

Adversarial Search

Chapter 5

Mausam

(Based on slides of Stuart Russell, Henry Kautz, Linda Shapiro & UW AI Faculty)

Game Playing

Why do AI researchers study game playing?

1. It's a good reasoning problem, formal and nontrivial.
2. Direct comparison with humans and other computer programs is easy.

What Kinds of Games?

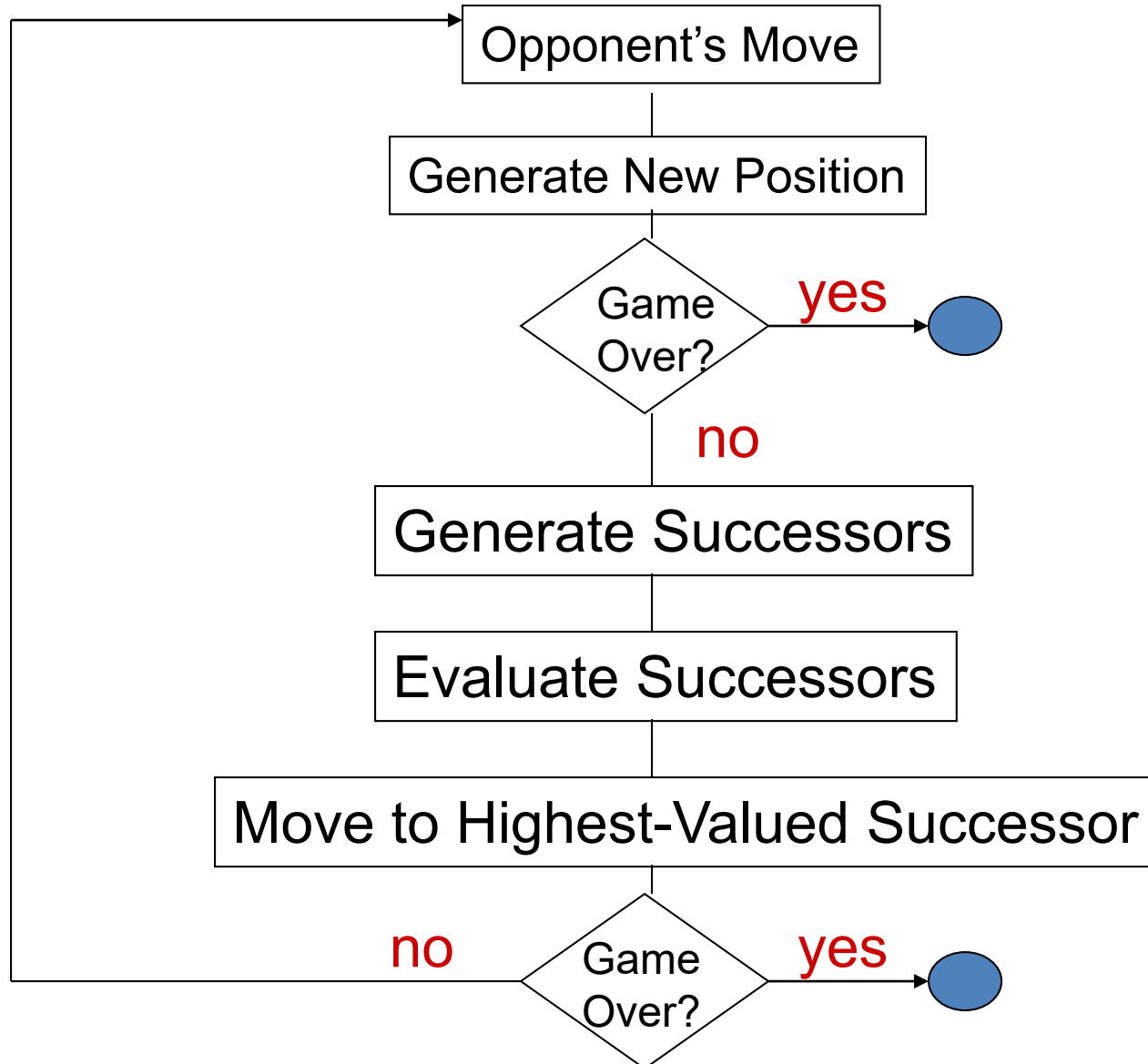
Mainly games of strategy with the following characteristics:

1. Sequence of **moves** to play
2. Rules that specify **possible moves**
3. Rules that specify a **payment** for each move
4. Objective is to **maximize** your payment

Games vs. Search Problems

- **Unpredictable opponent** → specifying a move for every possible opponent reply
- **Time limits** → unlikely to find goal, must approximate

Two-Player Game



Games as Adversarial Search

- States:
 - board configurations
- Initial state:
 - the board position and which player will move
- Successor function:
 - returns list of (move, state) pairs, each indicating a legal move and the resulting state
- Terminal test:
 - determines when the game is over
- Utility function:
 - gives a numeric value in terminal states
(e.g., -1, 0, +1 for loss, tie, win)

Game Tree (2-player, Deterministic, Turns)

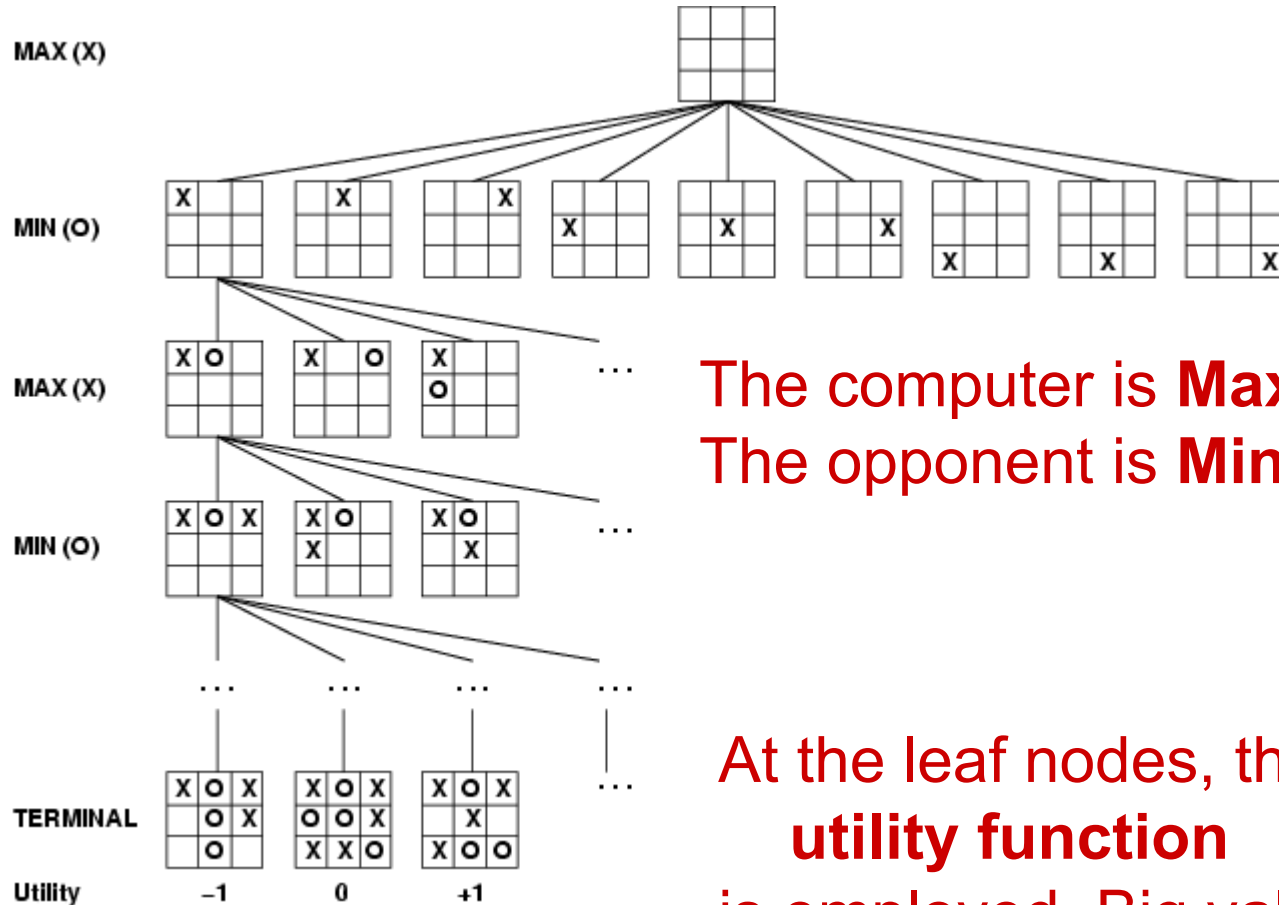
computer's
turn

opponent's
turn

computer's
turn

opponent's
turn

leaf nodes
are evaluated



The computer is **Max**.
The opponent is **Min**.

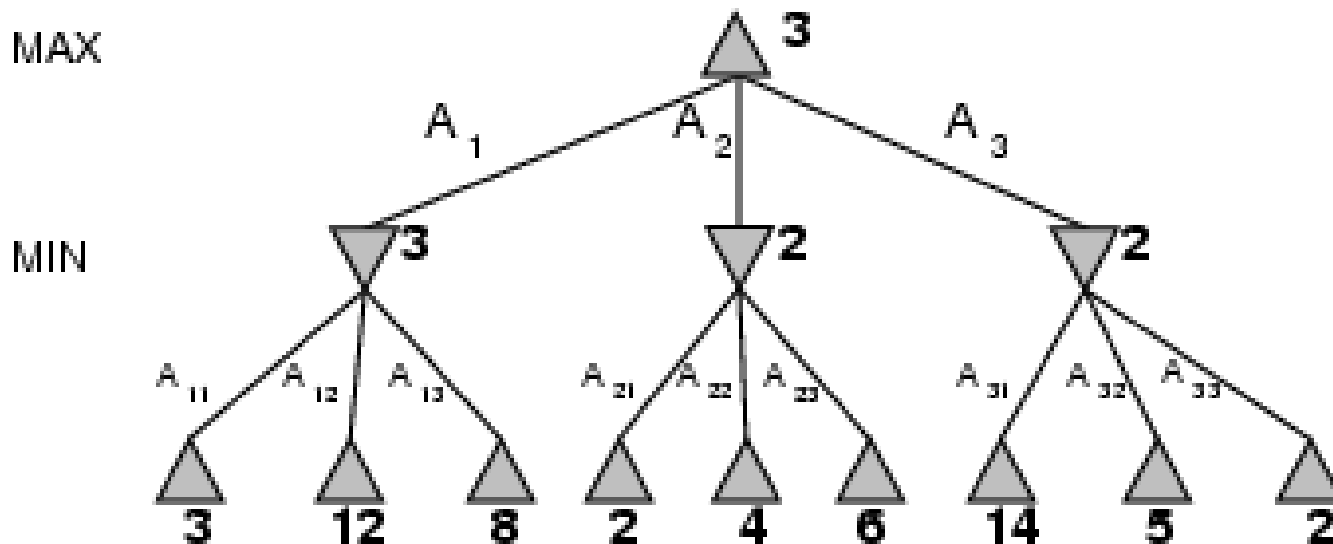
At the leaf nodes, the
utility function
is employed. Big value
means good, small is bad.

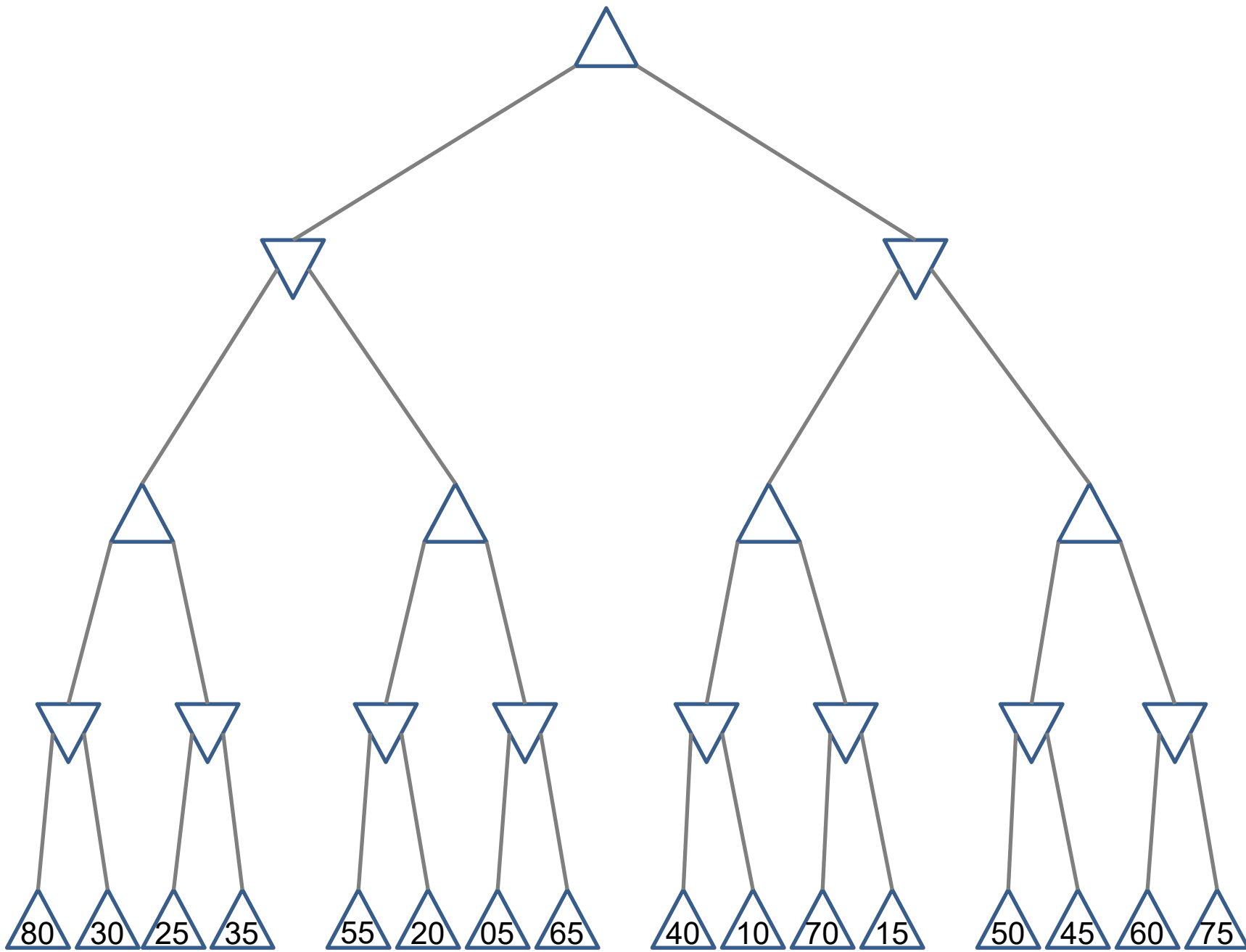
Mini-Max Terminology

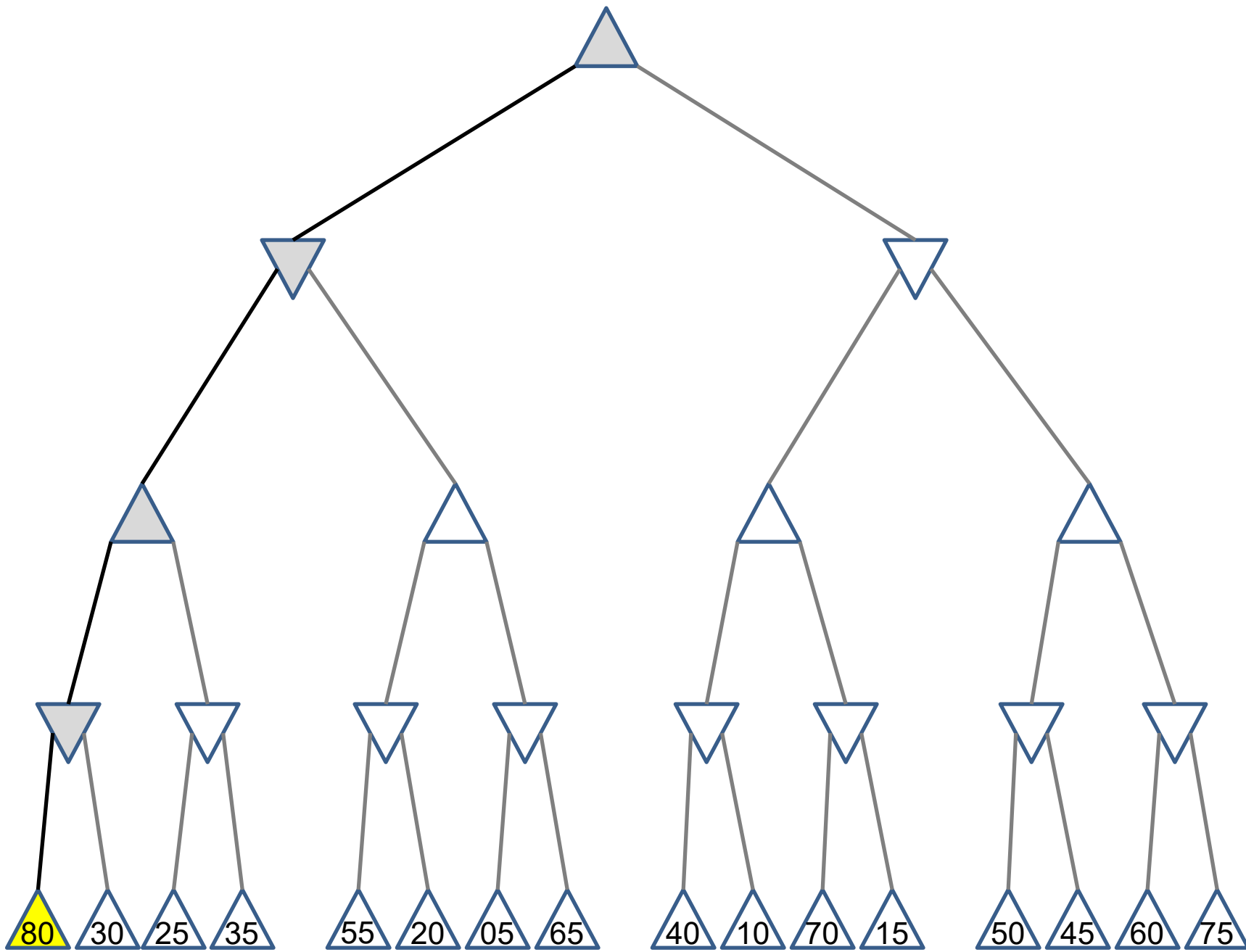
- **move**: a move by both players
- **ply**: a half-move
- **utility function**: the function applied to leaf nodes
- **backed-up value**
 - of a **max-position**: the value of its largest successor
 - of a **min-position**: the value of its smallest successor
- **minimax procedure**: search down several levels; at the bottom level apply the utility function, back-up values all the way up to the root node, and that node selects the move.

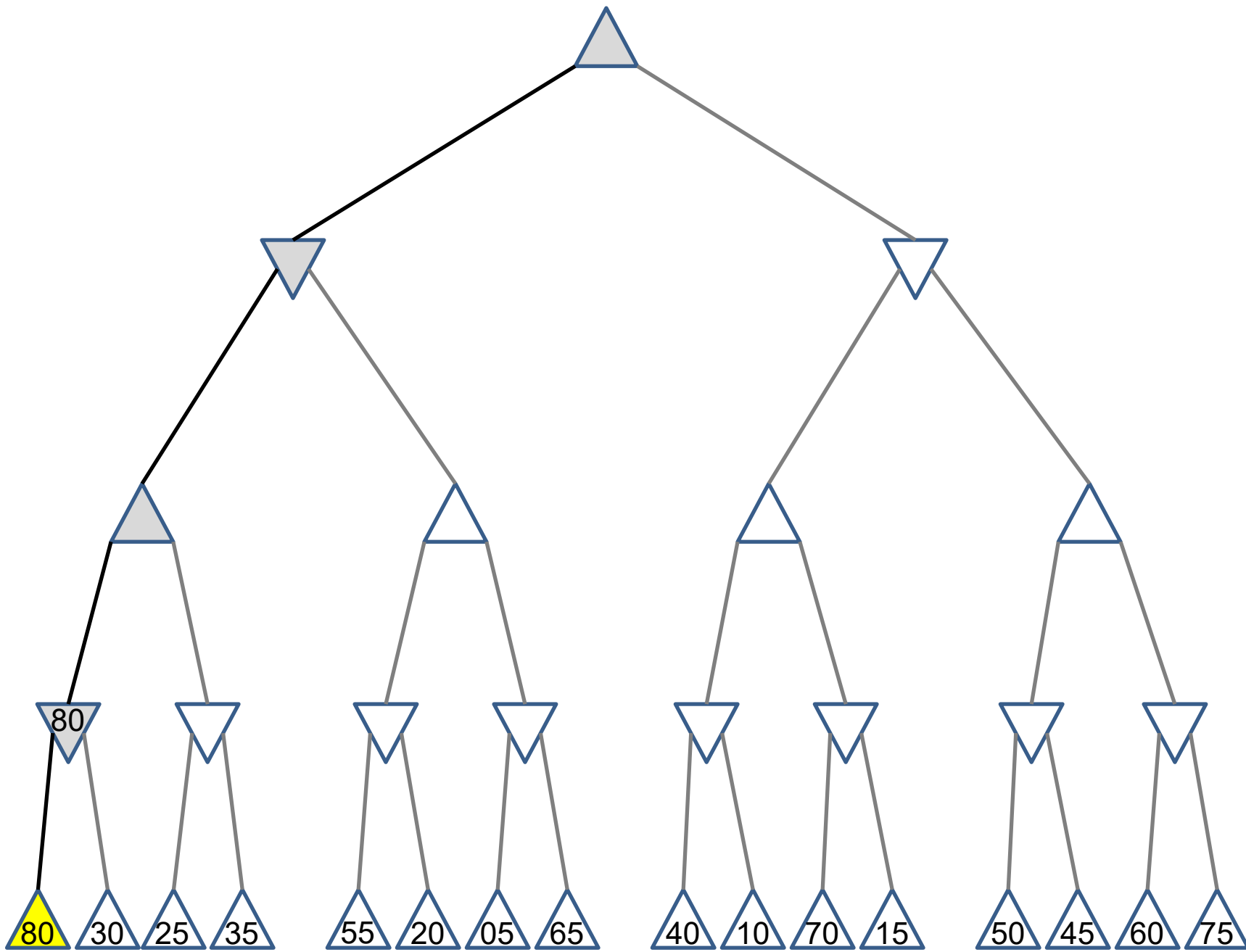
Minimax

- Perfect play for deterministic games
- Idea: choose move to position with highest **minimax value**
= best achievable payoff against best play
- E.g., 2-ply game:

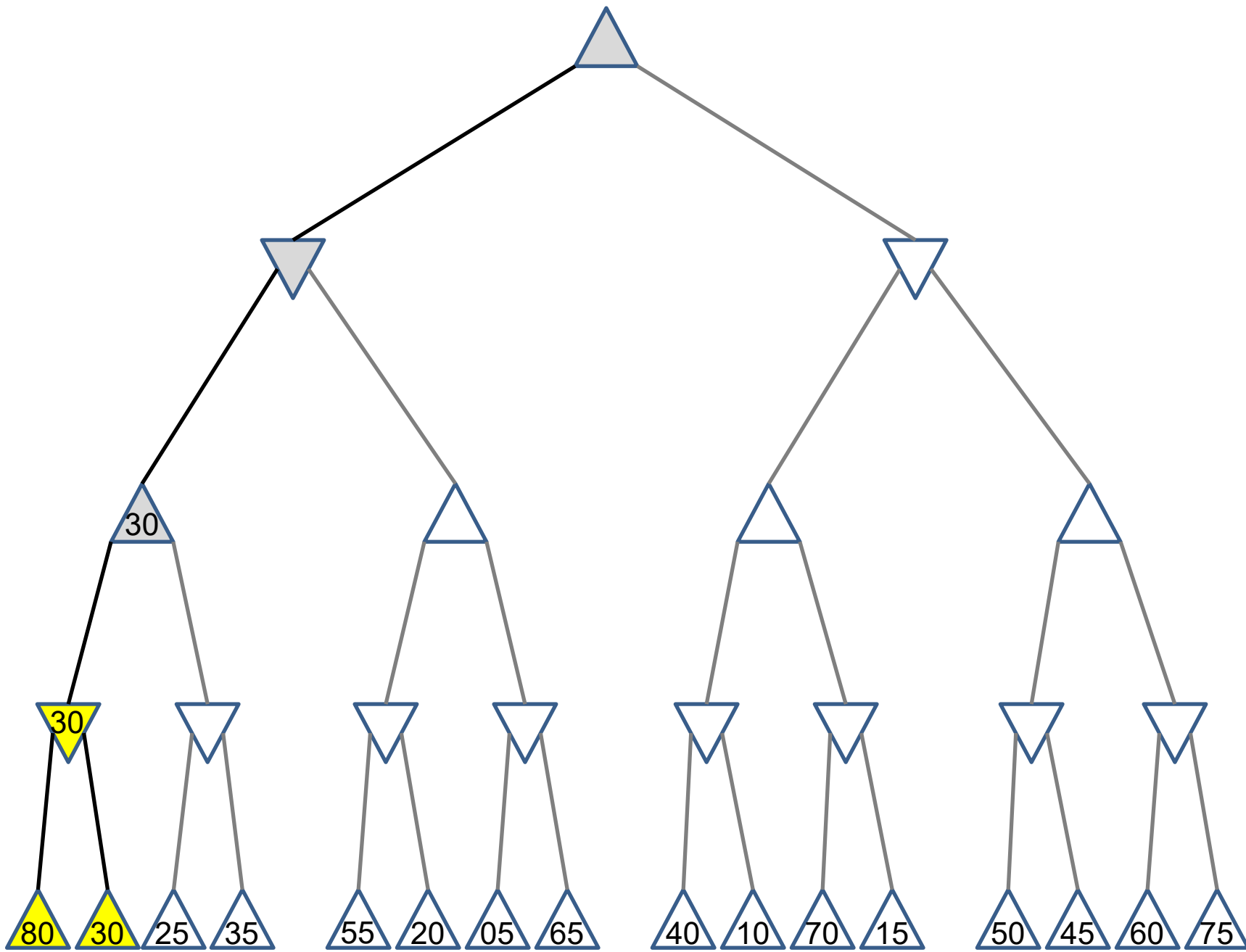


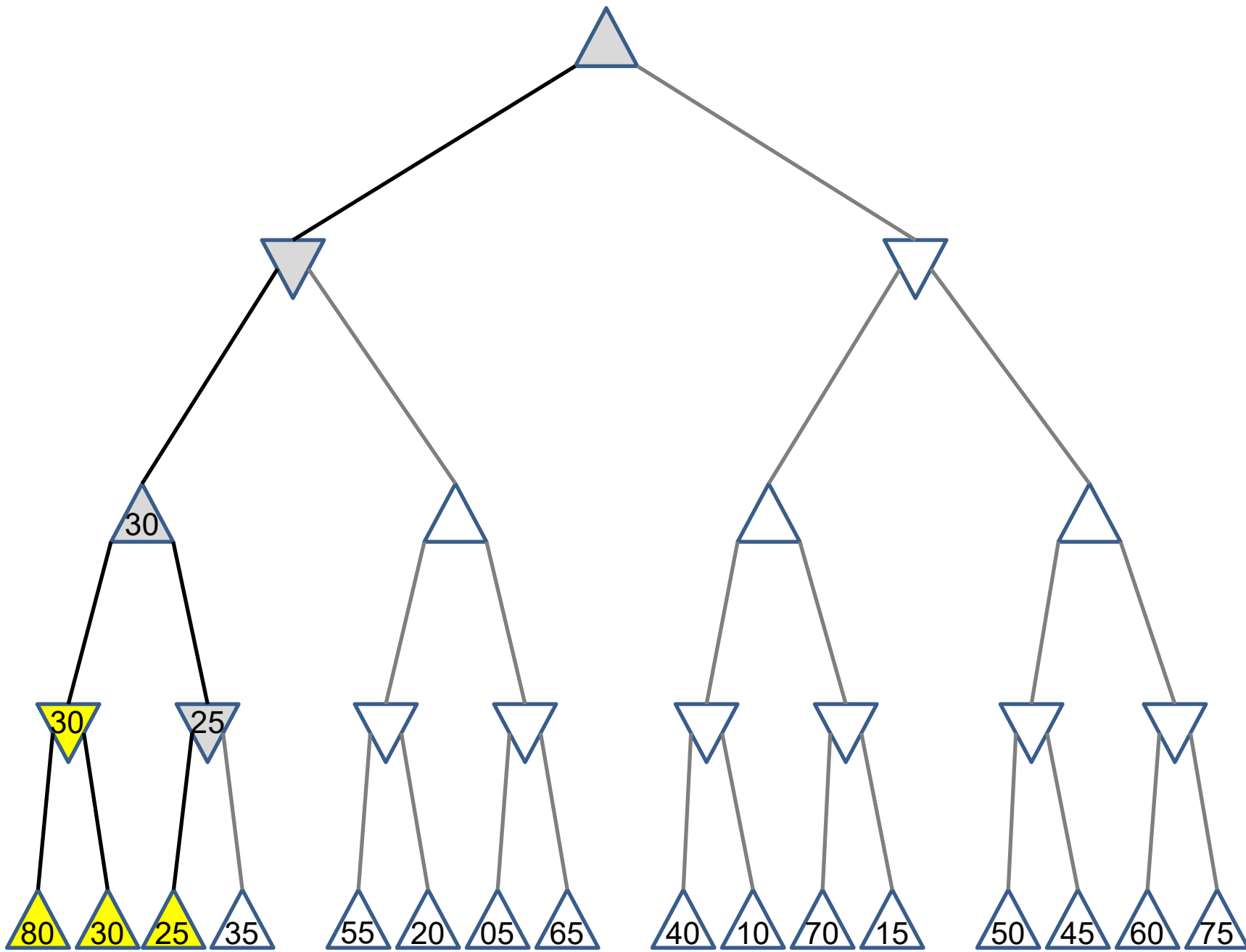


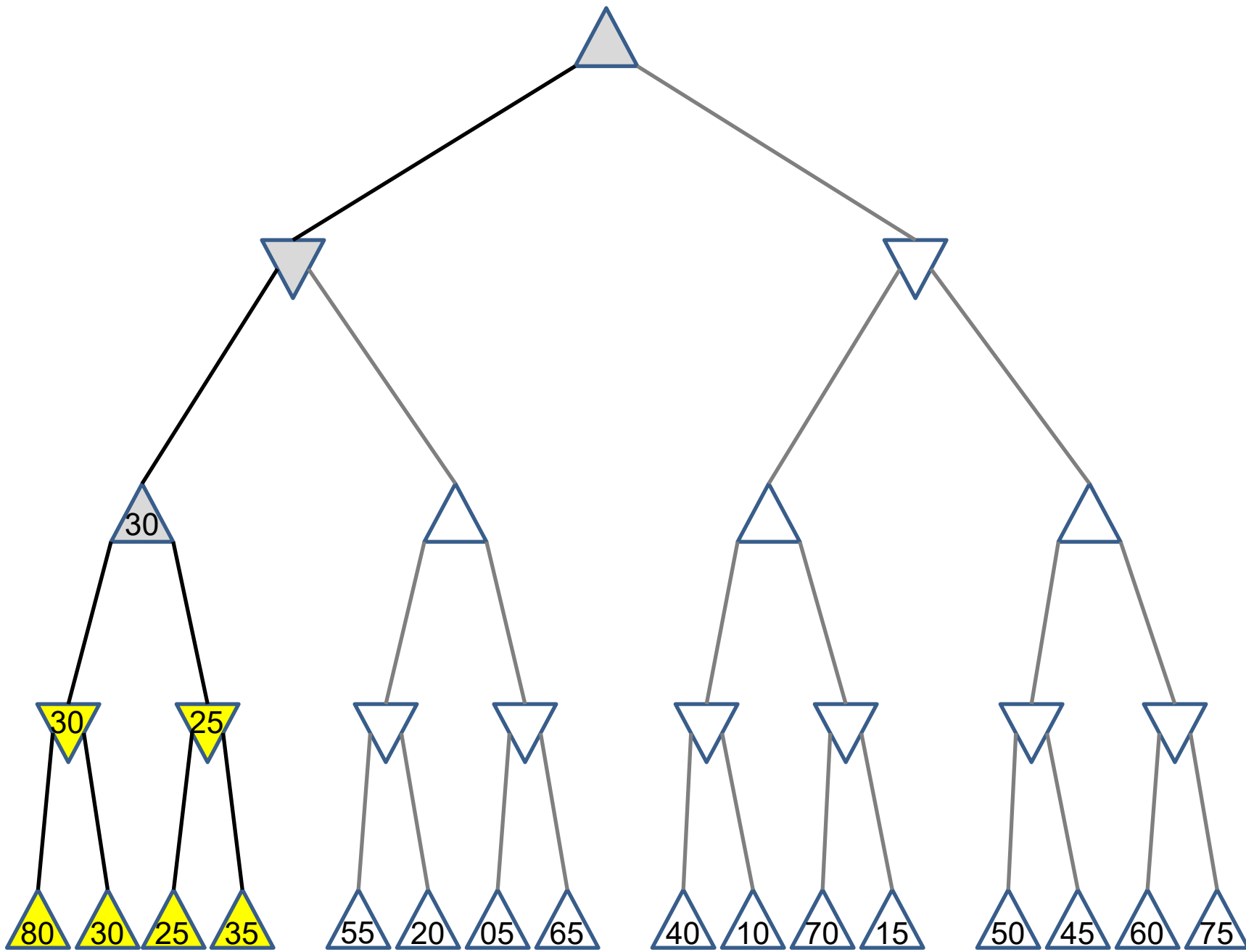




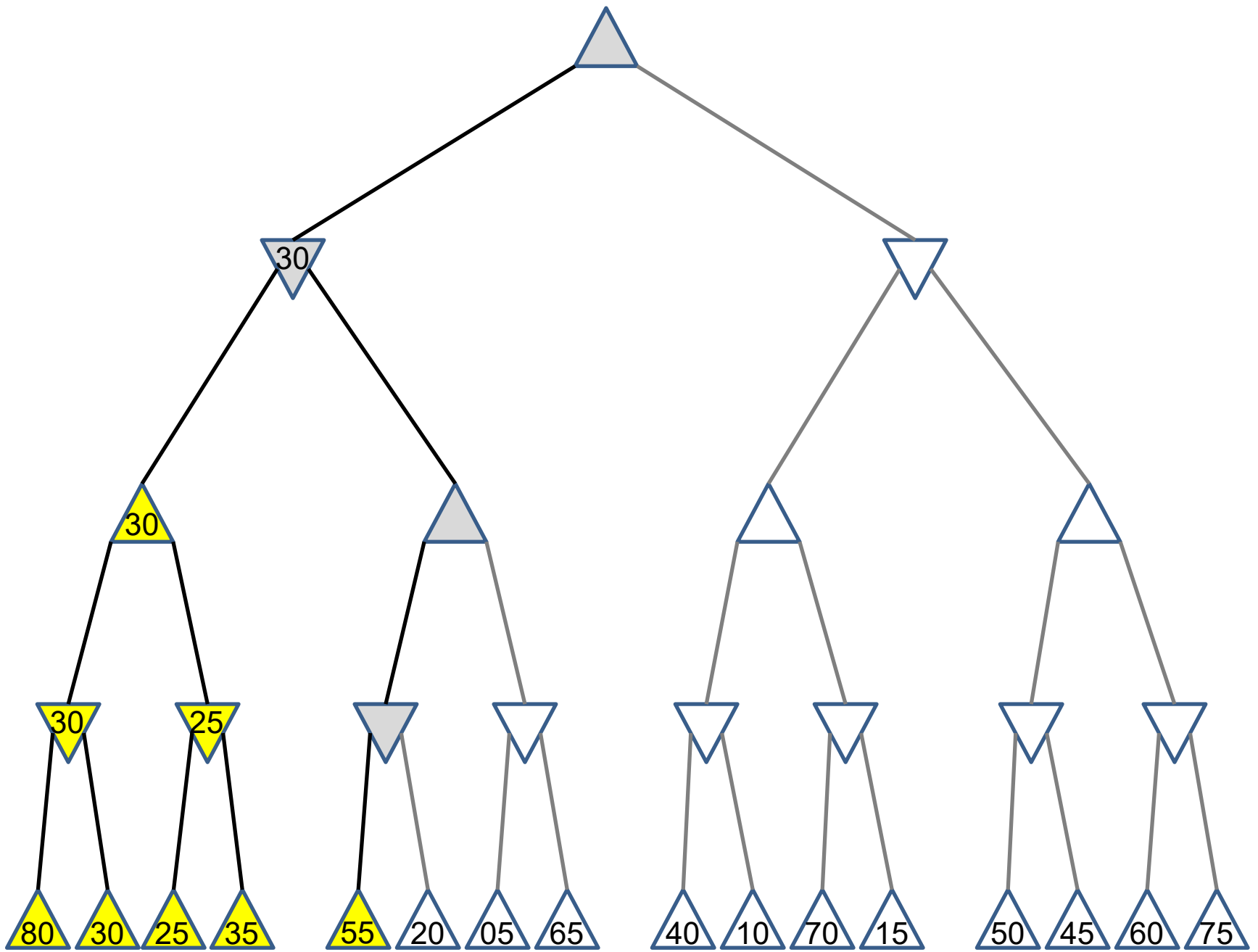


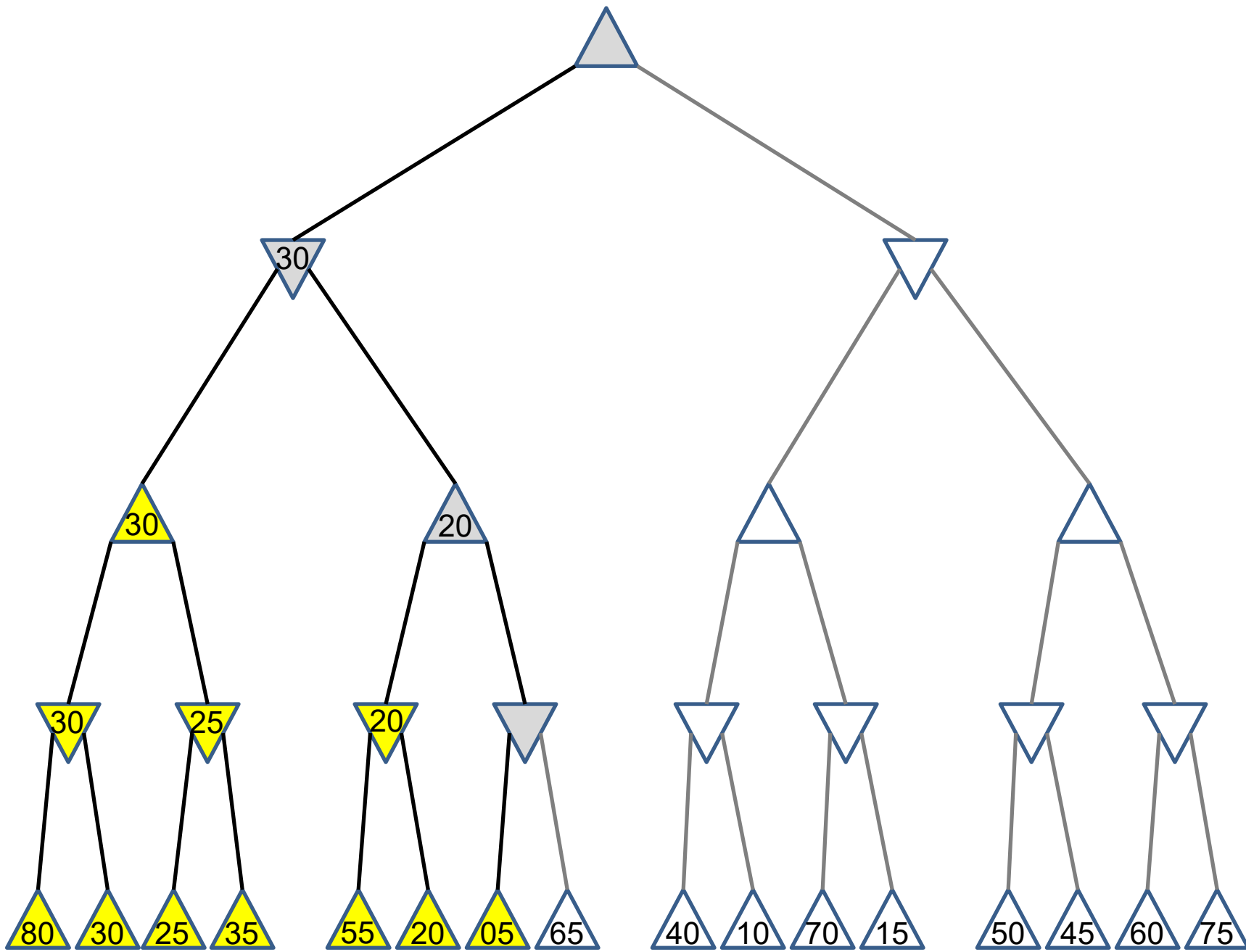


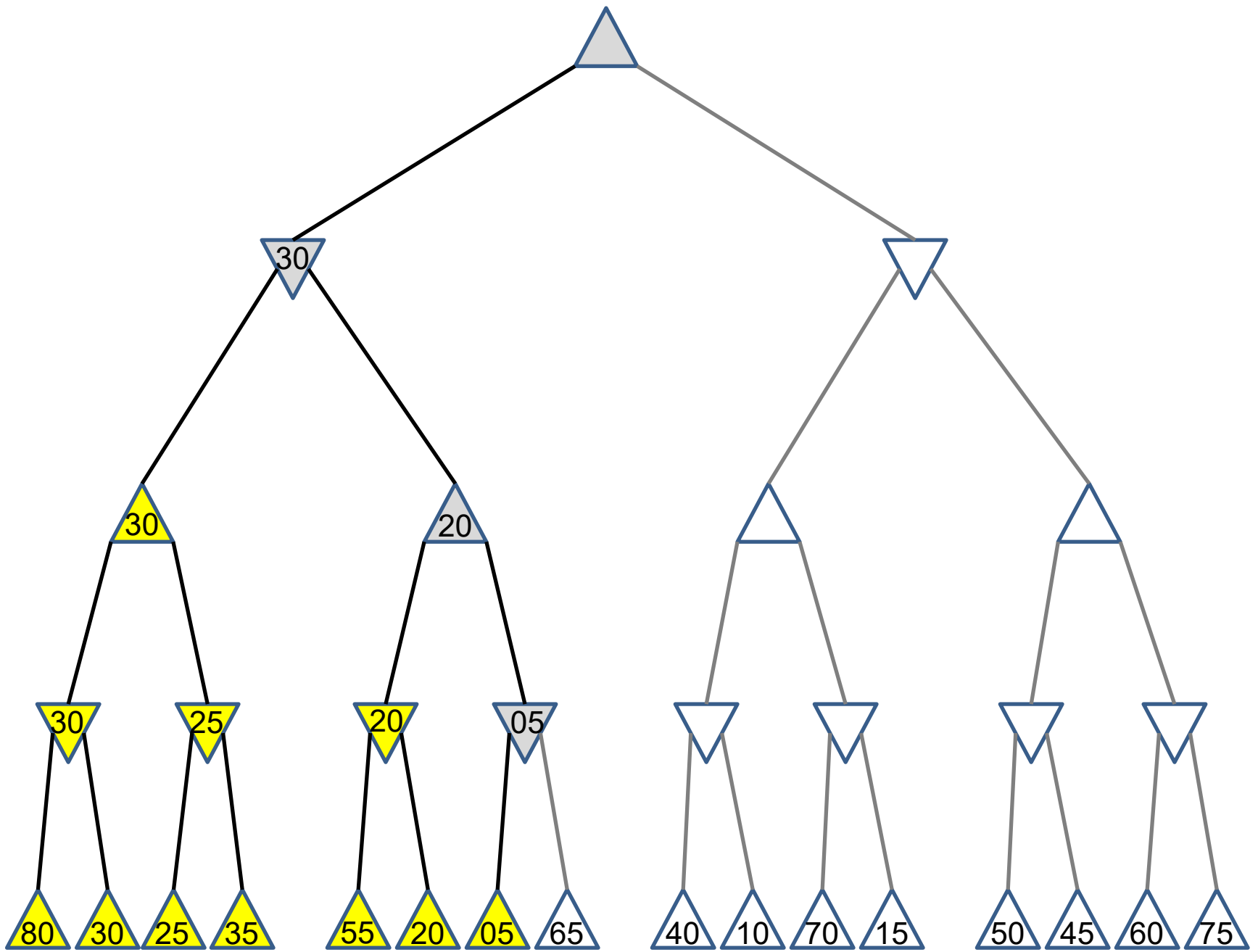


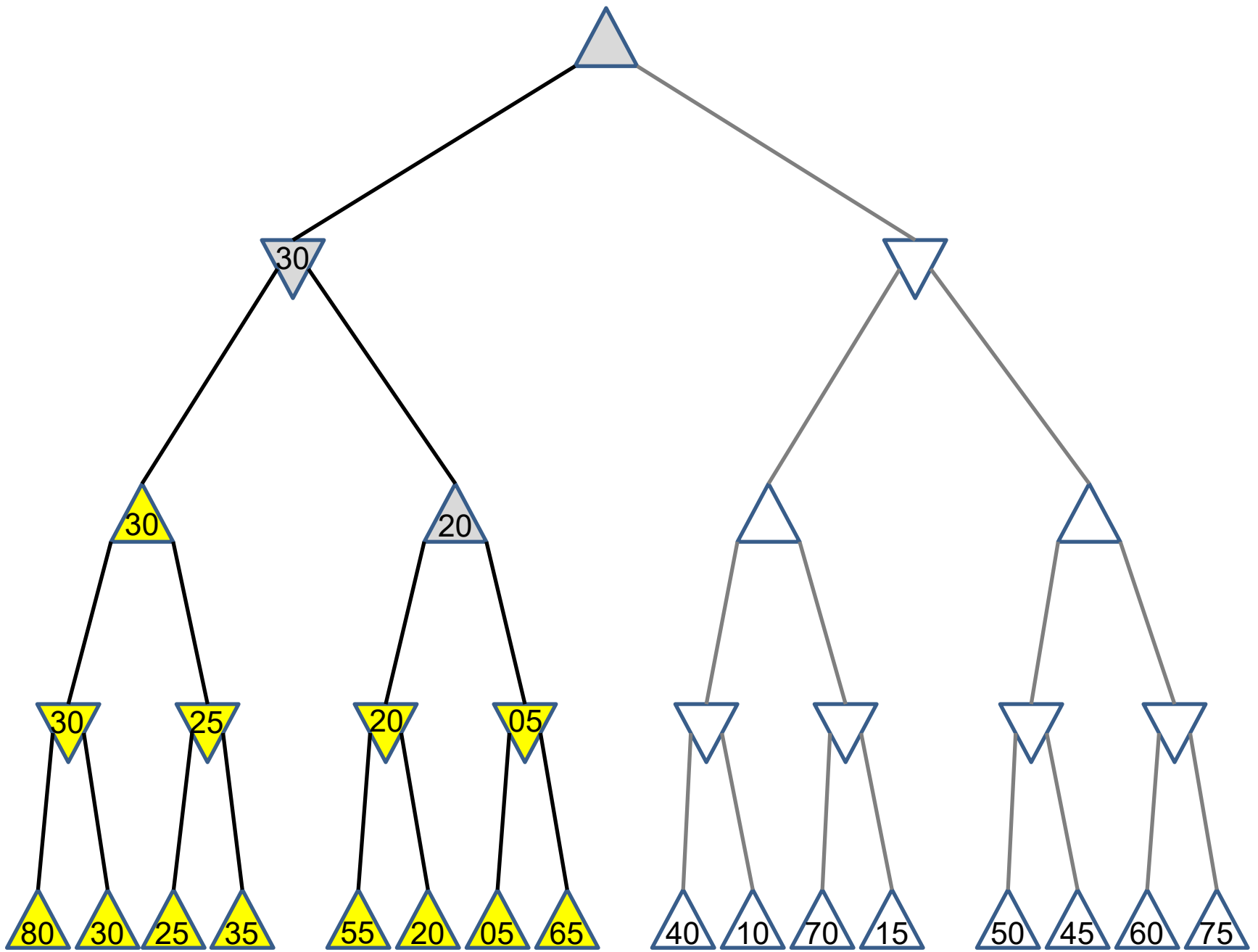




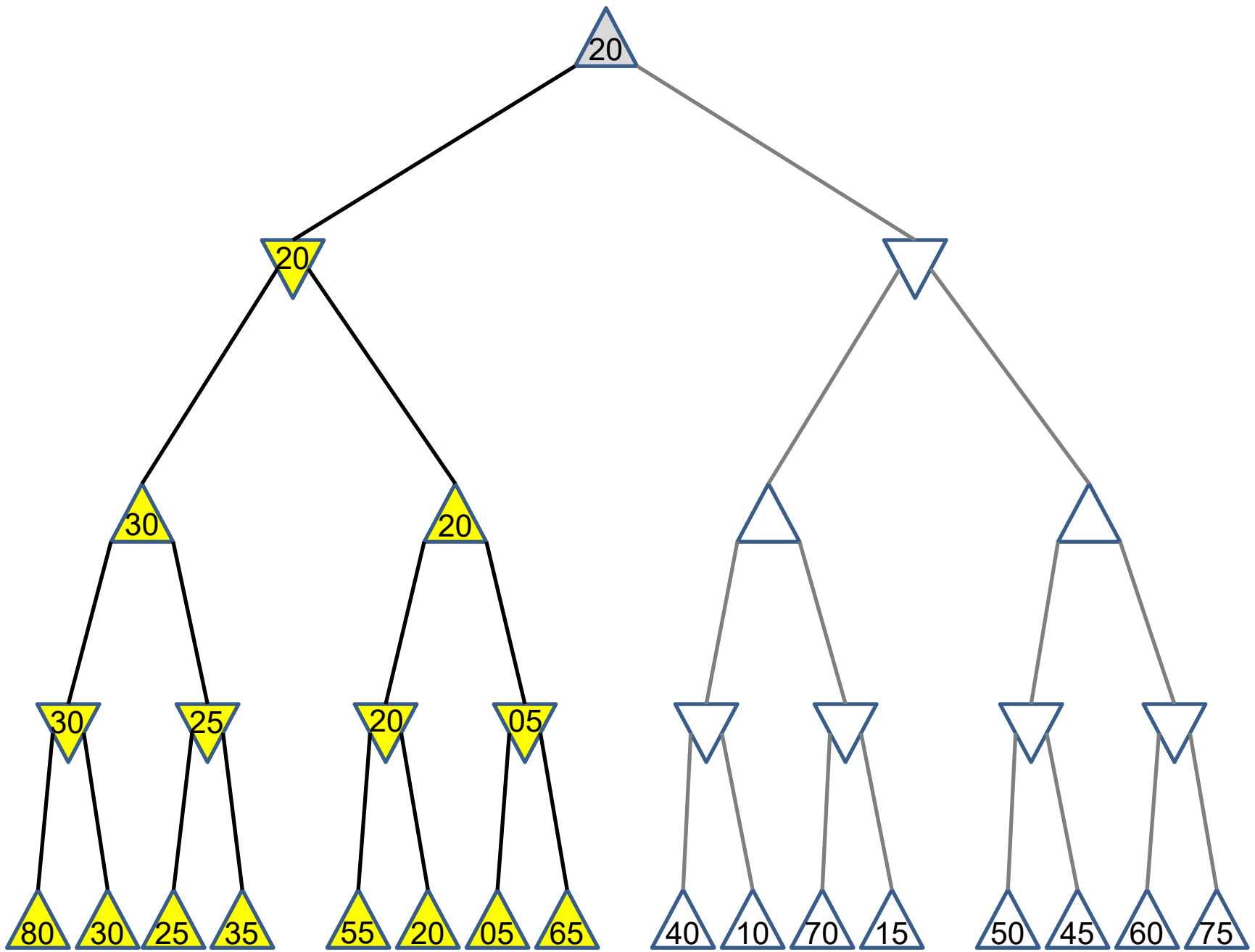


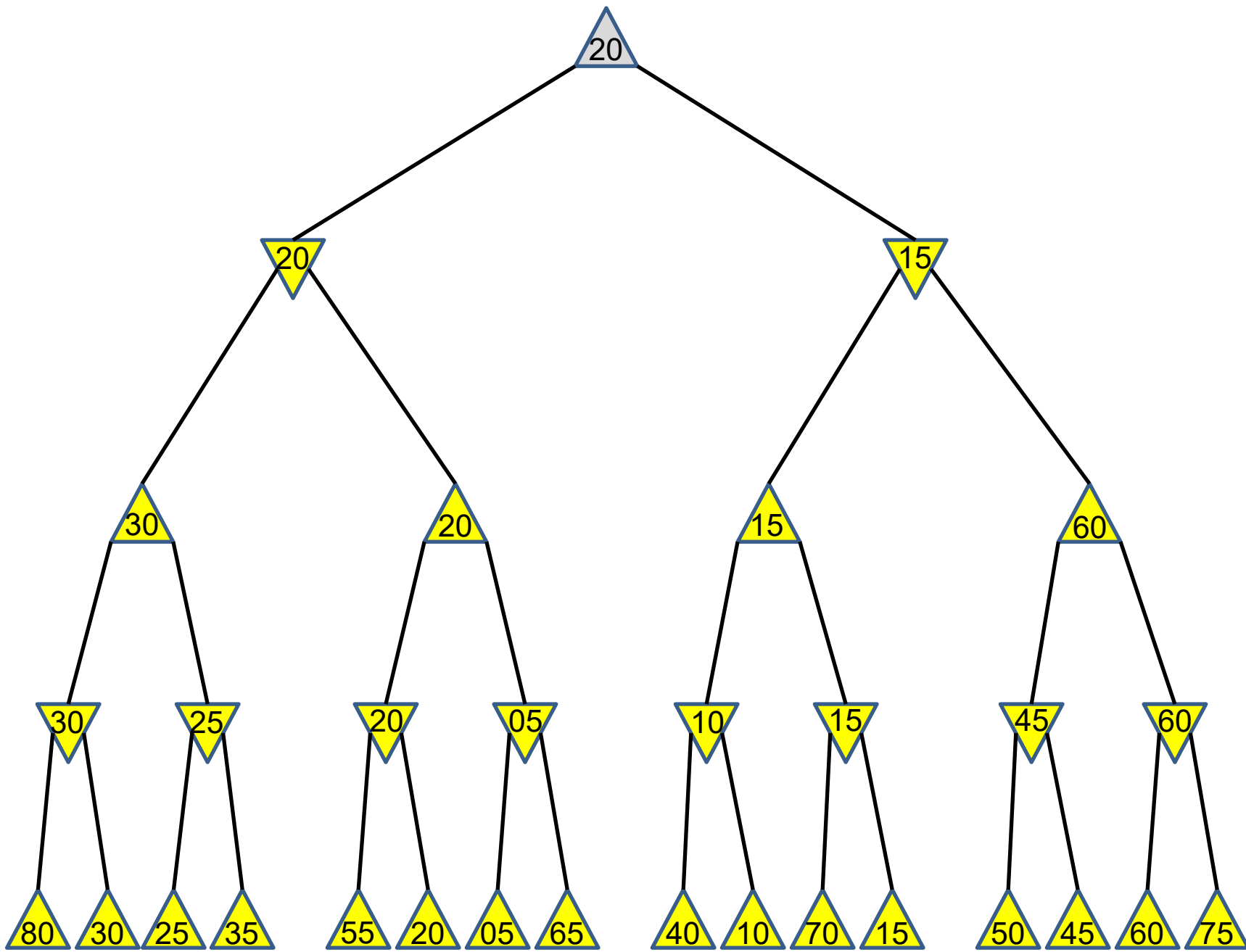


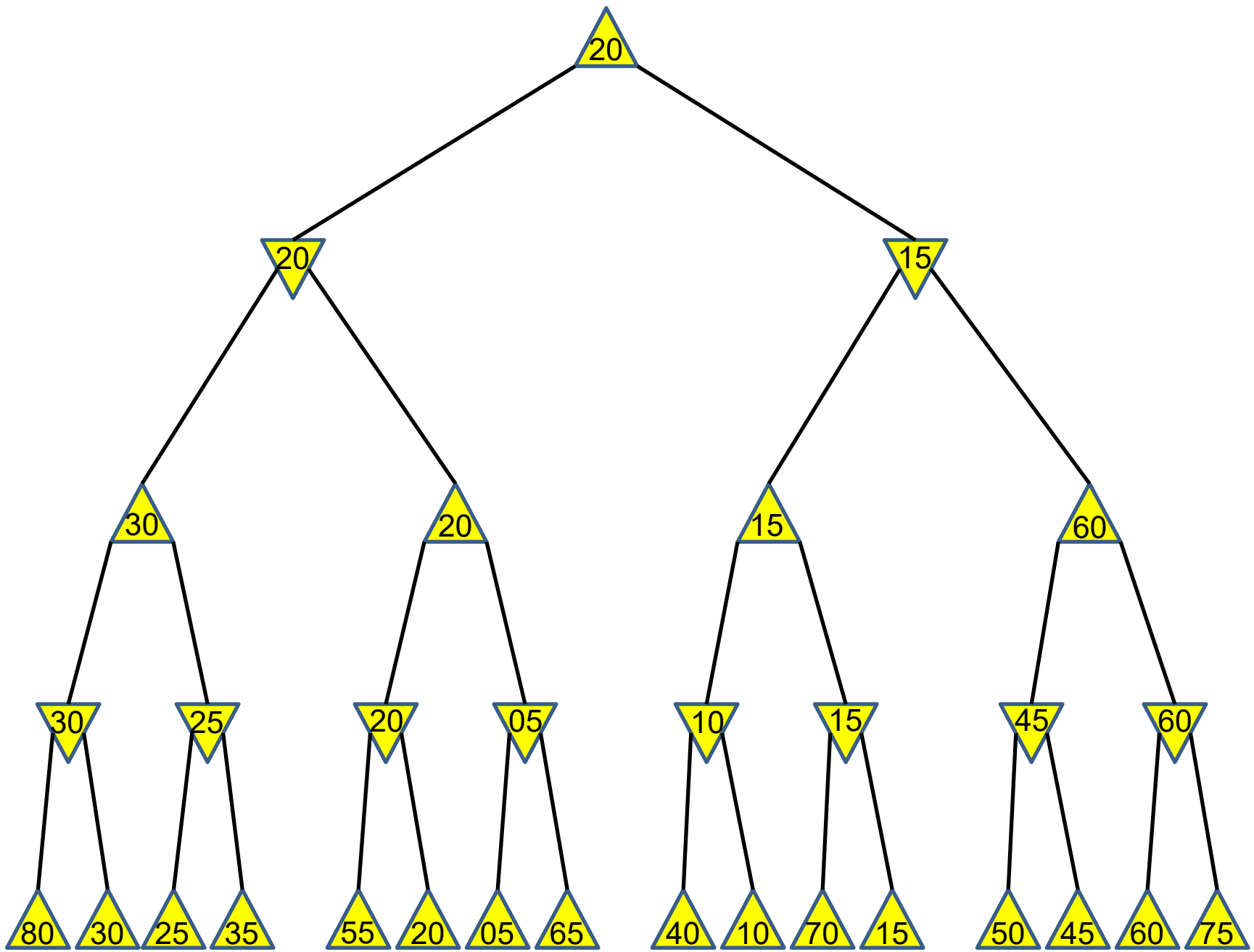














Minimax Strategy

- Why do we take the **min** value every other level of the tree?
- These nodes represent the **opponent's** choice of move.
- The computer assumes that the human will choose that move that is of **least value** to the computer.

Minimax algorithm

Adversarial analogue of DFS

function MINIMAX-DECISION(*state*) *returns an action*

$v \leftarrow \text{MAX-VALUE}(\textit{state})$

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*)

$v \leftarrow -\infty$

for *a, s* in SUCCESSORS(*state*) **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(s))$

return *v*

function MIN-VALUE(*state*) *returns a utility value*

if TERMINAL-TEST(*state*) **then return** UTILITY(*state*)

$v \leftarrow \infty$

for *a, s* in SUCCESSORS(*state*) **do**

$v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(s))$

return *v*

Properties of Minimax

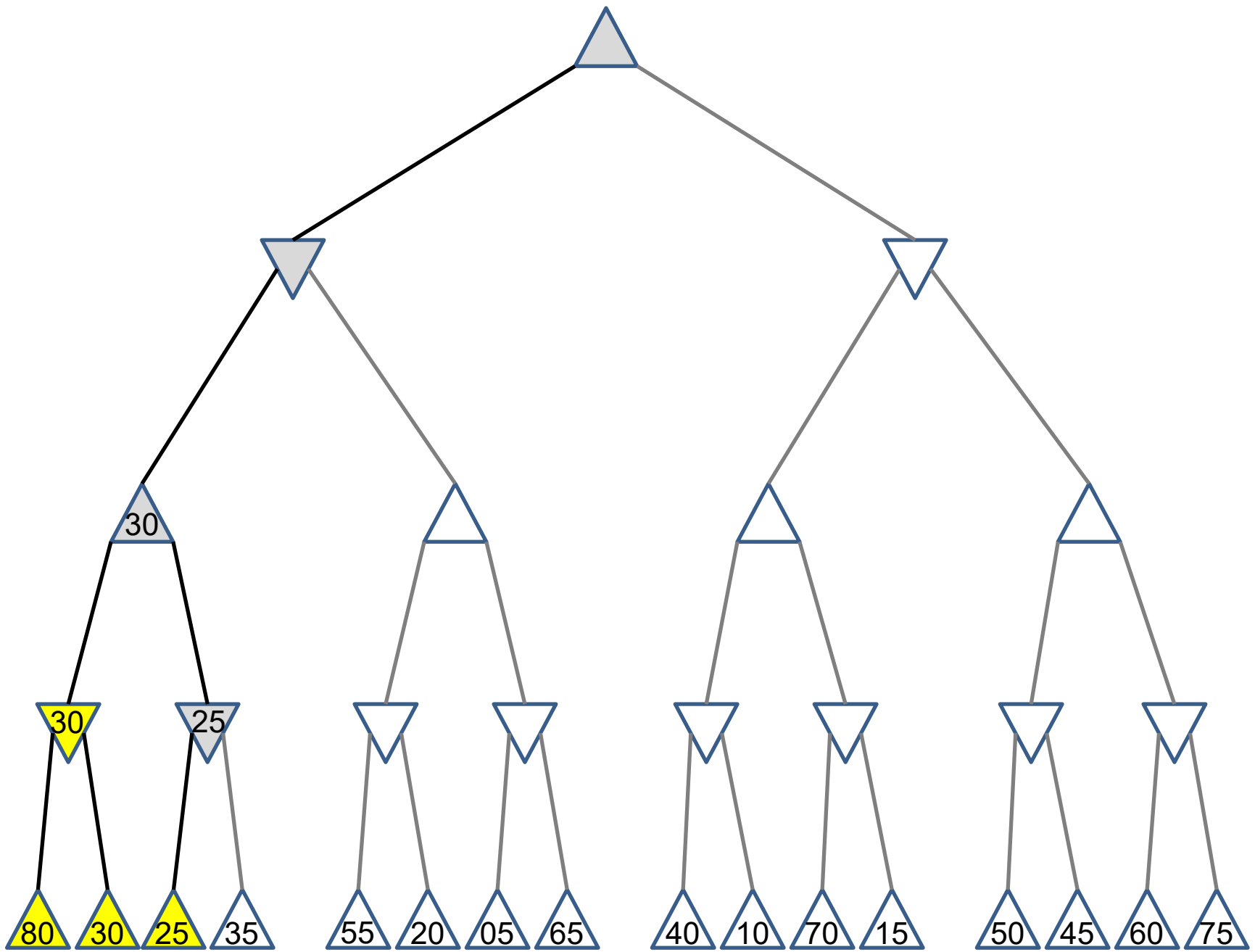
- Complete?
 - Yes (if tree is finite)
- Optimal?
 - Yes (against an optimal opponent)
 - No (does not exploit opponent weakness against suboptimal opponent)
- Time complexity?
 - $O(b^m)$
- Space complexity?
 - $O(bm)$ (depth-first exploration)

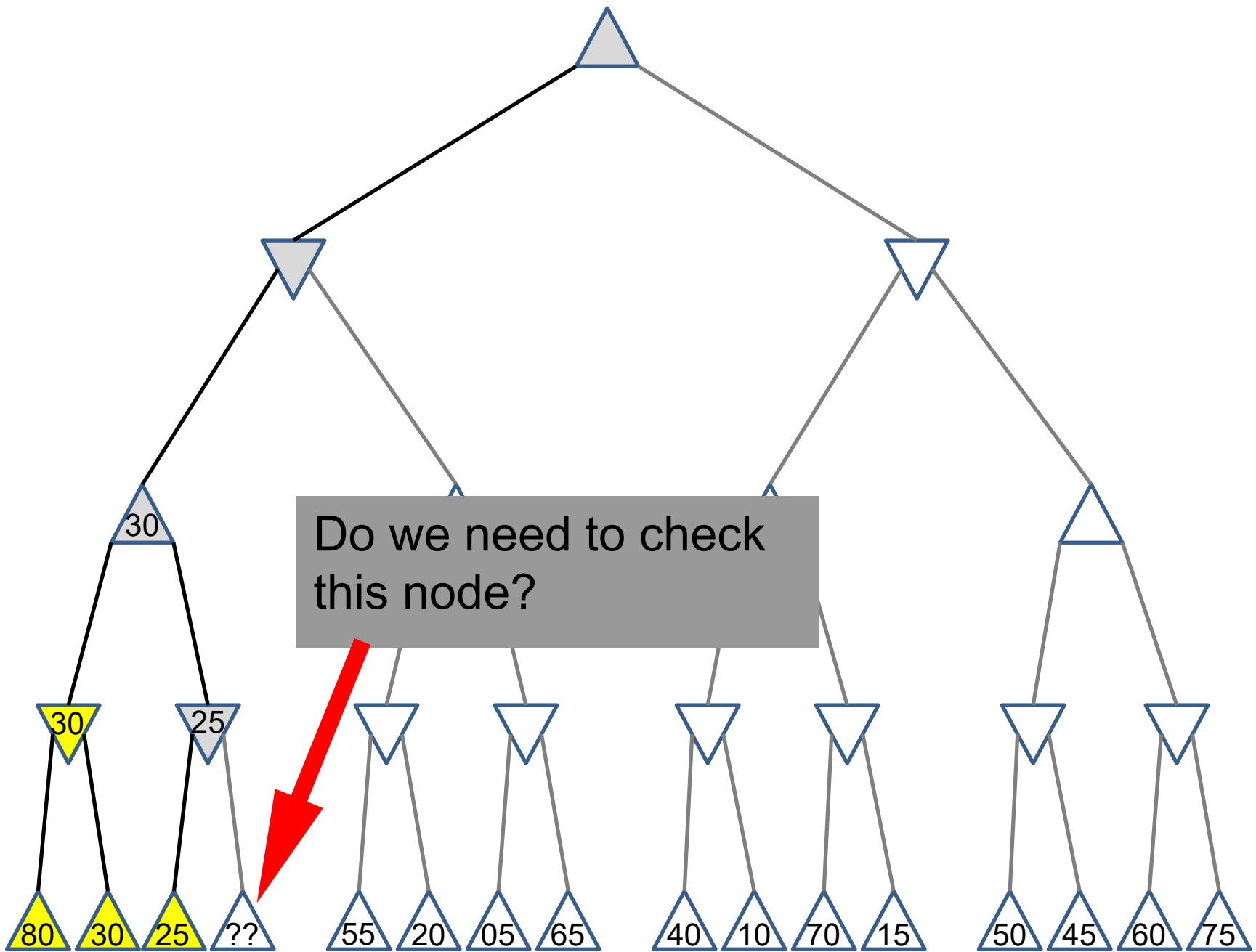
Good Enough?

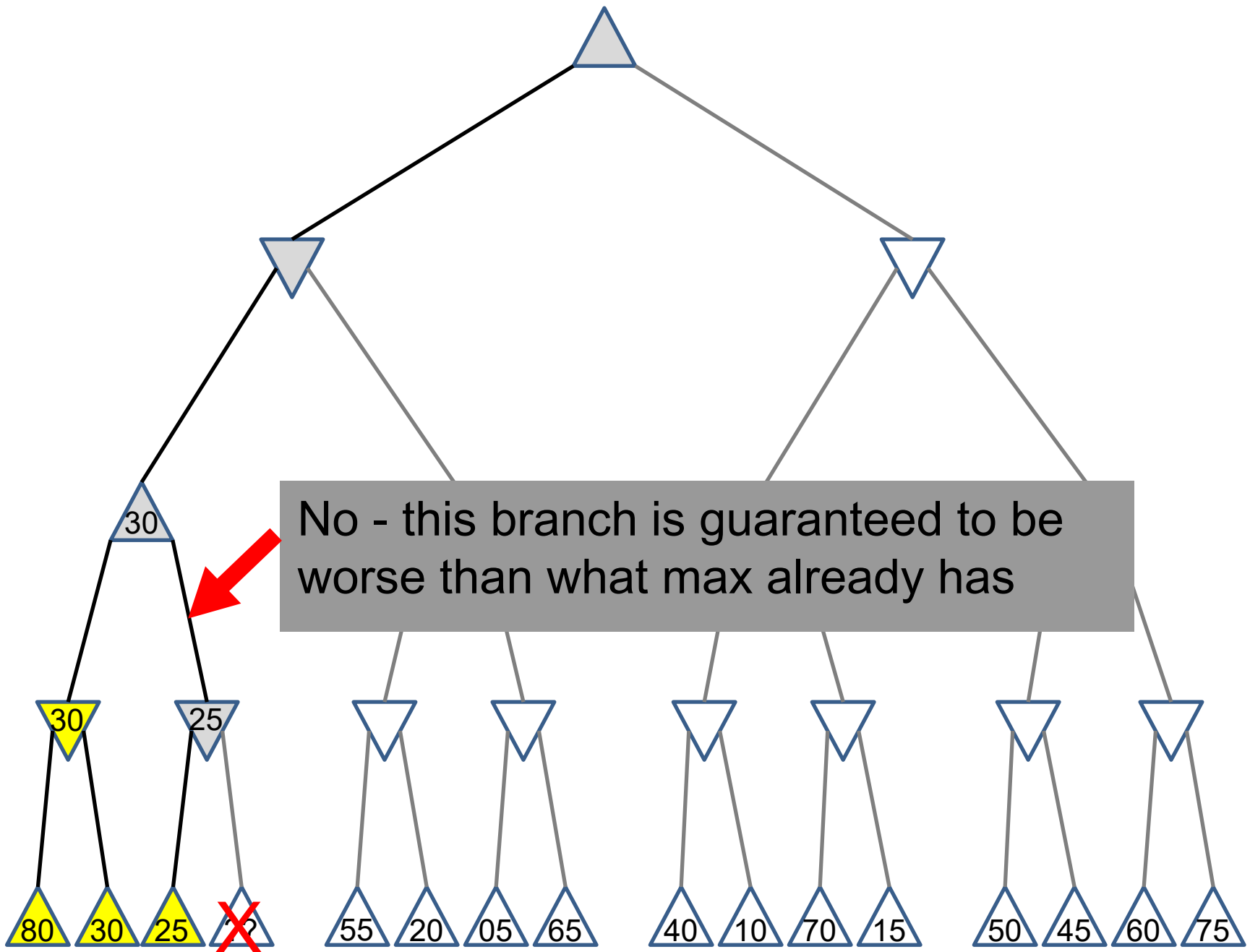
- Chess:
 - branching factor $b \approx 35$
 - game length $m \approx 100$
 - search space $b^m \approx 35^{100} \approx 10^{154}$
- The Universe:
 - number of atoms $\approx 10^{78}$
 - age $\approx 10^{18}$ seconds
 - 10^8 moves/sec $\times 10^{78} \times 10^{18} = 10^{104}$
- Exact solution completely infeasible

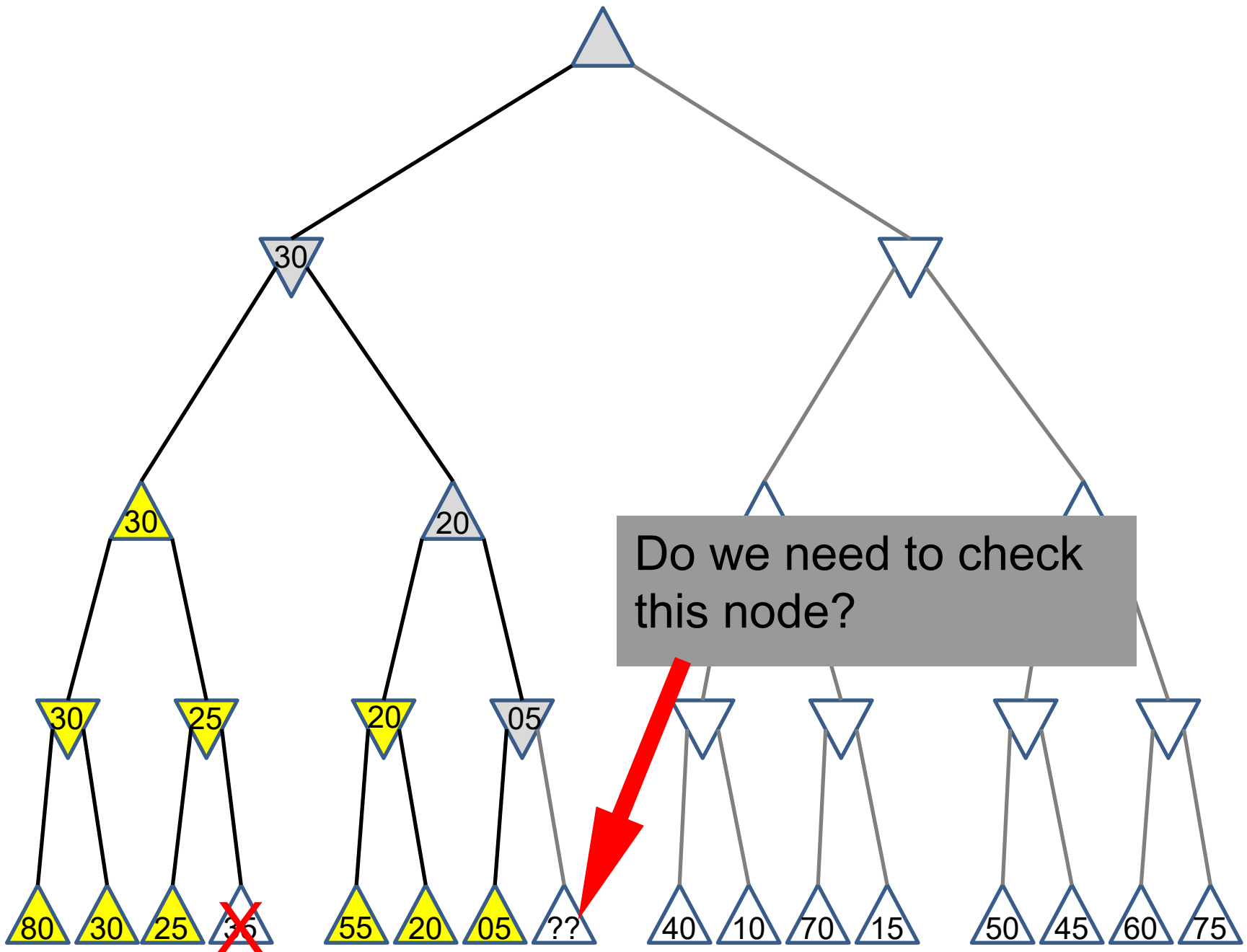


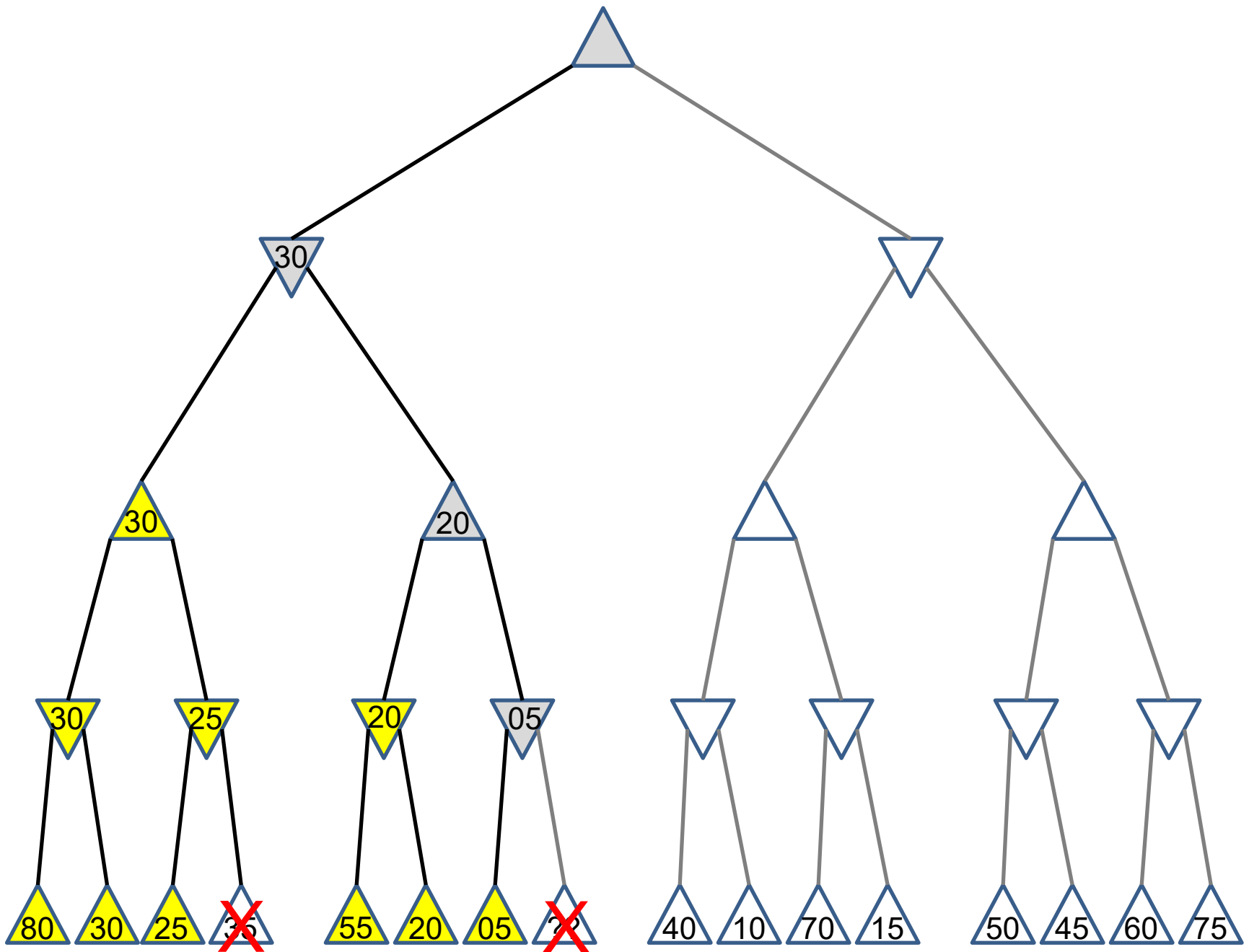












Alpha-Beta

- The alpha-beta procedure can speed up a depth-first minimax search.
- Alpha: a lower bound on the value that a max node may ultimately be assigned

$$v \geq \alpha$$

- Beta: an upper bound on the value that a minimizing node may ultimately be assigned

$$v \leq \beta$$

Alpha-Beta

```
MinVal(state, alpha, beta){  
    if (terminal(state))  
        return utility(state);  
    for (s in children(state)){  
        child = MaxVal(s,alpha,beta);  
        beta = min(beta,child);  
        if (alpha>=beta) return child;  
    }  
    return best child (min); }
```

alpha = the **highest** value for **MAX** along the path

beta = the **lowest** value for **MIN** along the path

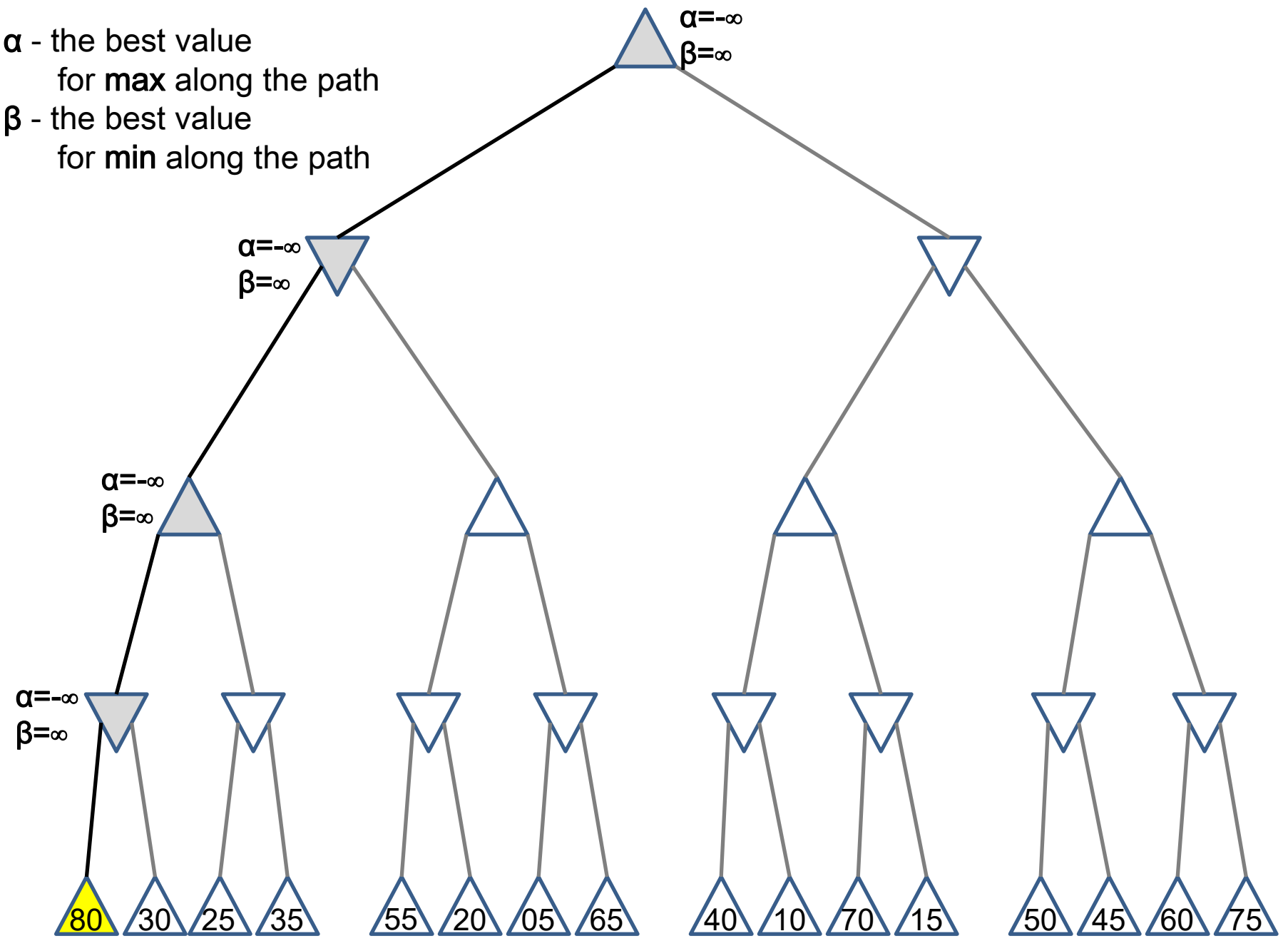
Alpha-Beta

```
MaxVal(state, alpha, beta){  
    if (terminal(state))  
        return utility(state);  
    for (s in children(state)){  
        child = MinVal(s,alpha,beta);  
        alpha = max(alpha,child);  
        if (alpha>=beta) return child;  
    }  
    return best child (max); }
```

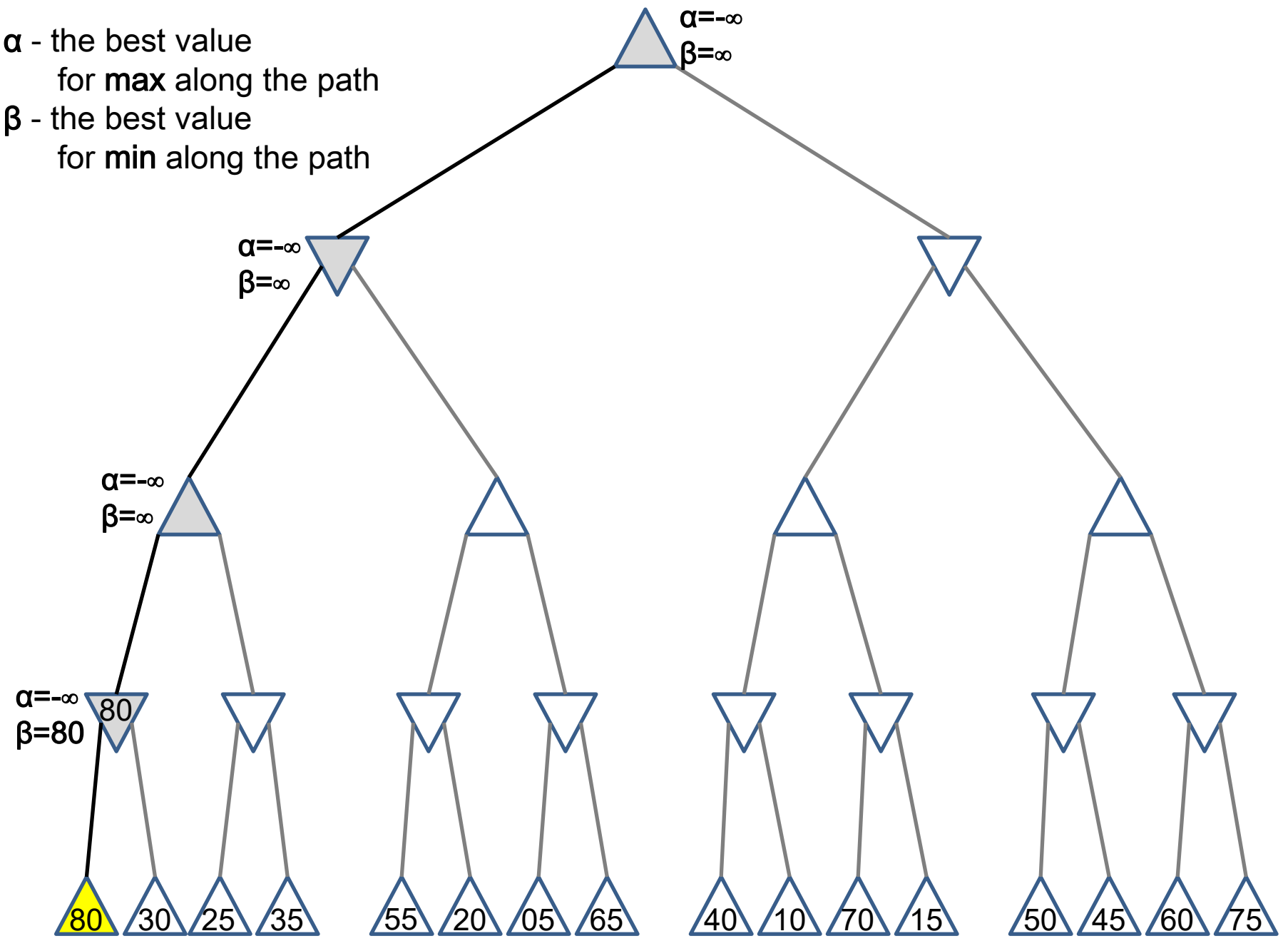
alpha = the highest value for MAX along the path

beta = the lowest value for MIN along the path

α - the best value
for **max** along the path
 β - the best value
for **min** along the path

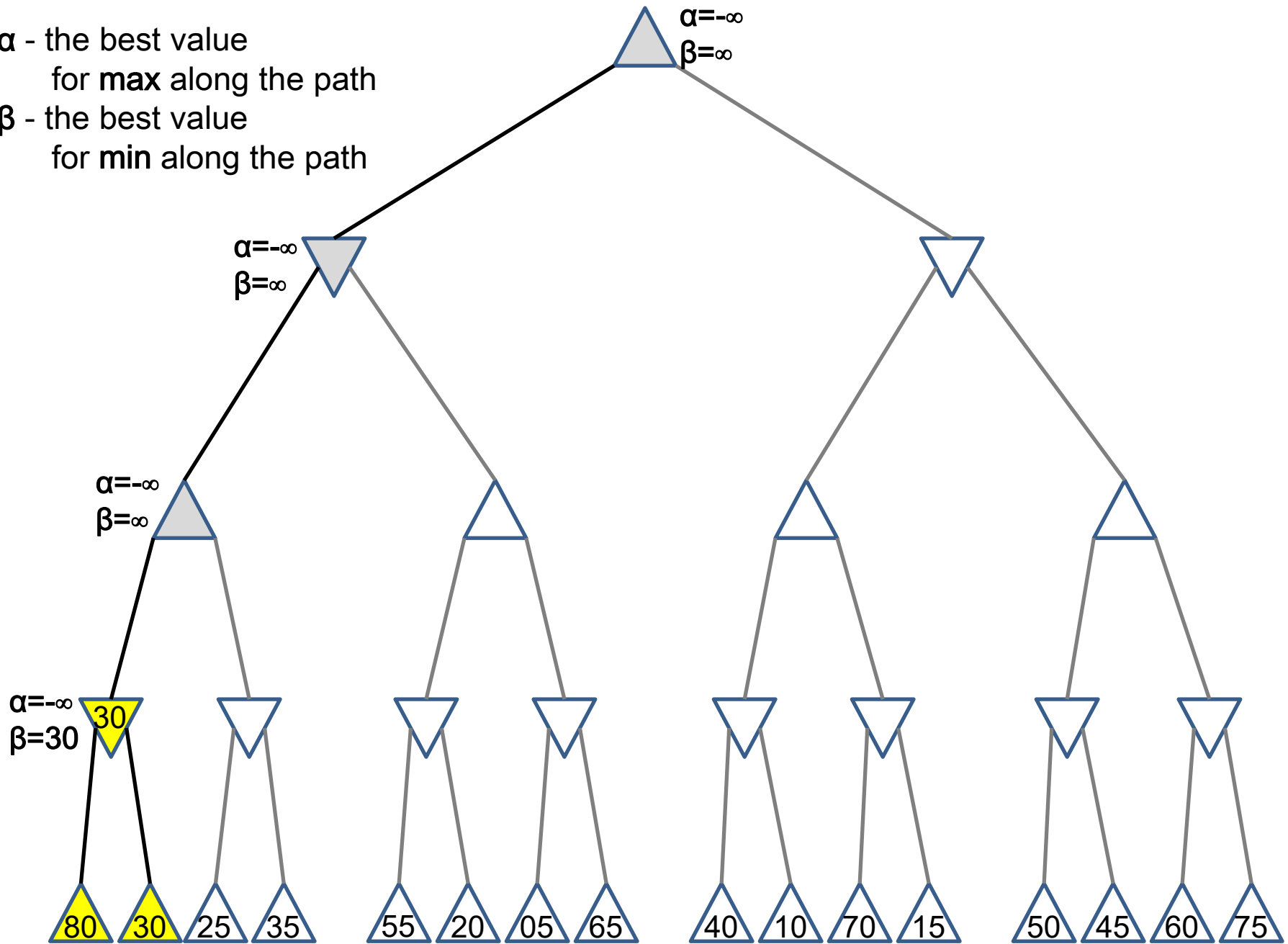


α - the best value
for **max** along the path
 β - the best value
for **min** along the path

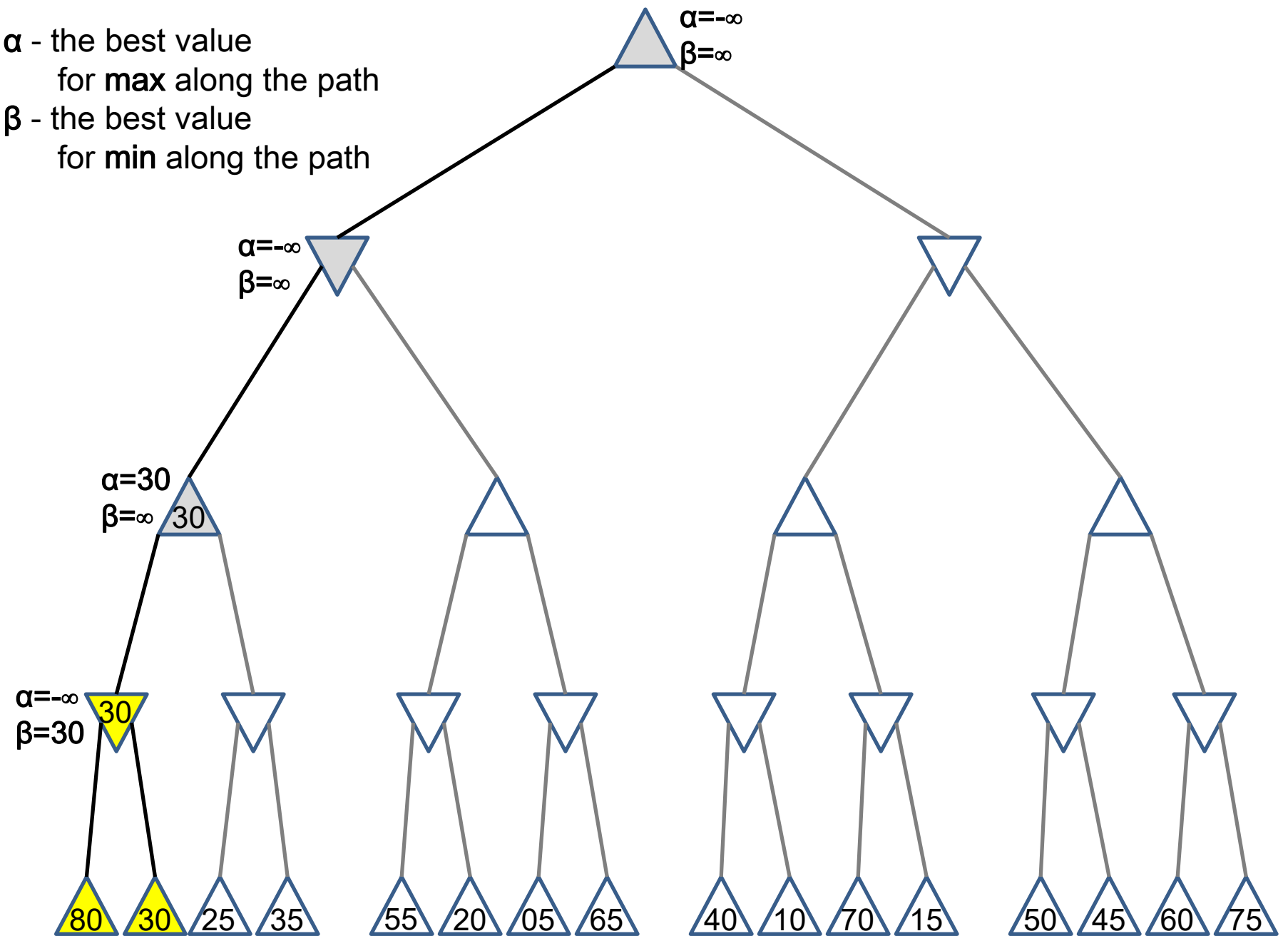


α - the best value
for **max** along the path

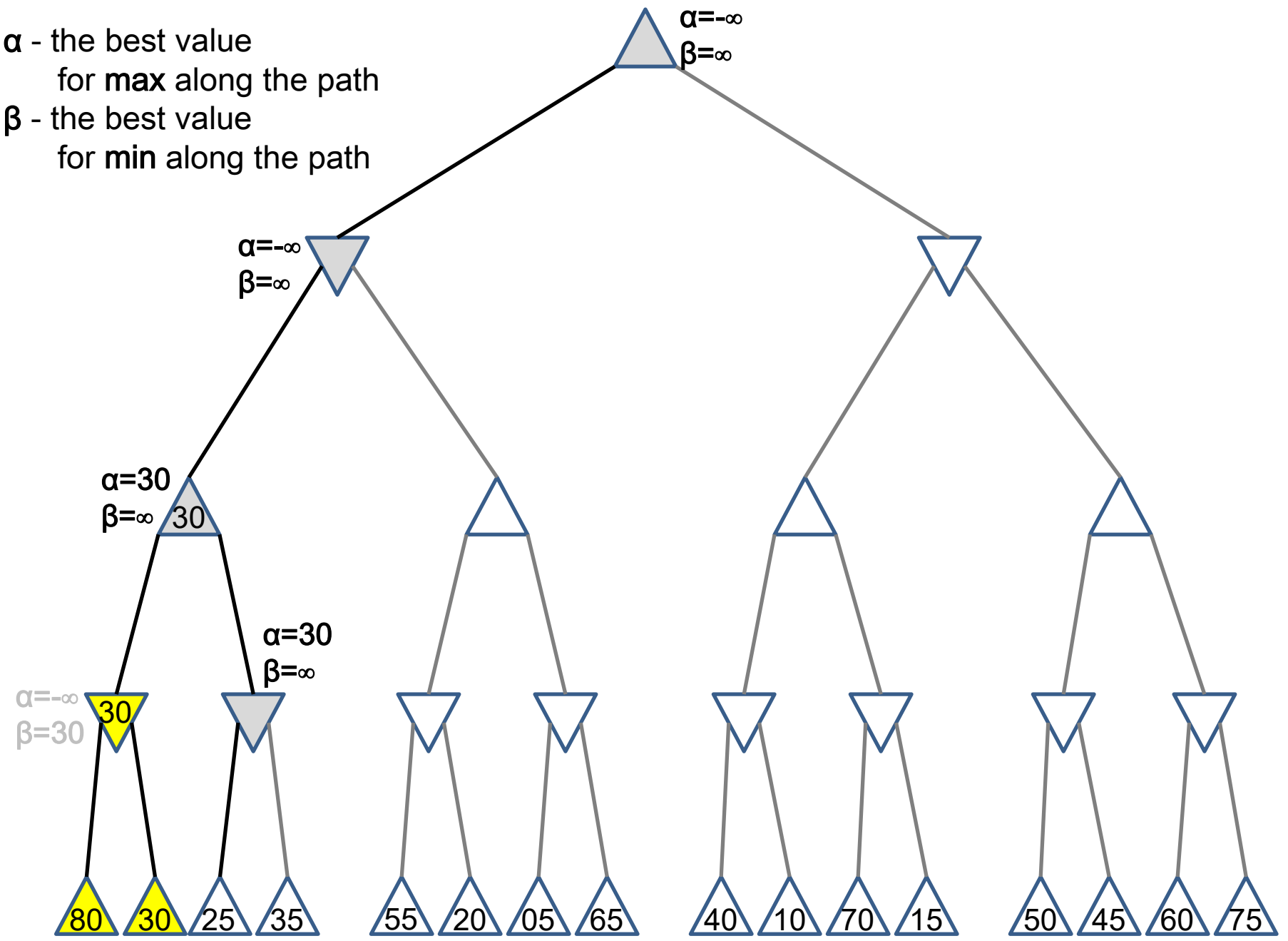
β - the best value
for **min** along the path



α - the best value
for **max** along the path
 β - the best value
for **min** along the path

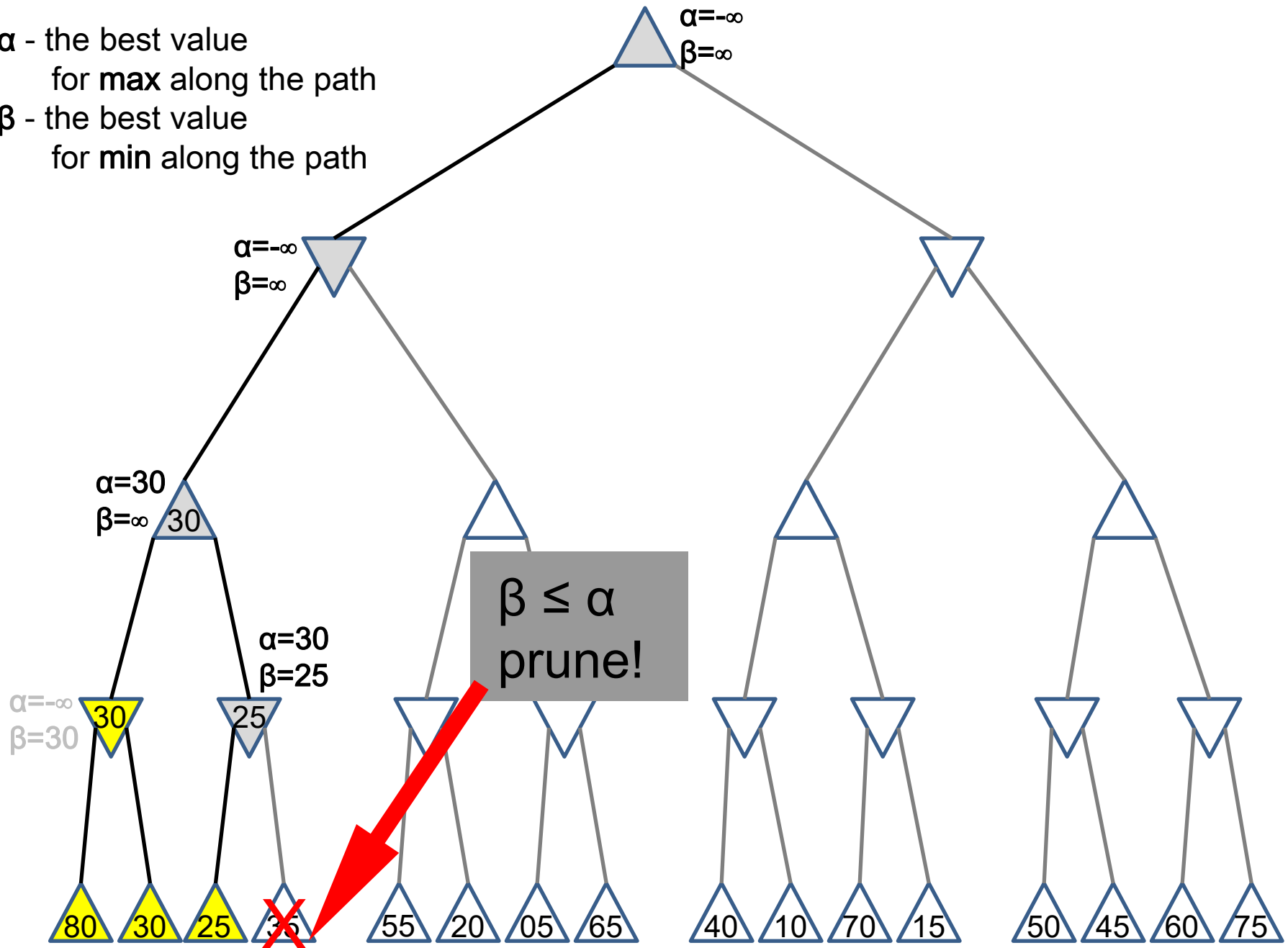


α - the best value
for **max** along the path
 β - the best value
for **min** along the path



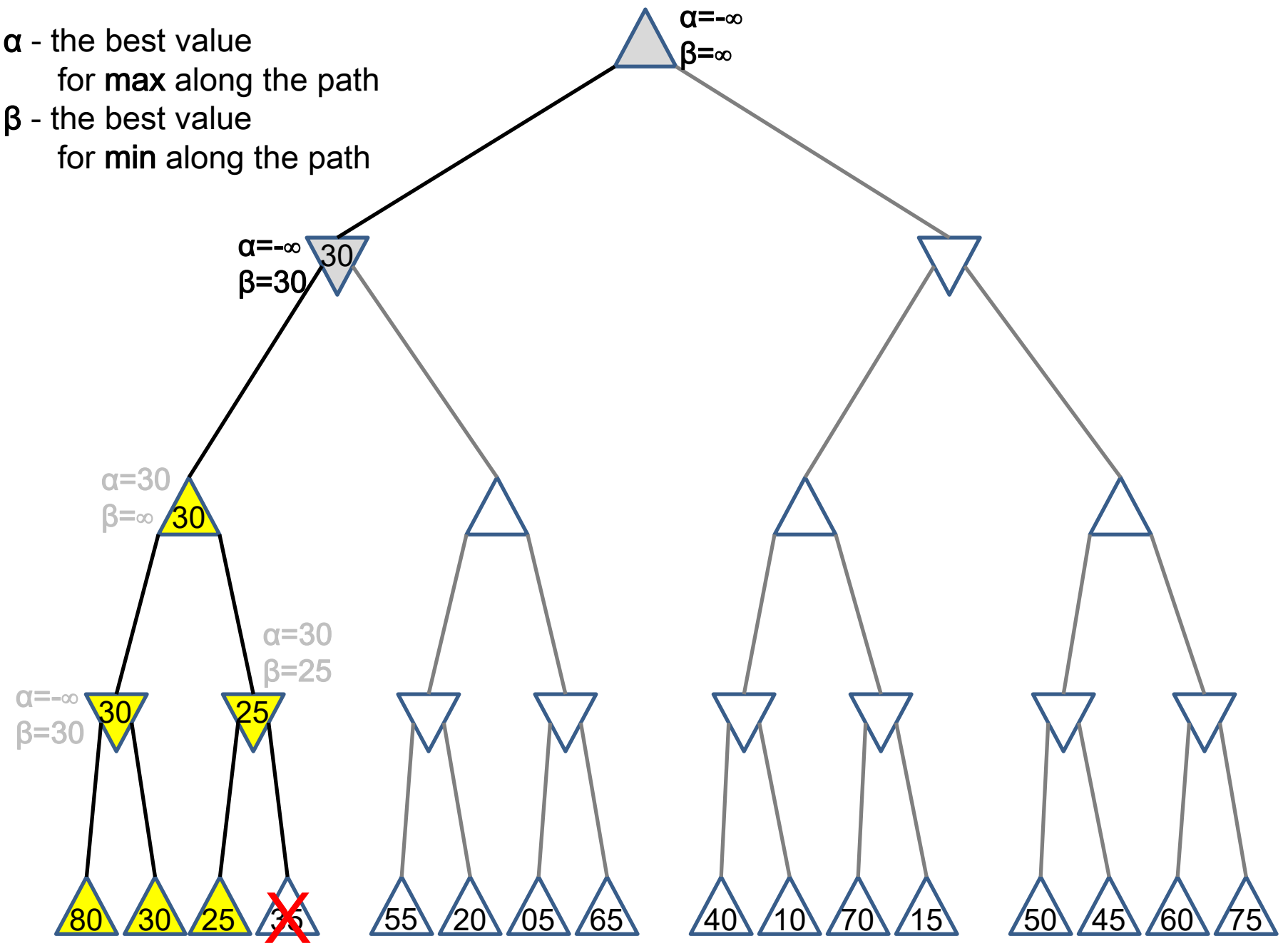
α - the best value
for **max** along the path

β - the best value
for **min** along the path



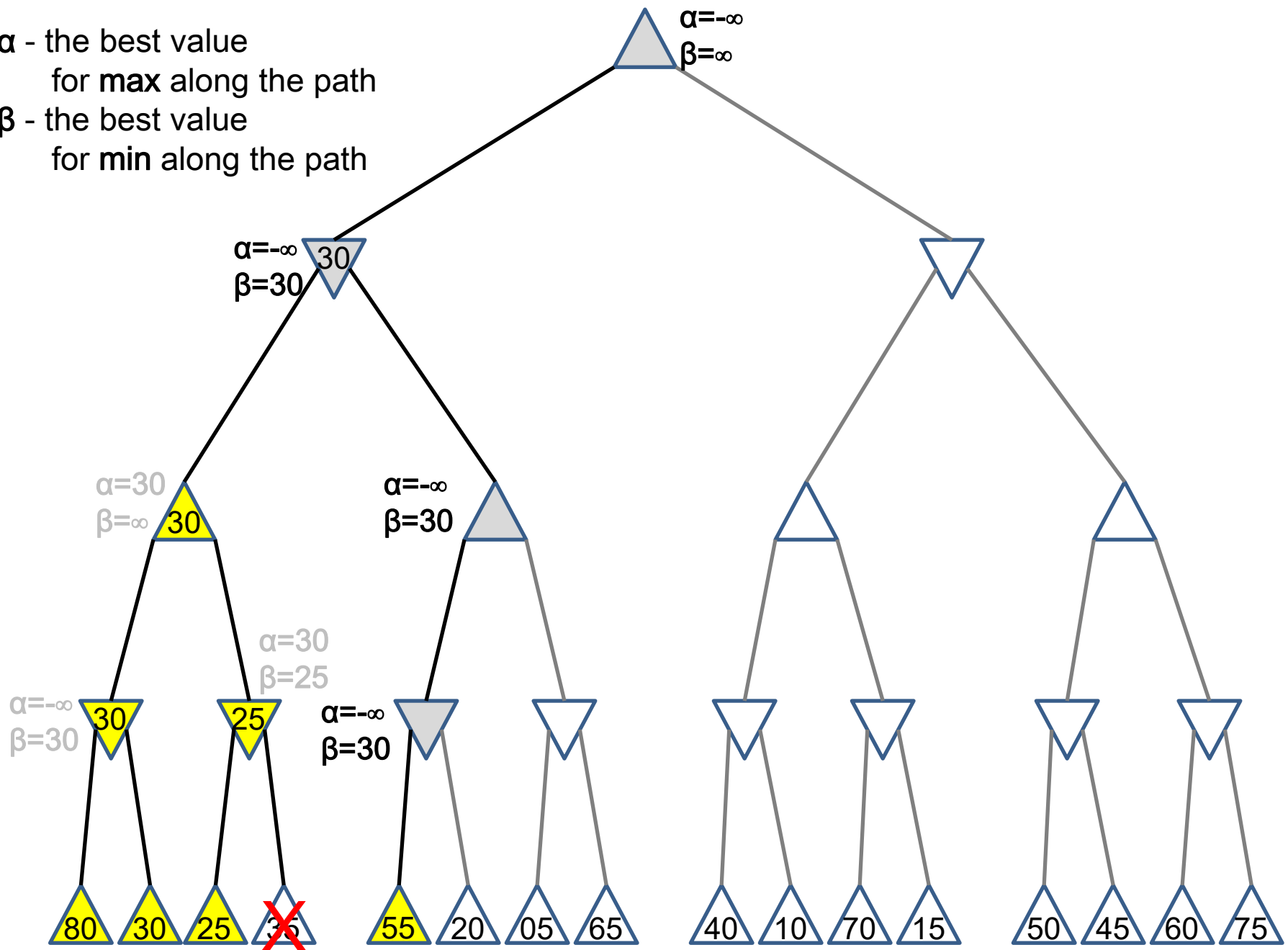
α - the best value
for **max** along the path

β - the best value
for **min** along the path



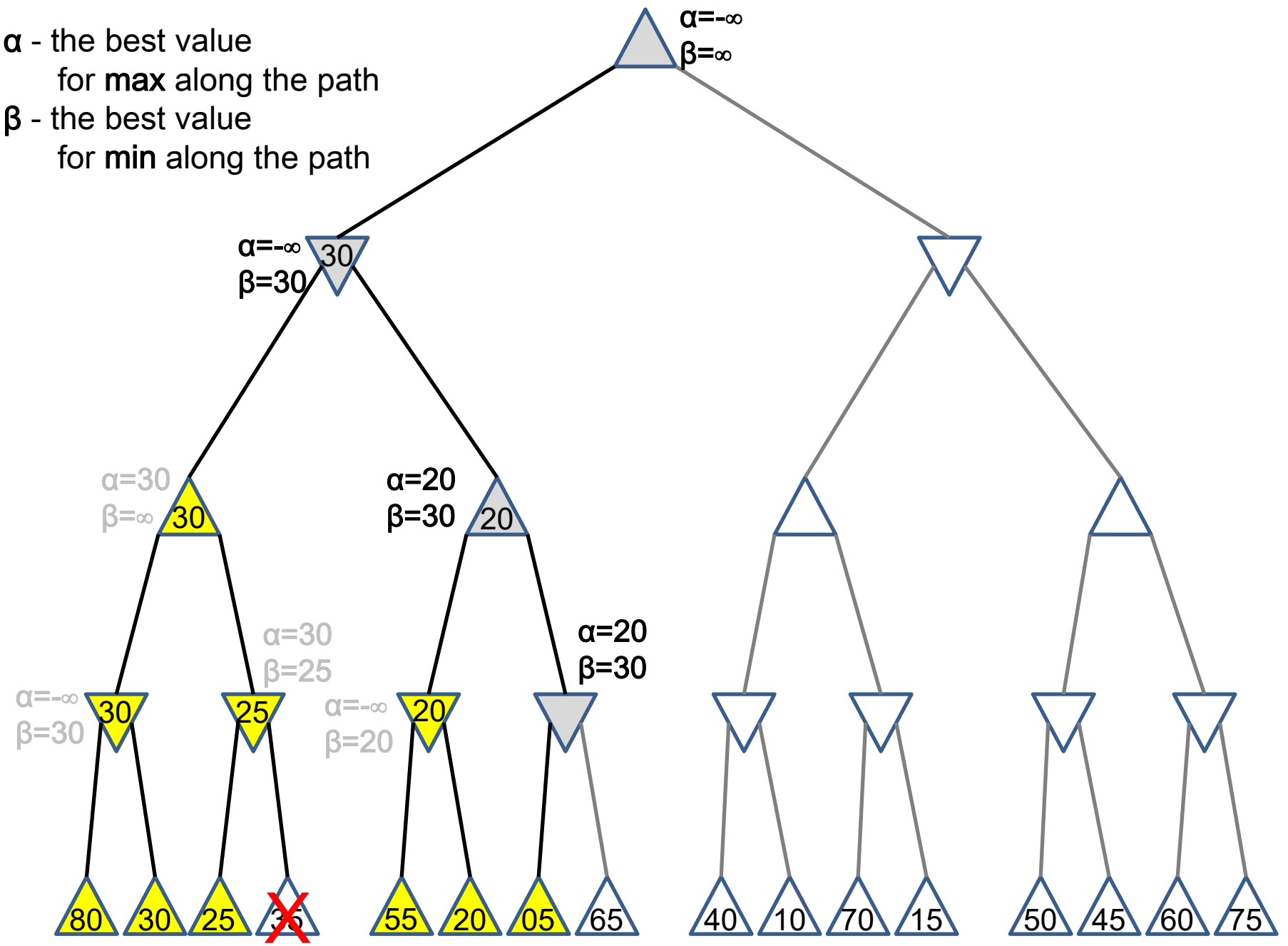
α - the best value
for **max** along the path

β - the best value
for **min** along the path



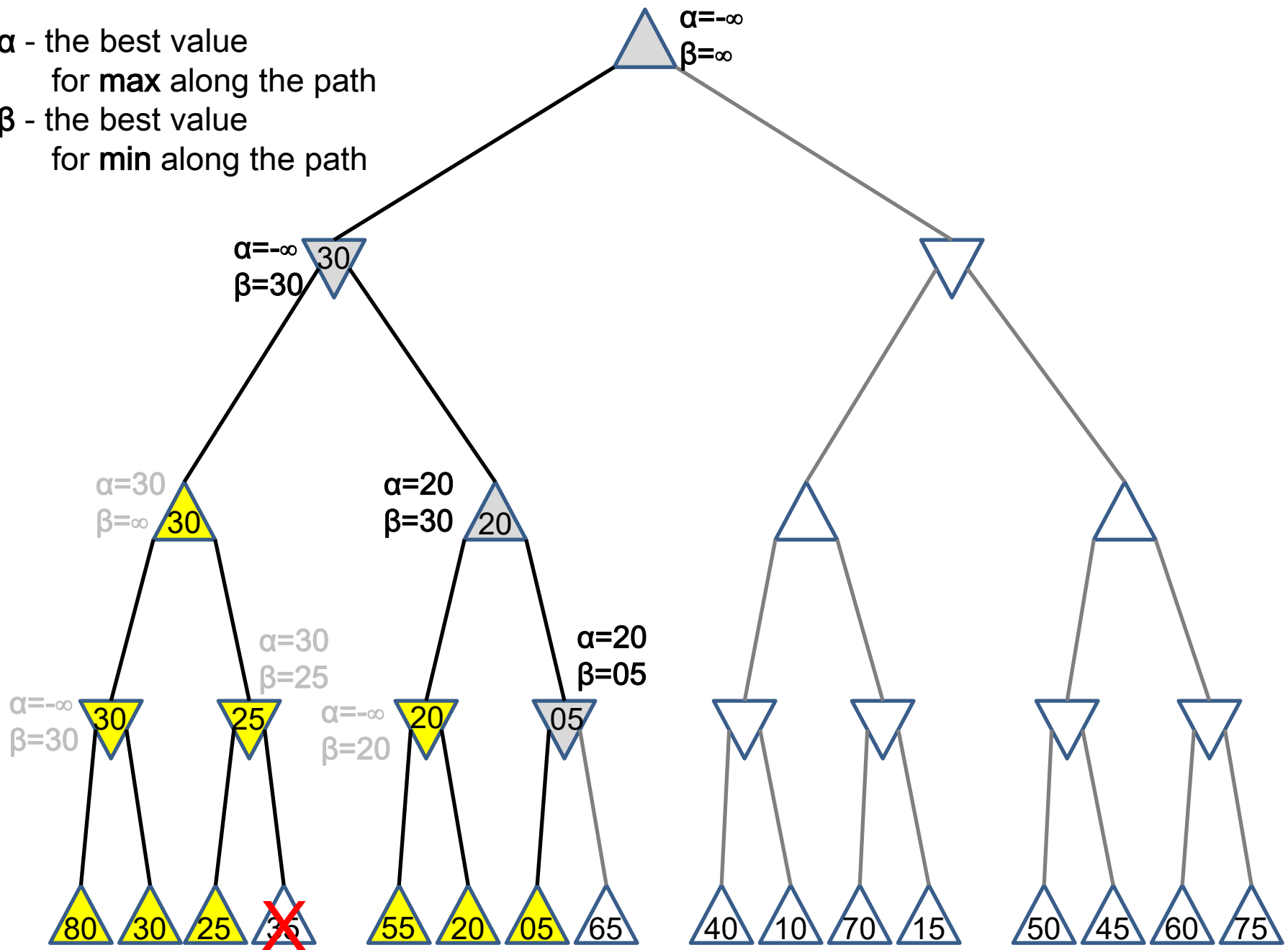
α - the best value
for **max** along the path

β - the best value
for **min** along the path

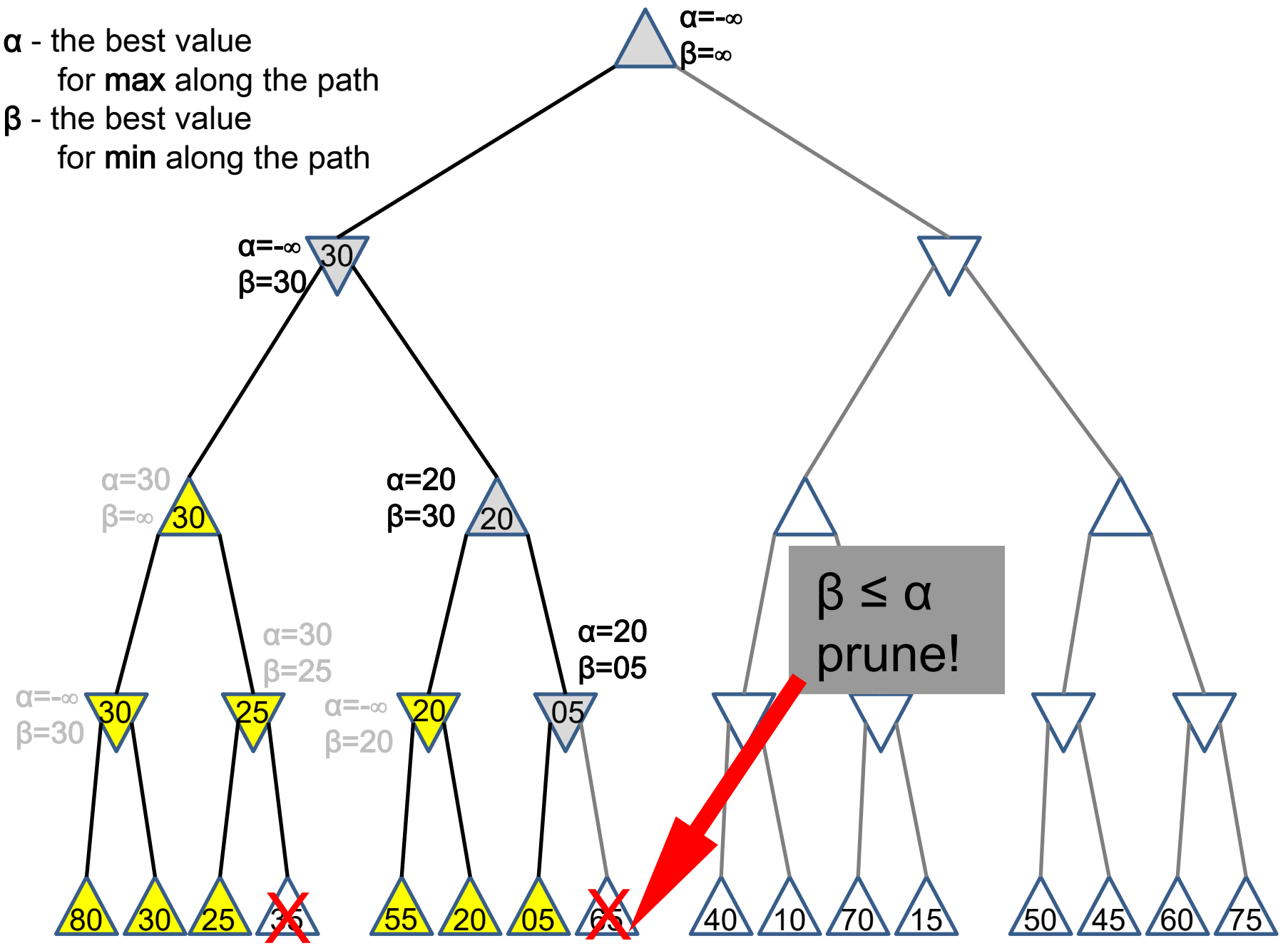


α - the best value
for **max** along the path

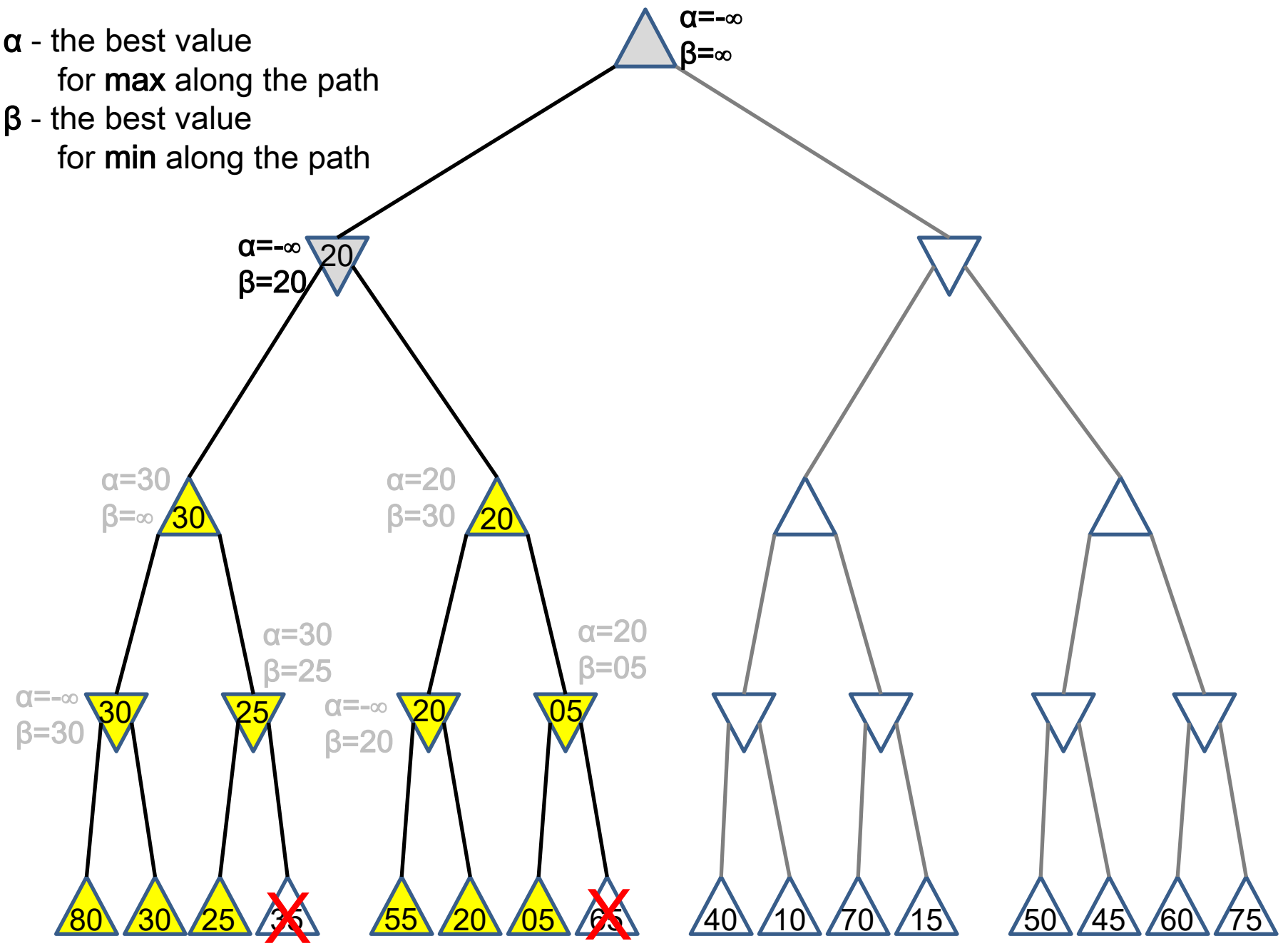
β - the best value
for **min** along the path



α - the best value
for **max** along the path
 β - the best value
for **min** along the path

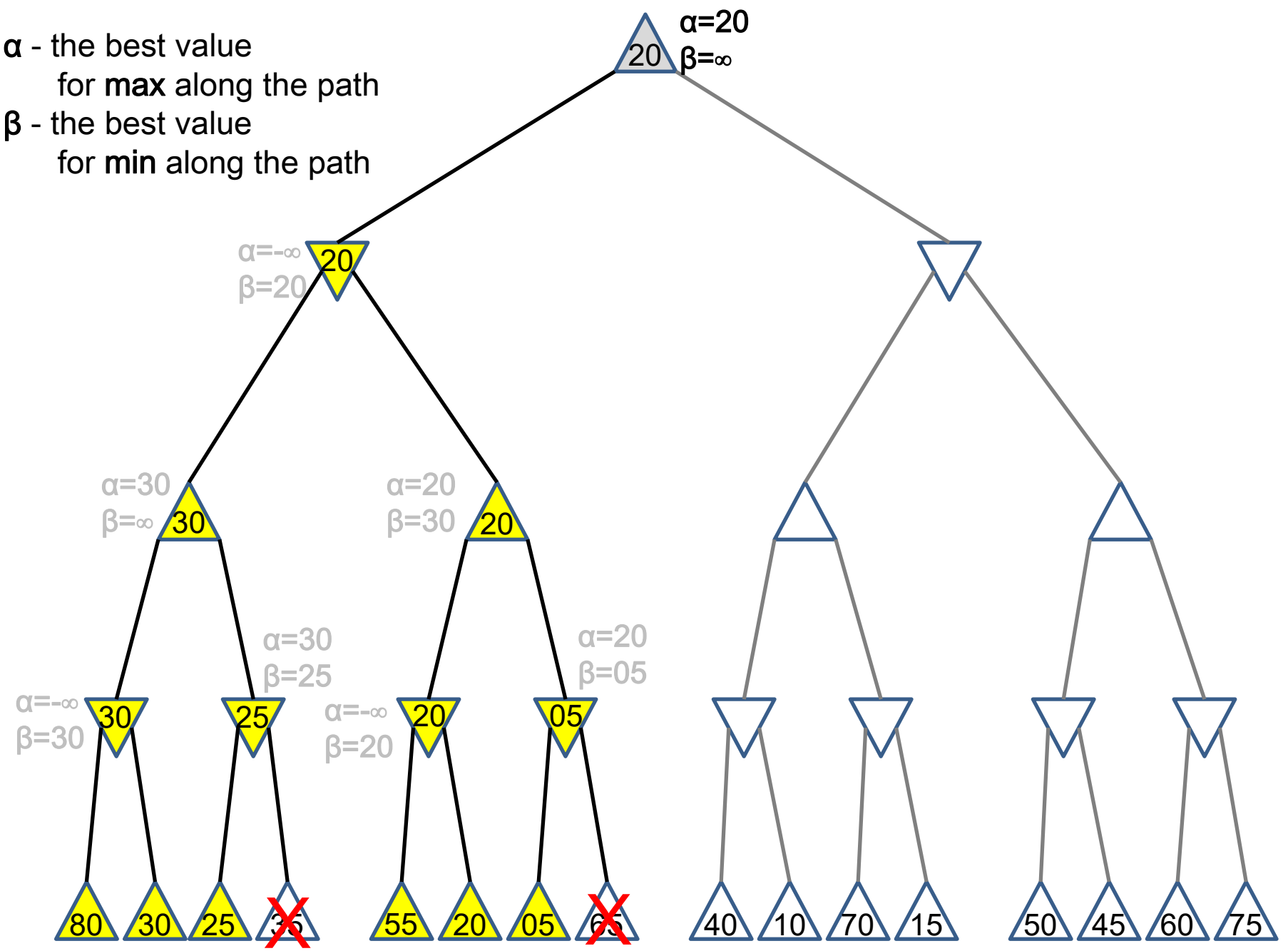


α - the best value
 for **max** along the path
 β - the best value
 for **min** along the path

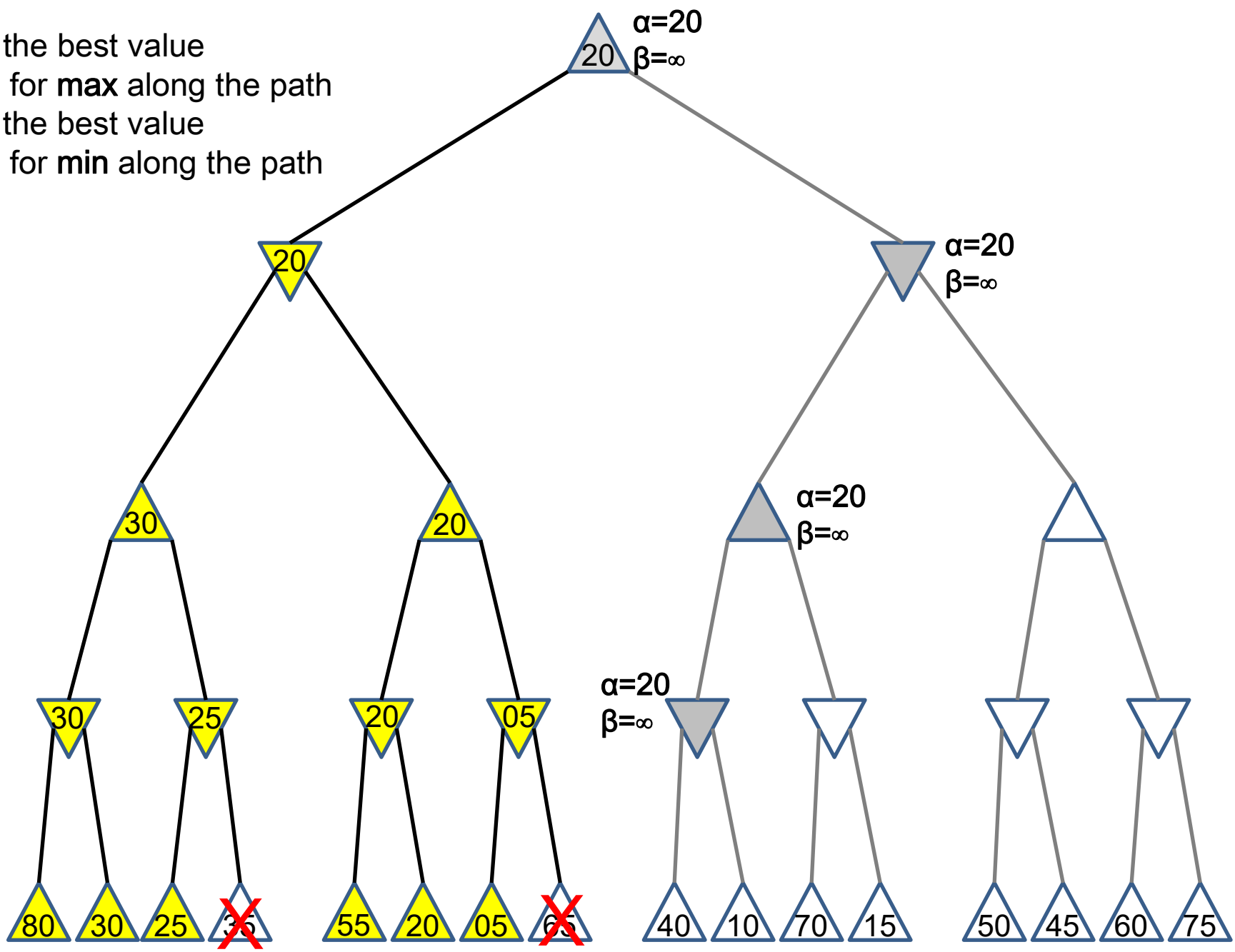


α - the best value
for **max** along the path

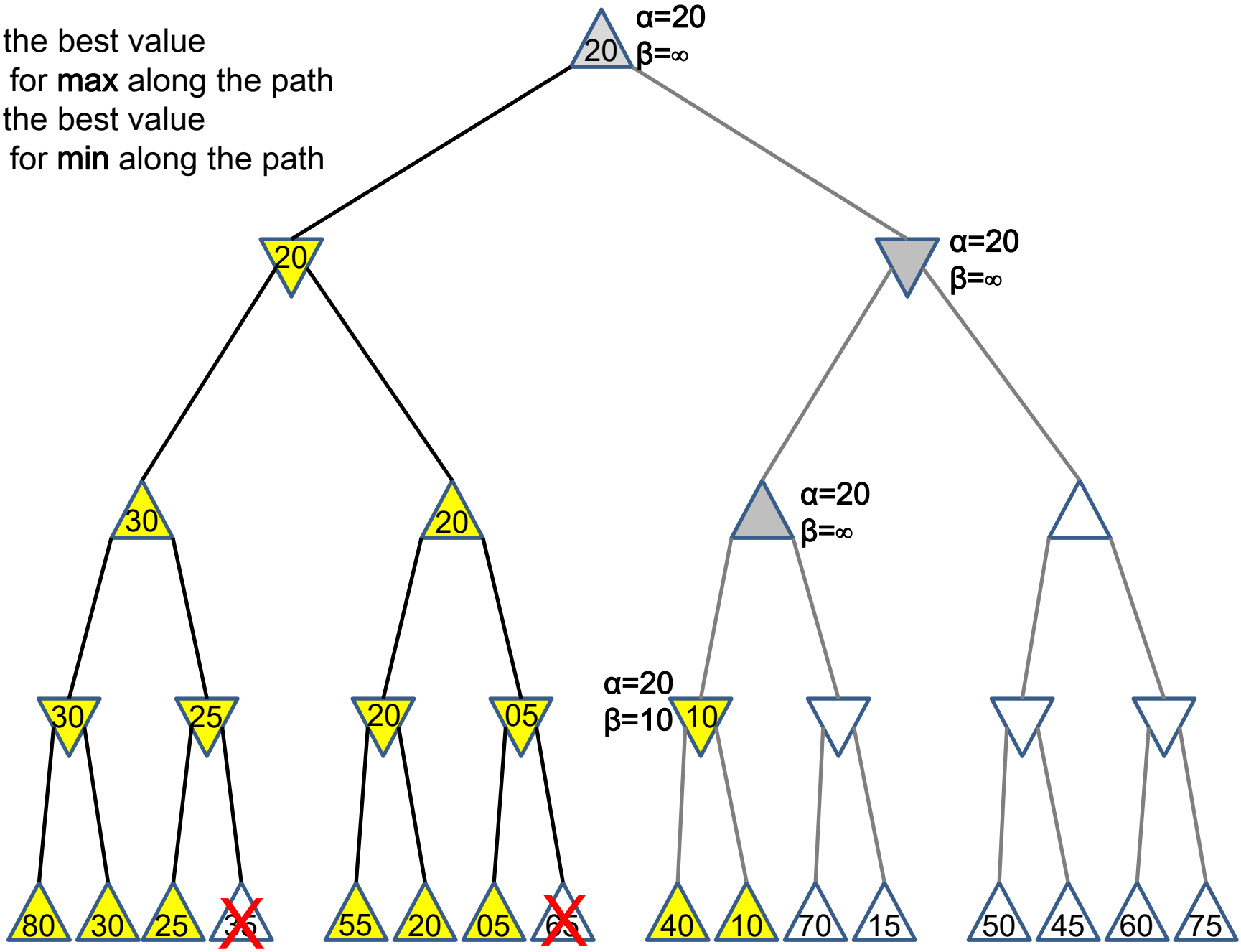
β - the best value
for **min** along the path



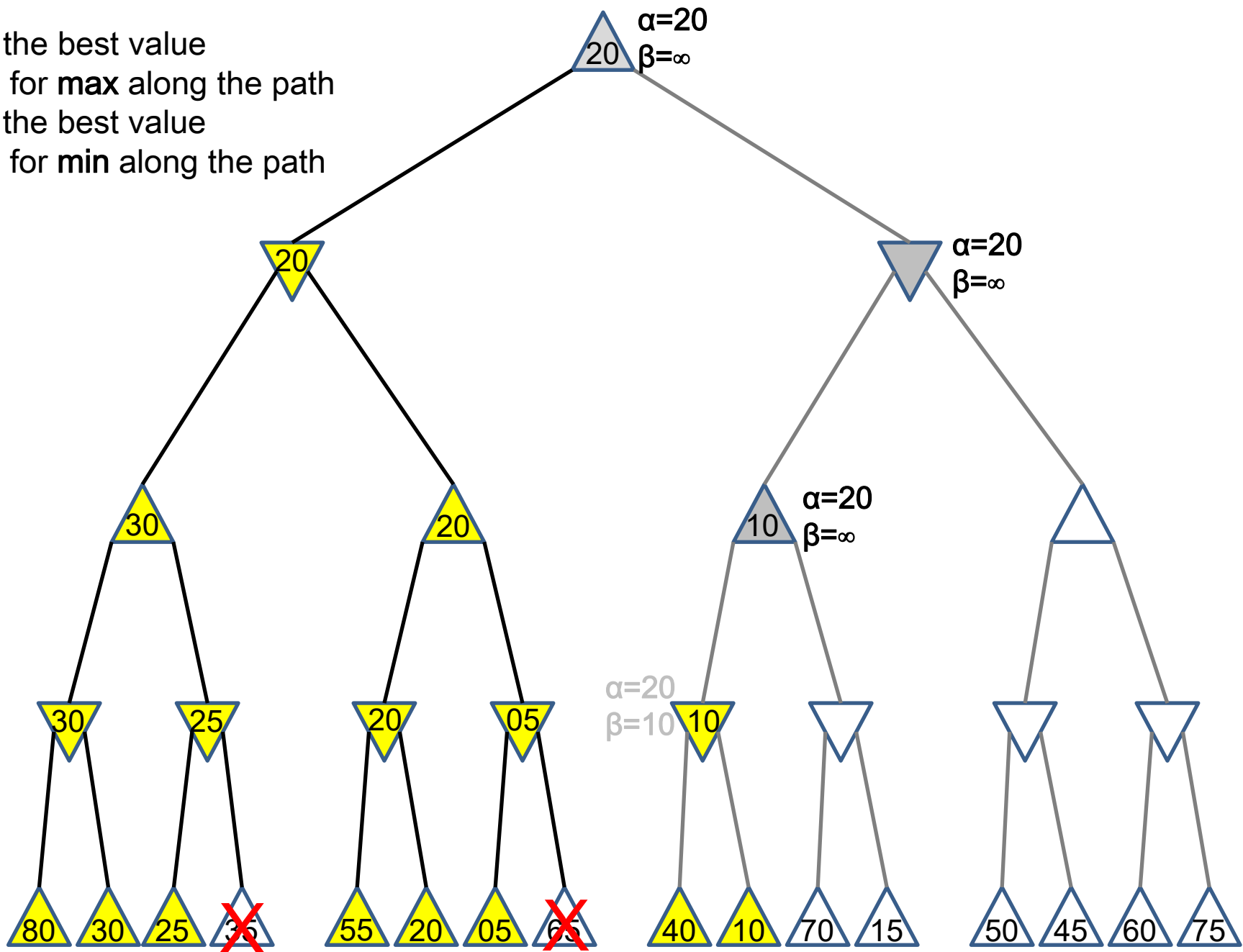
α - the best value
for **max** along the path
 β - the best value
for **min** along the path



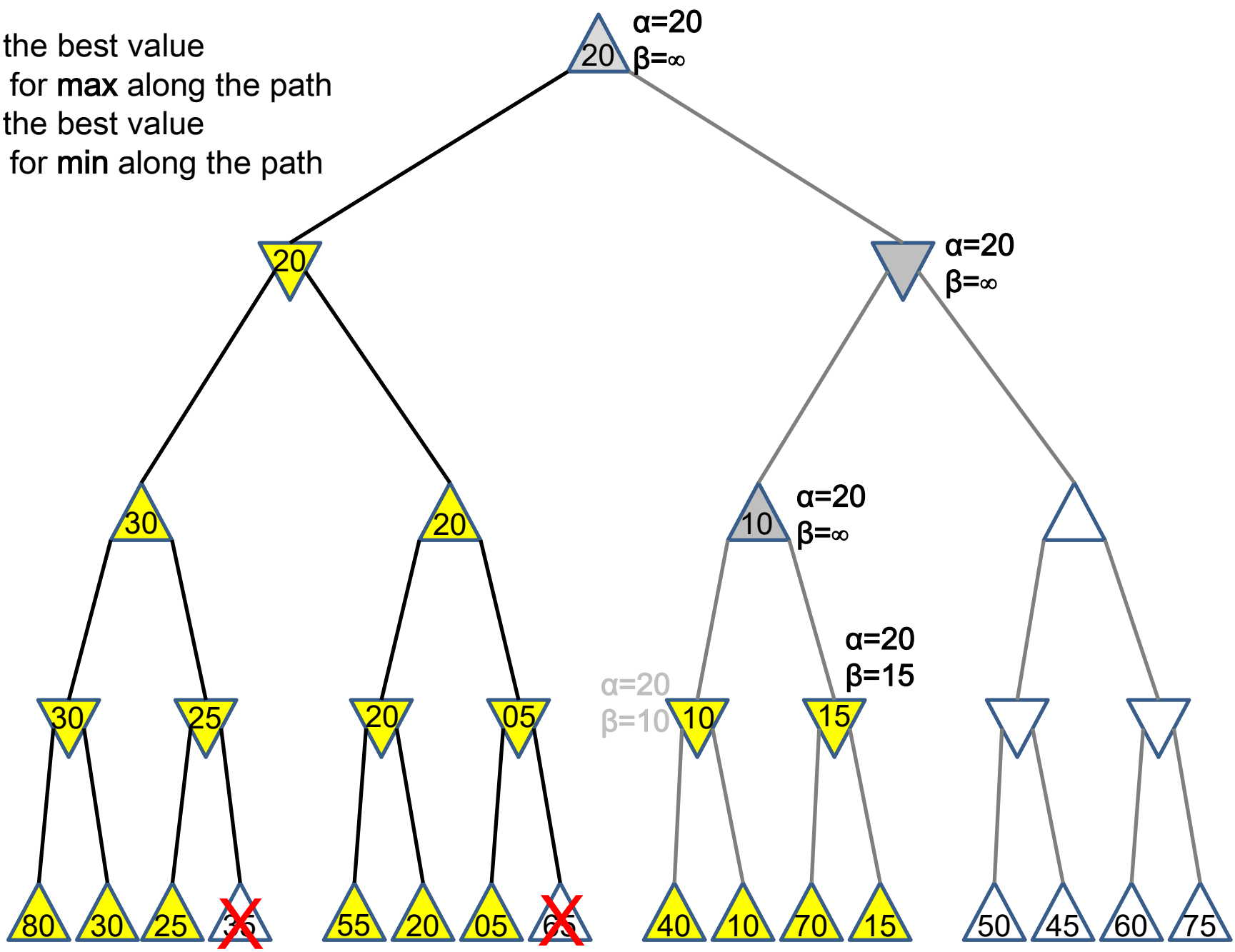
α - the best value
for **max** along the path
 β - the best value
for **min** along the path



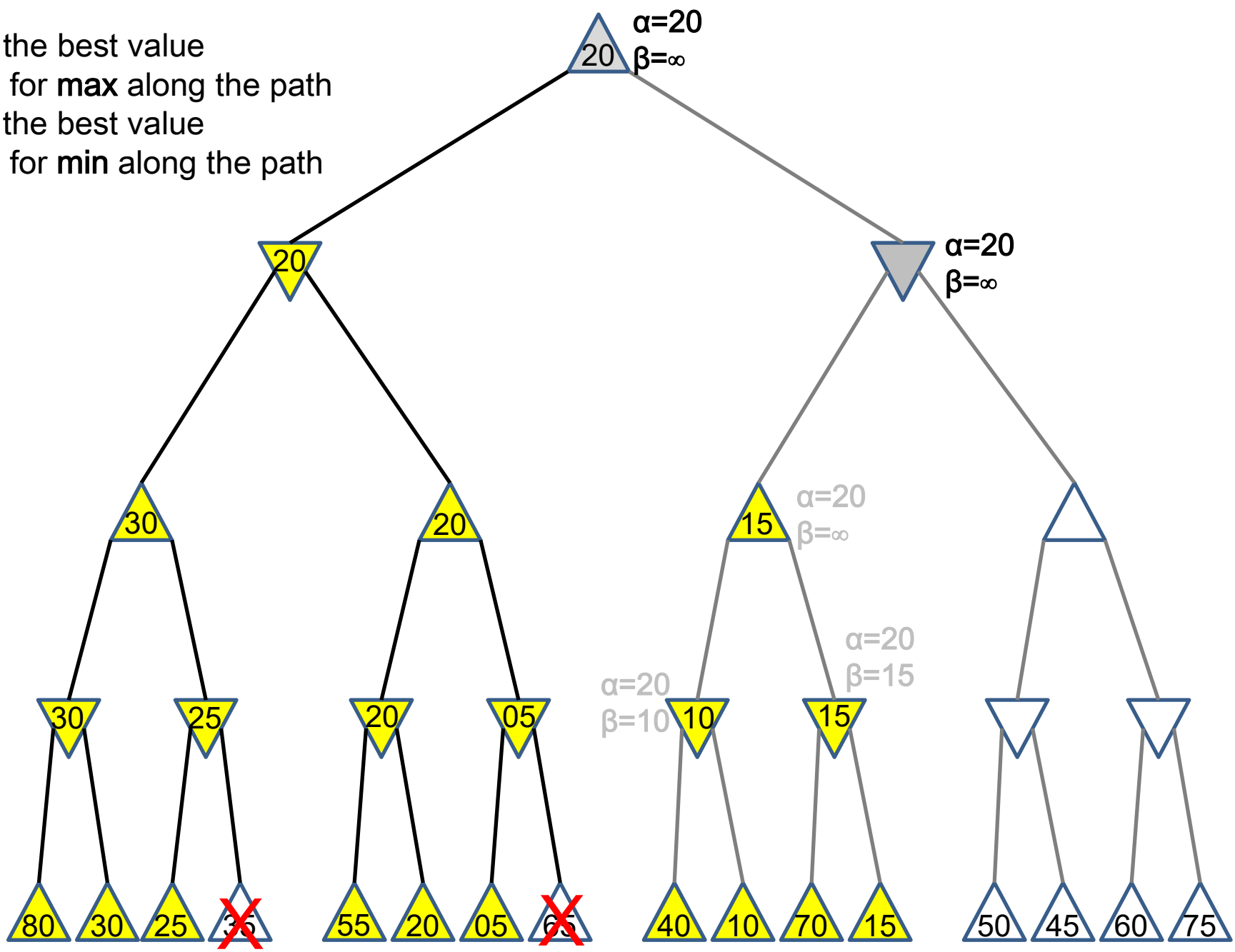
- α - the best value for **max** along the path
- β - the best value for **min** along the path



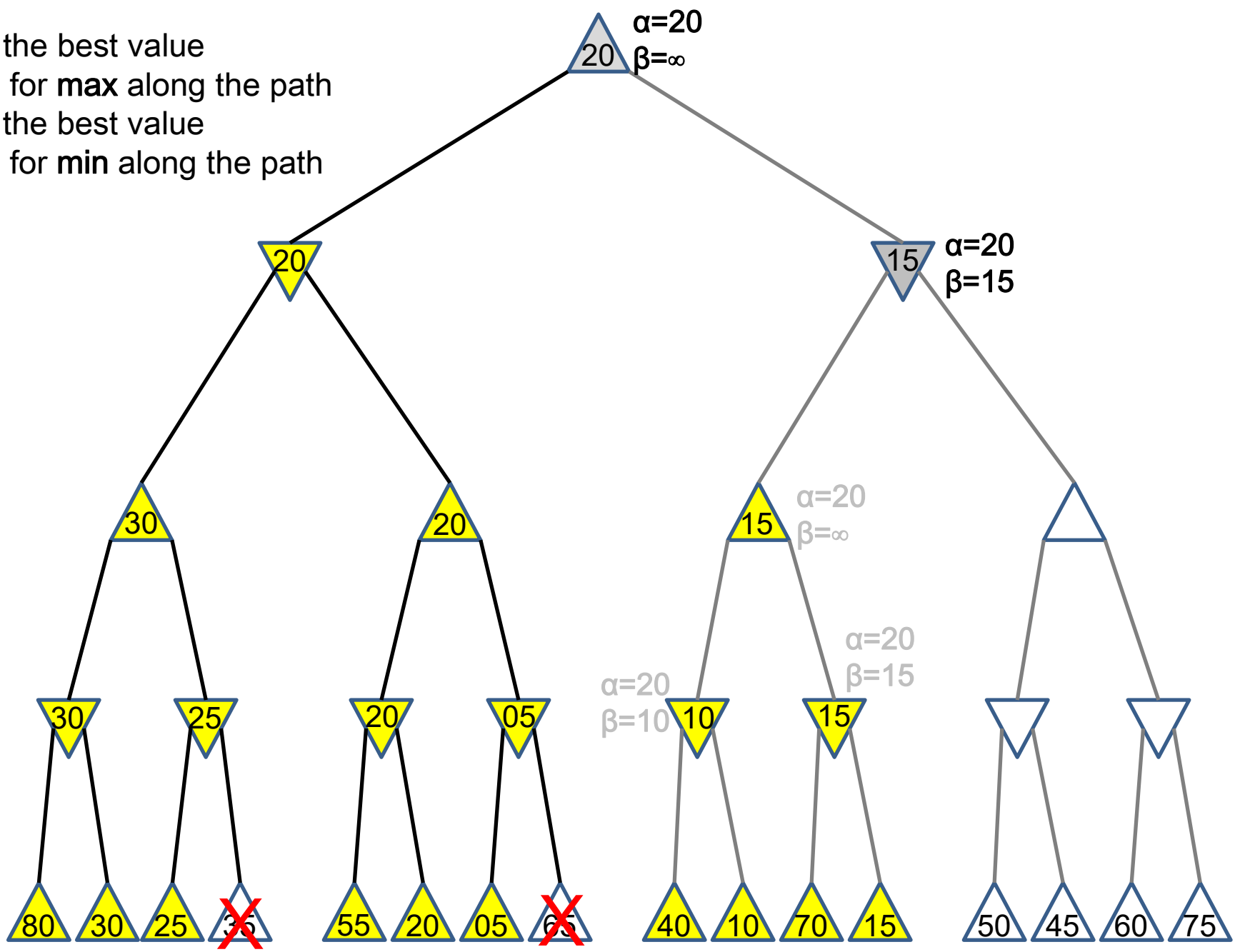
α - the best value
for **max** along the path
 β - the best value
for **min** along the path



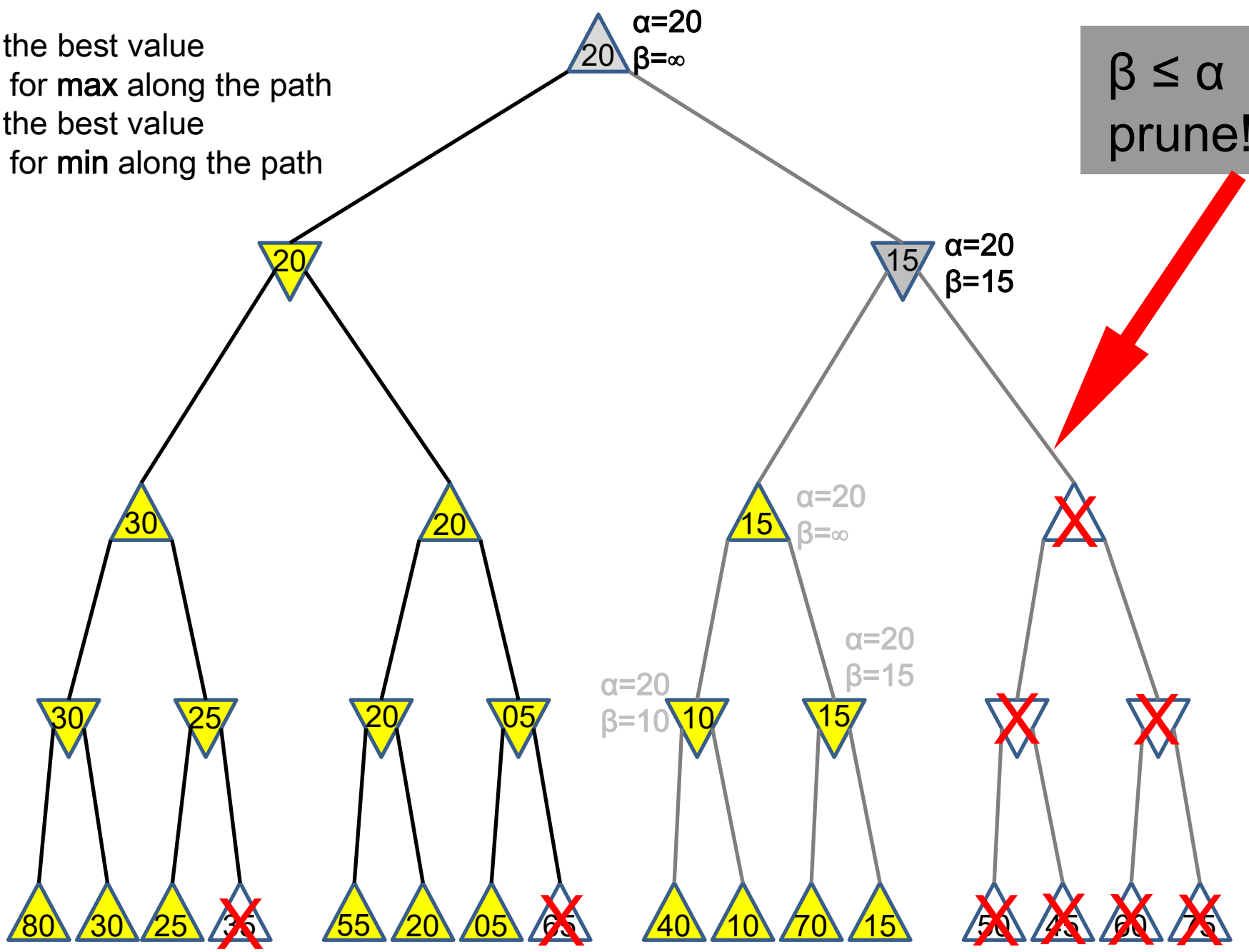
α - the best value
for **max** along the path
 β - the best value
for **min** along the path



α - the best value
for **max** along the path
 β - the best value
for **min** along the path

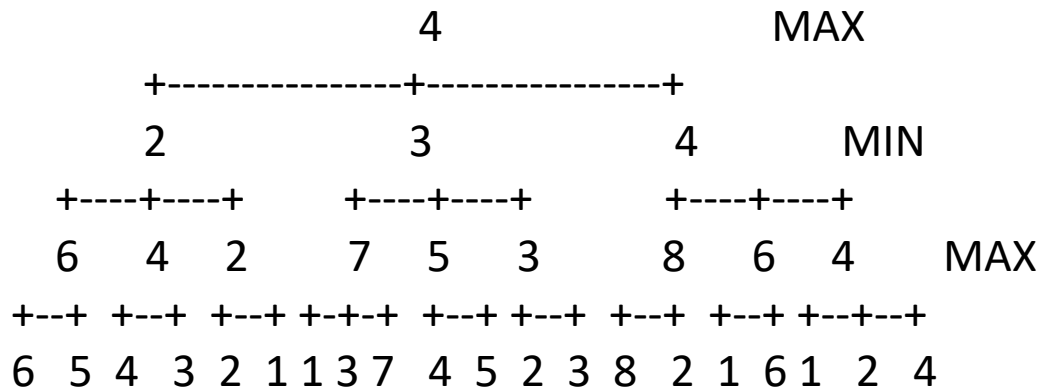


α - the best value
for **max** along the path
 β - the best value
for **min** along the path

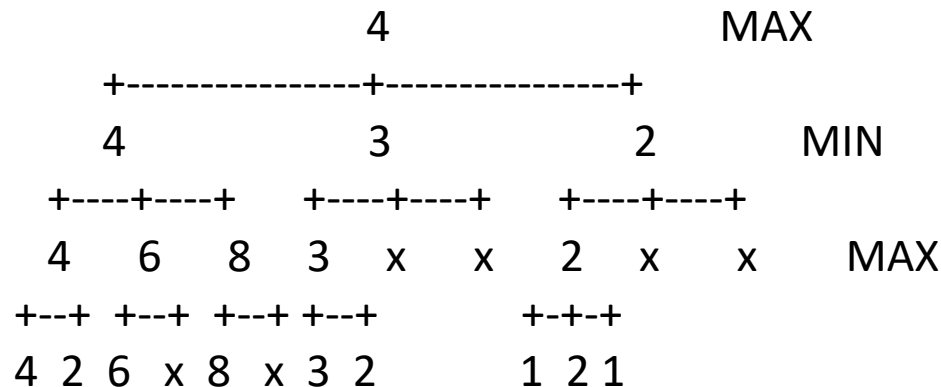


Bad and Good Cases for Alpha-Beta Pruning

- Bad: Worst moves encountered first



- Good: Good moves ordered first



- If we can order moves, we can get more benefit from alpha-beta pruning

Properties of α - β

- Pruning **does not** affect final result. This means that it **gets the exact same result as does full minimax**.
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
→ **doubles** depth of search
- A simple example of reasoning about 'which computations are relevant' (a form of **metareasoning**)

Why $O(b^{m/2})$?

Let $T(m)$ be time complexity of search for depth m

Normally:

$$T(m) = b.T(m-1) + c \Rightarrow T(m) = O(b^m)$$

With ideal α - β pruning:

$$T(m) = T(m-1) + (b-1)T(m-2) + c \Rightarrow T(m) = O(b^{m/2})$$

Node Ordering

Iterative deepening search

Use evaluations of the previous search for order

Also helps in returning a move in given time

Good Enough?

- Chess:

- branching factor $b \approx 35$

- game length $m \approx 100$

- search space $b^{m/2} \approx 35^{50} \approx 10^{77}$

**The universe
can play chess
- can we?**

- The Universe:

- number of atoms $\approx 10^{78}$

- age $\approx 10^{18}$ seconds

- 10^8 moves/sec $\times 10^{78} \times 10^{18} = 10^{104}$

Cutting off Search

MinimaxCutoff is identical to *MinimaxValue* except

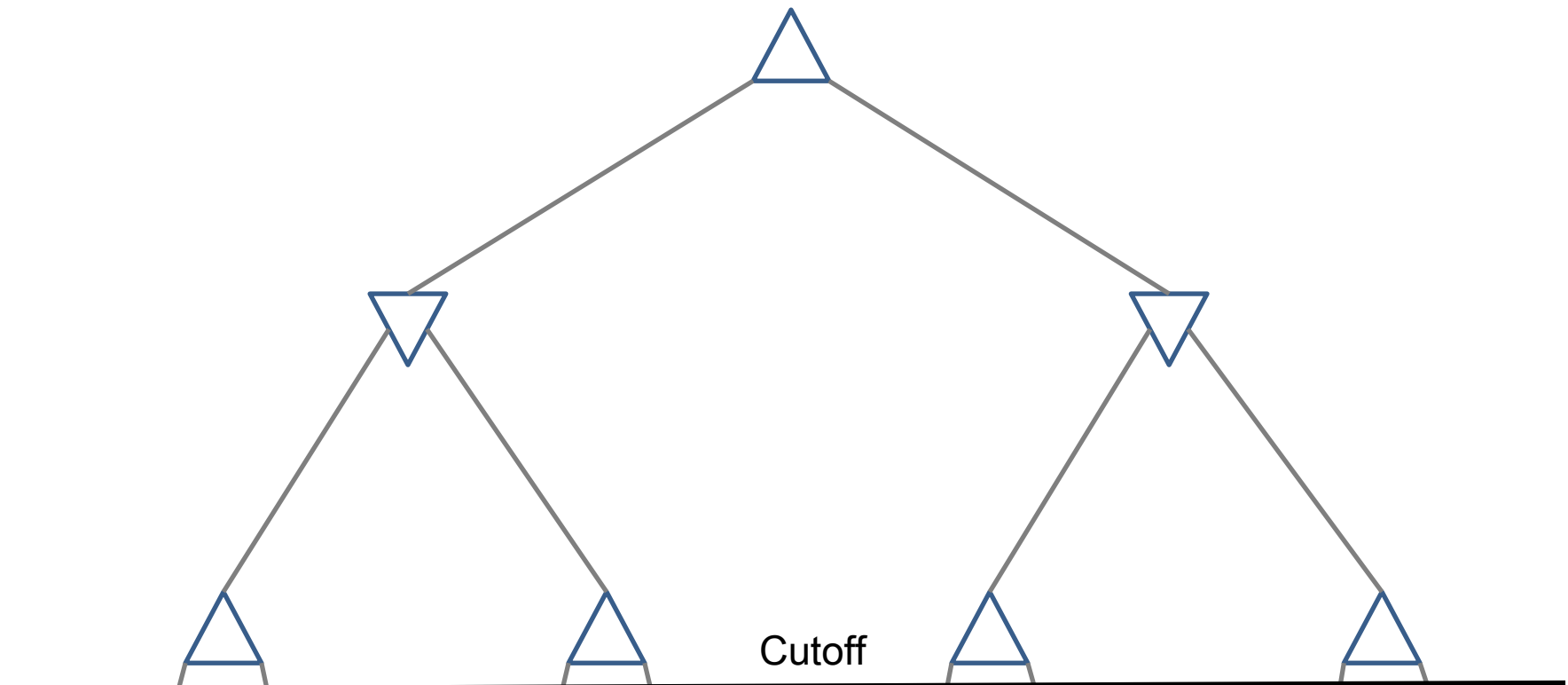
1. *Terminal?* is replaced by *Cutoff?*
2. *Utility* is replaced by *Eval*

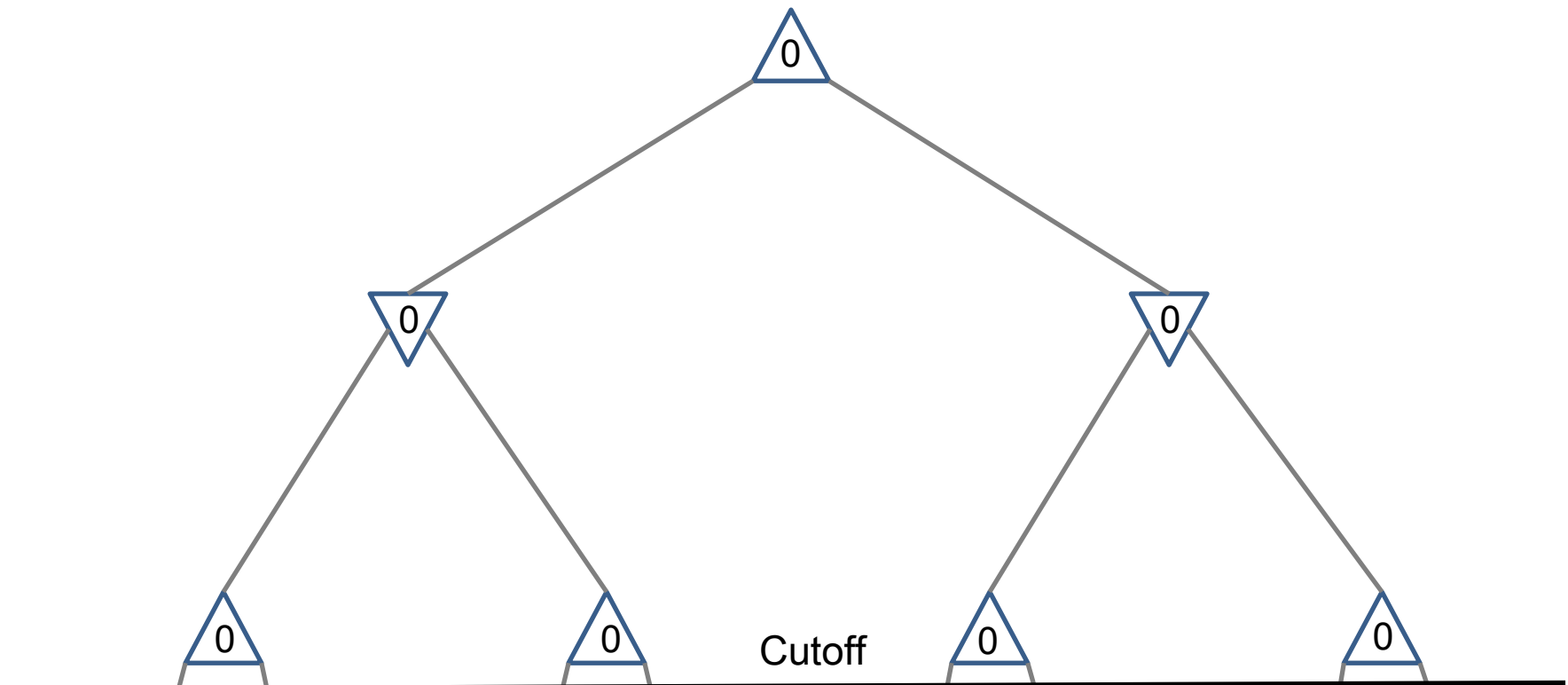
Does it work in practice?

$$b^m = 10^6, b=35 \rightarrow m=4$$

4-ply lookahead is a hopeless chess player!

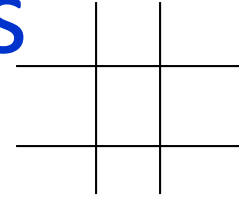
- 4-ply \approx human novice
- 8-ply \approx typical PC, human master
- 12-ply \approx Deep Blue, Kasparov





Evaluation Functions

Tic Tac Toe



- Let p be a position in the game
- Define the utility function $f(p)$ by
 - $f(p) =$
 - largest positive number if p is a win for computer
 - smallest negative number if p is a win for opponent
 - $RCDC - RCDO$
 - where $RCDC$ is number of rows, columns and diagonals in which computer could still win
 - and $RCDO$ is number of rows, columns and diagonals in which opponent could still win.

Sample Evaluations

- X = Computer; O = Opponent

	O	
	X	

O	O	X
X	X	

	X	O
rows		
cols		
diags		

	X	O
rows		
cols		
diags		

Evaluation functions

- For chess/checkers, typically **linear** weighted sum of **features**

$$Eval(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_m f_m(s)$$

e.g., $w_1 = 9$ with

$f_1(s) = (\text{number of white queens}) - (\text{number of black queens}),$
etc.

Example: Samuel's Checker-Playing Program

- It uses a linear evaluation function

$$f(n) = w_1 f_1(n) + w_2 f_2(n) + \dots + w_m f_m(n)$$

For example: $f = 6K + 4M + U$

- K = King Advantage
- M = Man Advantage
- U = Undenied Mobility Advantage (number of moves that Max where Min has no jump moves)

Samuel's Checker Player

- In learning mode
 - Computer acts as 2 players: **A** and **B**
 - **A** adjusts its coefficients after every move
 - **B** uses the static utility function
 - If **A** wins, its function is given to **B**

Samuel's Checker Player

- How does A change its function?

Coefficient replacement

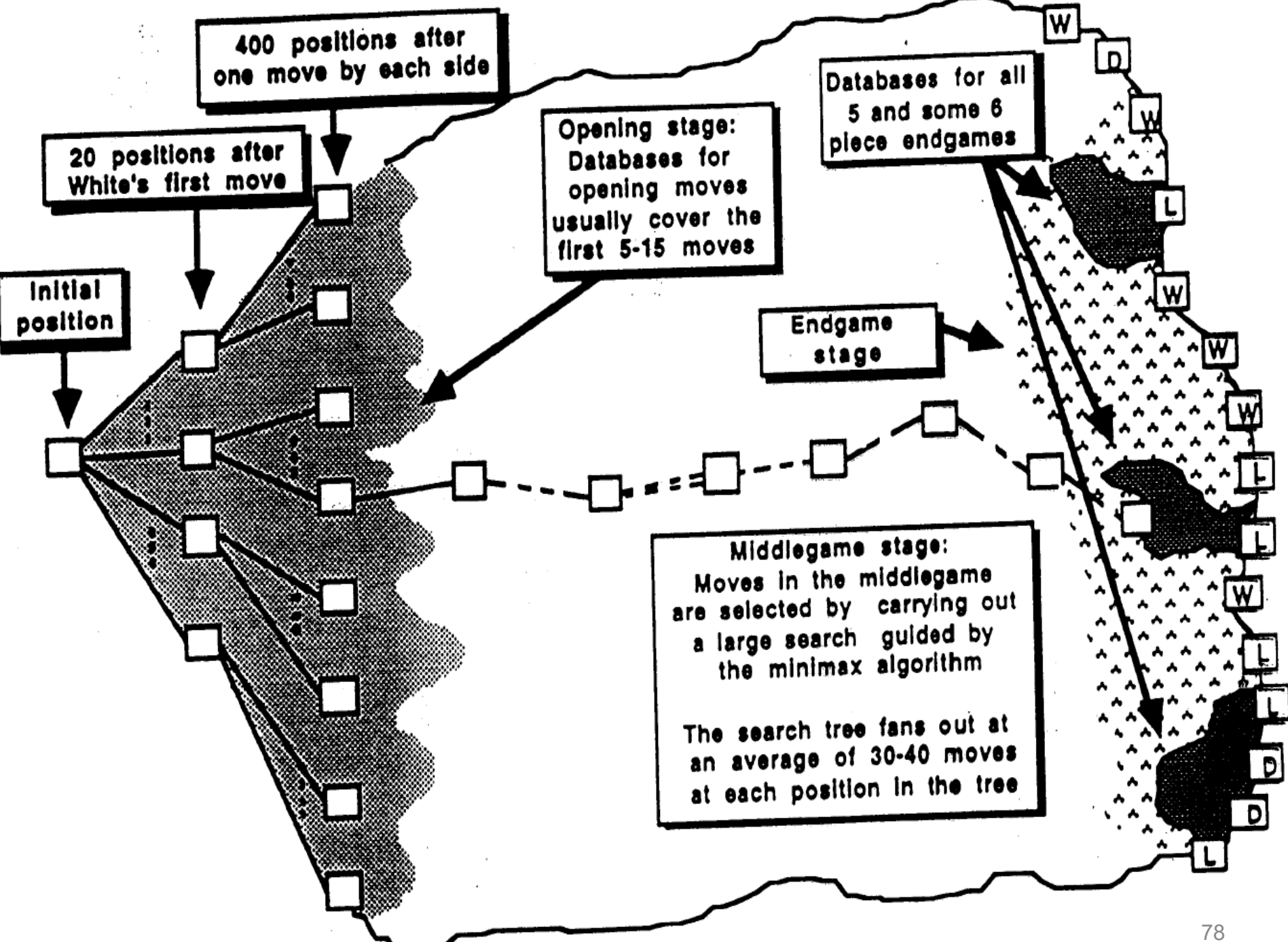
$\Delta(\text{node}) = \text{backed-up value}(\text{node}) - \text{initial value}(\text{node})$

if $\Delta > 0$ then terms that contributed **positively** are given more weight and terms that contributed negatively get less weight

if $\Delta < 0$ then terms that contributed **negatively** are given more weight and terms that contributed positively get less weight

Chess: Rich history of cumulative ideas

- Minimax search, evaluation function learning (1950).
- Alpha-Beta search (1966).
- Transposition Tables (1967).
- Iterative deepening DFS (1975).
- End game data bases ,singular extensions(1977, 1980)
- Parallel search and evaluation(1983 ,1985)
- Circuitry (1987)

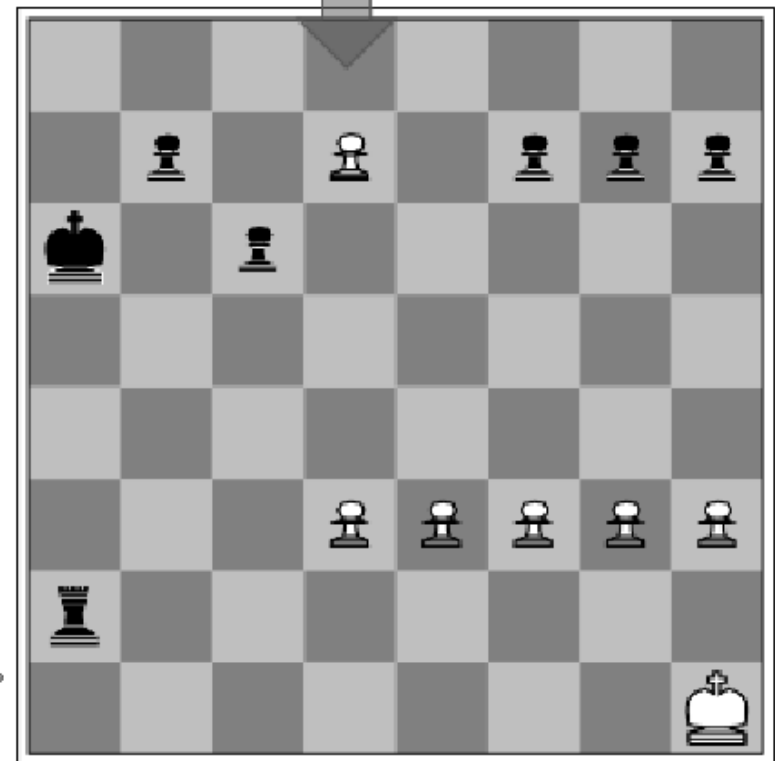


Problem with fixed depth Searches

if we only
it may be
catastroph
sequence
make any

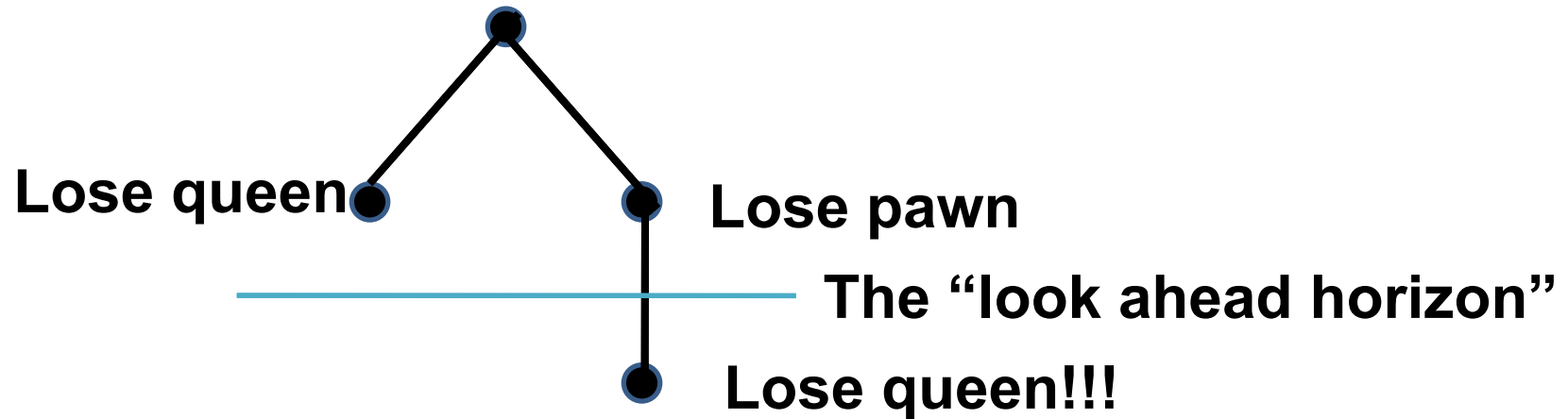
also works
(good mov

Fixed depth search
thinks it can avoid
the queening move



Black to move

Problems with a fixed ply: The Horizon Effect



- Inevitable losses are postponed
- Unachievable goals appear achievable
- Short-term gains mask unavoidable consequences (traps)

Solutions

- How to counter the horizon effect
 - Feedover
 - Do not cut off search at non-quiet board positions (dynamic positions)
 - Example, king in danger
 - Keep searching down that path until reach quiet (stable) nodes
 - Secondary Search
 - Search further down selected path to ensure this is the best move

Quiescence Search

This involves searching past the terminal search nodes (depth of 0) and testing all the non-quiescent or 'violent' moves until the situation becomes calm, and only then apply the evaluator.

Enables programs to detect long capture sequences and calculate whether or not they are worth initiating.

Expand searches to avoid evaluating a position where tactical disruption is in progress.

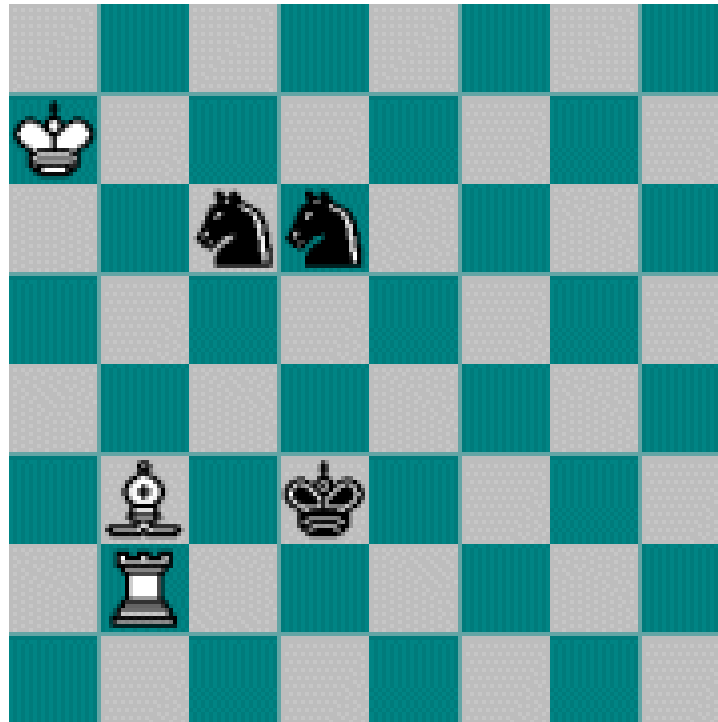
Additional Refinements

- **Probabilistic Cut:** cut branches probabilistically based on shallow search and global depth-level statistics (forward pruning)
- **Openings/Endgames:** for some parts of the game (especially initial and end moves), keep a catalog of best moves to make.
- **Singular Extensions:** find obviously good moves and try them at cutoff.

End-Game Databases

- Ken Thompson - all 5 piece end-games
- Lewis Stiller - all 6 piece end-games
 - Refuted common chess wisdom: many positions thought to be ties were really forced wins -- 90% for white
 - Is perfect chess a win for white?

The MONSTER



White wins in 255 moves
(Stiller, 1991)

Deterministic Games in Practice

- **Checkers:** Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used a precomputed endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 444 billion positions. Checkers is now solved!
- **Chess:** Deep Blue defeated human world champion Garry Kasparov in a six-game match in 1997. Deep Blue searches 200 million positions per second, uses very sophisticated evaluation, and undisclosed methods for extending some lines of search up to 40 ply. Current programs are even better, if less historic!
- **Othello:** human champions refuse to compete against computers, who are too good.
- **Go:** until recently, human champions refused to compete against computers, who were too bad. In Go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves, along with aggressive pruning. In 2016, DeepMind's AlphaGo defeated Lee Sedol 4-1 to end the human reign.

Game of Go

human champions refused to compete against computers, because software used to be too bad.

	Chess	Go
Size of board	8 x 8	19 x 19
Average no. of moves per game	100	300
Avg branching factor per turn	35	235
Additional complexity		Players can pass

AlphaGo (2016)

- Combination of
 - Deep Neural Networks
 - Monte Carlo Tree Search
- More details later.

Other Games

deterministic

chance

perfect
information

chess,
checkers, go,
othello

backgammon,
monopoly

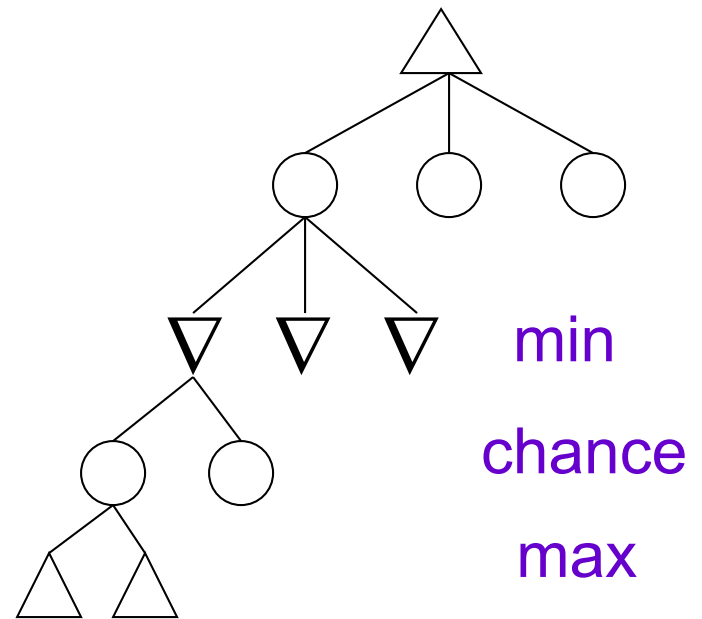
imperfect
information

stratego

bridge, poker,
scrabble

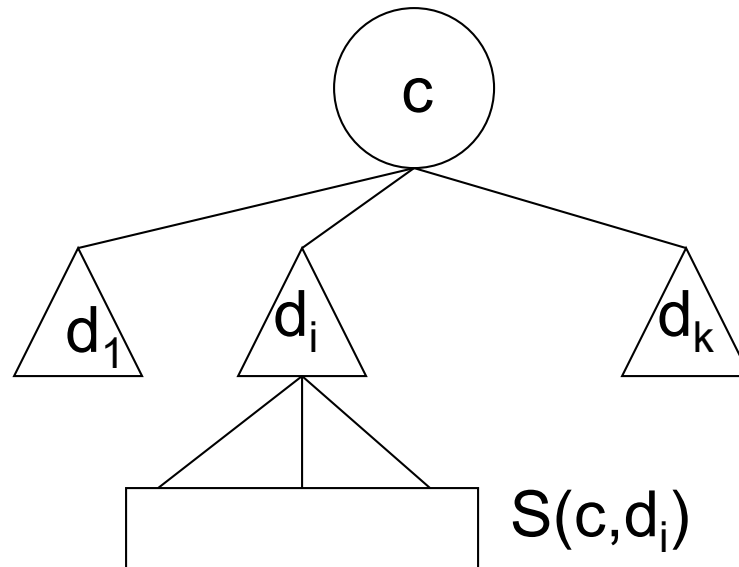
Games of Chance

- What about games that involve chance, such as
 - rolling dice
 - picking a card
- Use three kinds of nodes:
 - max nodes
 - min nodes
 - chance nodes



Games of Chance

Expectiminimax

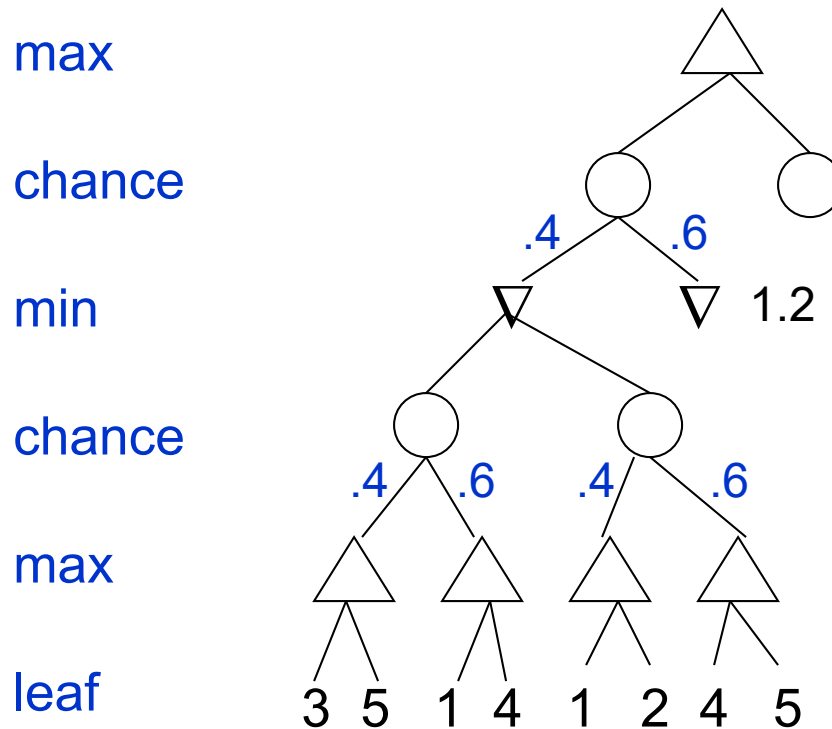


chance node with
max children

$$\text{expectimax}(c) = \sum_i P(d_i) \max_{s \in S(c, d_i)} (\text{backed-up-value}(s))$$

$$\text{expectimin}(c') = \sum_i P(d_i) \min_{s \in S(c, d_i)} (\text{backed-up-value}(s))$$

Example Tree with Chance



Complexity

- Instead of $O(b^m)$, it is $O(b^m n^m)$ where n is the number of chance outcomes.
- Since the complexity is higher (both time and space), we cannot search as deeply.
- Pruning algorithms may be applied.

Imperfect Information

- E.g. card games, where opponents' initial cards unknown
- Idea: For all deals consistent with what you can see
 - compute the minimax value of available actions for each of possible deals
 - compute the expected value over all deals



Status of AI Game Players

- Tic Tac Toe
 - Tied for best player in world
- Othello
 - [Computer](#) better than any human
 - Human champions now refuse to play computer
- Scrabble
 - Maven beat world champions Joel Sherman and Matt Graham
- Backgammon
 - 1992, [Tesauro](#) combines 3-ply search & neural networks (with 160 hidden units) yielding top-3 player
- Bridge
 - [Gib](#) ranked among top players in the world
- Poker
 - 2015, Heads-up limit hold'em poker is solved
- Checkers
 - 1994, [Chinook](#) ended 40-year reign of human champion Marion Tinsley
- Chess
 - 1997, [Deep Blue](#) beat human champion Gary Kasparov in six-game match
 - Deep Blue searches 200M positions/second, up to 40 ply
 - Now looking at other applications (molecular dynamics, drug synthesis)
- Go
 - 2016, Deepmind's AlphaGo defeated Lee Sedol & 2017 defeated Ke Jie

Summary

- Games are fun to work on!
- They illustrate several important points about AI.
- Perfection is unattainable → must approximate.
- Game playing programs have shown the world what AI can do.