Alpha Beta Procedure

- Idea:
  - Do Depth first search to generate partial game tree.
  - Give static evaluation function to leaves.
  - compute bound on internal nodes.
- Alpha, Beta bounds:
  - Alpha value for Max node means that Max real value is at least alpha.
  - Beta for Min node means that Min can guarantee a value below Beta.
- Computation:
  - Alpha of a Max node is the maximum value of its seen children.
  - Beta of a Min node is the lowest value seen of its child node .

When to Prune

- Pruning
  - Below a Min node whose beta value is lower than or equal to the alpha value of its ancestors.
  - Below a Max node having an alpha value greater than or equal to the beta value of any of its Min nodes ancestors.

The Alpha-Beta Procedure

The Alpha-Beta Procedure

- Now let us specify how to prune the Minimax tree in the case of a static evaluation function.

The Alpha-Beta Procedure

- Now let us specify how to prune the Minimax tree in the case of a static evaluation function.
- Use two variables alpha (associated with MAX nodes) and beta (associated with MIN nodes).

The Alpha-Beta Procedure

- Now let us specify how to prune the Minimax tree in the case of a static evaluation function.
- Use two variables alpha (associated with MAX nodes) and beta (associated with MIN nodes).
- These variables contain the best (highest or lowest, resp.) e(p) value at a node p that has been found so far.

The Alpha-Beta Procedure

- Now let us specify how to prune the Minimax tree in the case of a static evaluation function.
- Use two variables alpha (associated with MAX nodes) and beta (associated with MIN nodes).
- These variables contain the best (highest or lowest, resp.) e(p) value at a node p that has been found so far.
- Notice that alpha can never decrease, and beta can never increase.

The Alpha-Beta Procedure

The Alpha-Beta Procedure

- There are two rules for terminating search:

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.

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 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.

Alpha-Beta procedure

```
def value(state, α, β):
  if (state is a max node):
    max-value((state, α, β)
  else:
    min-value((state, α, β)
```

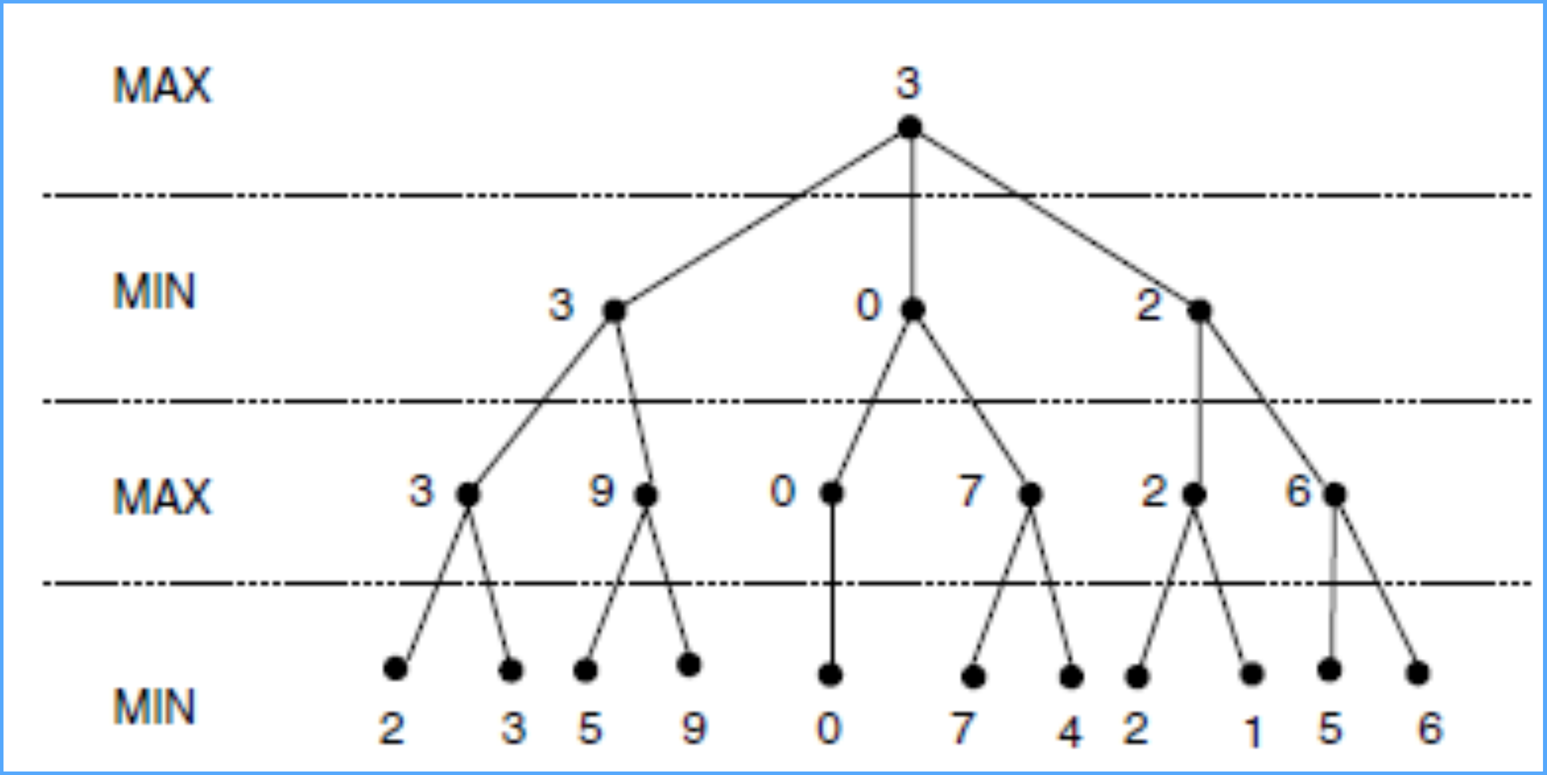
α: MAX's best option on path to root

β: MIN's best option on path to root

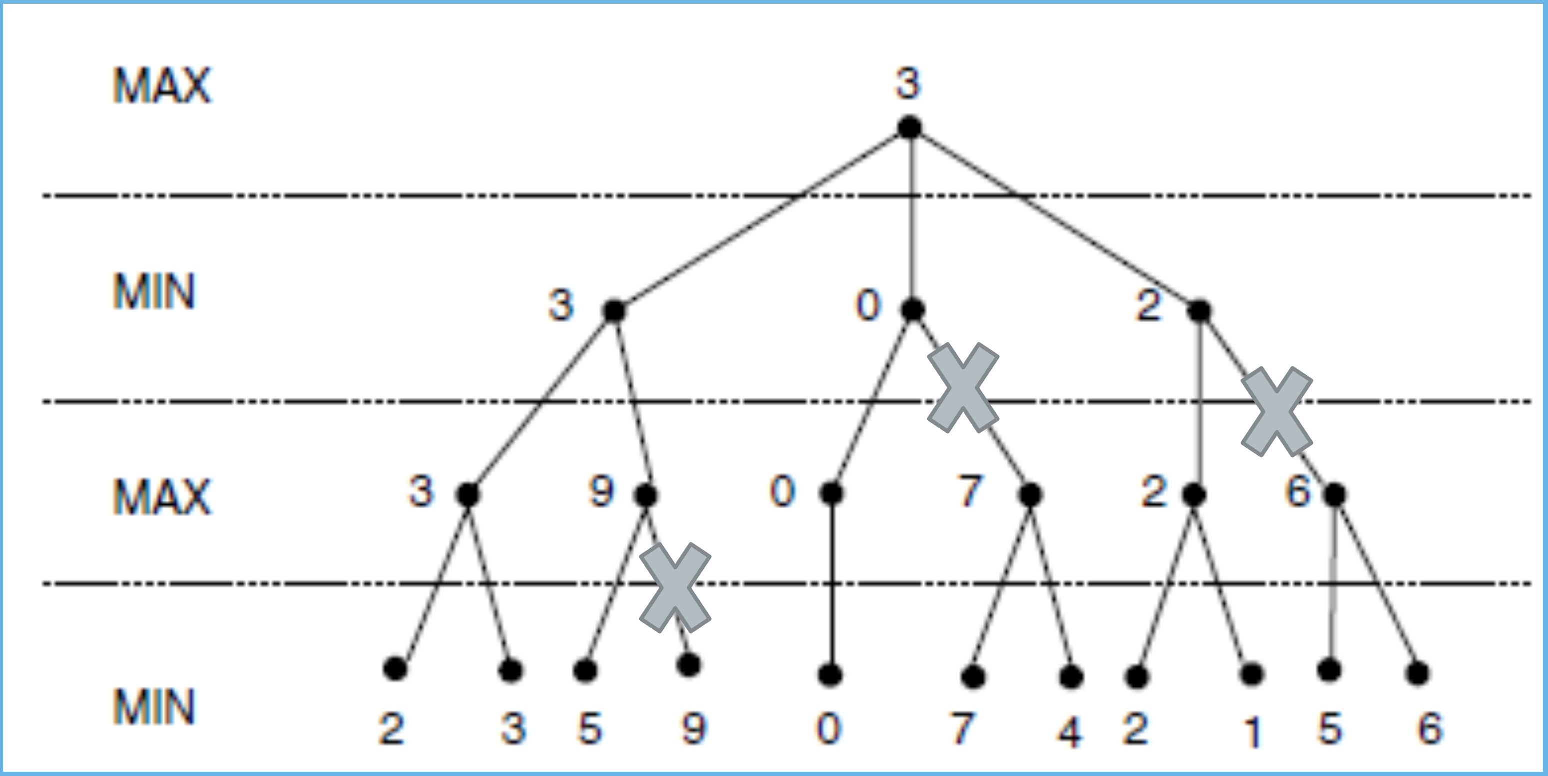
```
def max-value(state, α, β):
    initialize v = -∞
    if leaf(state):
        return value(state)
    for each successor of state:
        v = max(v, value(successor, α, β))
        if v ≥ β return v
        α = max(α, v)
    return v
```

```
def min-value(state , α, β):
    initialize v = +∞
    if leaf(state):
        return value(state)
    for each successor of state:
        v = min(v, value(successor, α, β))
        if v ≤ α return v
        β = min(β, v)
    return v
```

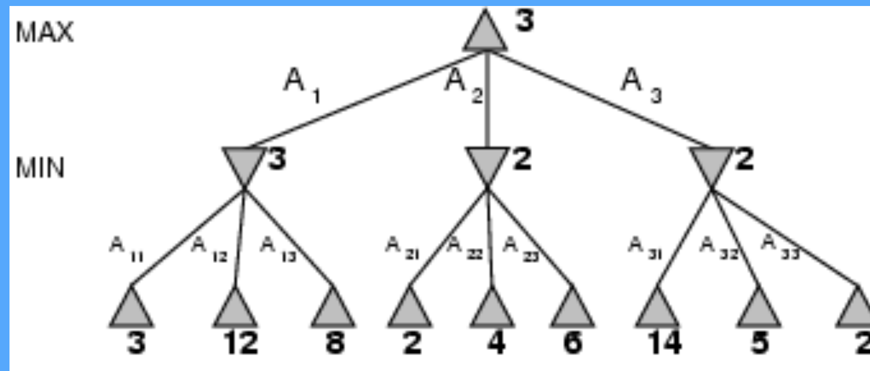
Pruning the tree Example 1

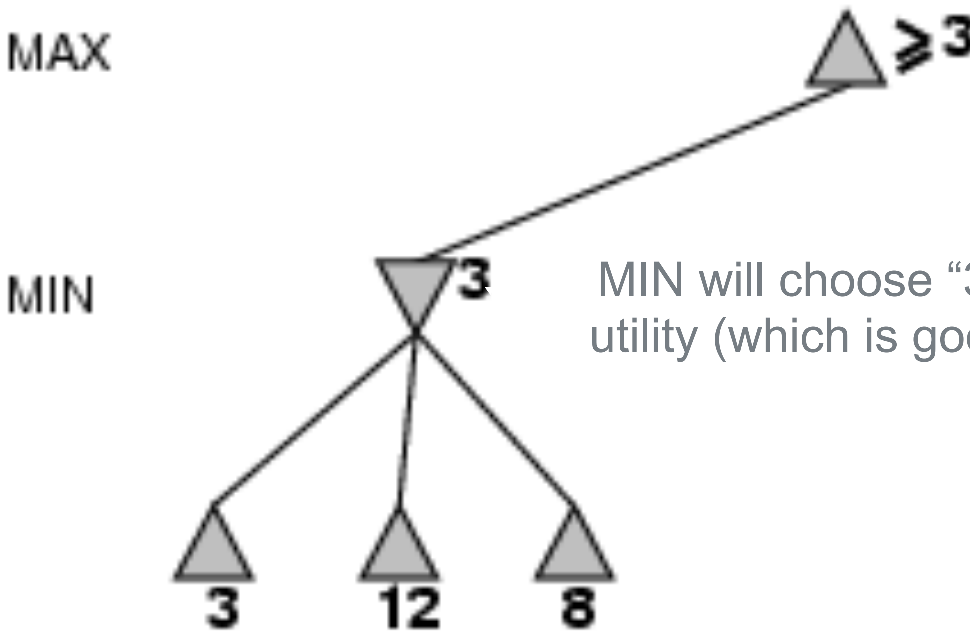


Pruning the tree Example



α-β pruning example 2

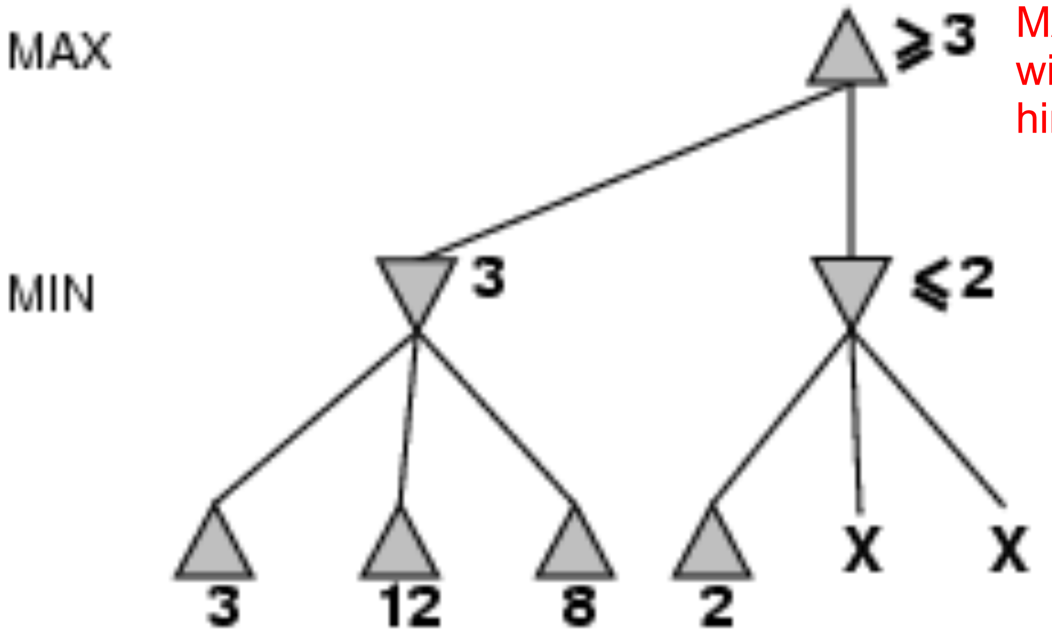




MAX knows that it can at least get "3" by playing this branch

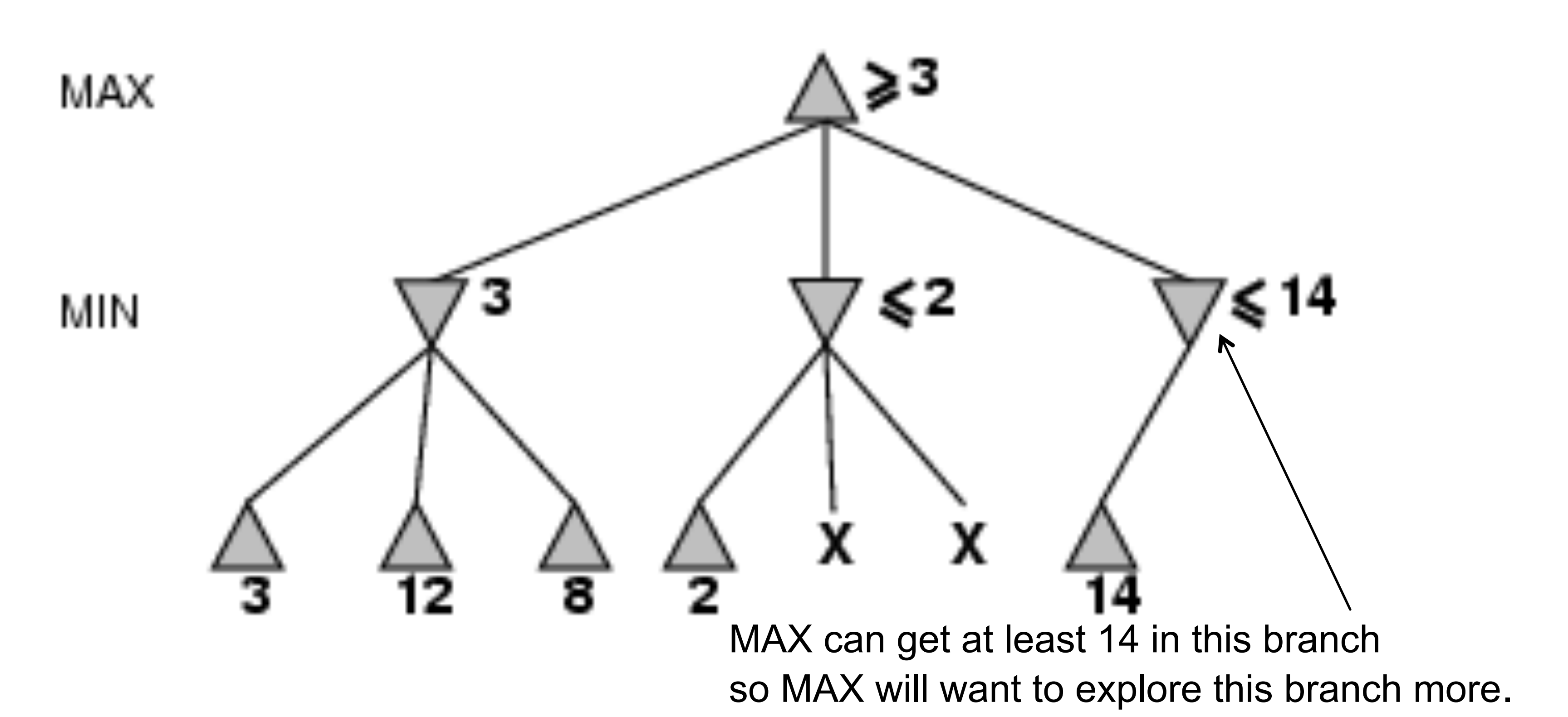
MIN will choose "3", because it minimizes the utility (which is good for MIN)

α-β pruning example 2

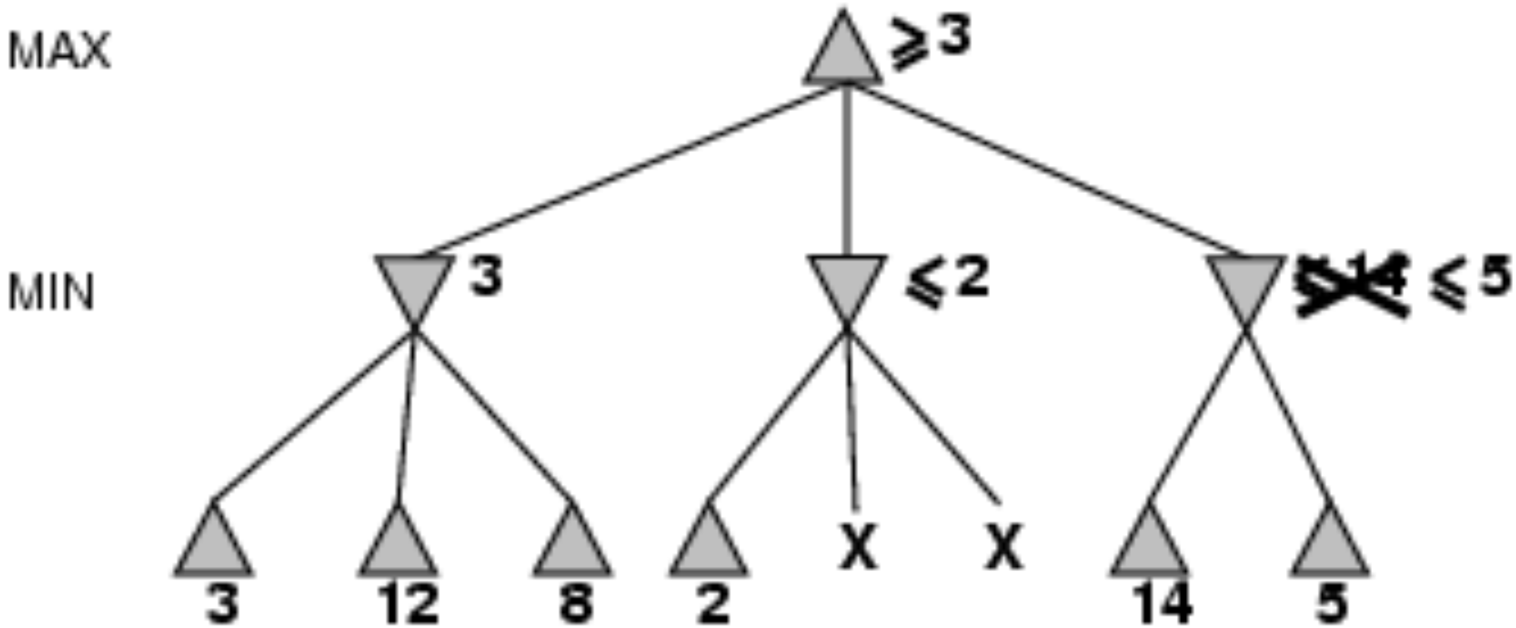


MAX knows that the new branch will never be better than 2 for him. It can *ignore* it.

α-β pruning example 2

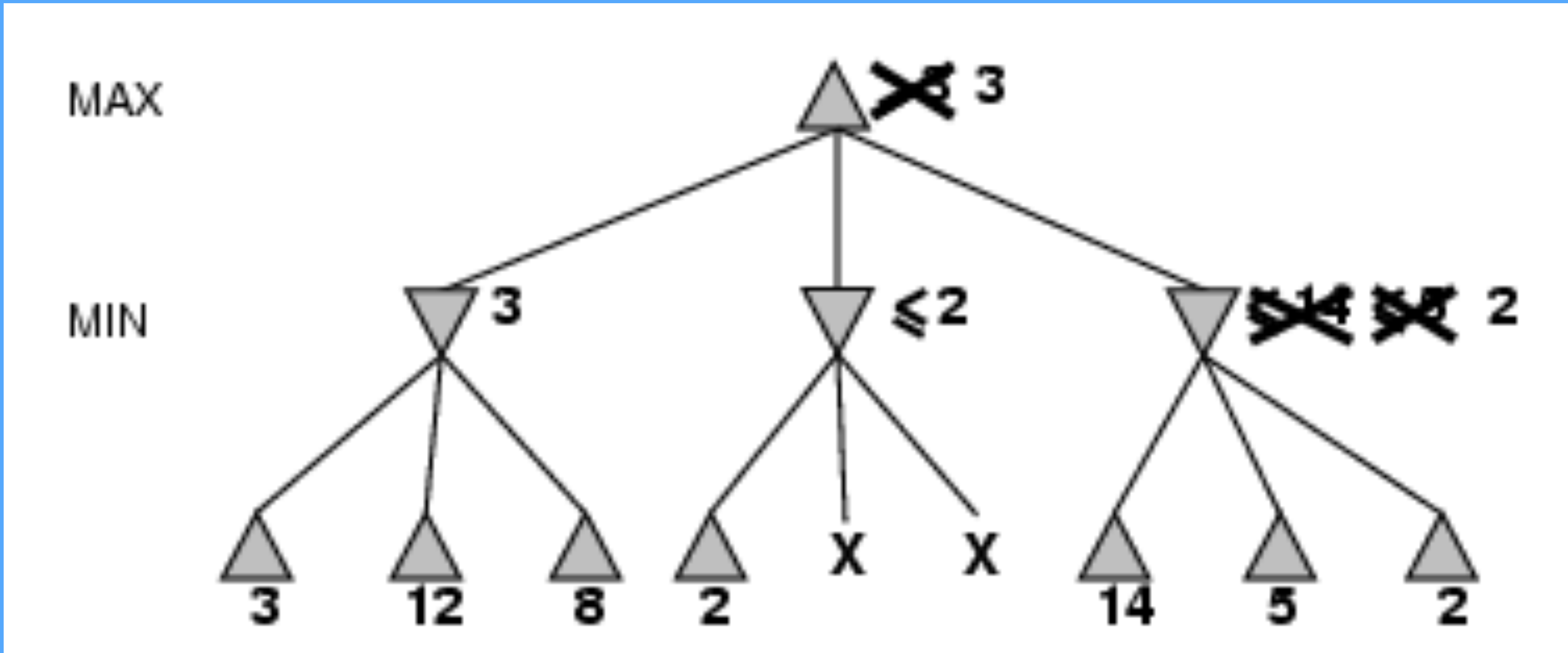


α-β pruning example 2



Should continue to explore

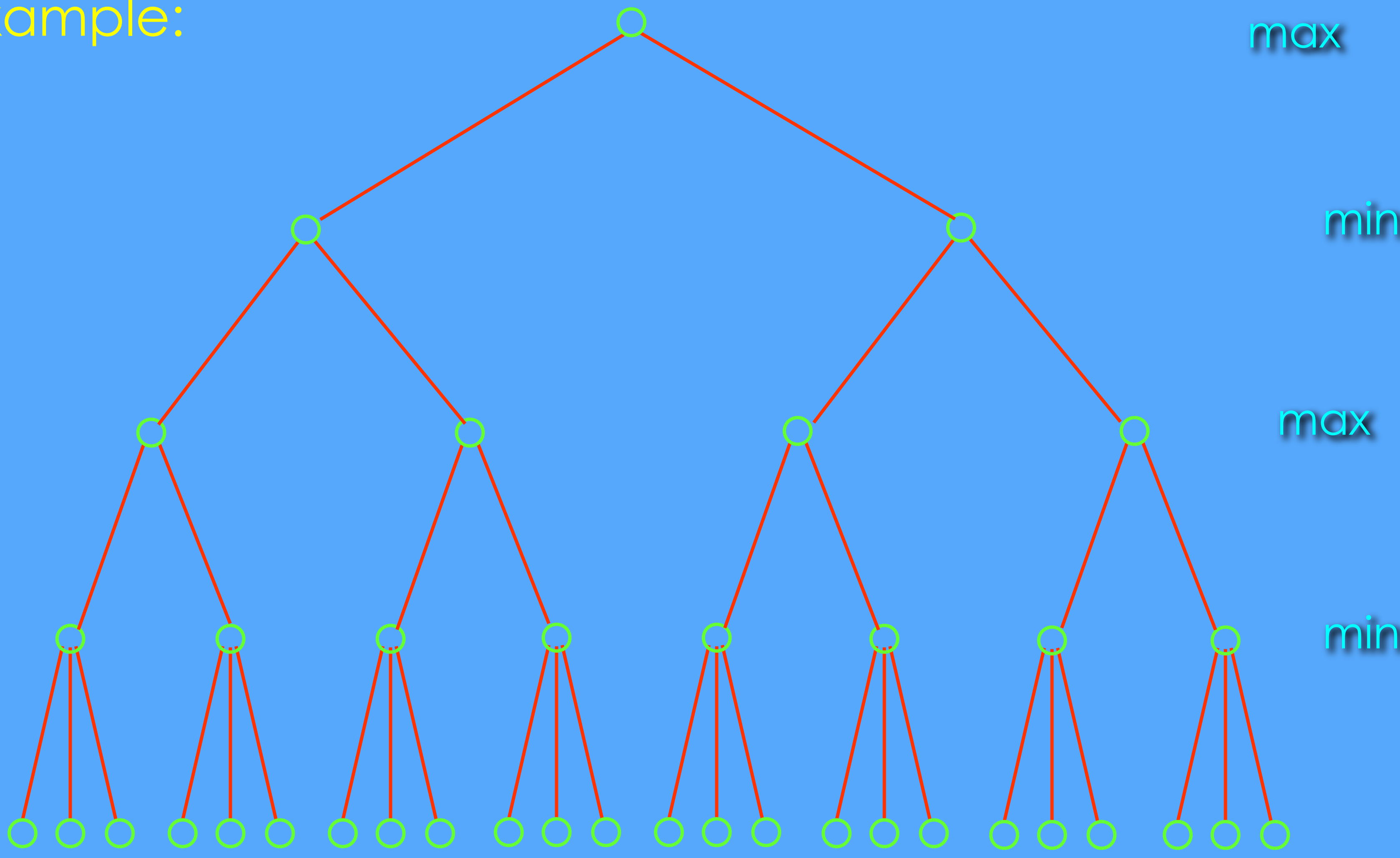
α-β pruning example 2



Search completed without exploring two branches

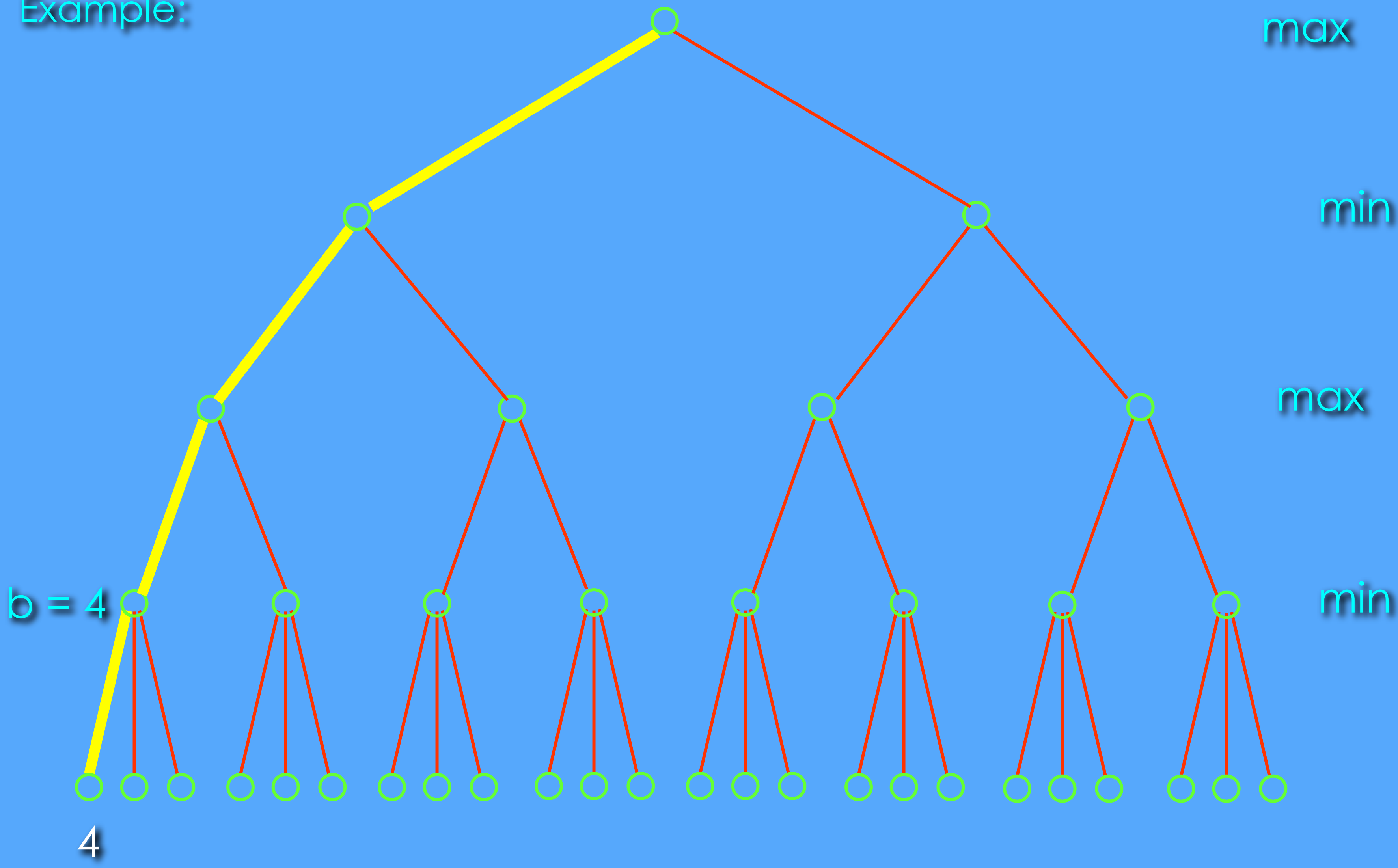
The Alpha-Beta Procedure Example 3

Example:

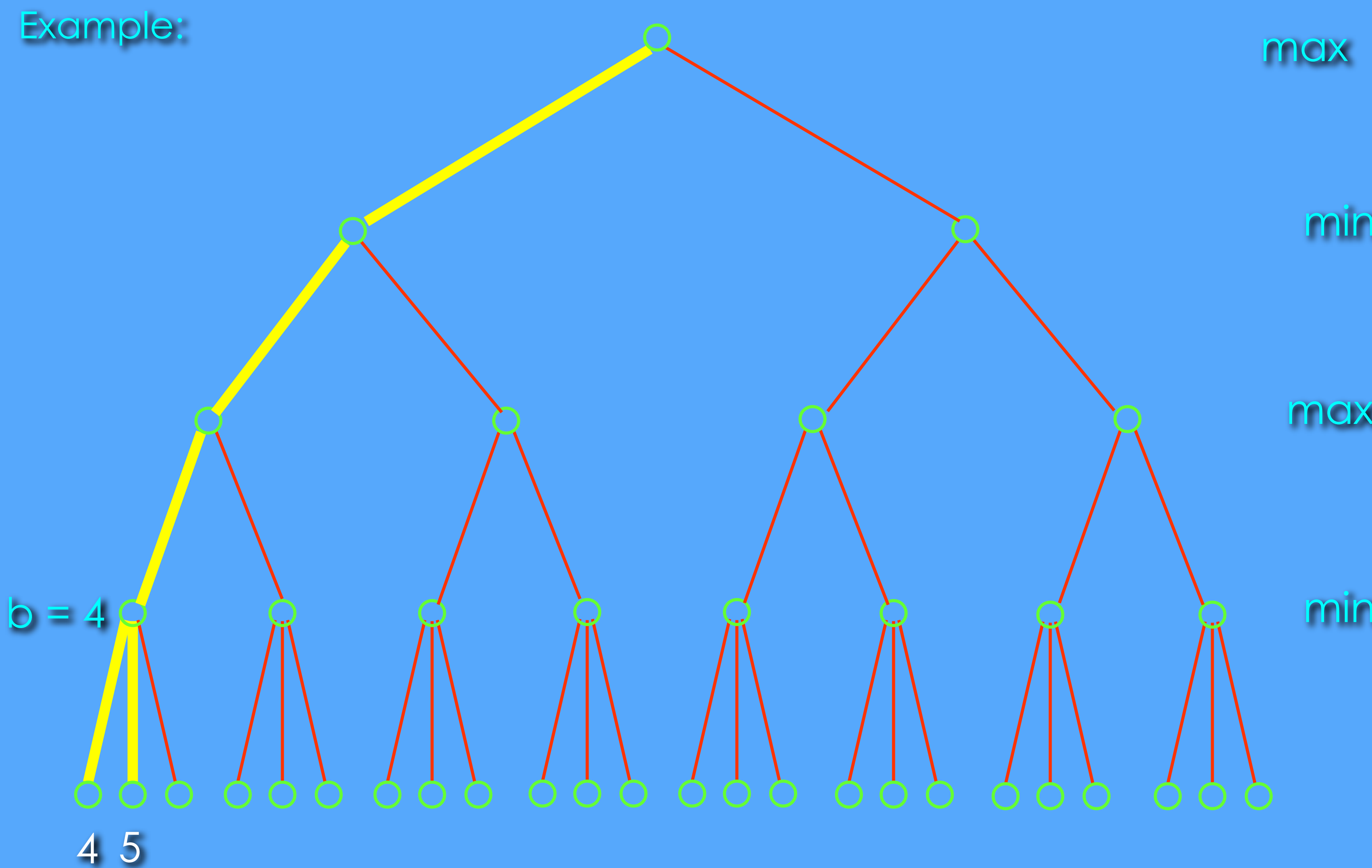


The Alpha-Beta Procedure

Example:

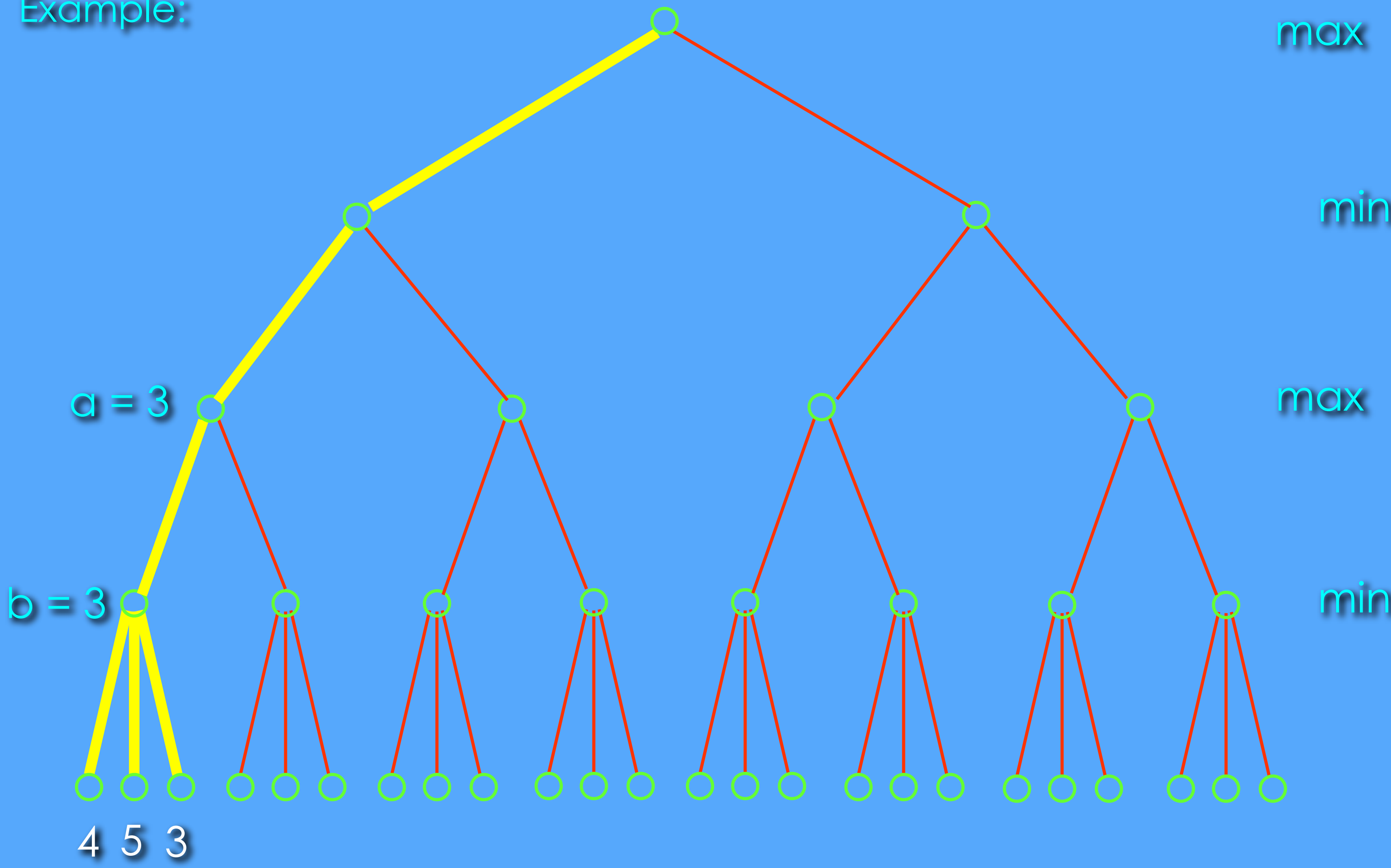


The Alpha-Beta Procedure



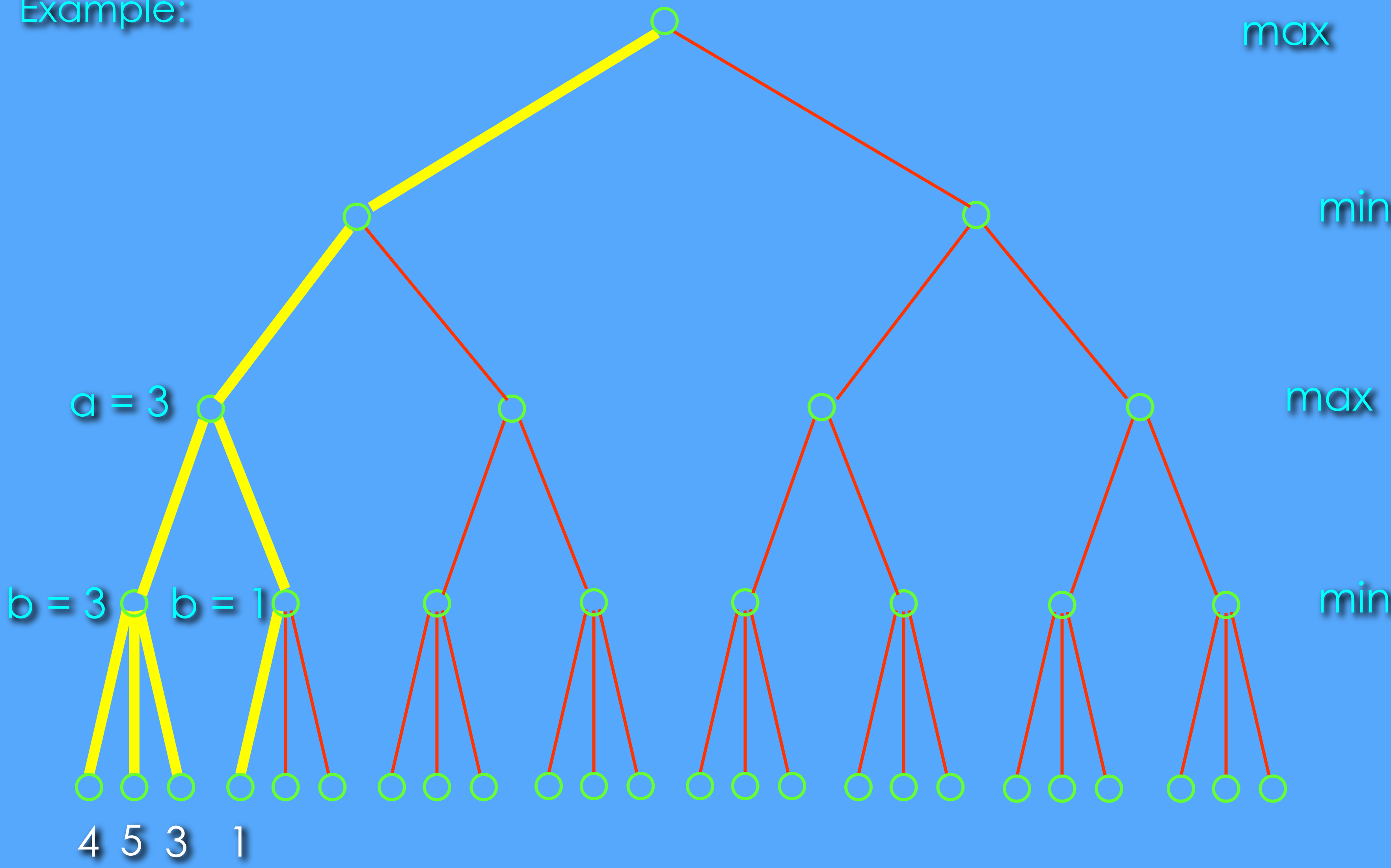
The Alpha-Beta Procedure

Example:



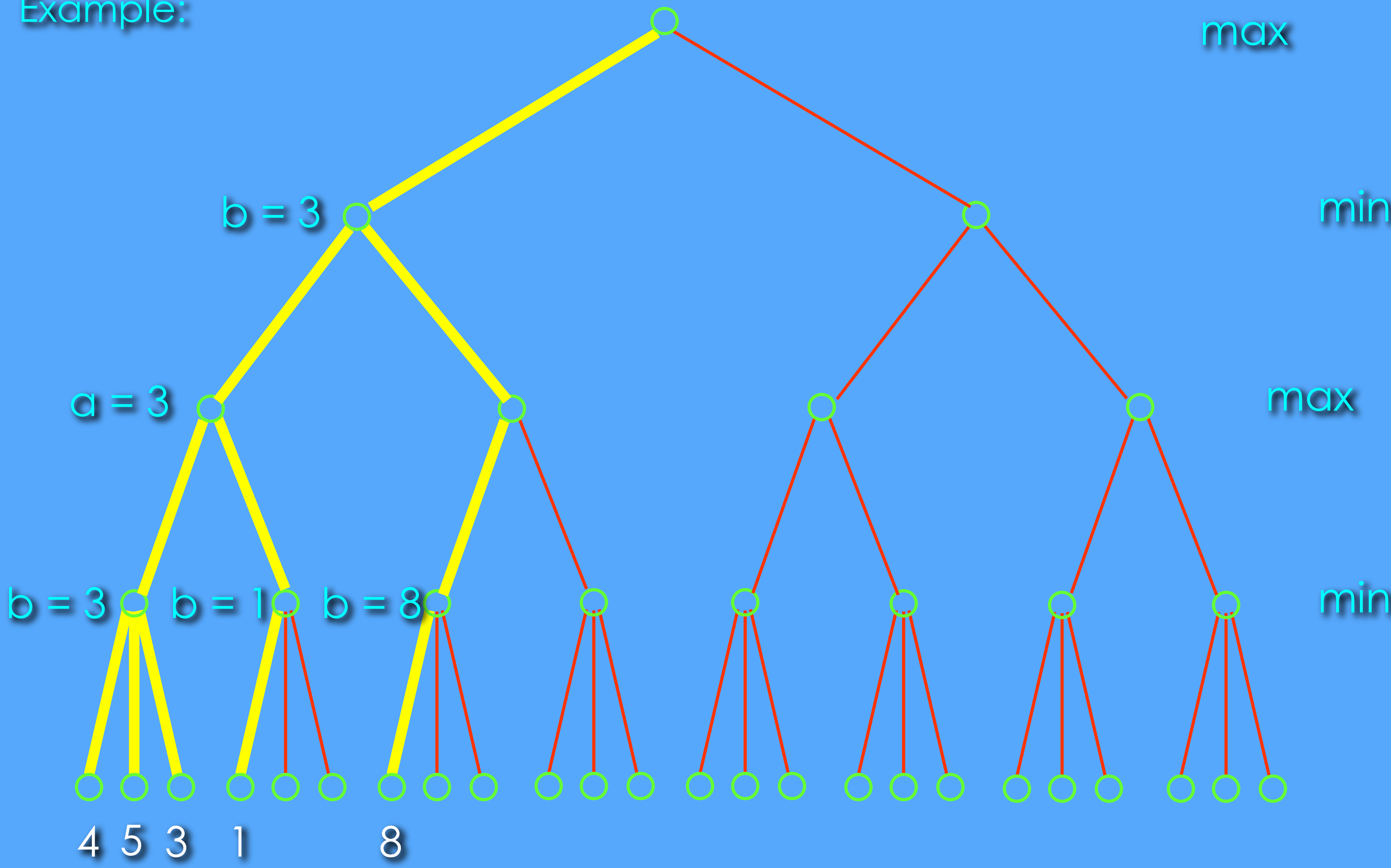
The Alpha-Beta Procedure

Example:



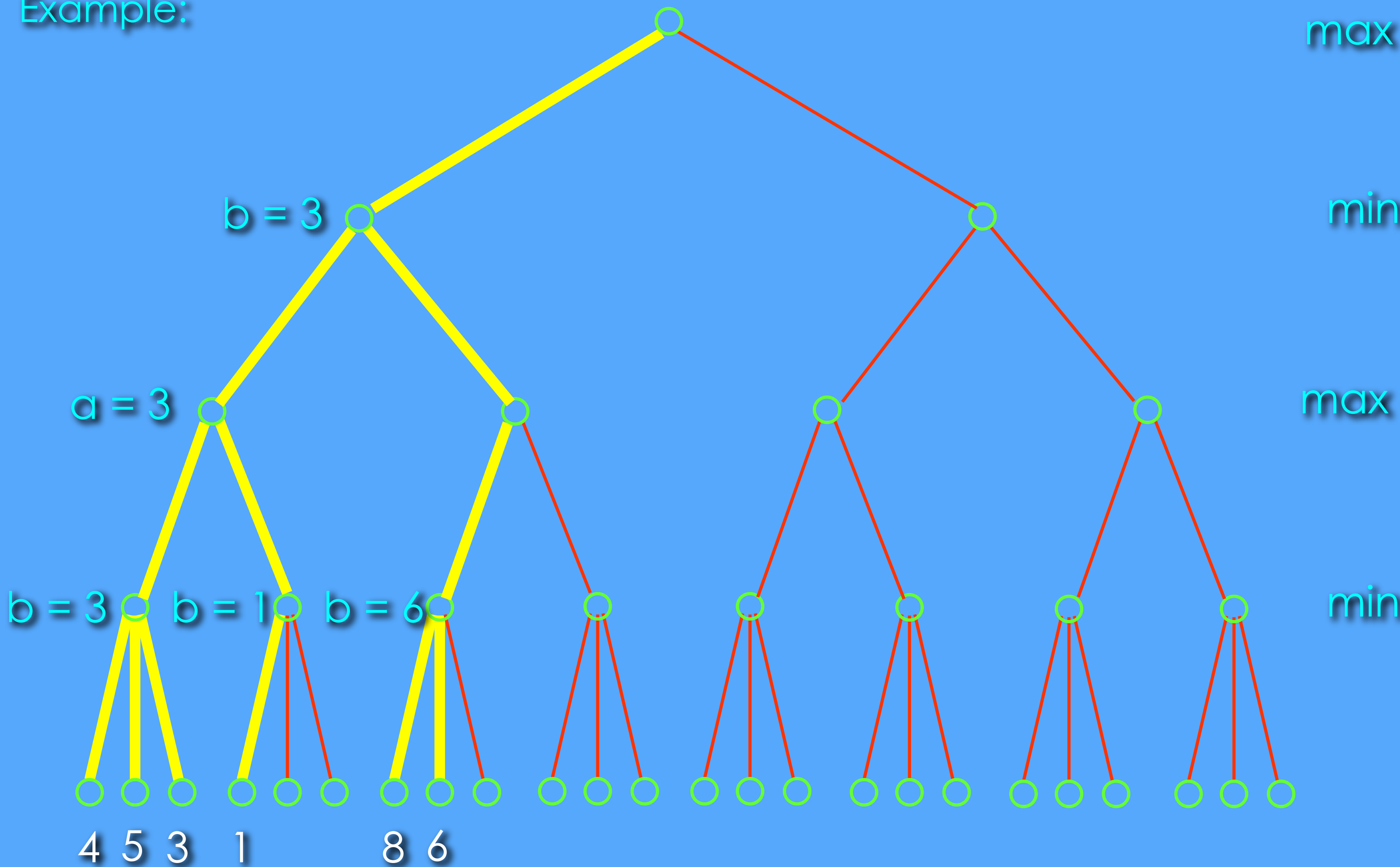
The Alpha-Beta Procedure

Example:



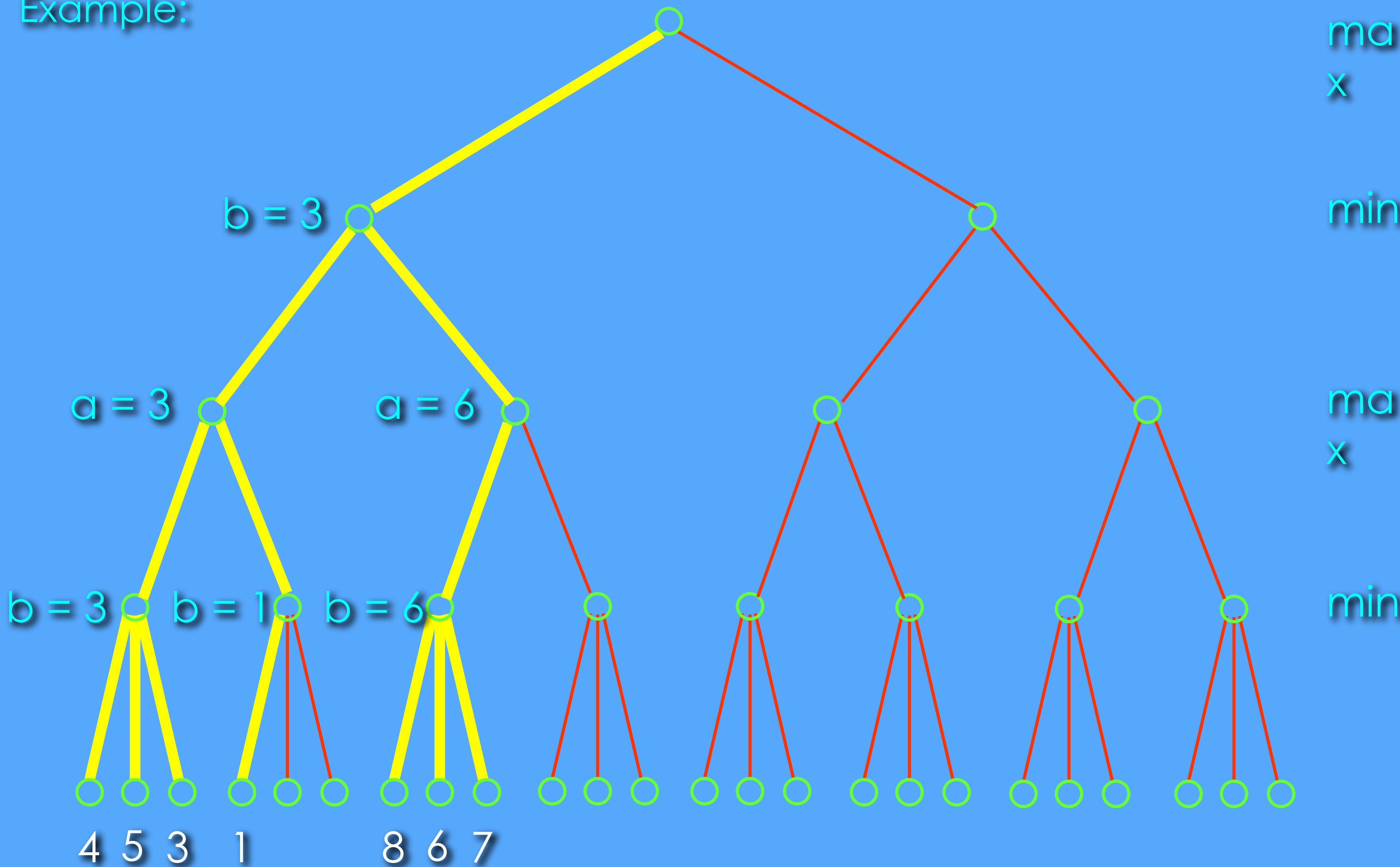
The Alpha-Beta Procedure

Example:



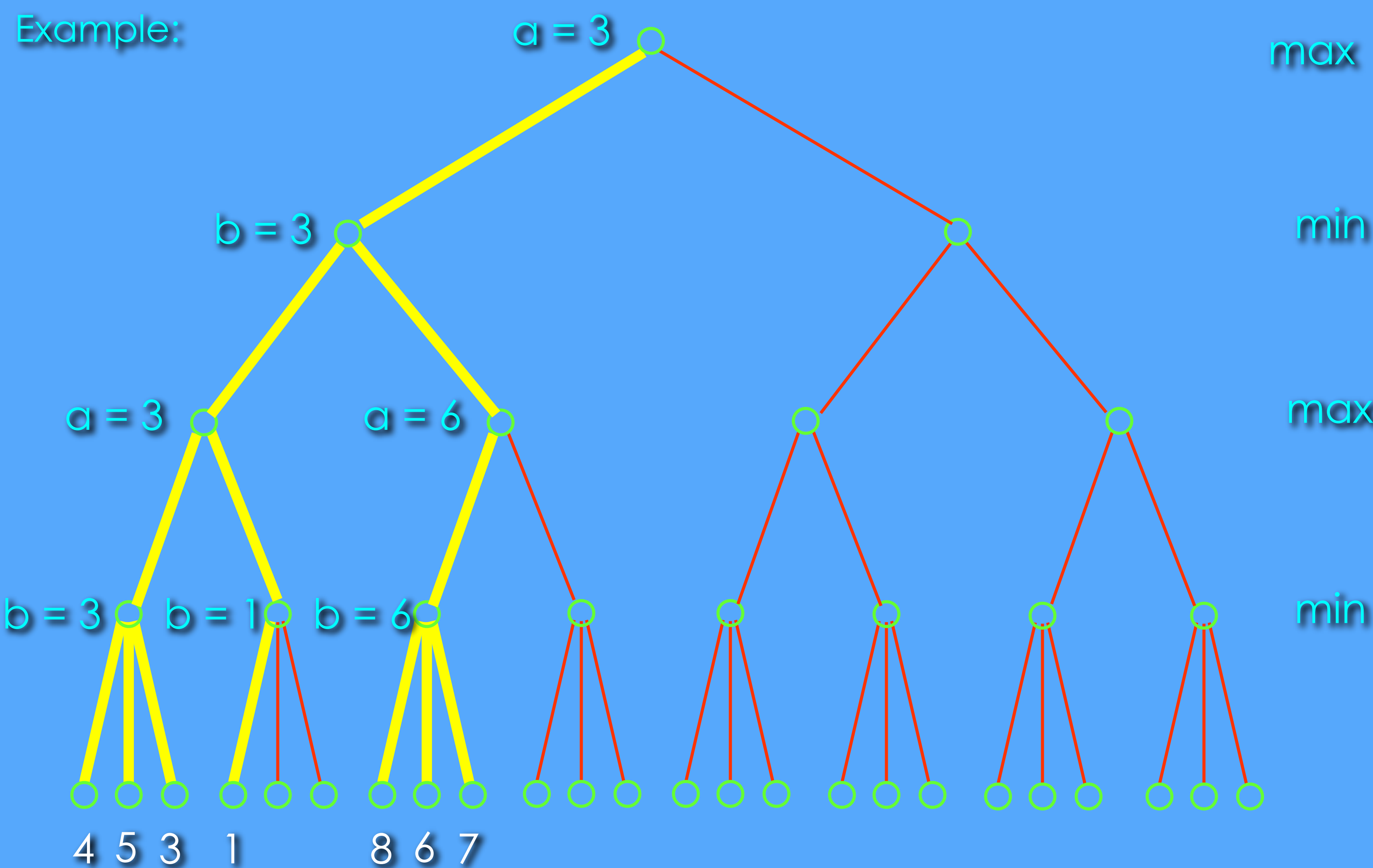
The Alpha-Beta Procedure

Example:



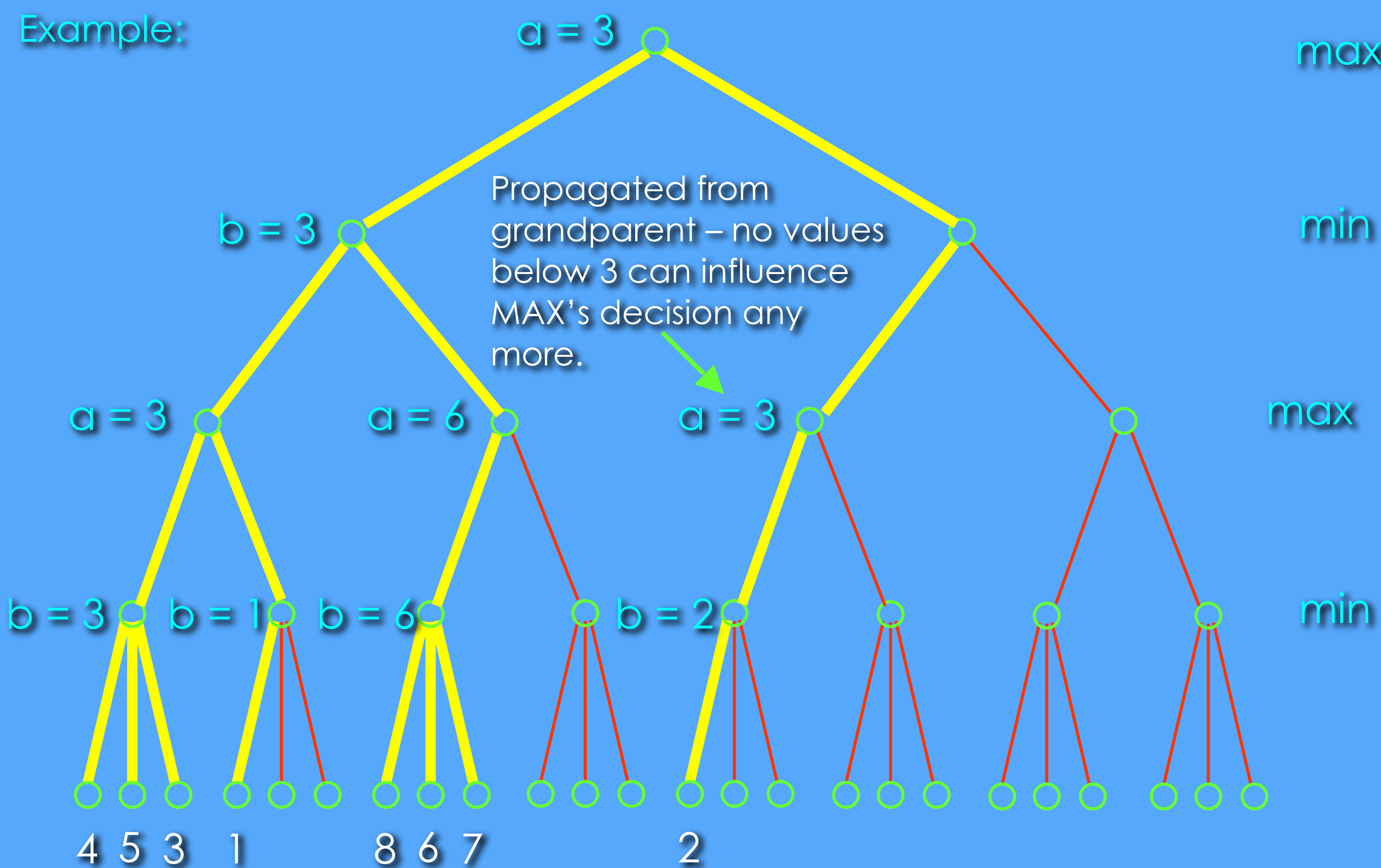
The Alpha-Beta Procedure

Example:



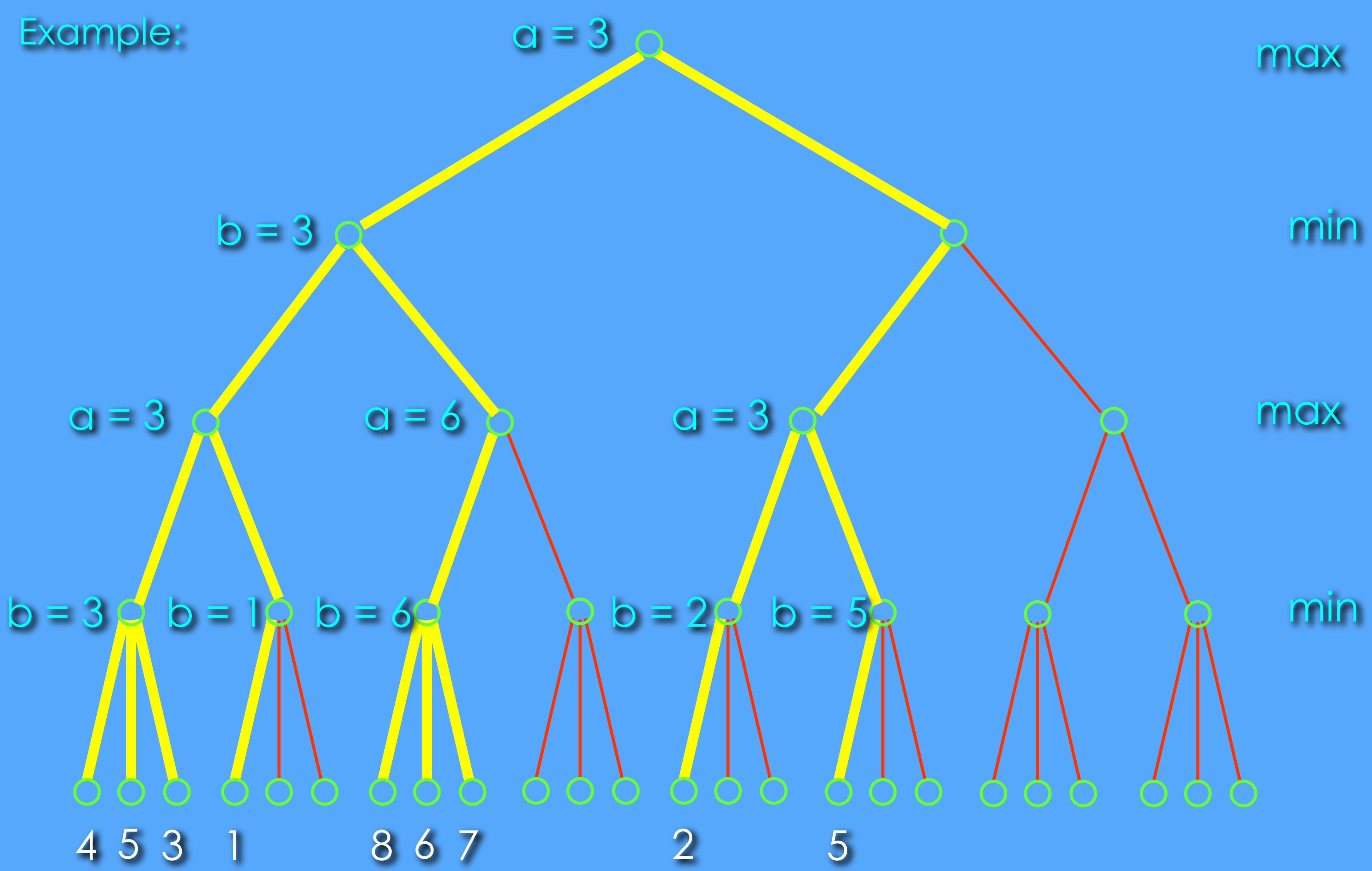
The Alpha-Beta Procedure

Example:



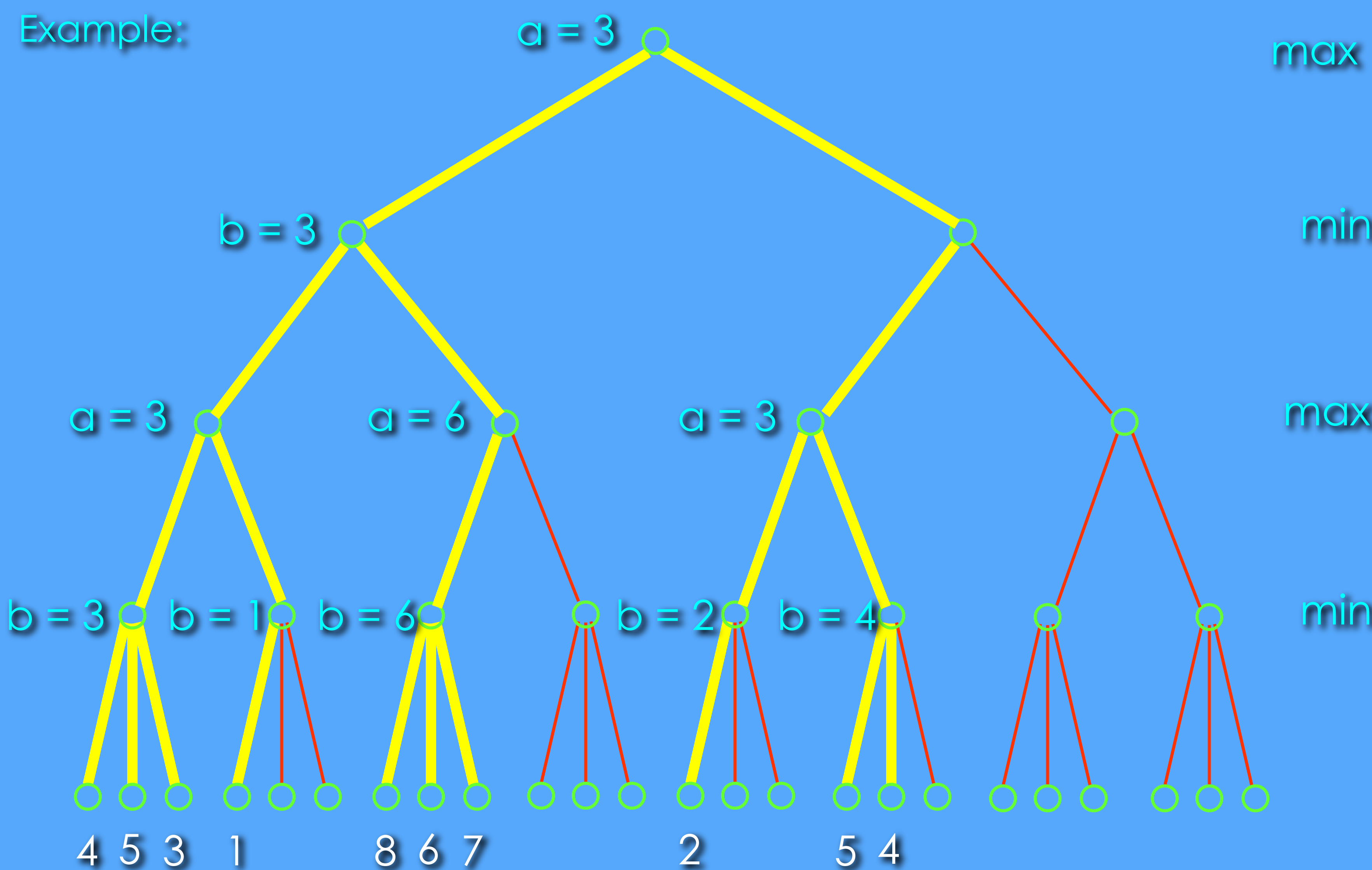
The Alpha-Beta Procedure

Example:



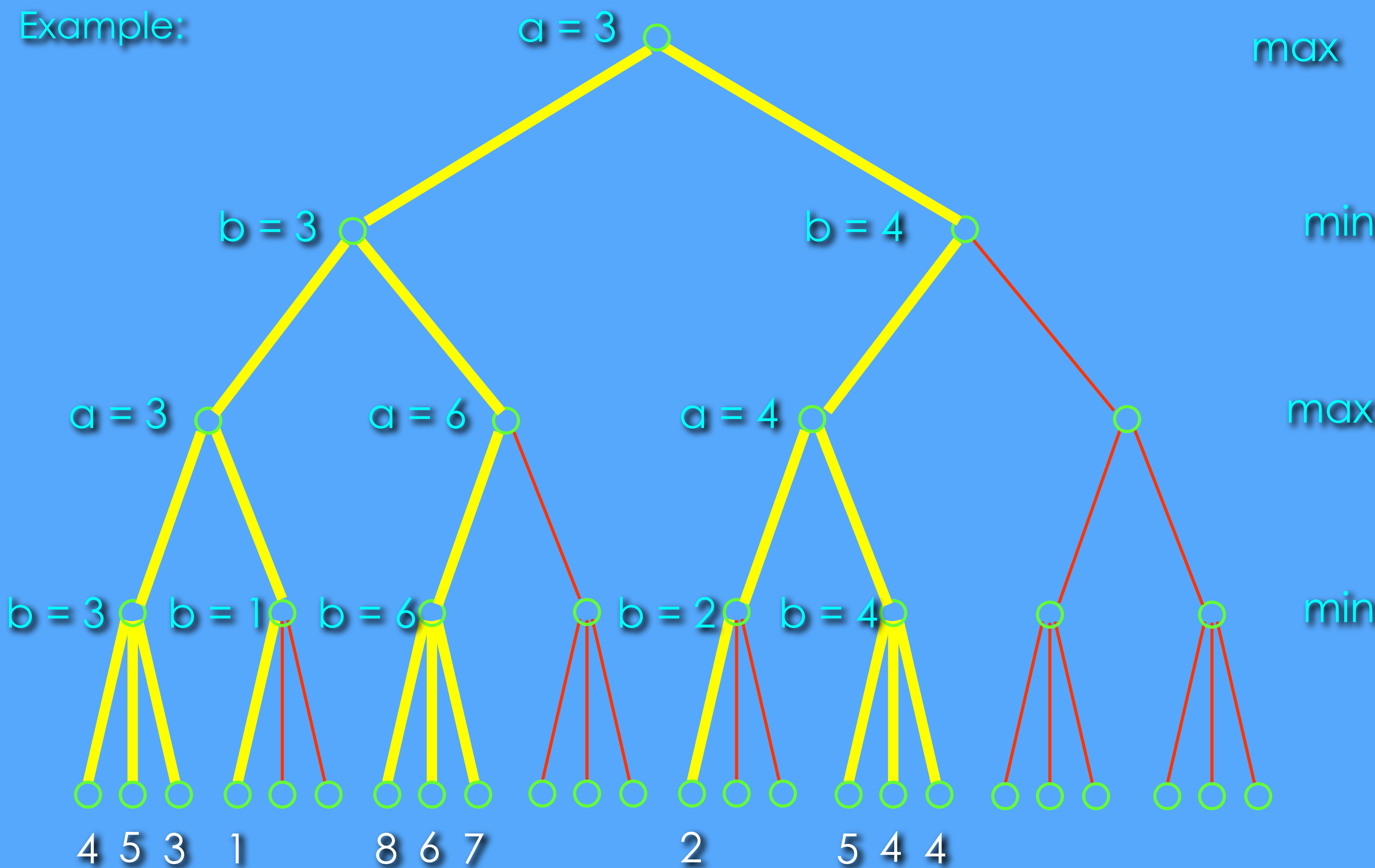
The Alpha-Beta Procedure

Example:



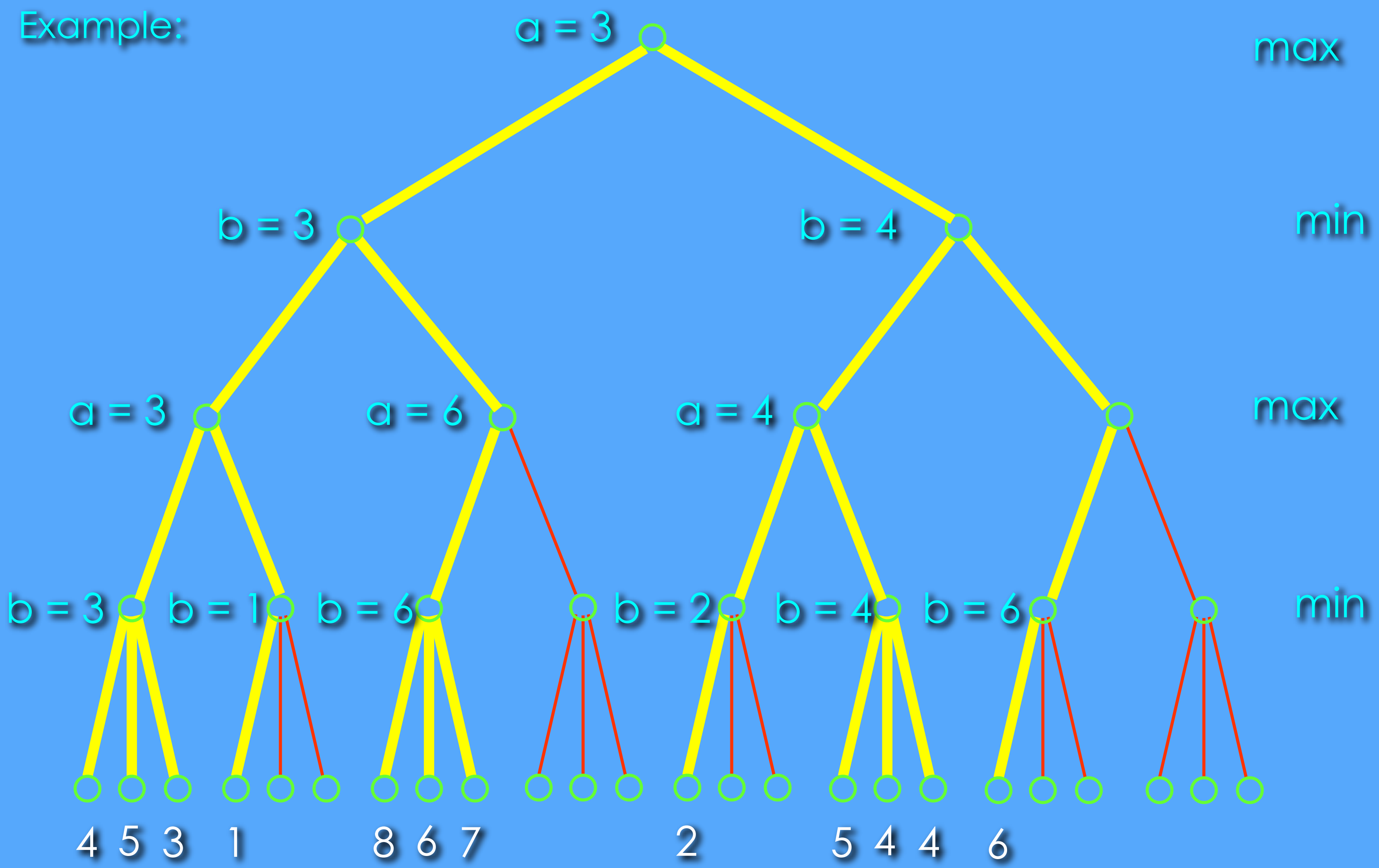
The Alpha-Beta Procedure

Example:



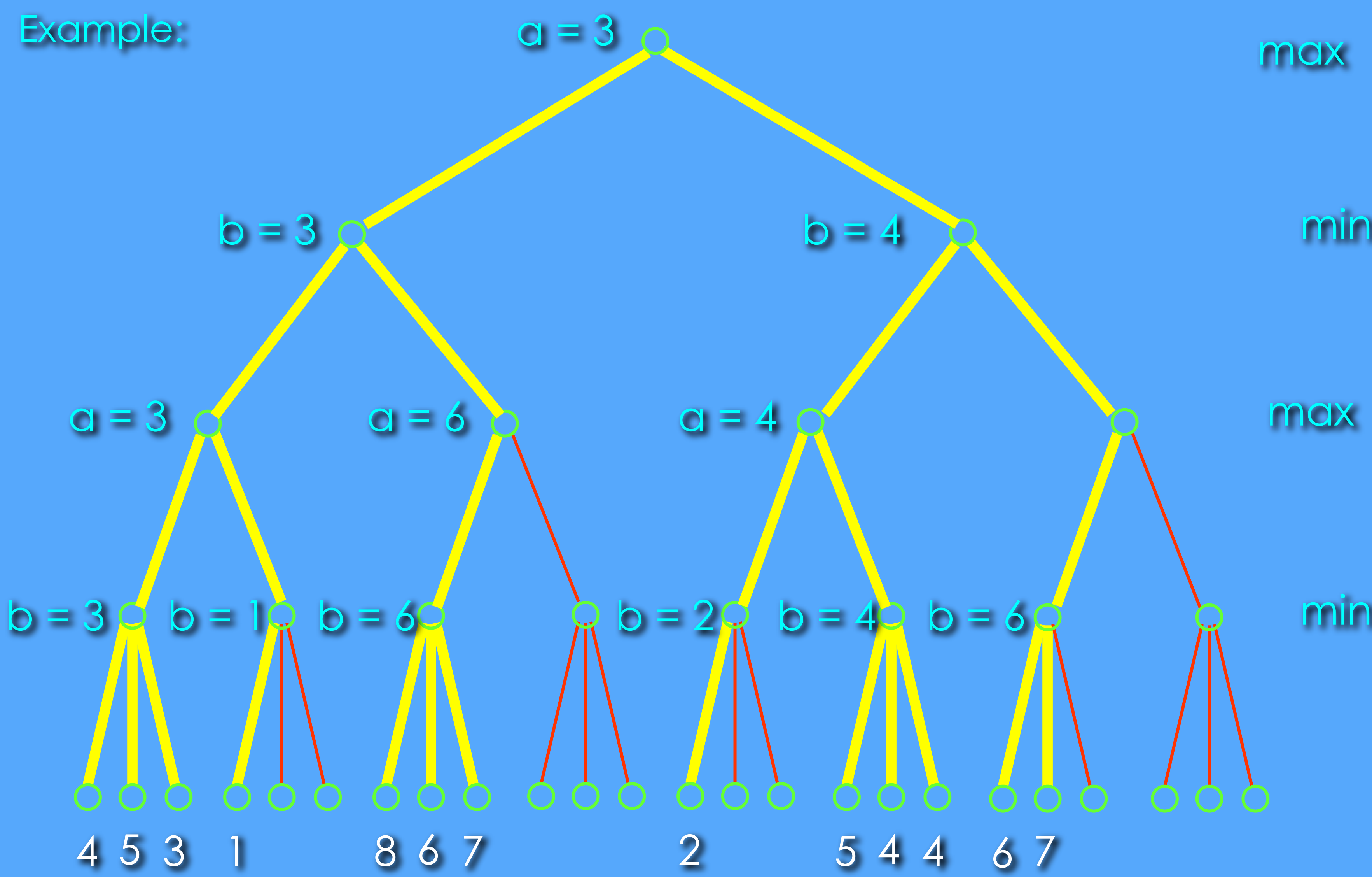
The Alpha-Beta Procedure

Example:



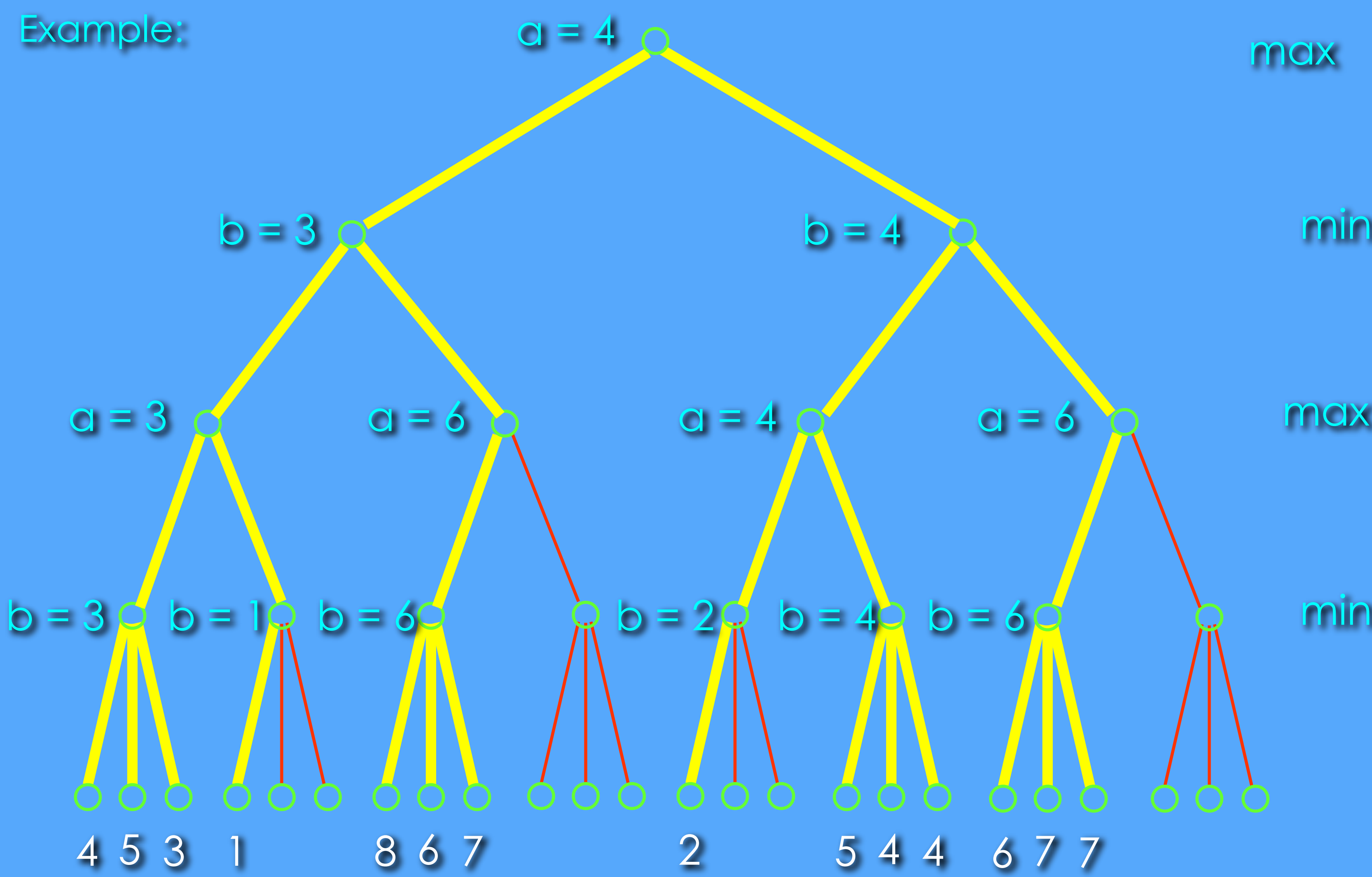
The Alpha-Beta Procedure

Example:



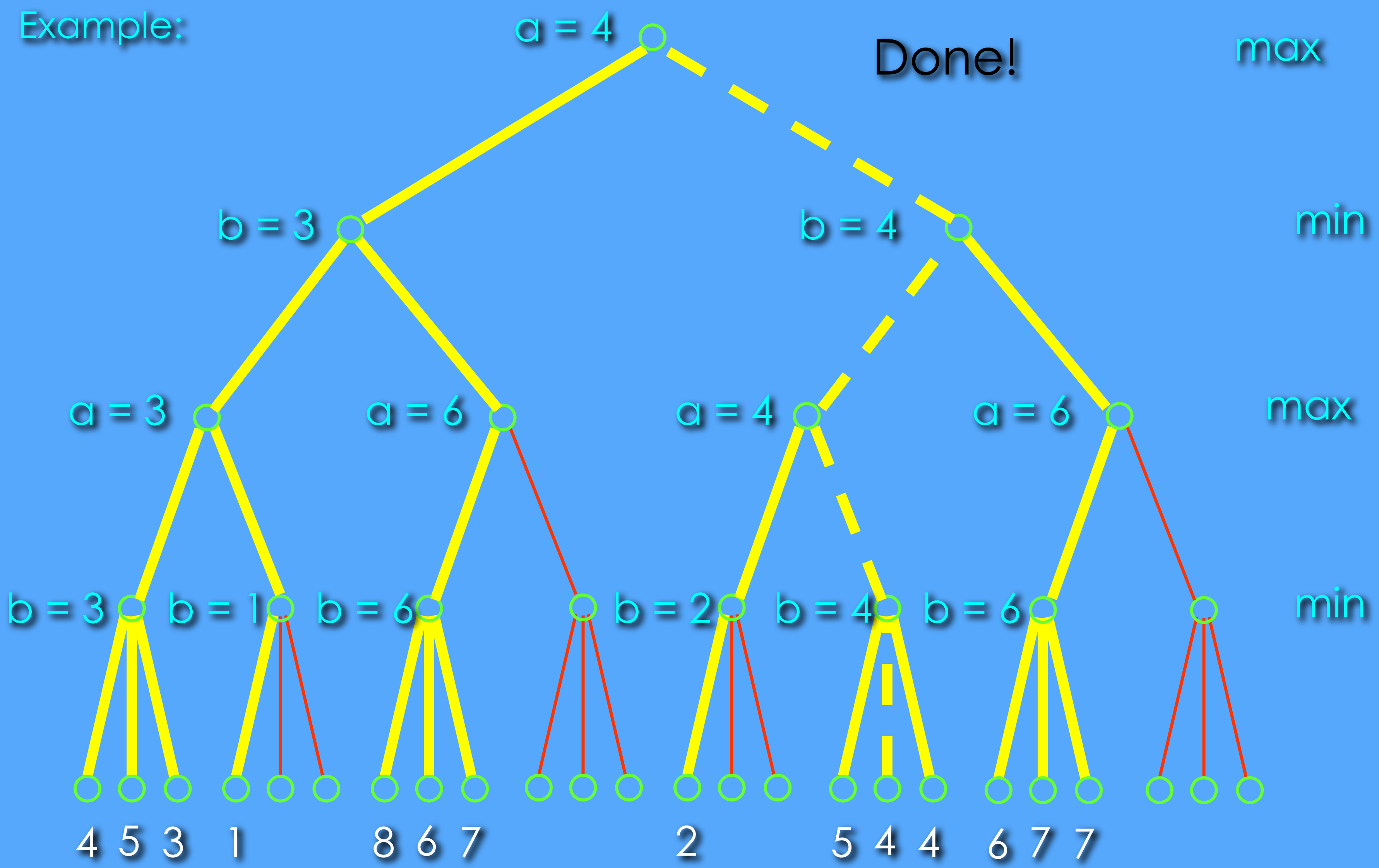
The Alpha-Beta Procedure

Example:

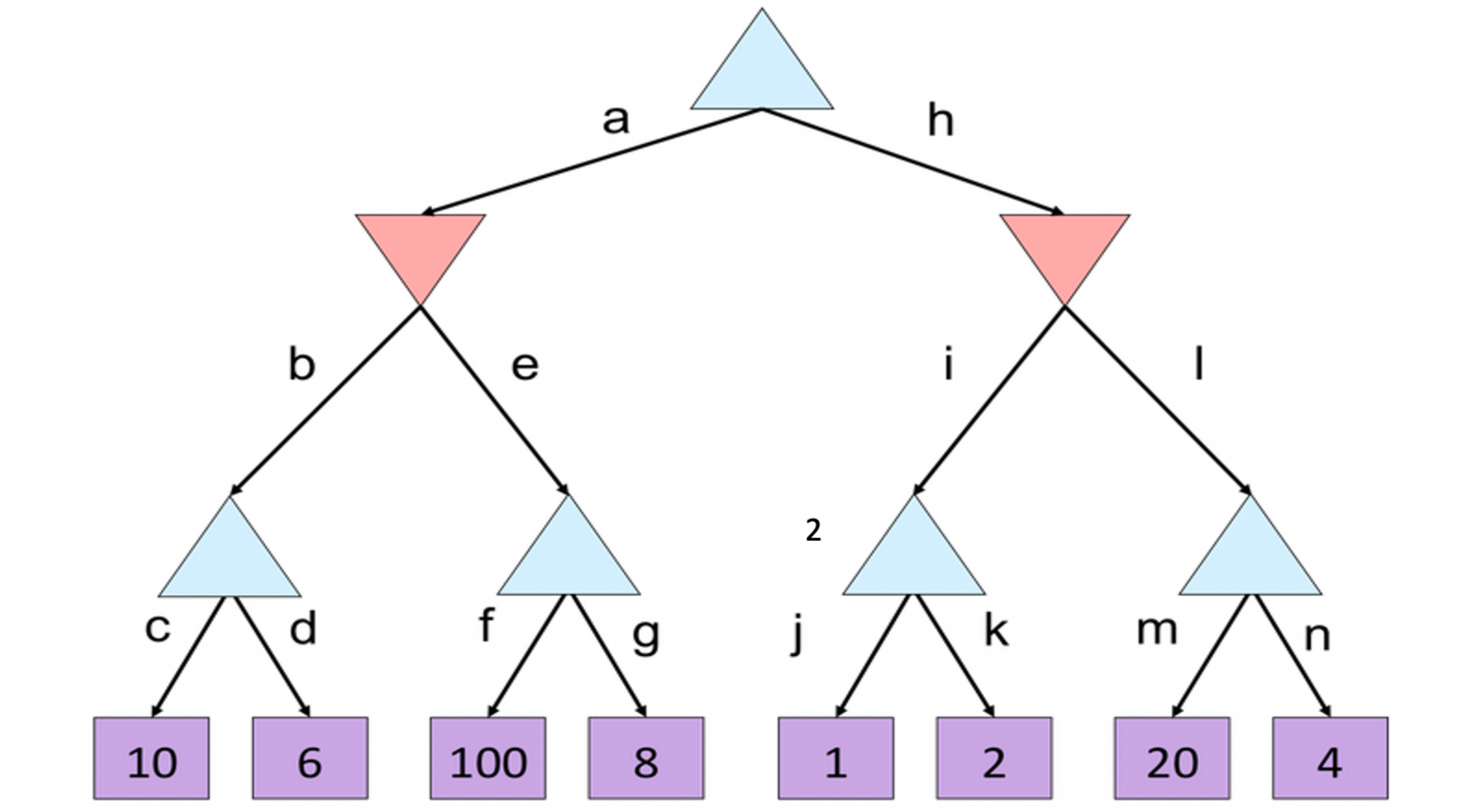


The Alpha-Beta Procedure

Example:



α-β pruning quiz



Properties of α-β pruning

- Pruning does not affect final result (it is exact).
- Good move ordering improves effectiveness of pruning (see last branch in example).
- The values at intermediate nodes may not be the same as the values computed by the minmax algorithm.

The Alpha-Beta Procedure

The Alpha-Beta Procedure

- What is the efficiency benefit of the alpha-beta method? (It is no better than min-max procedure of the same depth in terms of the quality of the move.)

The Alpha-Beta Procedure

- What is the efficiency benefit of the alpha-beta method? (It is no better than min-max procedure of the same depth in terms of the quality of the move.)

- Suppose that there is a game that always allows a player to choose among **b** different moves, and we want to look **d** moves ahead.

The Alpha-Beta Procedure

- What is the efficiency benefit of the alpha-beta method? (It is no better than min-max procedure of the same depth in terms of the quality of the move.)

- Suppose that there is a game that always allows a player to choose among **b** different moves, and we want to look **d** moves ahead.

- Then our search tree has **b^d leaves**.

The Alpha-Beta Procedure

- What is the efficiency benefit of the alpha-beta method? (It is no better than min-max procedure of the same depth in terms of the quality of the move.)

- Suppose that there is a game that always allows a player to choose among **b** different moves, and we want to look **d** moves ahead.

- Then our search tree has **b^d leaves**.

- If we do not use alpha-beta pruning, we would have to apply the static evaluation function $N_d = b^d$ times.

The Alpha-Beta Procedure

The Alpha-Beta Procedure

- if we assume that new children of a node are explored in the "most beneficial" order - those nodes p are explored first that will yield maximum values e(p) at depth d for MAX and minimum values for MIN - the number of nodes to be evaluated is:

Timing Issues

Timing Issues

- It is very difficult to predict for a given game situation how many operations a depth d look-ahead will require.

Timing Issues

- It is very difficult to predict for a given game situation how many operations a depth d look-ahead will require.
- Since we want the computer to respond within a certain amount of time, it is a good idea to apply the idea of ***iterative deepening***.

Timing Issues

- It is very difficult to predict for a given game situation how many operations a depth d look-ahead will require.
- Since we want the computer to respond within a certain amount of time, it is a good idea to apply the idea of ***iterative deepening***.
- First, the computer finds the best move according to a ***one-move look-ahead*** search.

Timing Issues

- It is very difficult to predict for a given game situation how many operations a depth d look-ahead will require.
- Since we want the computer to respond within a certain amount of time, it is a good idea to apply the idea of ***iterative deepening***.
- First, the computer finds the best move according to a ***one-move look-ahead*** search.
- Then, the computer determines the best move for a ***two-move look-ahead***, and remembers it as the new best move.

Timing Issues

- It is very difficult to predict for a given game situation how many operations a depth d look-ahead will require.
- Since we want the computer to respond within a certain amount of time, it is a good idea to apply the idea of ***iterative deepening***.
- First, the computer finds the best move according to a ***one-move look-ahead*** search.
- Then, the computer determines the best move for a ***two-move look-ahead***, and remembers it as the new best move.
- This is ***continued*** until the time runs out. Then the currently remembered best move is executed.

How to Find Static Evaluation Functions

How to Find Static Evaluation Functions

- Often, a static evaluation function e(p) first computes an appropriate feature vector f(p) that contains information about features of the current game configuration that are important for its evaluation.

How to Find Static Evaluation Functions

- Often, a static evaluation function e(p) first computes an appropriate feature vector f(p) that contains information about features of the current game configuration that are important for its evaluation.
- There is also a weight vector w(p) that indicates the weight (= importance) of each feature for the assessment of the current situation.

How to Find Static Evaluation Functions

- Often, a static evaluation function e(p) first computes an appropriate feature vector f(p) that contains information about features of the current game configuration that are important for its evaluation.
- There is also a weight vector w(p) that indicates the weight (= importance) of each feature for the assessment of the current situation.
- Then e(p) is simply computed as the **scalar product** of f(p) and w(p).

How to Find Static Evaluation Functions

- Often, a static evaluation function e(p) first computes an appropriate feature vector f(p) that contains information about features of the current game configuration that are important for its evaluation.
- There is also a weight vector w(p) that indicates the weight (= importance) of each feature for the assessment of the current situation.
- Then e(p) is simply computed as the **scalar product** of f(p) and w(p).
- Both the identification of the most relevant features and the correct estimation of their relative importance are **crucial** for the strength of a game-playing program.

Evaluation function for chess

- For games like chess, typically linear weighted sum of features
- $Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s)$

e.g. $w_1 = 9$ with

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

- Weights 9 for queen, 5 for rook, 3 for bishop and knight and 1 for pawn – suggested by Shannon is still widely used.

How to Find Static Evaluation Functions

- Once we have found suitable features, the weights can be adapted algorithmically.

How to Find Static Evaluation Functions

- Once we have found suitable features, the weights can be adapted algorithmically.
- This can be achieved, for example, with a neural network.

How to Find Static Evaluation Functions

- Once we have found suitable features, the weights can be adapted algorithmically.
- This can be achieved, for example, with a neural network.
- So the challenge is in extracting the most informative features from a game configuration.

How to Find Static Evaluation Functions

- Once we have found suitable features, the weights can be adapted algorithmically.
- This can be achieved, for example, with a neural network.
- So the challenge is in extracting the most informative features from a game configuration.

Heuristics and Game Tree Search

- How to determine the depth of evaluation? (clearly the time or other resource is one guide.)

- The Horizon Effect
  - sometimes there’s a major “effect” (such as a piece being captured) which is just “below” the depth to which the tree has been expanded
  - the computer cannot see that this major event could happen
  - it has a “limited horizon”
  - there are heuristics to try to follow certain branches more deeply to detect to such important events.
  - this helps to avoid catastrophic losses due to “short-sightedness”

Heuristics and Game Tree Search

Heuristics for Tree Exploration

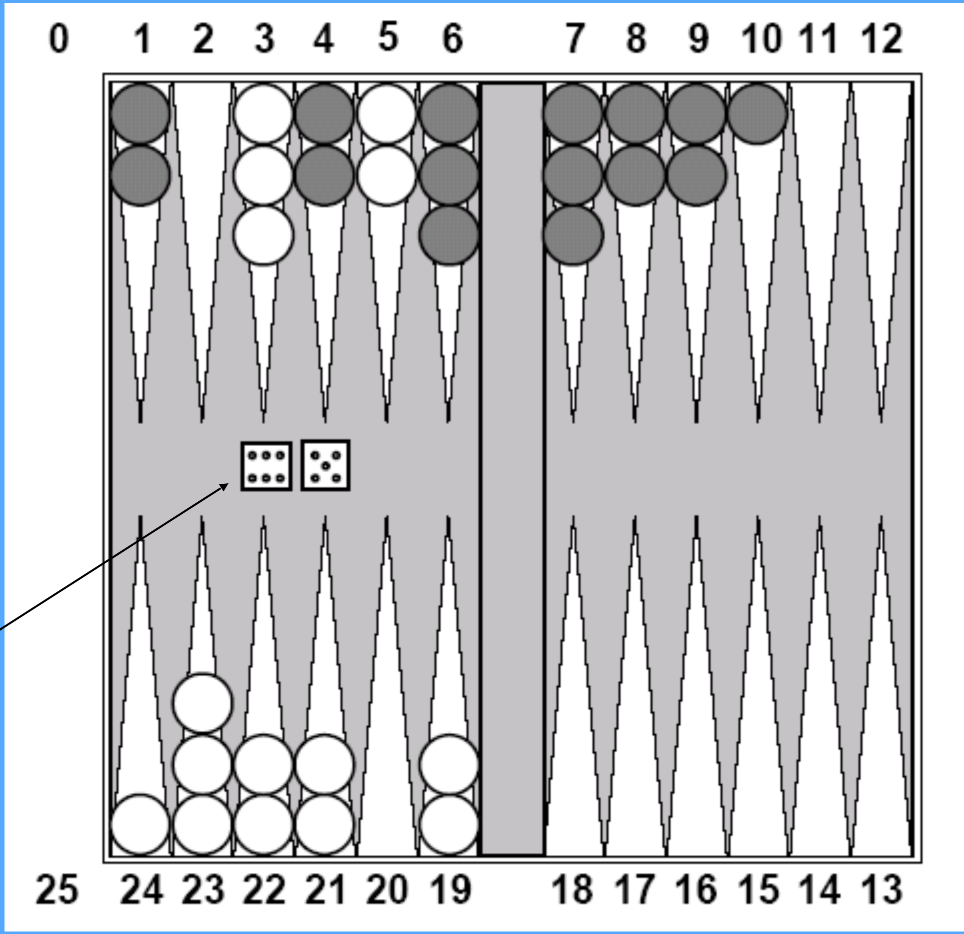
- it may be better to explore some branches more deeply in the allotted time (forward pruning)
- various heuristics exist to identify "promising" branches
  - expand to some depth k, and rank the nodes by the static evaluation function and choose the best m nodes to expand deeper.
  - for each node, determine the uncertainty of the evaluation function and go deeper on the most uncertain ones.
  - determine if a position is "active" or "passive". For active nodes, explore deeper.

Forward Pruning & Lookup tables

- Humans don’t consider all possible moves.
- Can we prune certain branches immediately?
- Use estimates (from past experience) of the uncertainty in the estimate of the node’s value and uses that to decide if a node can be pruned.
- Instead of search one can also store game states.
- Openings in chess are played from a library
- Endgames have often been solved and stored as well.

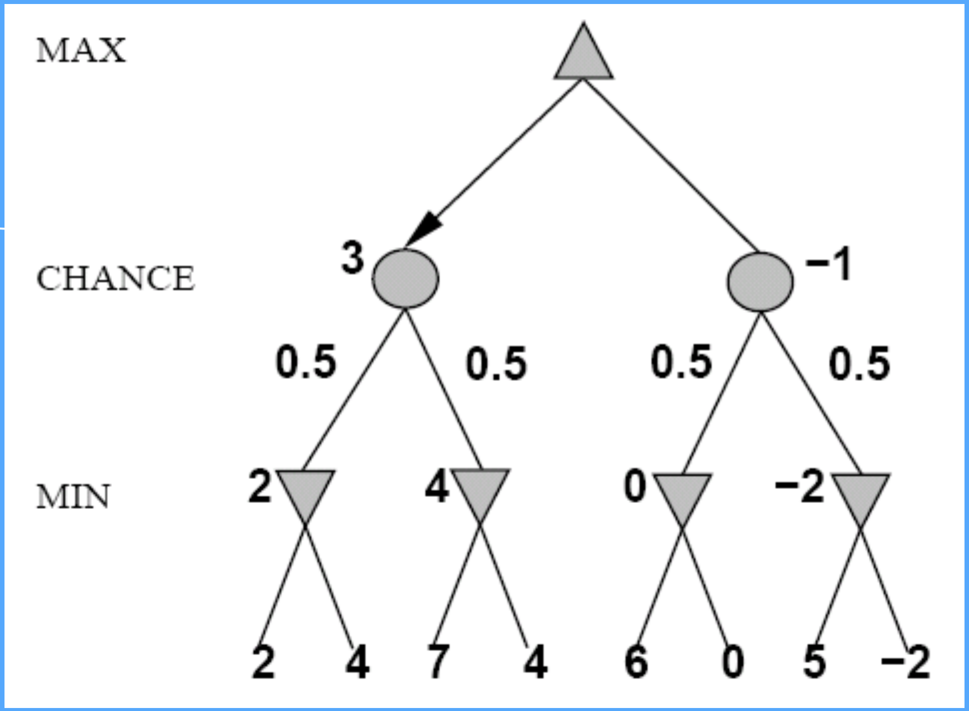
Chance Games

Backgammon



element of chance

Expected Minimax (Expectimax)



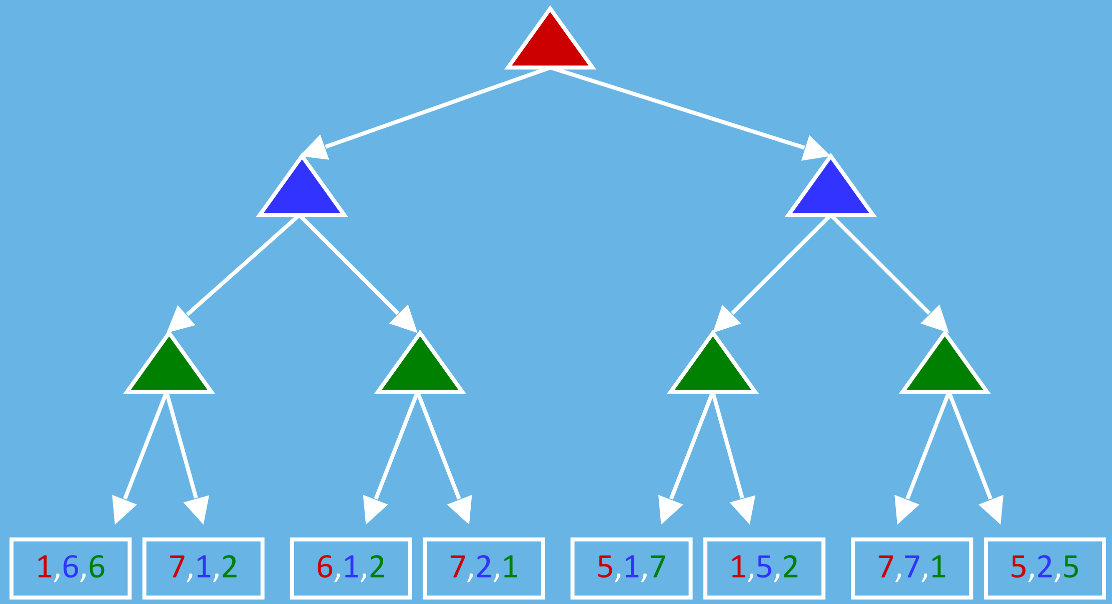
Again, the tree is constructed bottom-up.

Now we have even more nodes to search!

Multi-Agent Utilities

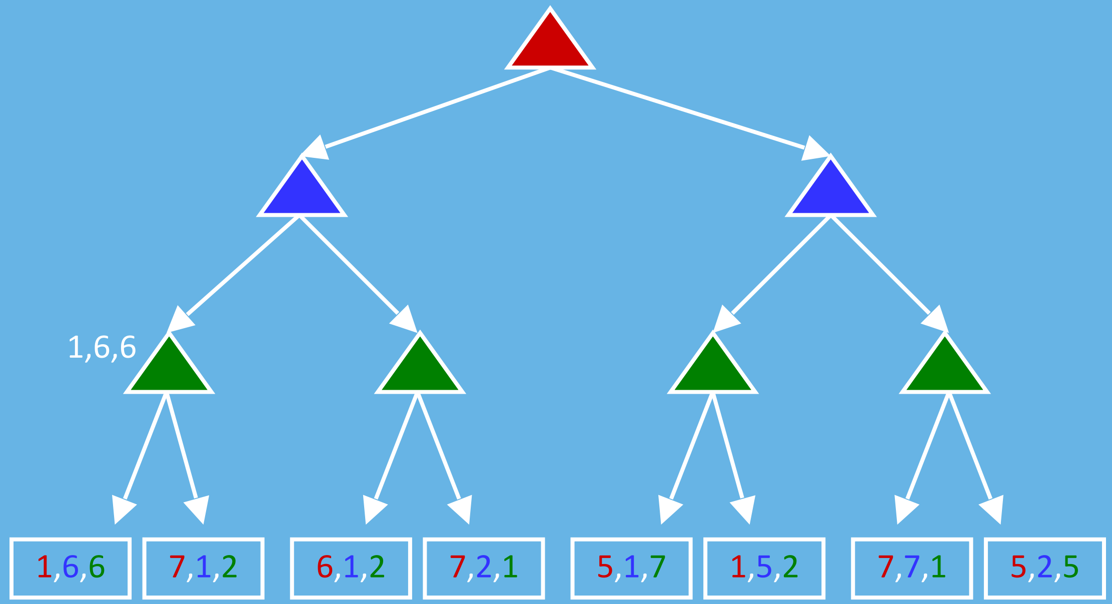
- What if the game is not zero-sum, or has multiple players?

- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and
  - competition dynamically…



Multi-Agent Utilities

- What if the game is not zero-sum, or has multiple players?
- Generalization of minimax:
  - Terminals have utility tuples
  - Node values are also utility tuples
  - Each player maximizes its own component
  - Can give rise to cooperation and
  - competition dynamically…



Summary

- Game playing is best 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.
- Minmax 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.