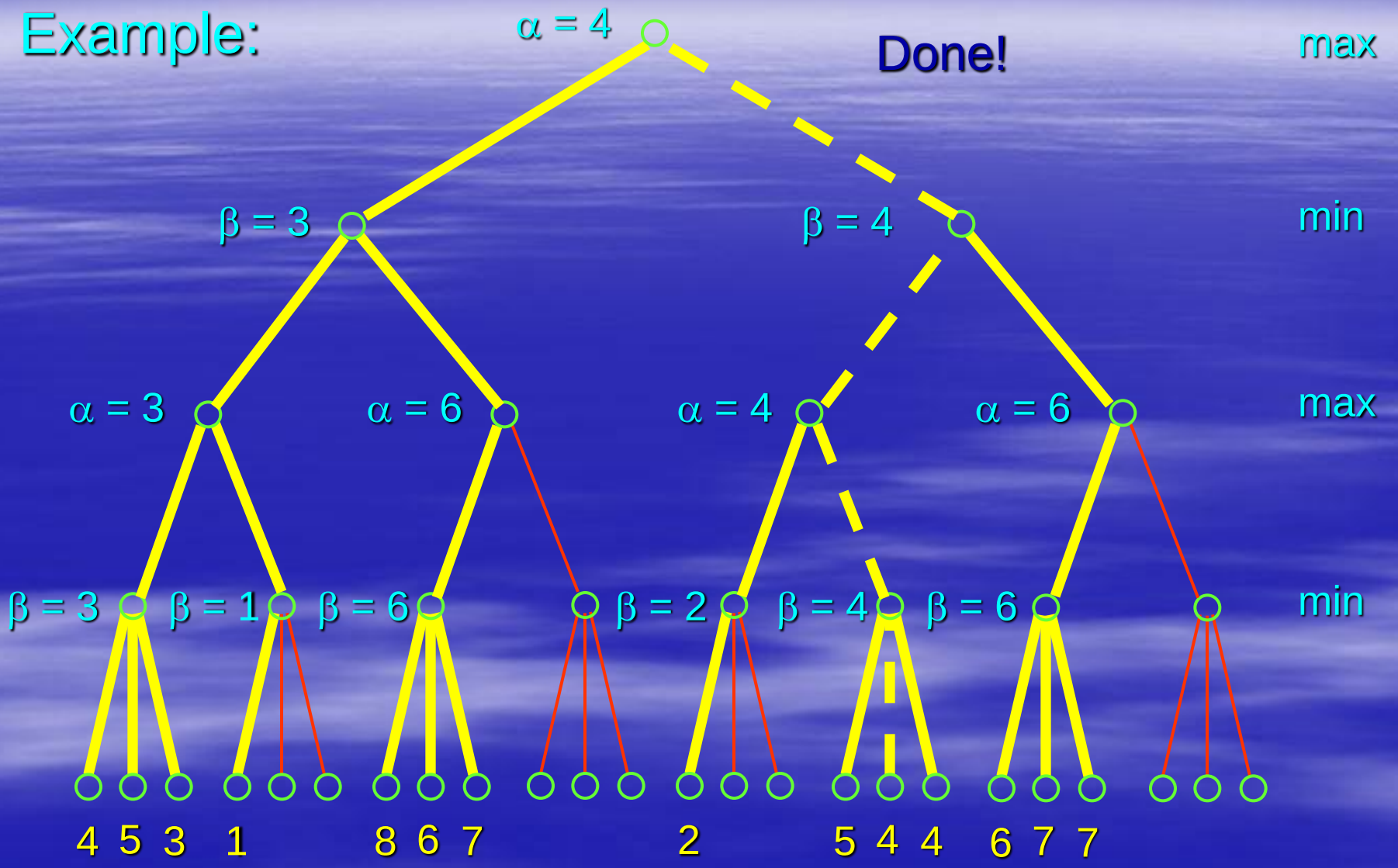
The Alpha-Beta Procedure

Example:

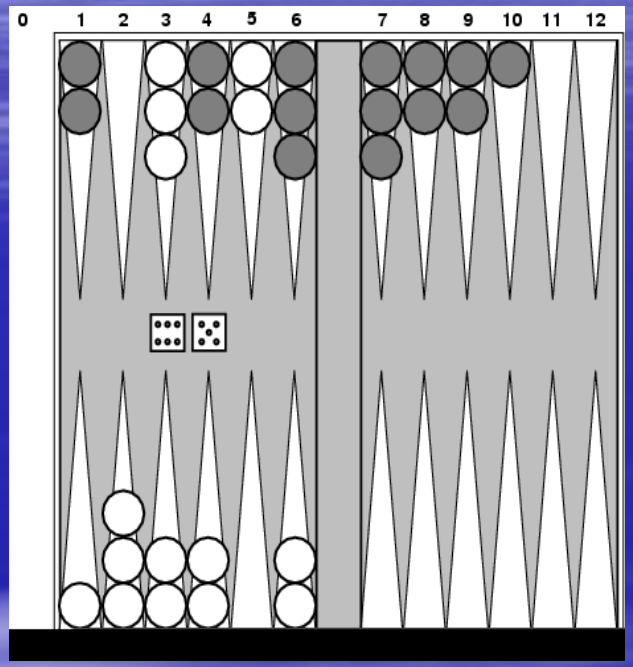


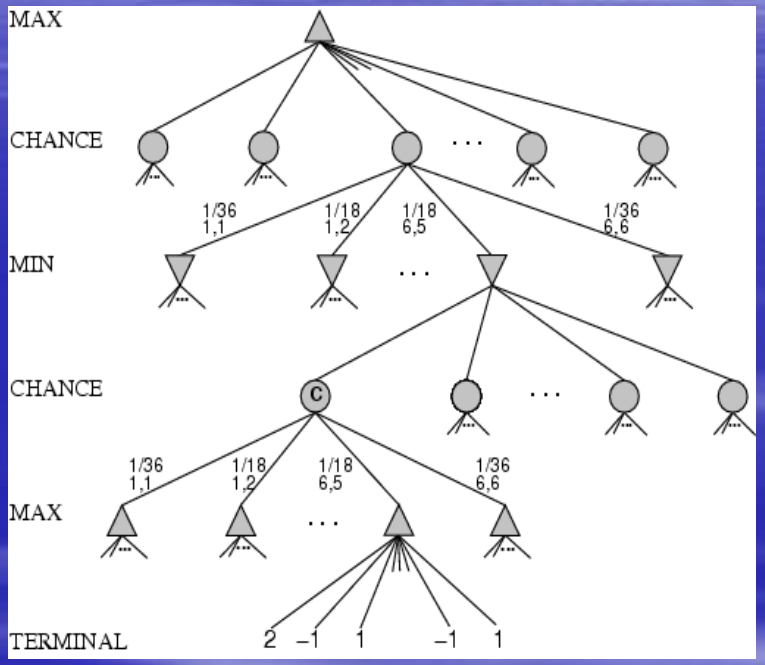
The Alpha-Beta Procedure

Example:



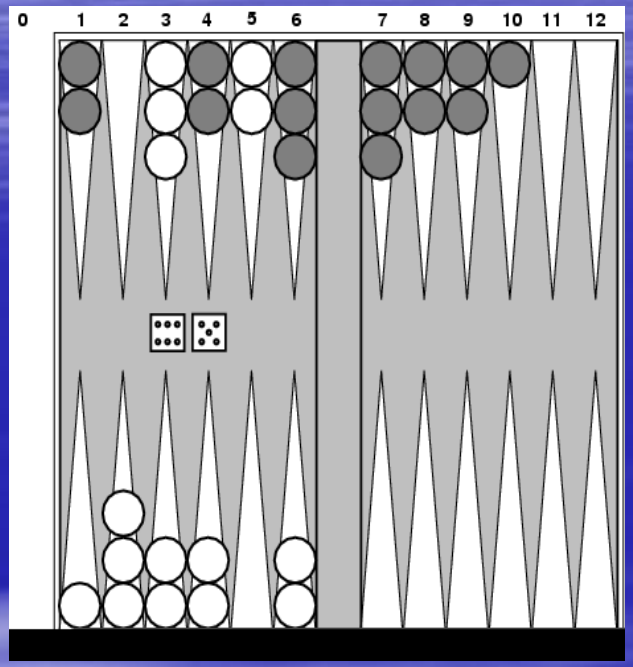
Games that include chance

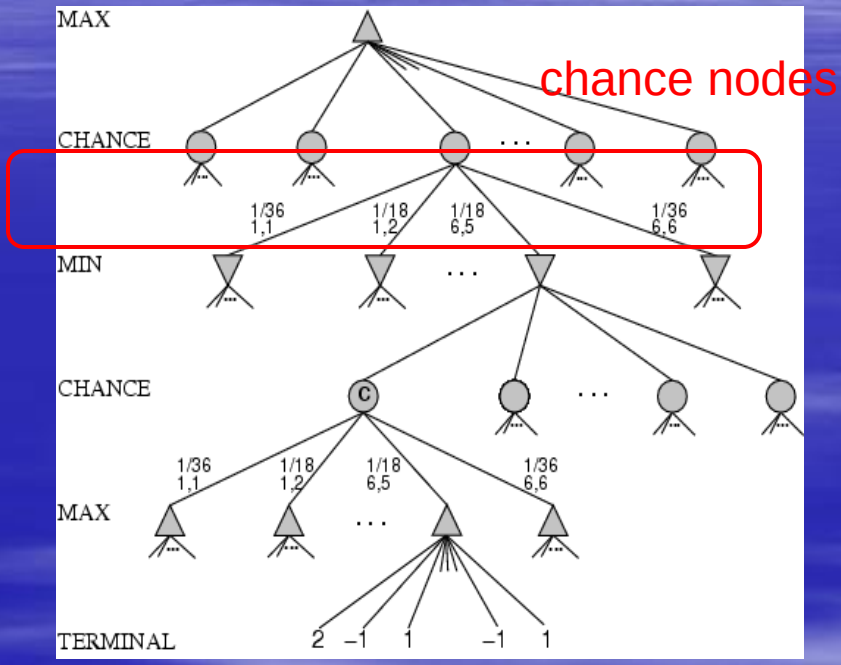




- Possible moves (5-10,5-11), (5-11,19-24),(5-10,10-16) and (5-11,11-16)

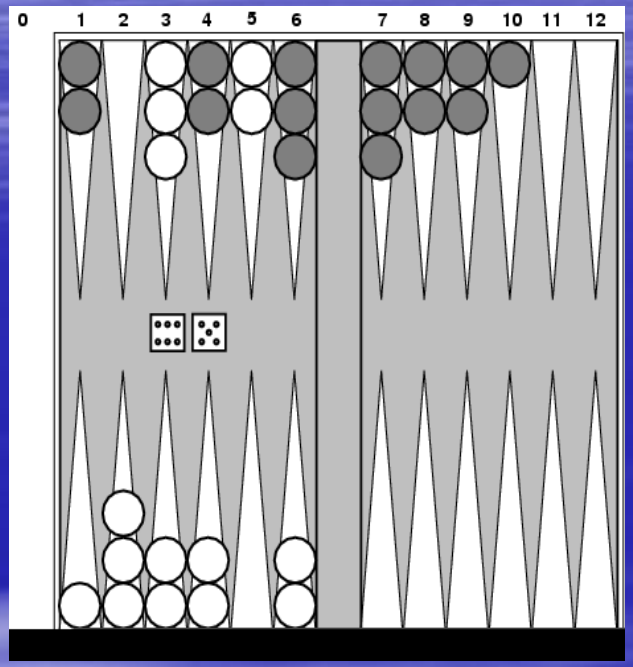
Games that include chance

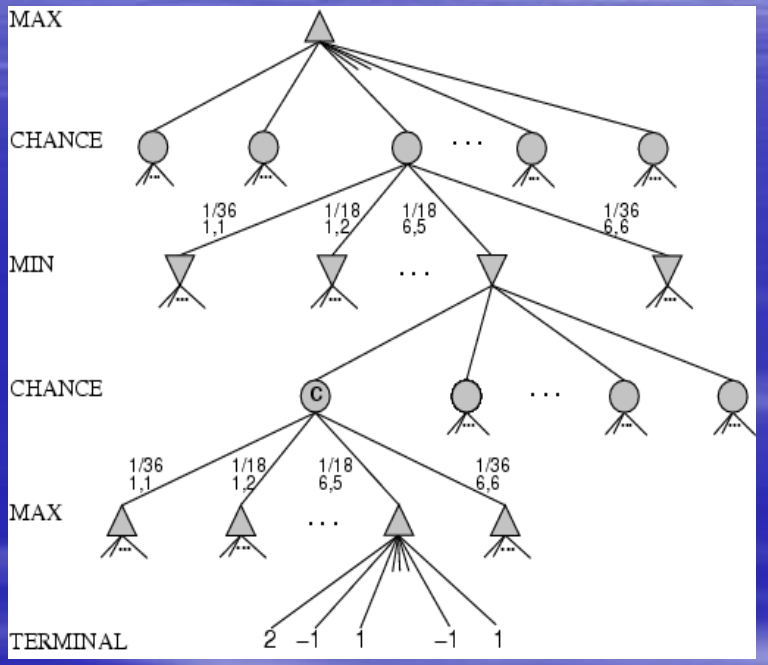




- Possible moves (5-10,5-11), (5-11,19-24),(5-10,10-16) and (5-11,11-16)
- [1,1], [6,6] chance 1/36, all other chance 1/18

Games that include chance





- [1,1], [6,6] chance 1/36, all other chance 1/18
- Cannot calculate definite minimax value, only *expected* value

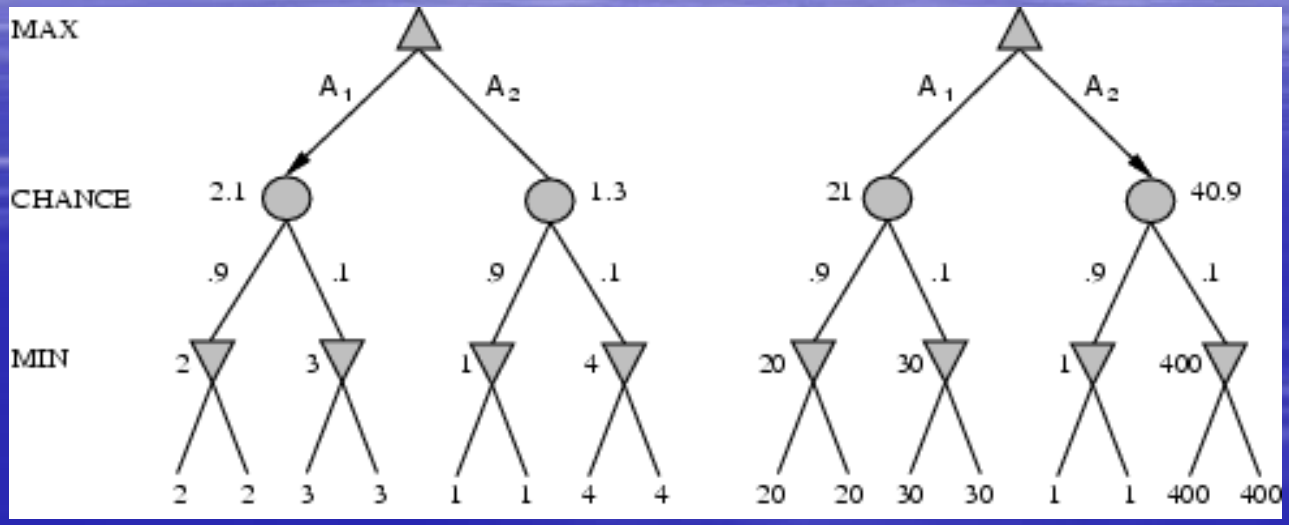
Expected minimax value

EXPECTED-MINIMAX-VALUE(n)=

UTILITY(n)	If n is a terminal
$\max_{s \in successors(n)}$ MINIMAX-VALUE(s)	If n is a max node
$\min_{s \in successors(n)}$ MINIMAX-VALUE(s)	If n is a max node
$\sum_{s \in successors(n)} P(s)$. EXPECTEDMINIMAX(s)	If n is a chance node

These equations can be backed-up recursively all the way to the root of the game tree.

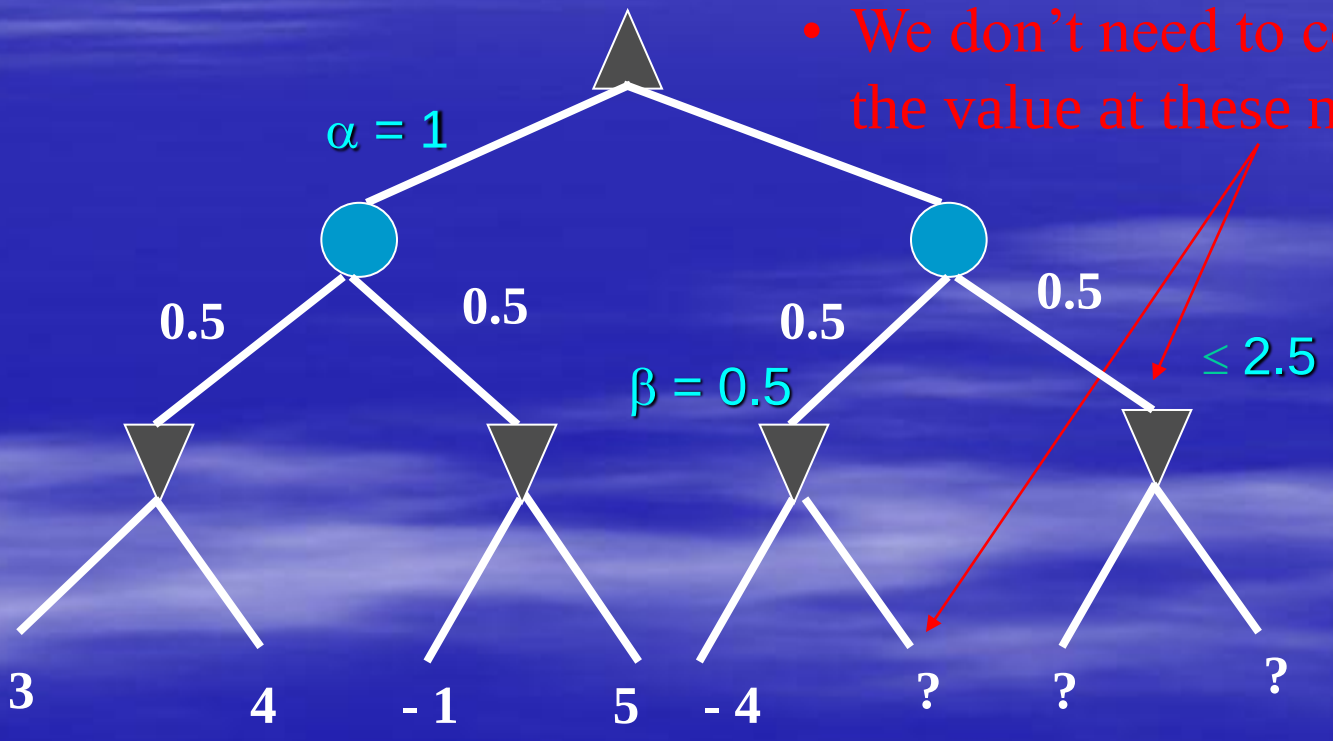
Position evaluation with chance nodes



- Left, A1 wins
- Right A2 wins
- Outcome of evaluation function may change when values are scaled differently.
- Behavior is preserved only by a *positive linear* transformation of EVAL.

Alpha-beta pruning in games with chance

- We can improve on the performance of the minimax algorithm for games with chance through alpha-beta pruning.



$\alpha = 3 * 0.5 + (-1) * 0.5 = 1.0$

Suppose $-5 \leq e(n) \leq 5$