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:

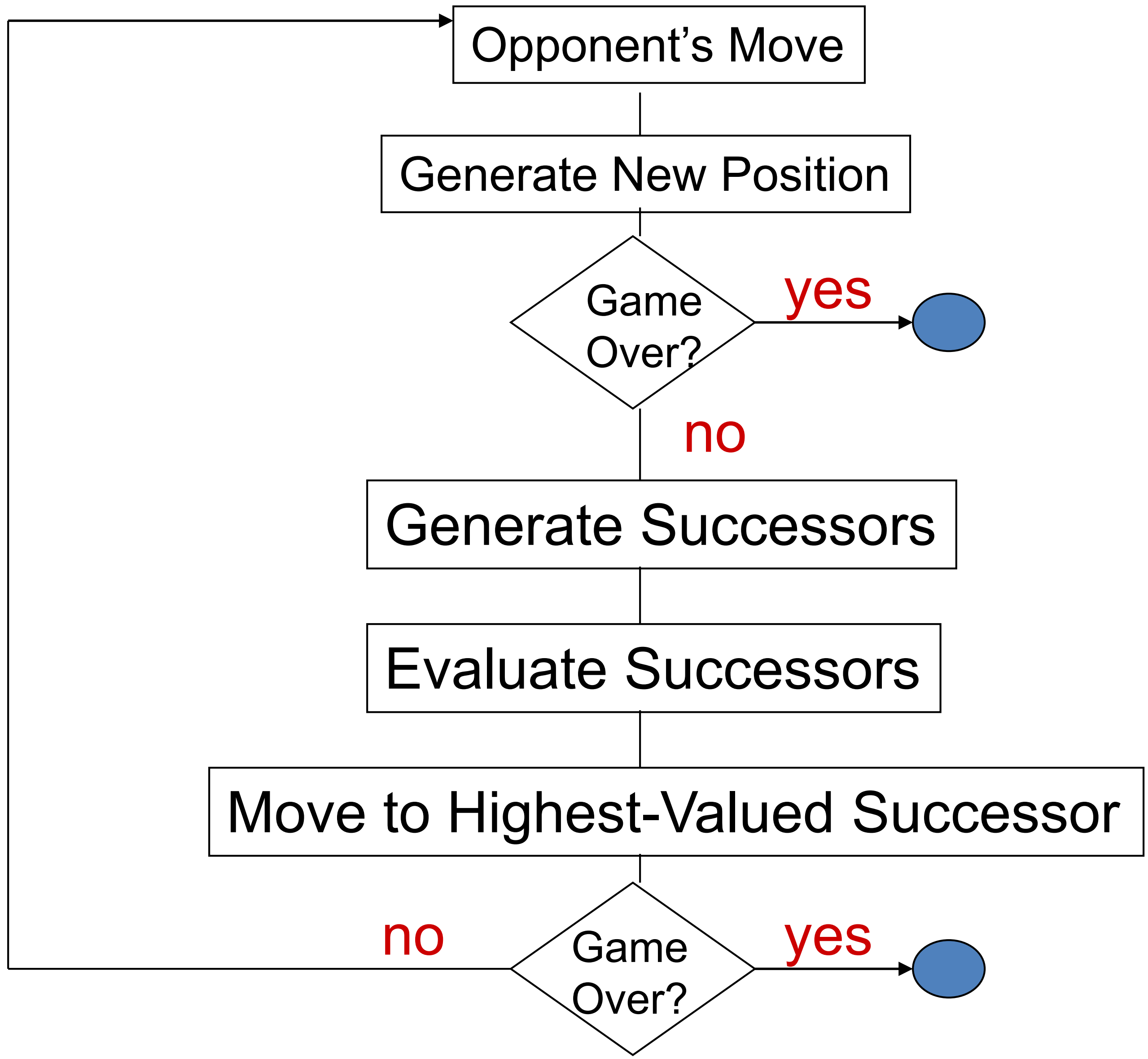
1. Sequence of moves to play
2. Rules that specify possible moves
3. Rules that specify a payment for each move
4. Objective is to maximize your payment

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)

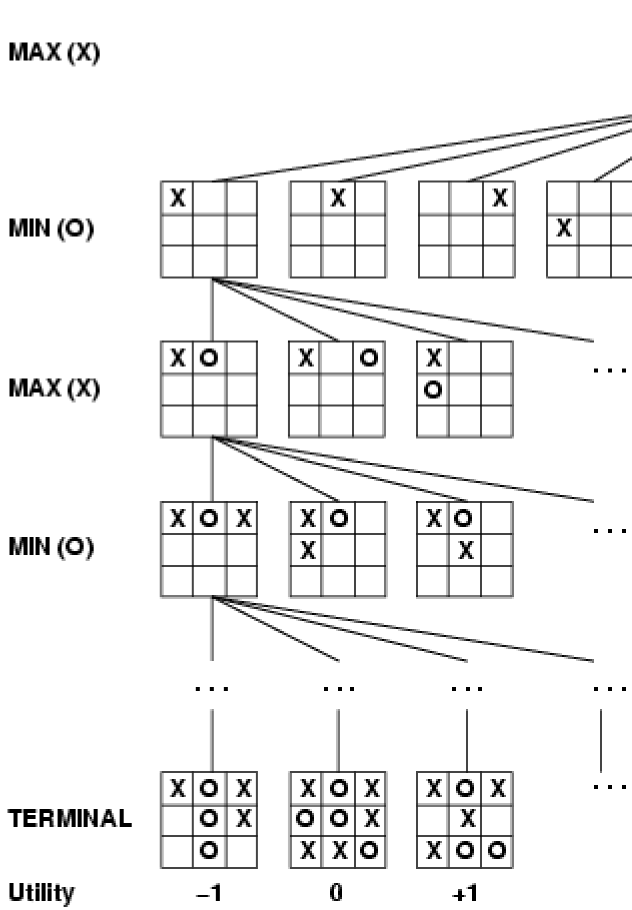
computer's turn

opponent's turn

computer's turn

opponent's turn

leaf nodes are evaluated



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

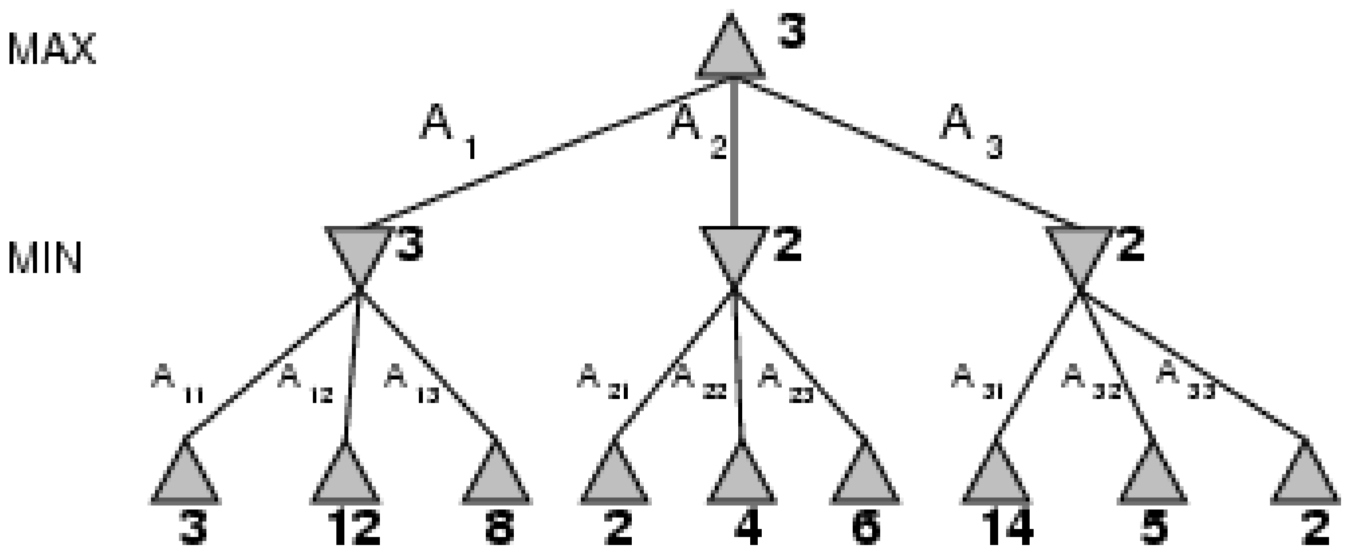
At the leaf nodes, the **utility function** is employed. Big value means good, small is bad.

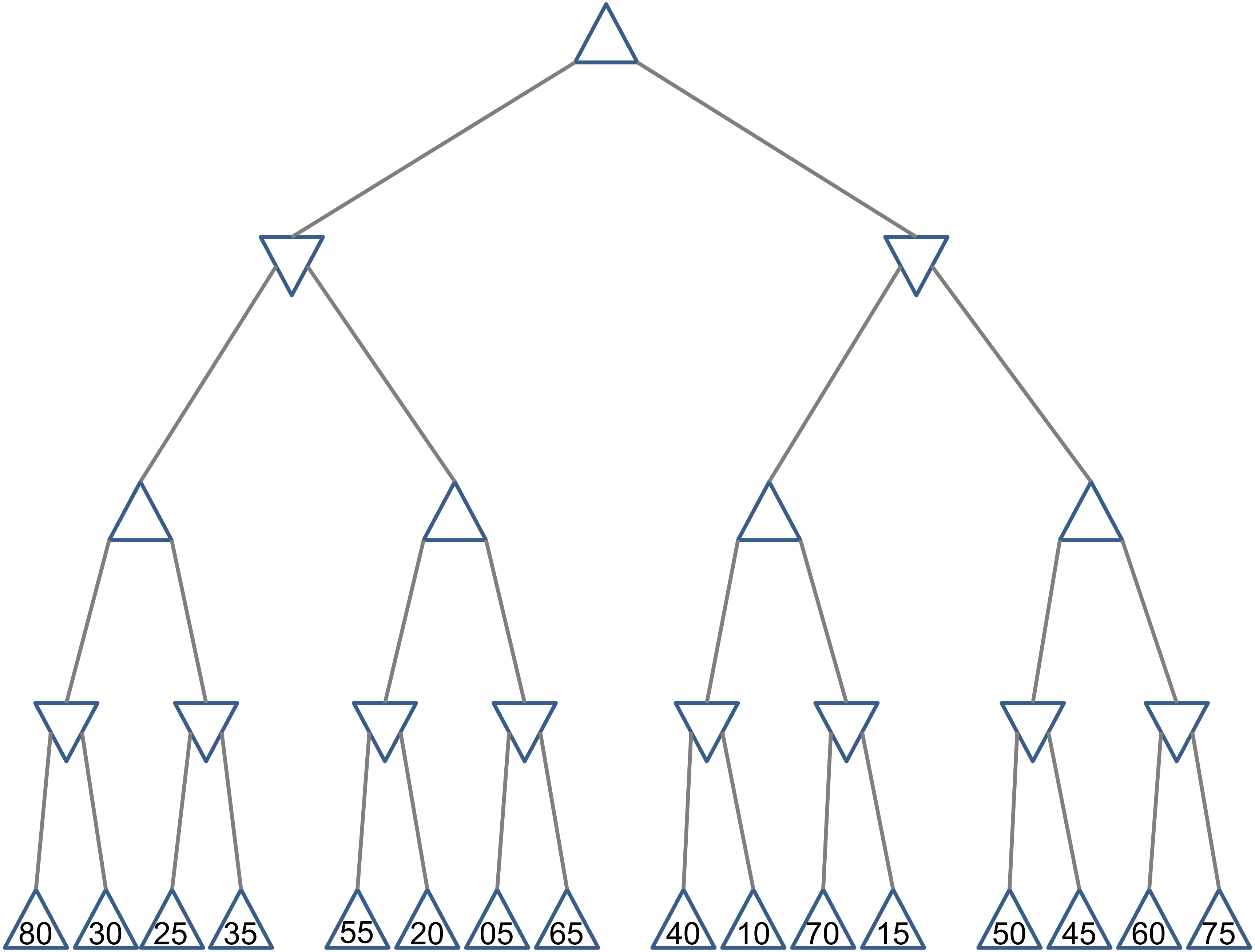
Mini-Max Terminology

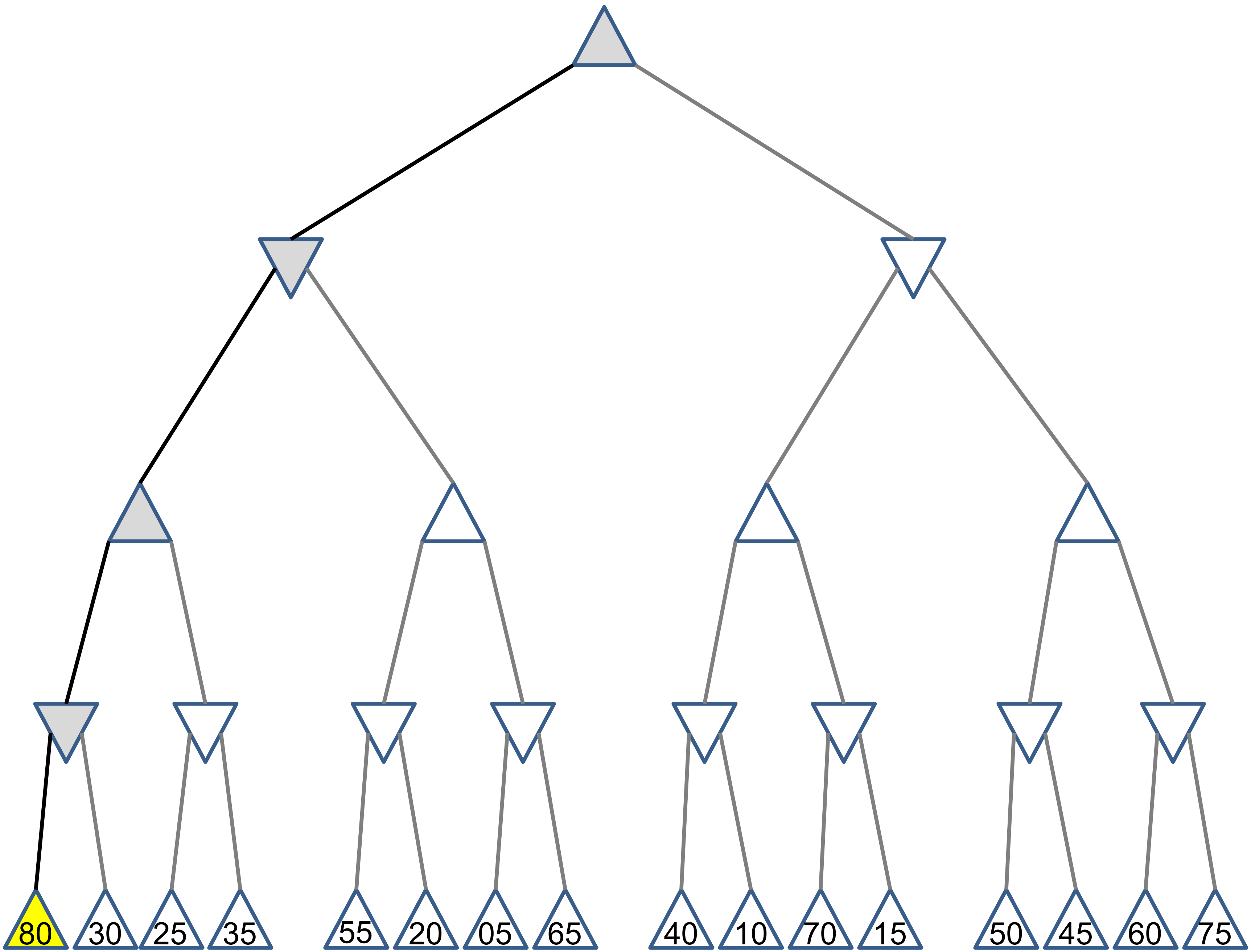
- move: a move by both players
- ply: a half-move
- utility function: the function applied to leaf nodes
- backed-up value
  - of a max-position: the value of its largest successor
  - of a min-position: the value of its smallest successor
- minimax procedure: search down several levels; at the bottom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

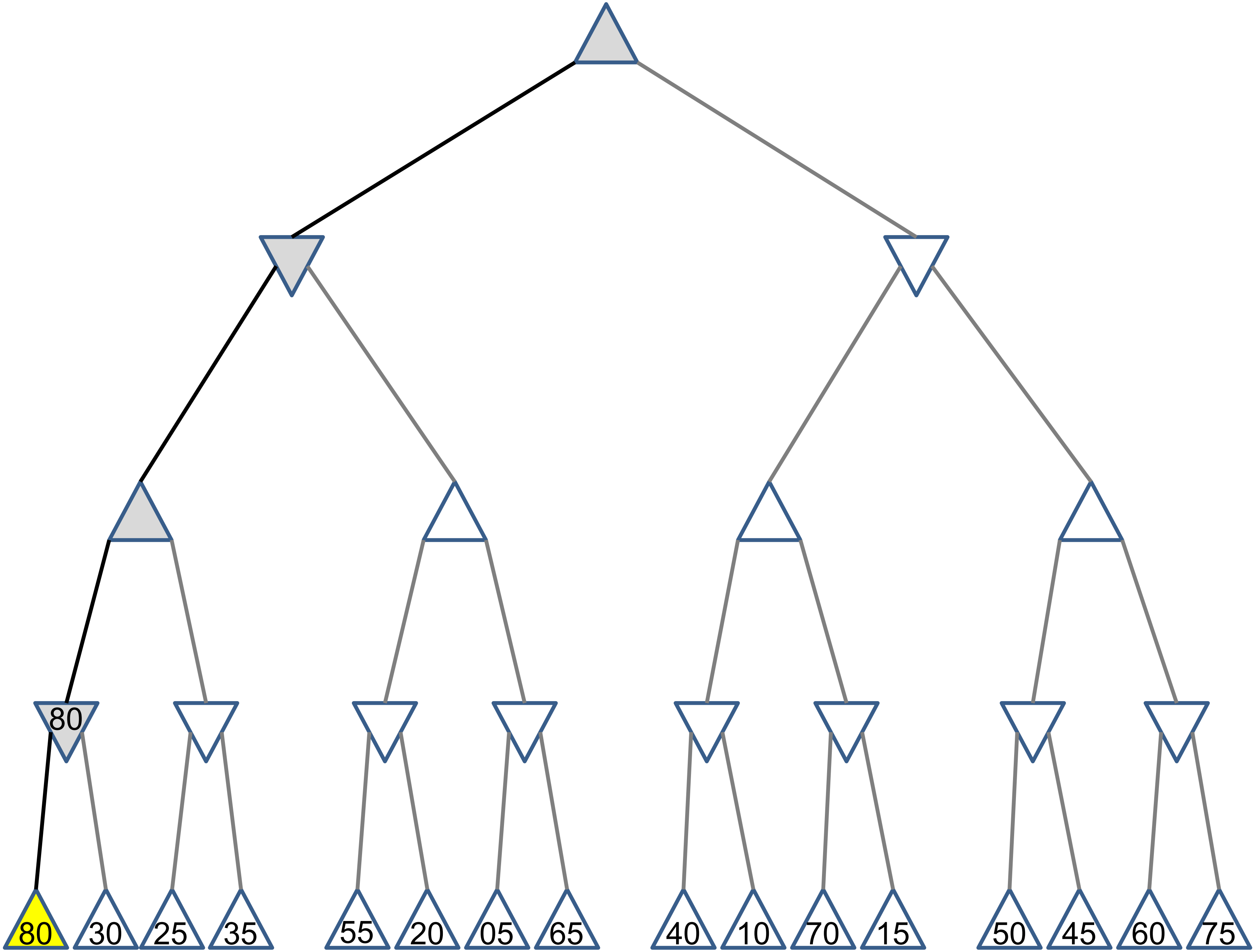
Minimax

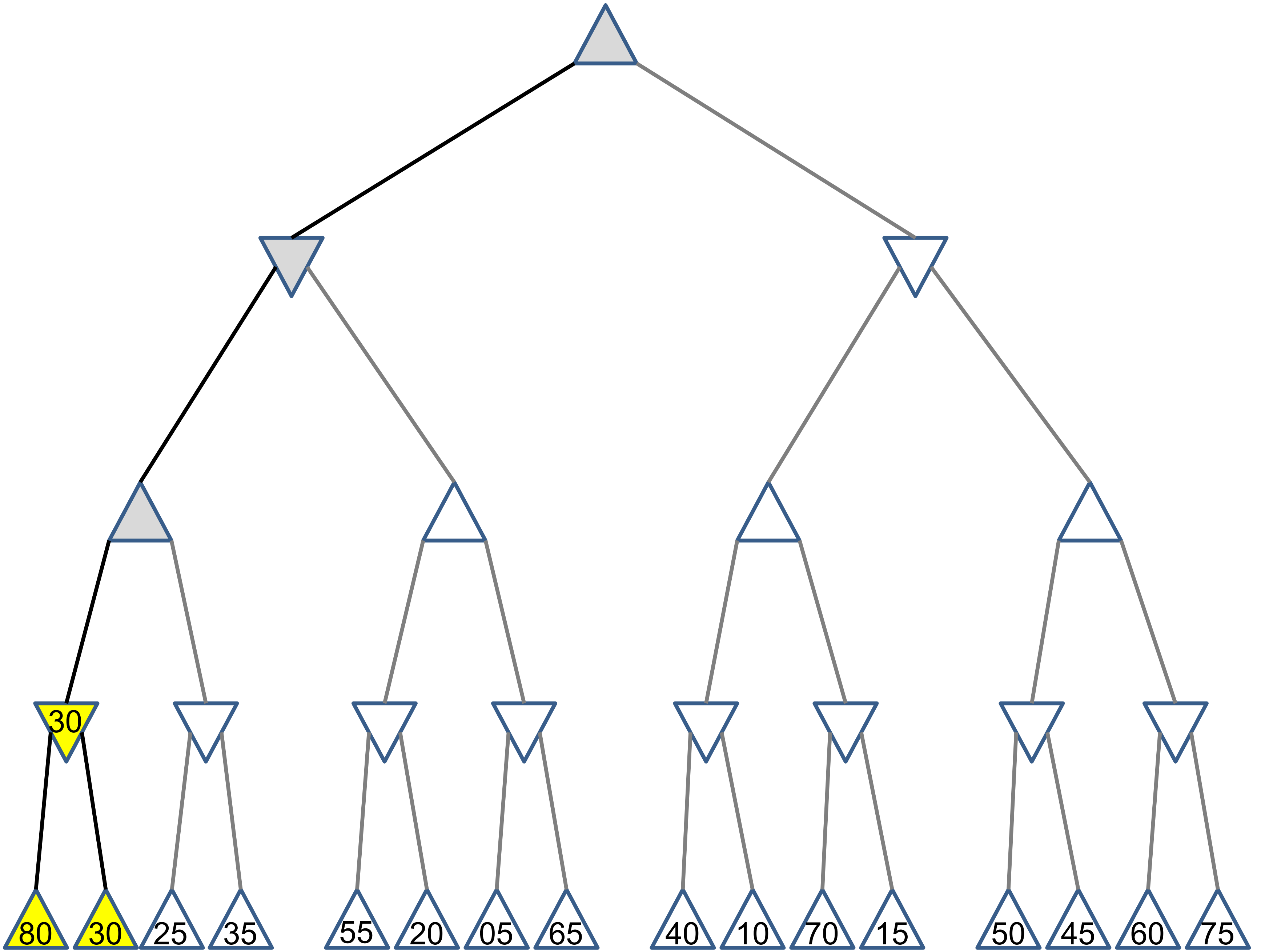
- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value
  = best achievable payoff against best play
- E.g., 2-ply game:

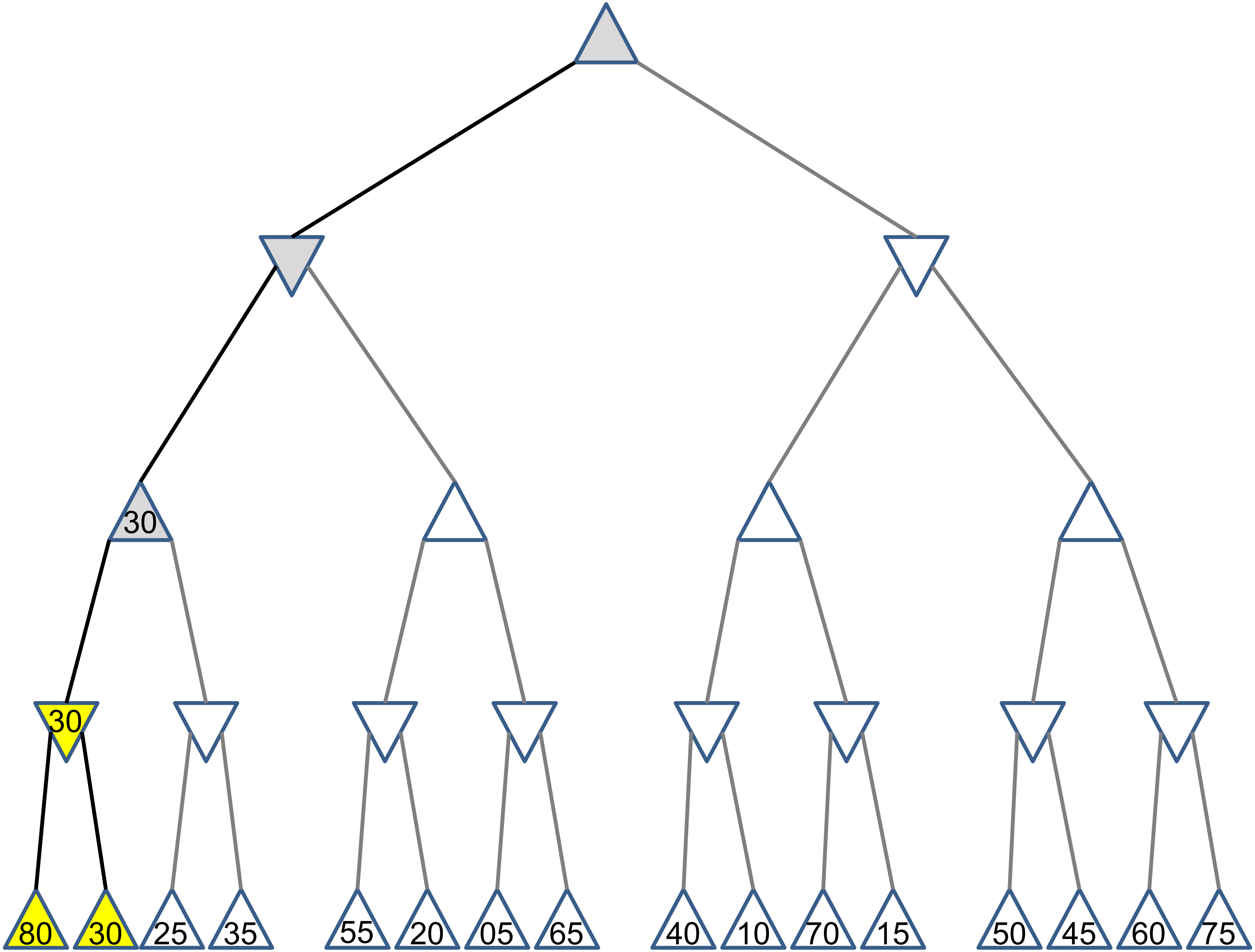


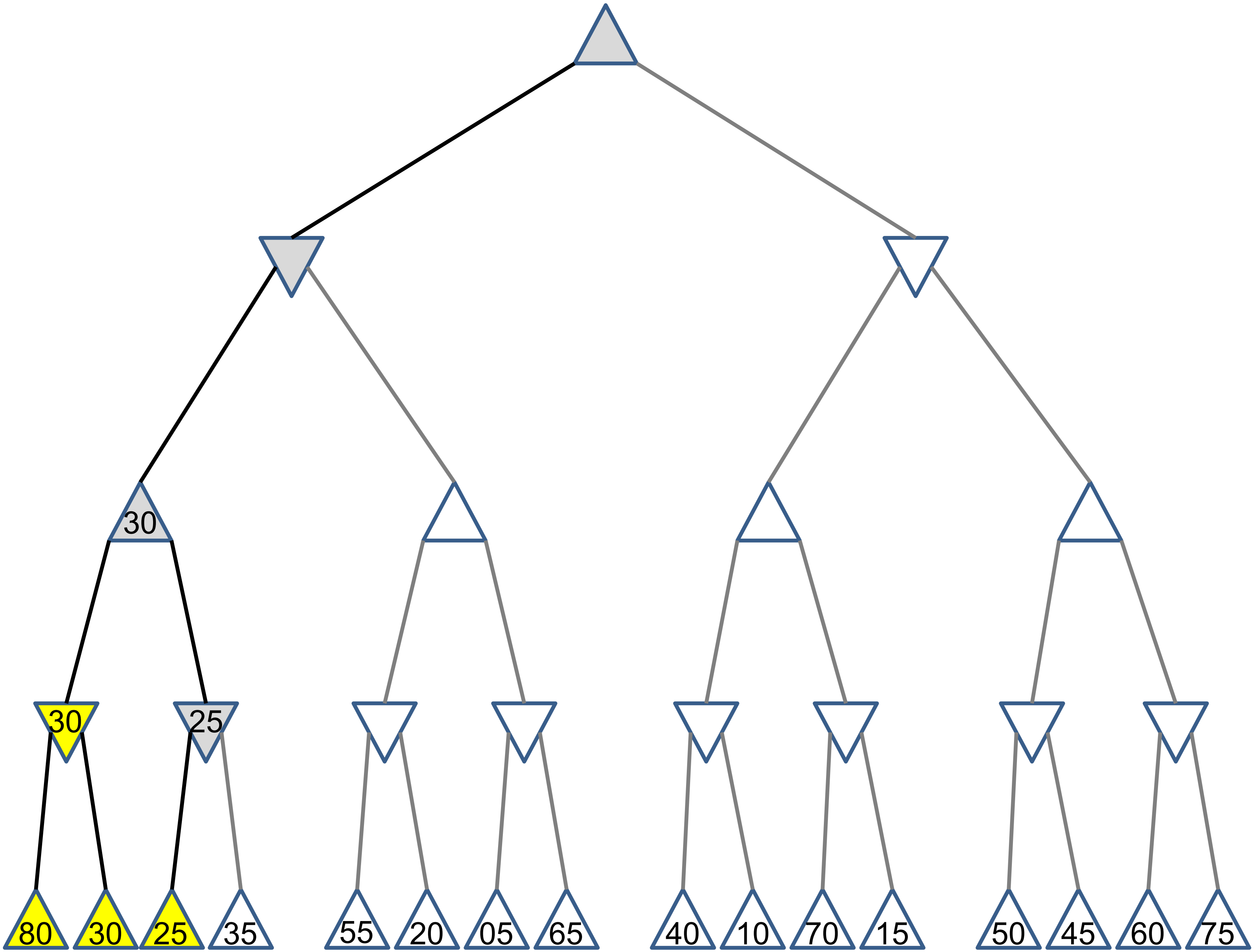

80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75


80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

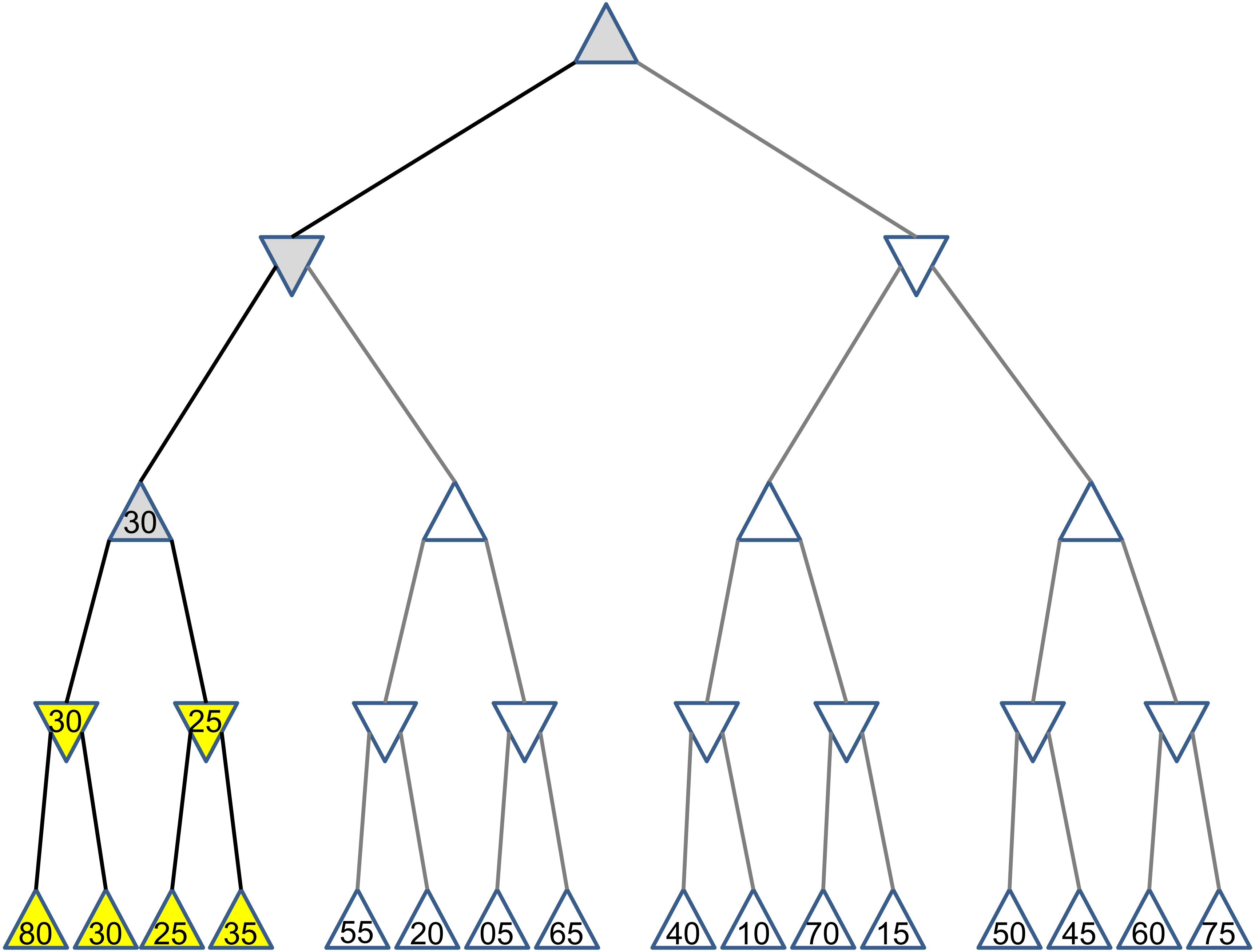

80
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

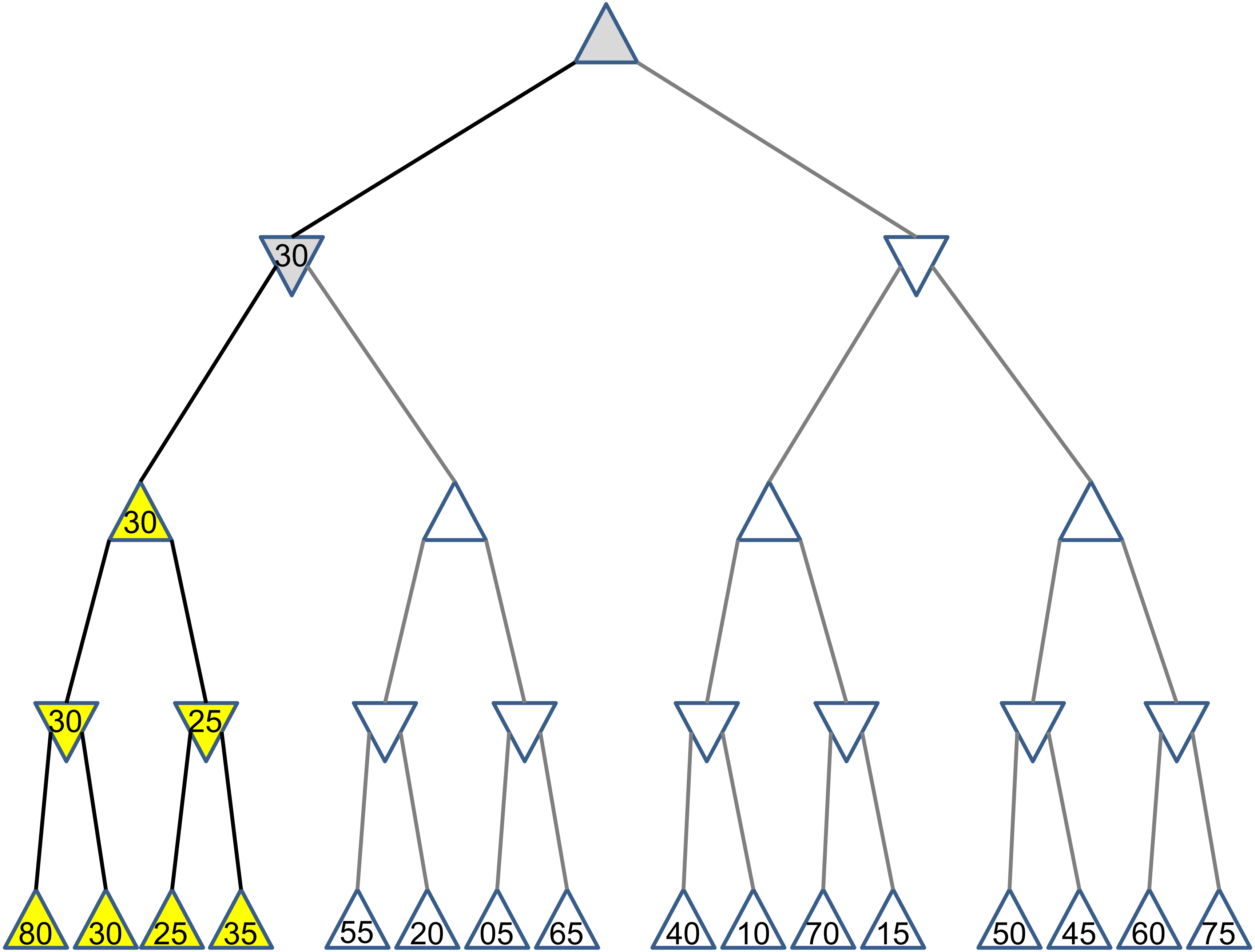

30
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

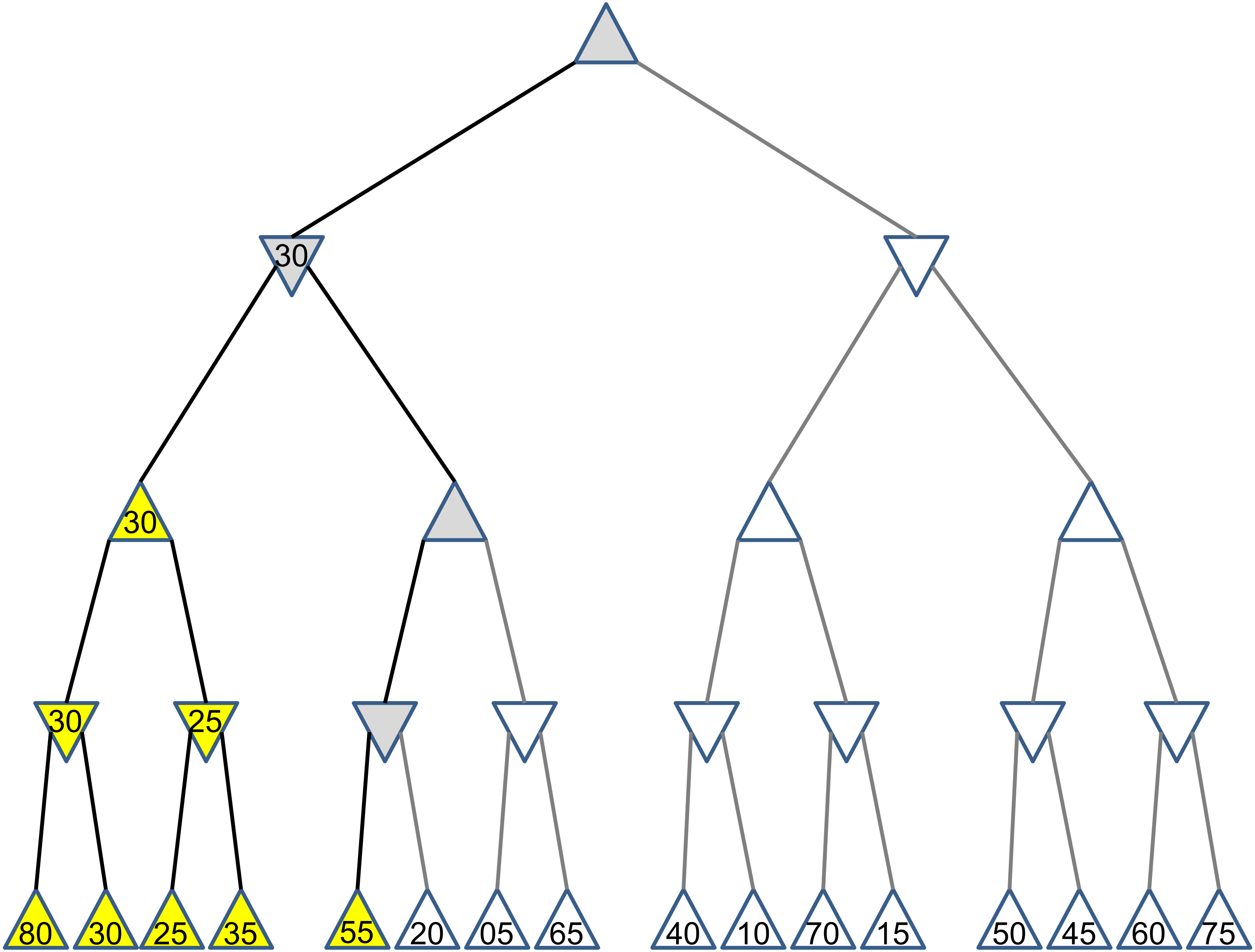

30
30
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

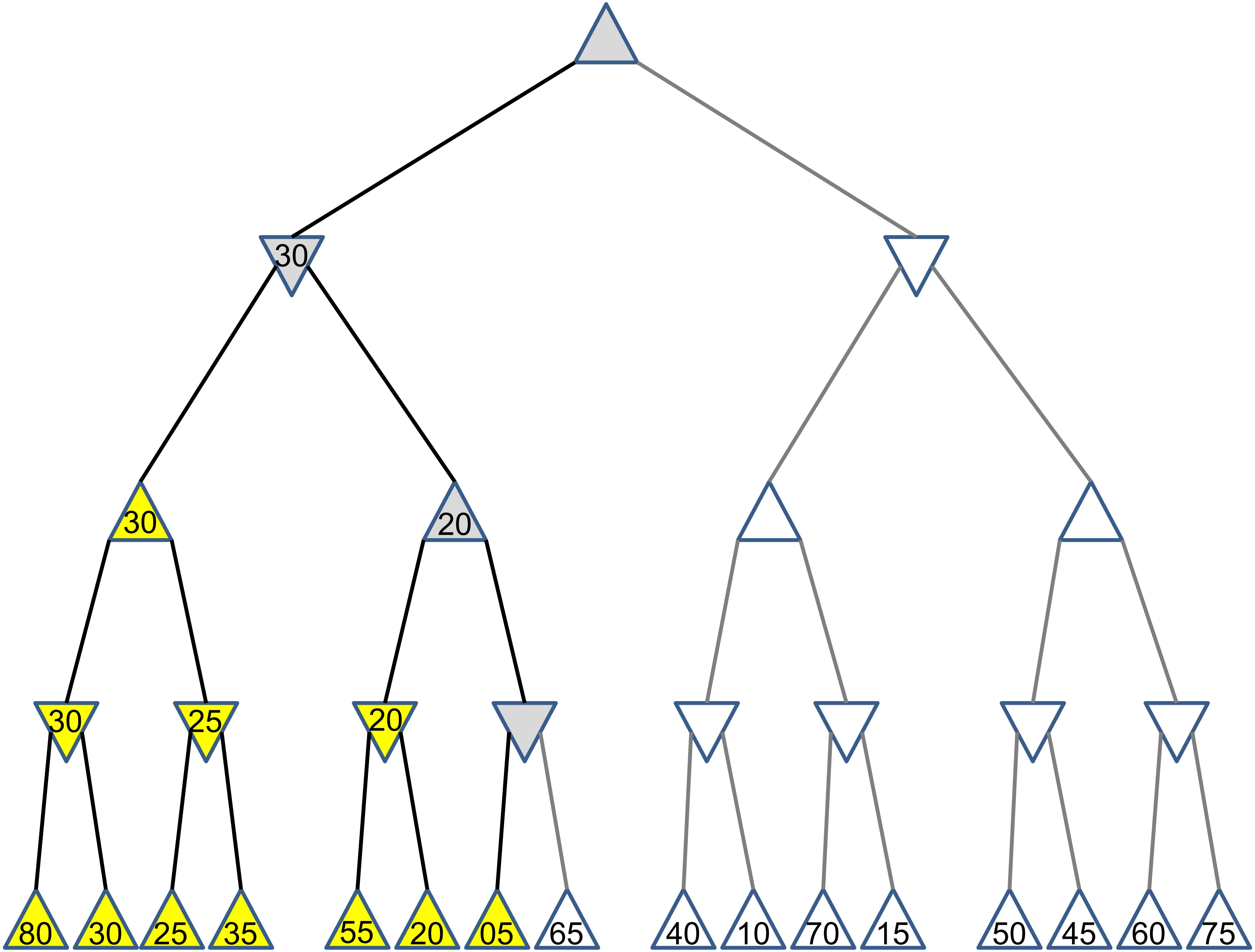


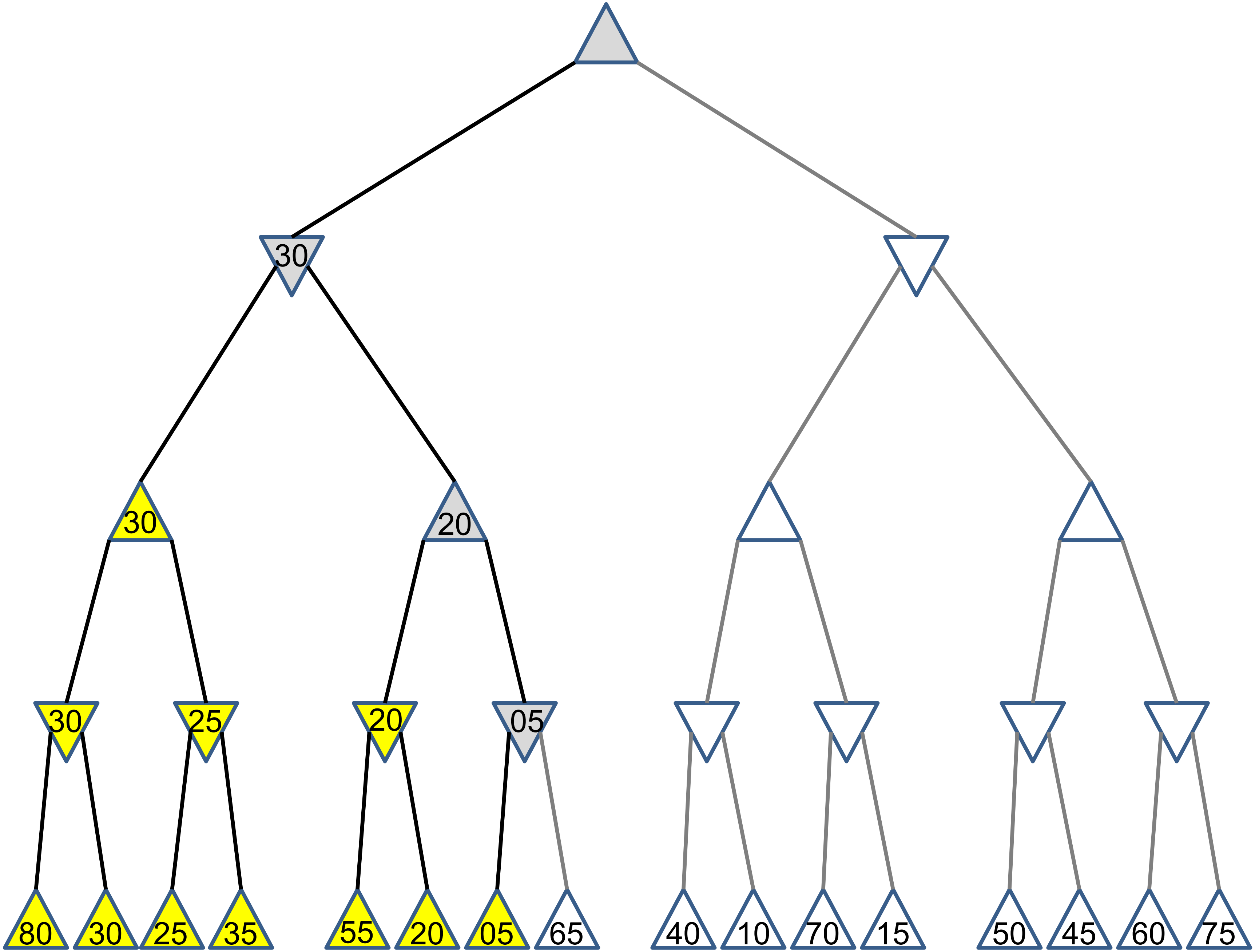
30
30
25
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

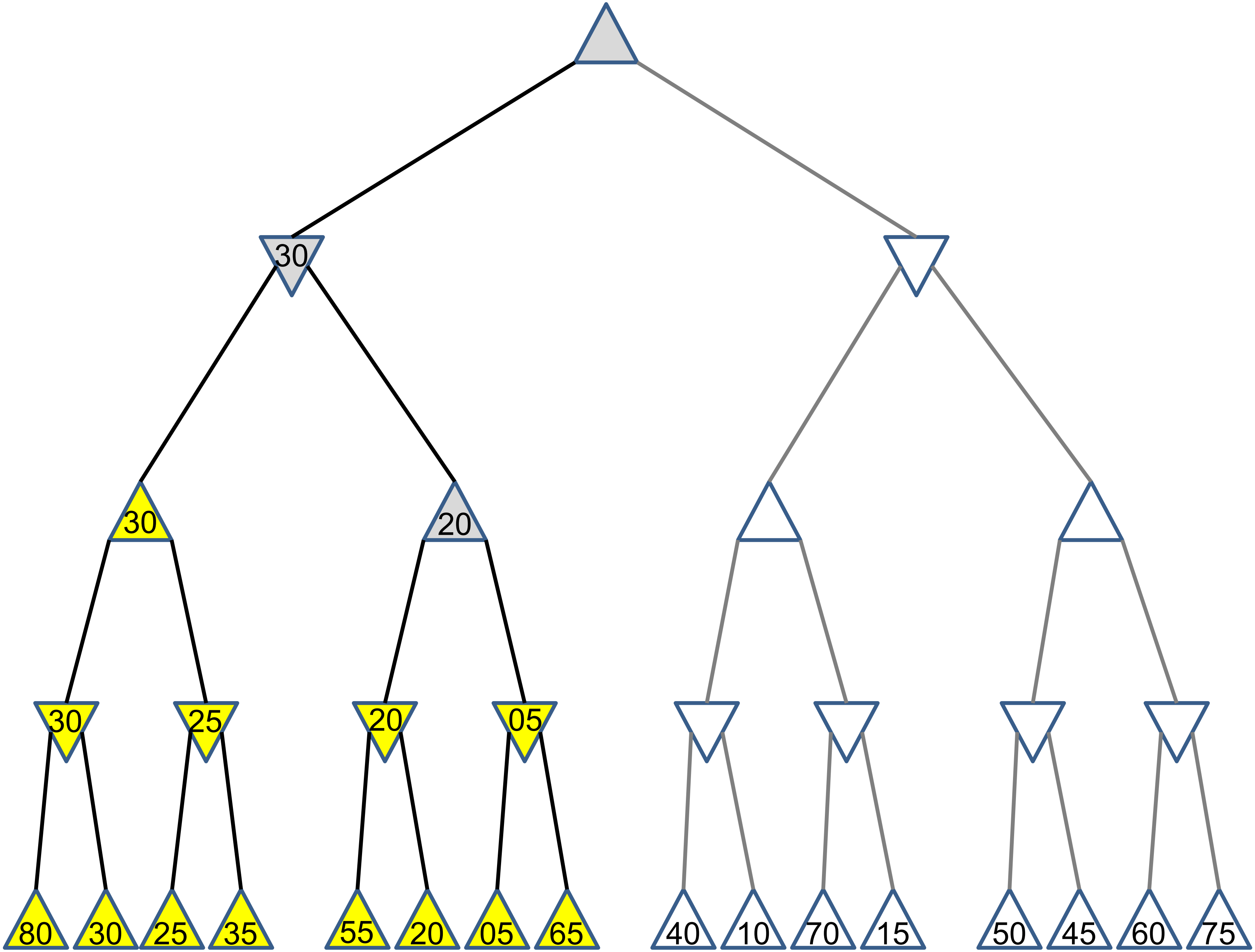

30
30
25
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

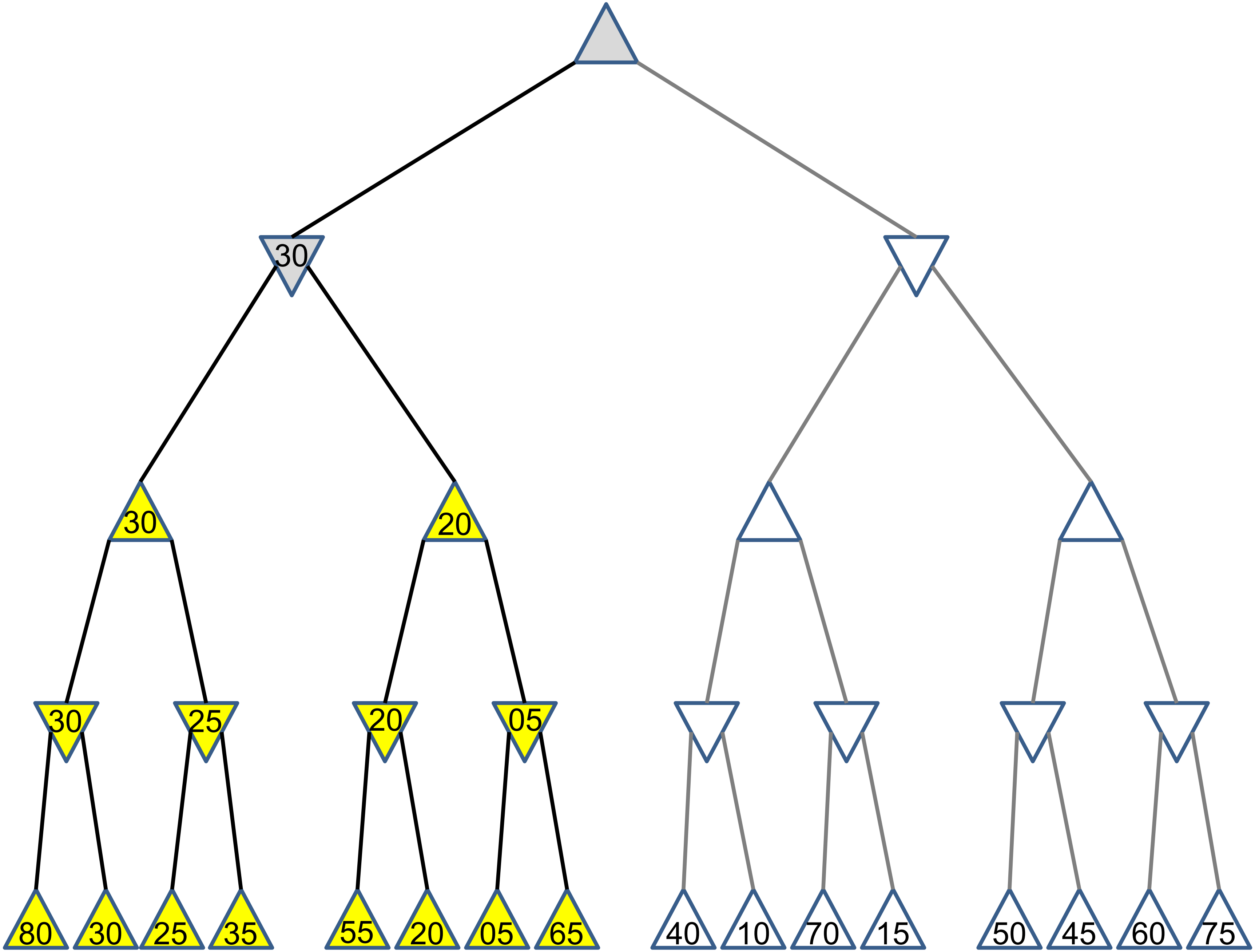

30
30
30
25
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

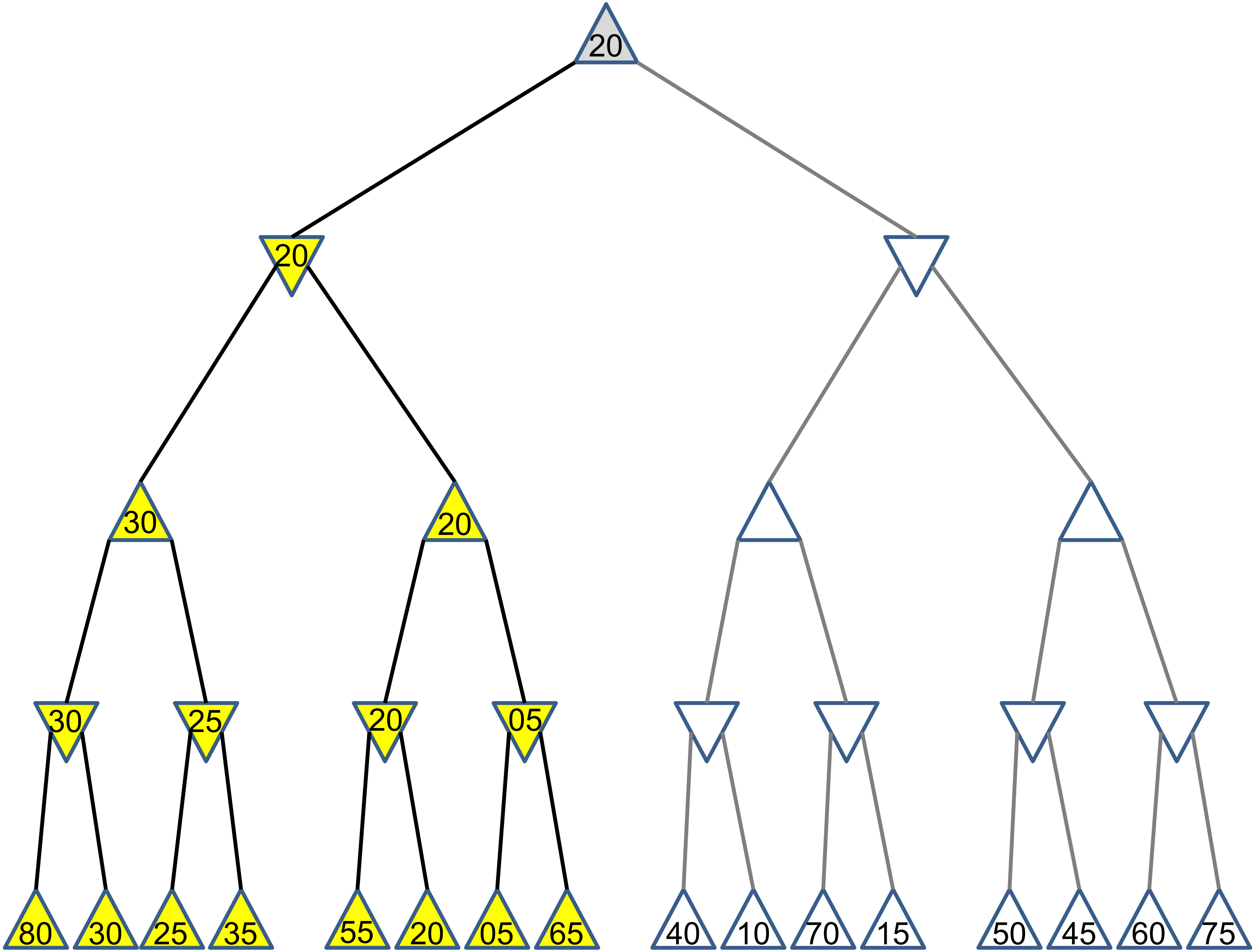

30
30
30
25
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

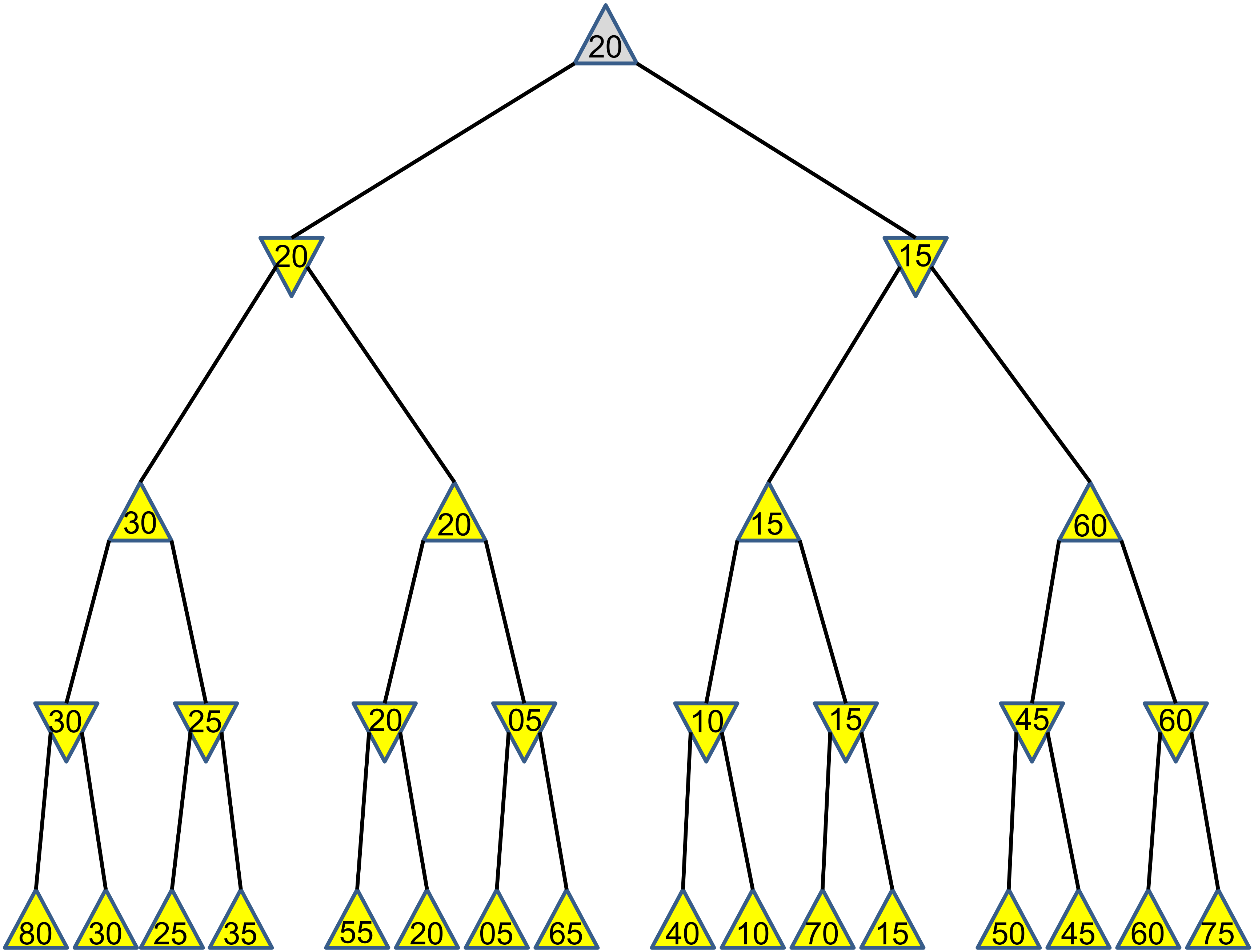

30
30
20
30
25
20
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

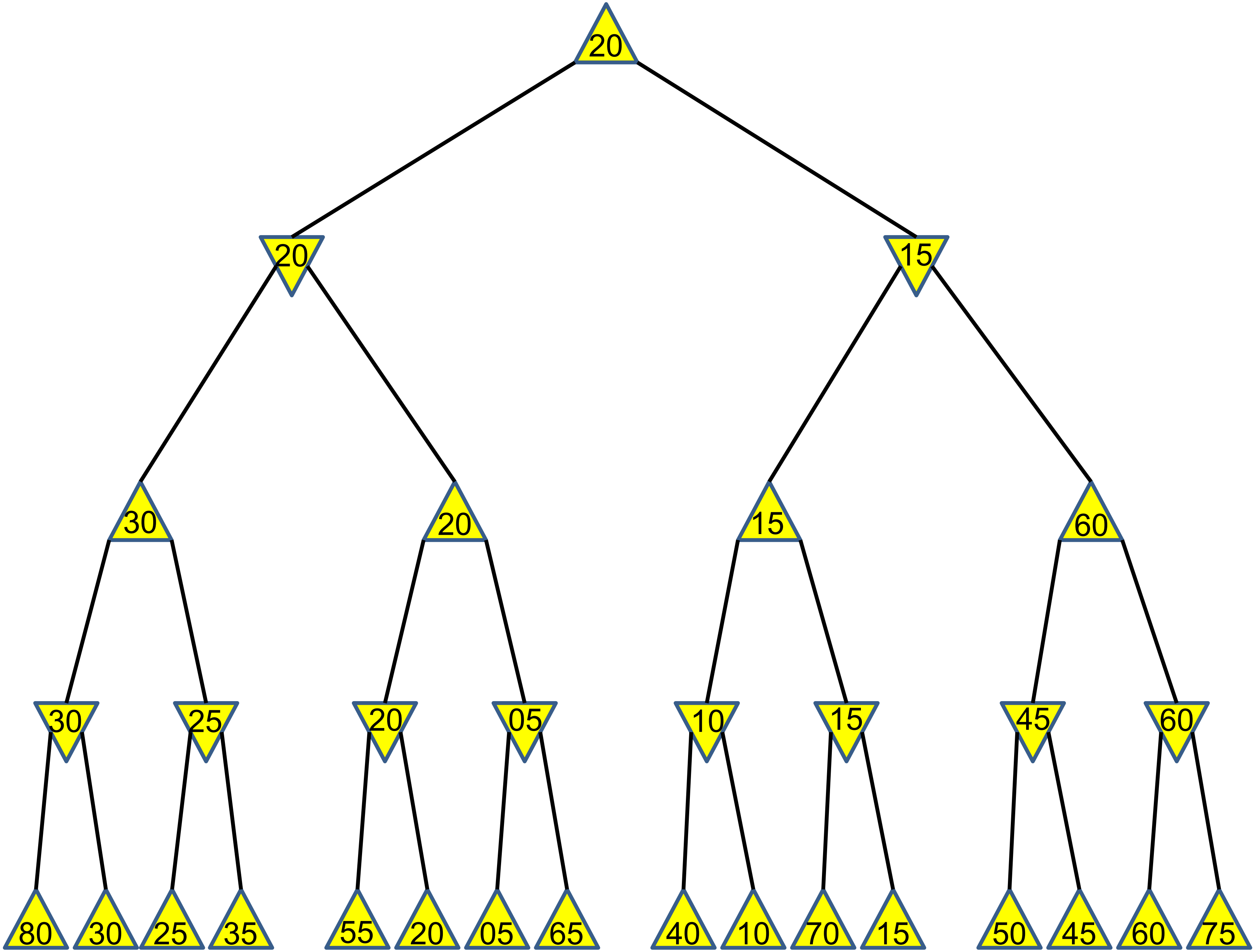

30
30
20
30
25
20
05
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

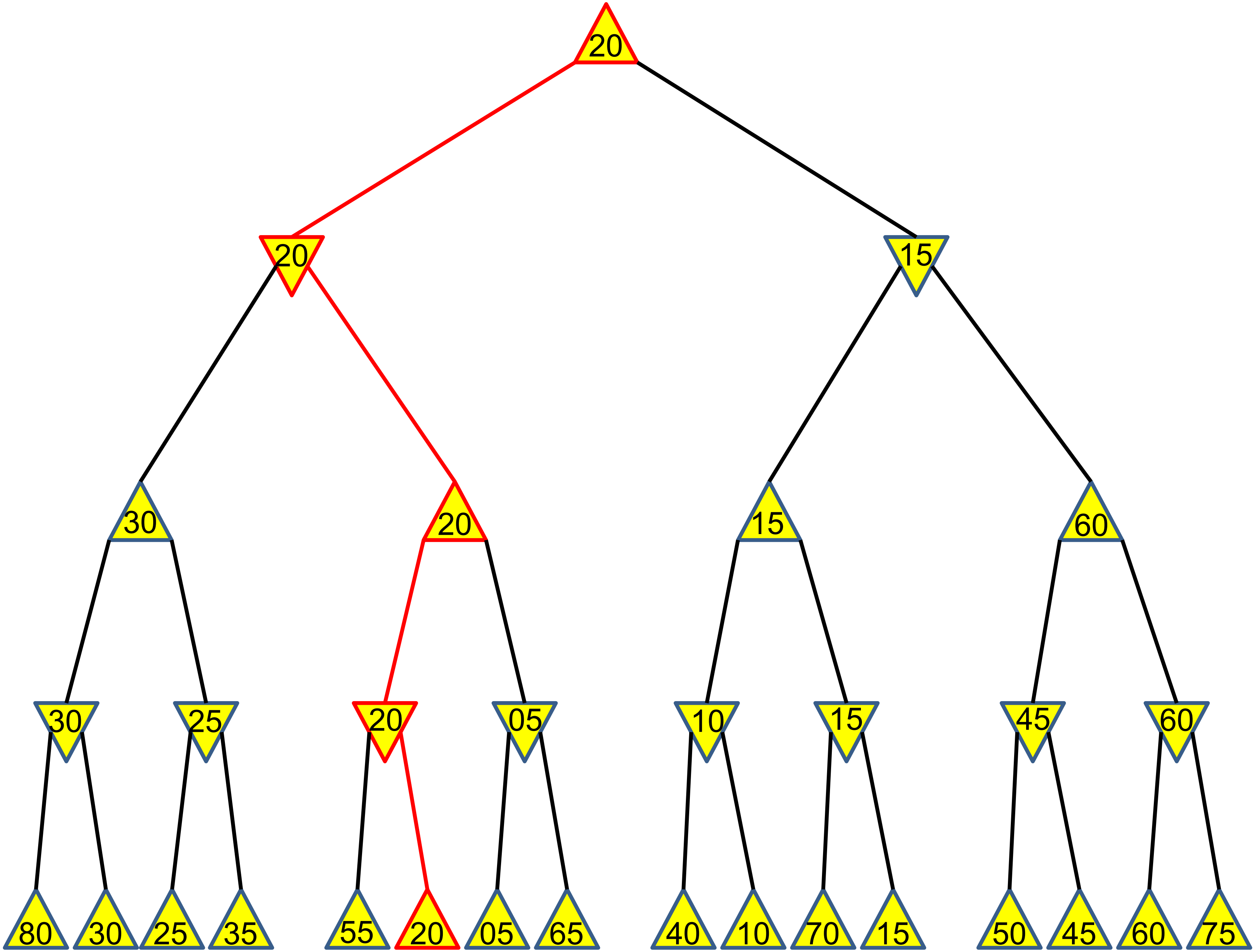

30
30
20
30
25
20
05
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75


30
30
20
30
25
20
05
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75


20
20
30
20
30
25
20
05
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75


20
20
15
30
20
15
60
30
25
20
05
10
15
45
60
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75


20
20
15
30
20
15
60
30
25
20
05
10
15
45
60
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75


20
20
15
30
20
15
60
30
25
20
05
10
15
45
60
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

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

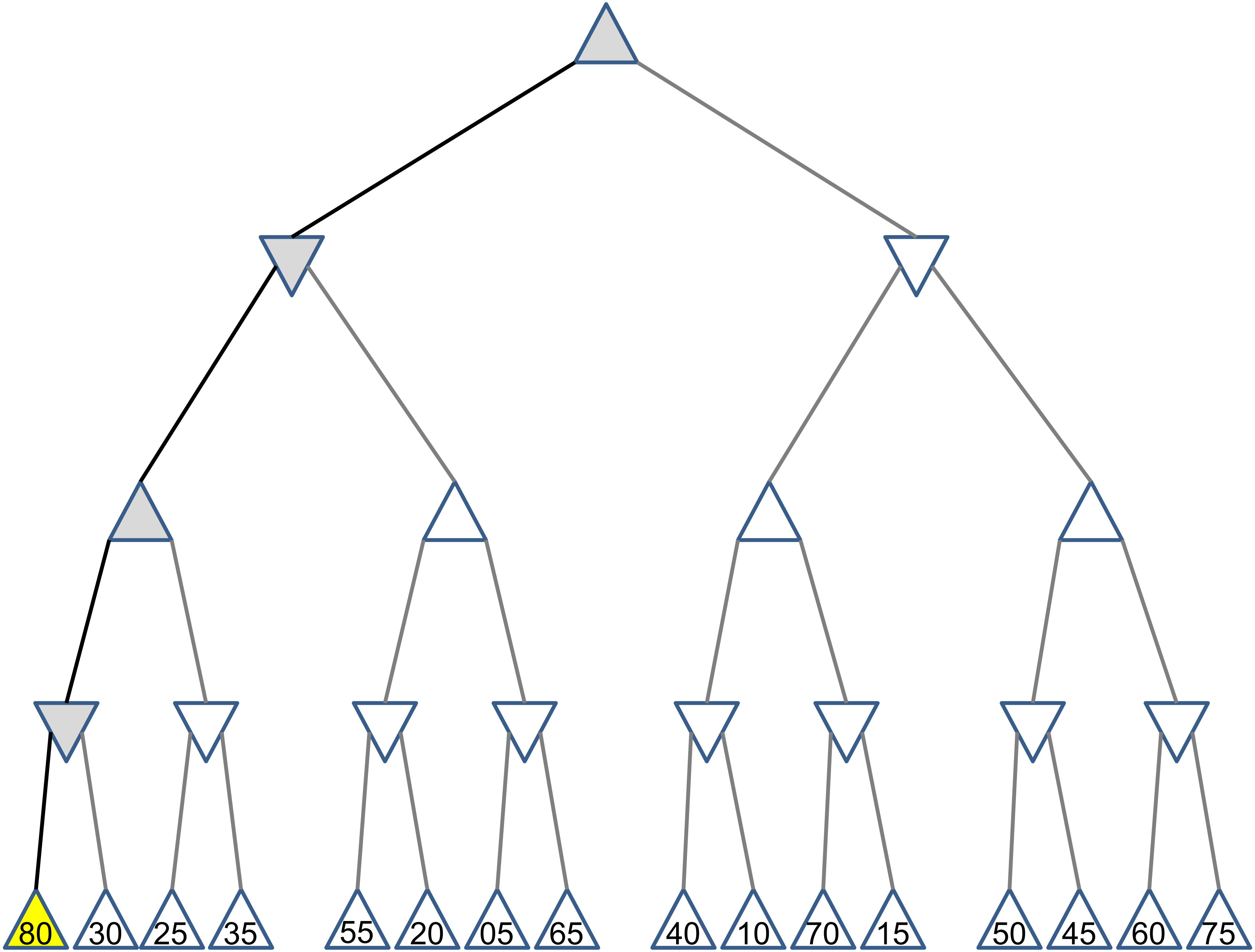
```
function MINIMAX-DECISION(state) returns an action
    v ← MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value v

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
```

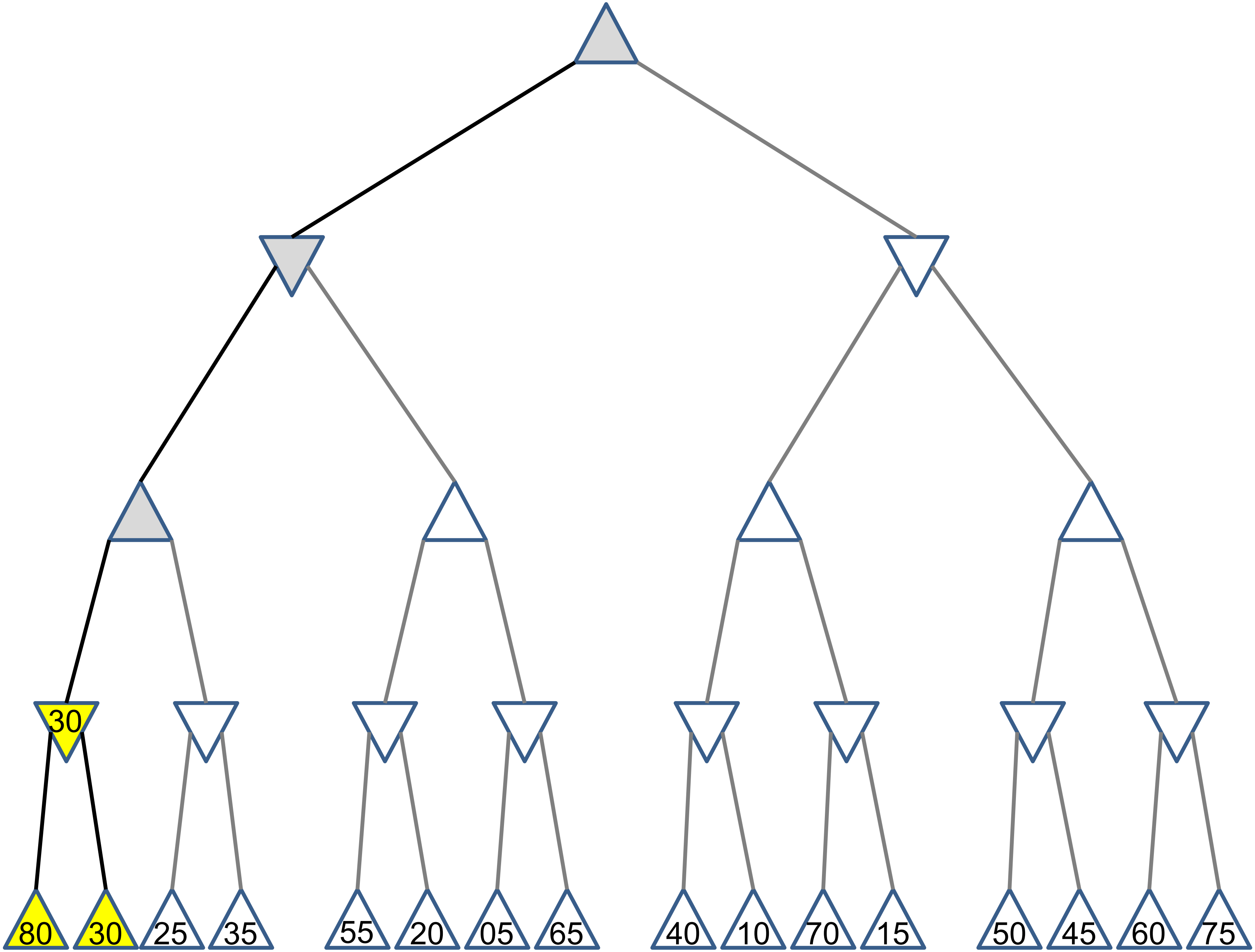
Properties of Minimax

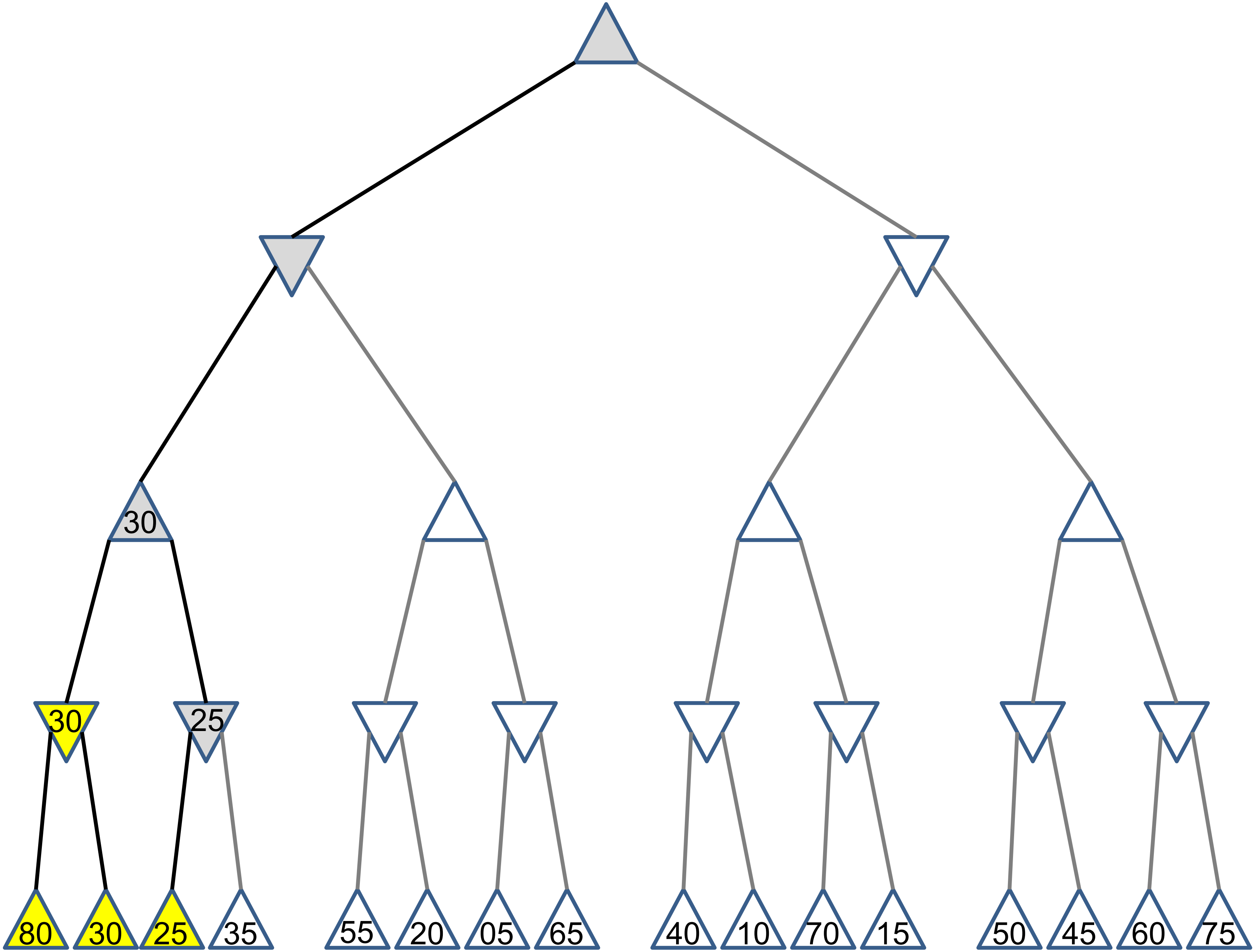
- Complete?
  - Yes (if tree is finite)
- Optimal?
  - Yes (against an optimal opponent)
  - No (does not exploit opponent weakness against suboptimal opponent)
- Time complexity?
  - $O(b^m)$
- Space complexity?
  - O(bm) (depth-first exploration)

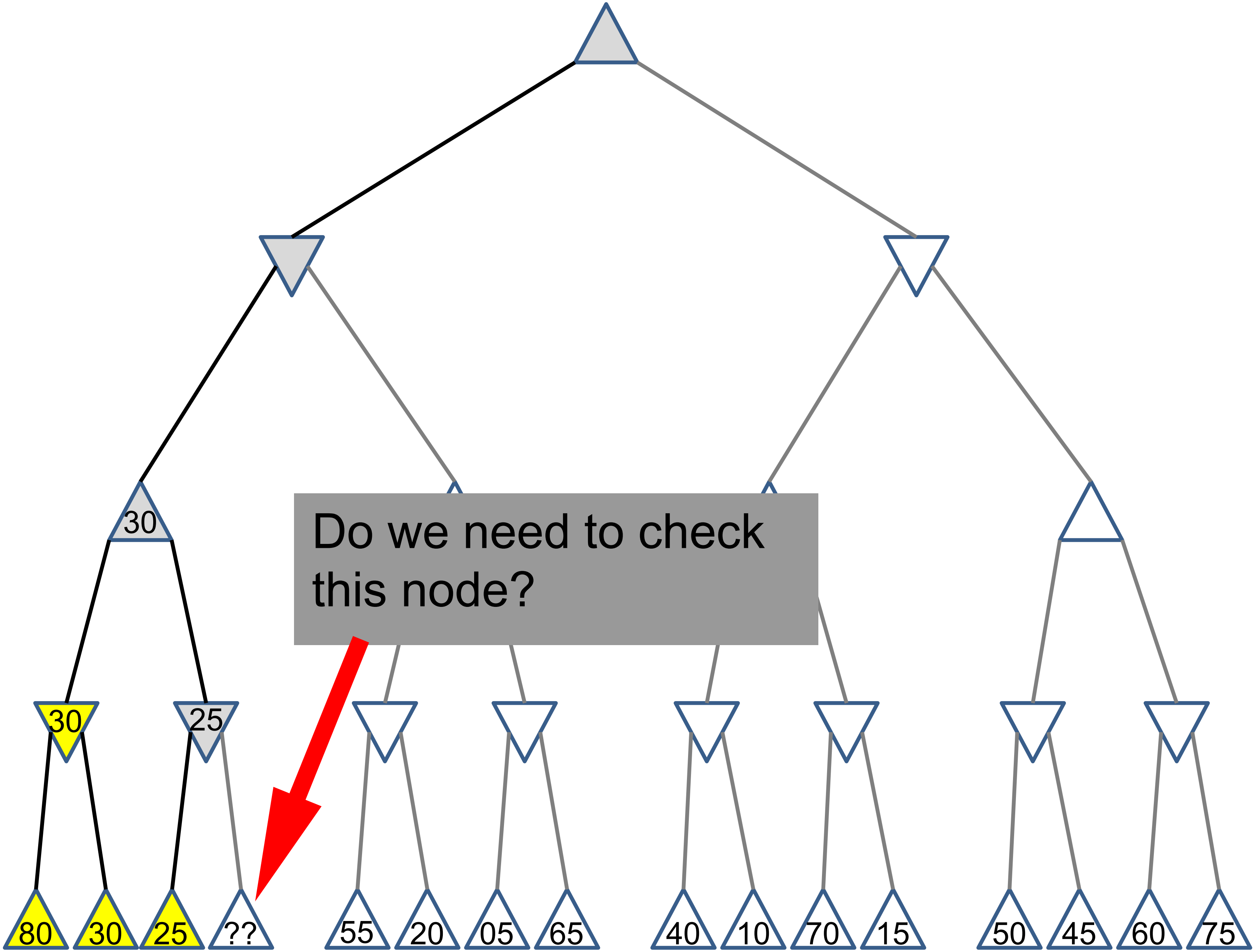
Good Enough?

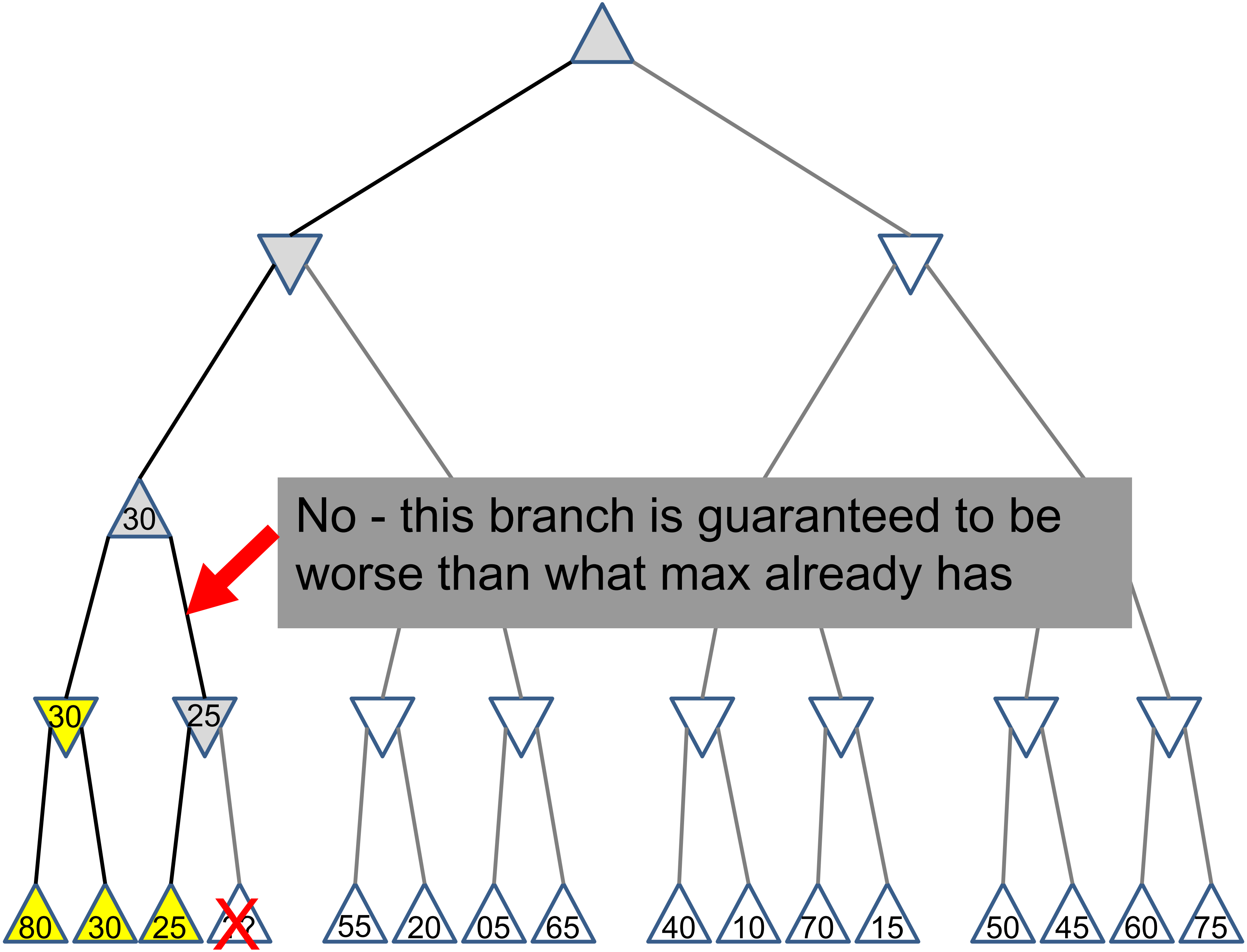
- Chess:
  - branching factor $b \approx 35$
  - game length $m \approx 100$
  - search space $b^m \approx 35^{100} \approx 10^{154}$
- The Universe:
  - number of atoms $\approx 10^{78}$
  - age $\approx 10^{18}$ seconds
  - 10^8 moves/sec x 10^{78} x $10^{18} = 10^{104}$
- Exact solution completely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


80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

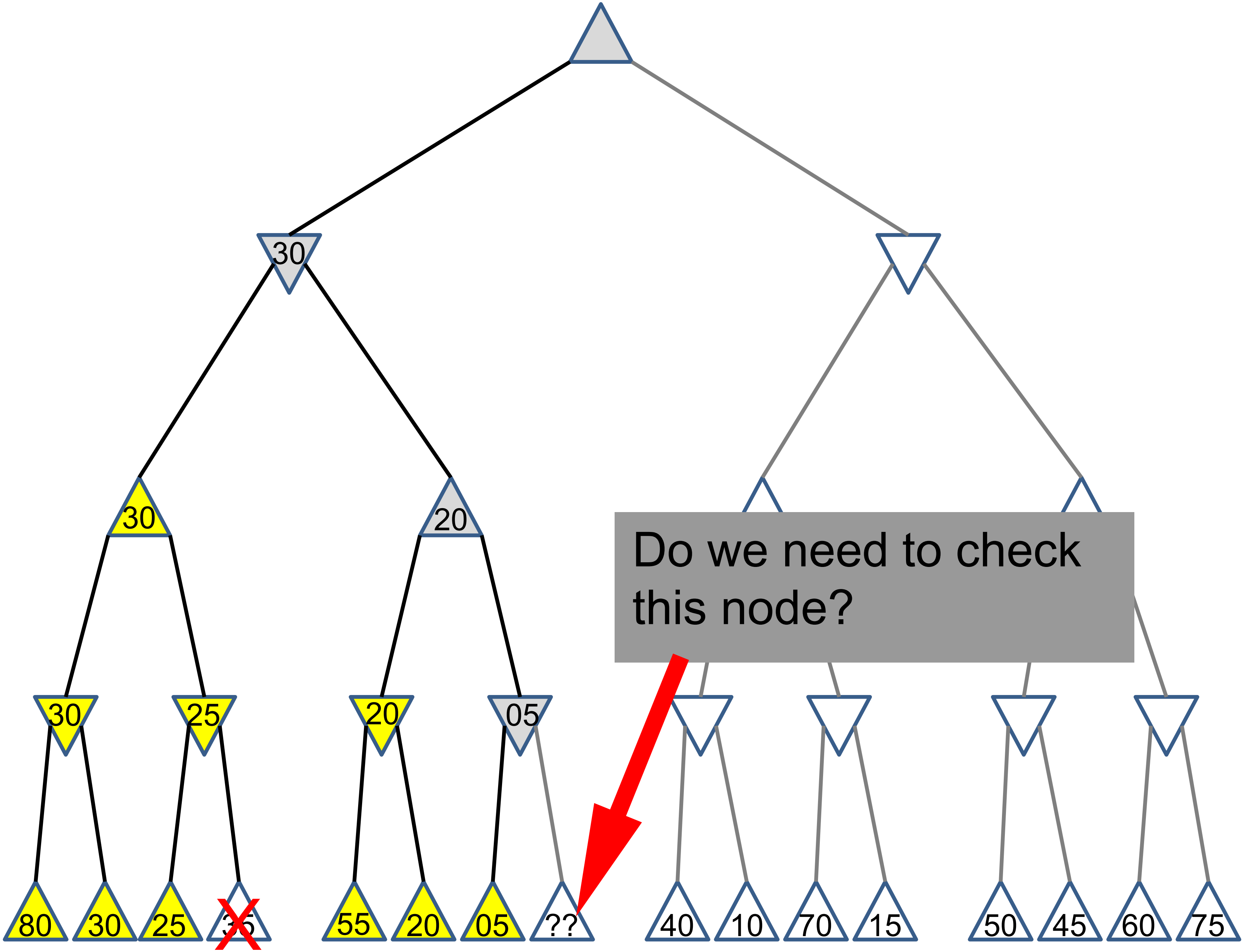

30
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

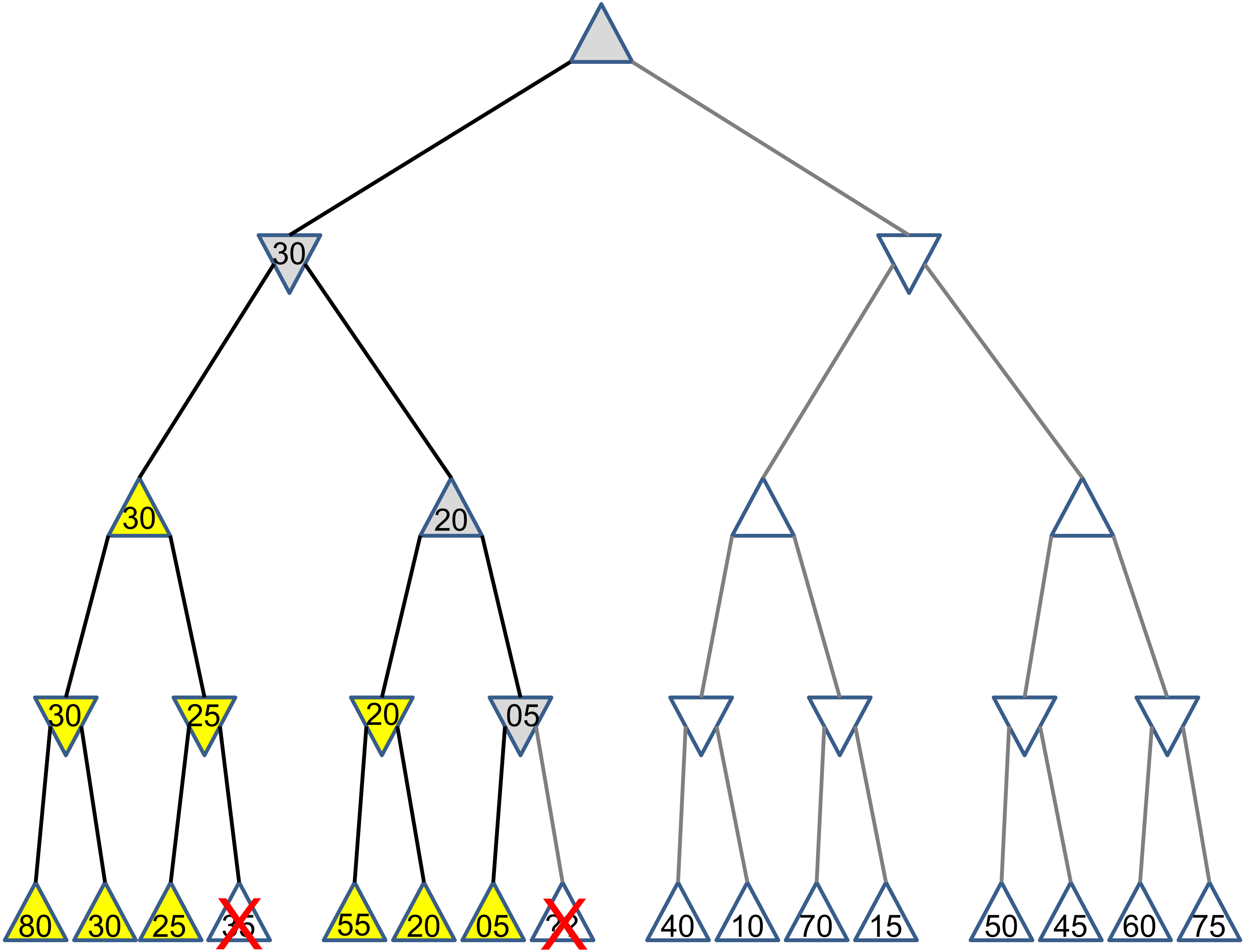

30
30
25
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75


30
Do we need to check this node?
30
25
80
30
25
??
55
20
05
65
40
10
70
15
50
45
60
75



No - this branch is guaranteed to be worse than what max already has


30
30
20
30
25
20
05
80
30
25
35
55
20
05
??
40
10
70
15
50
45
60
75
Do we need to check this node?


30
30
20
30
25
20
05
80
30
25
55
20
05
40
10
70
15
50
45
60
75

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

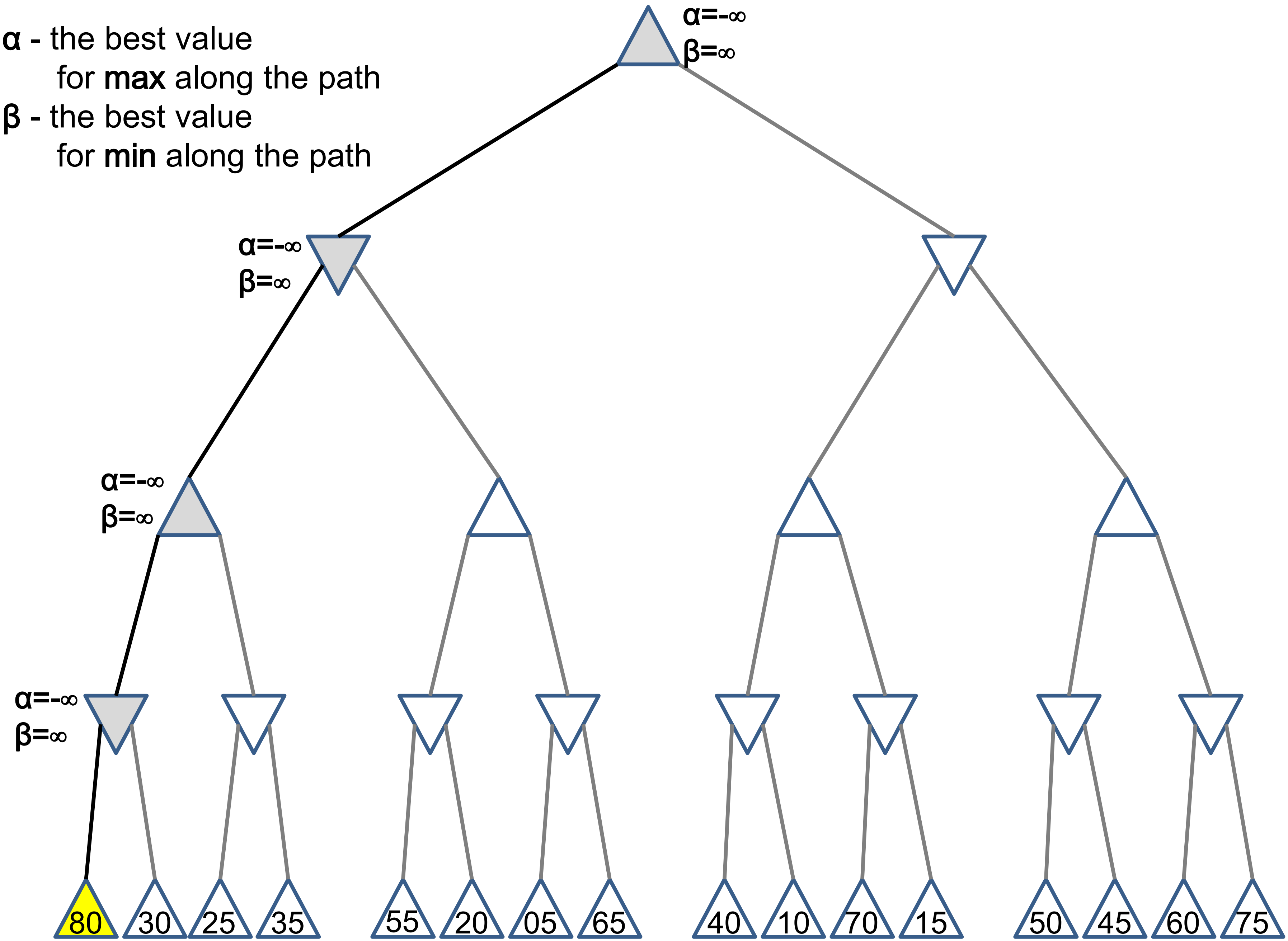
Alpha-Beta

```
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

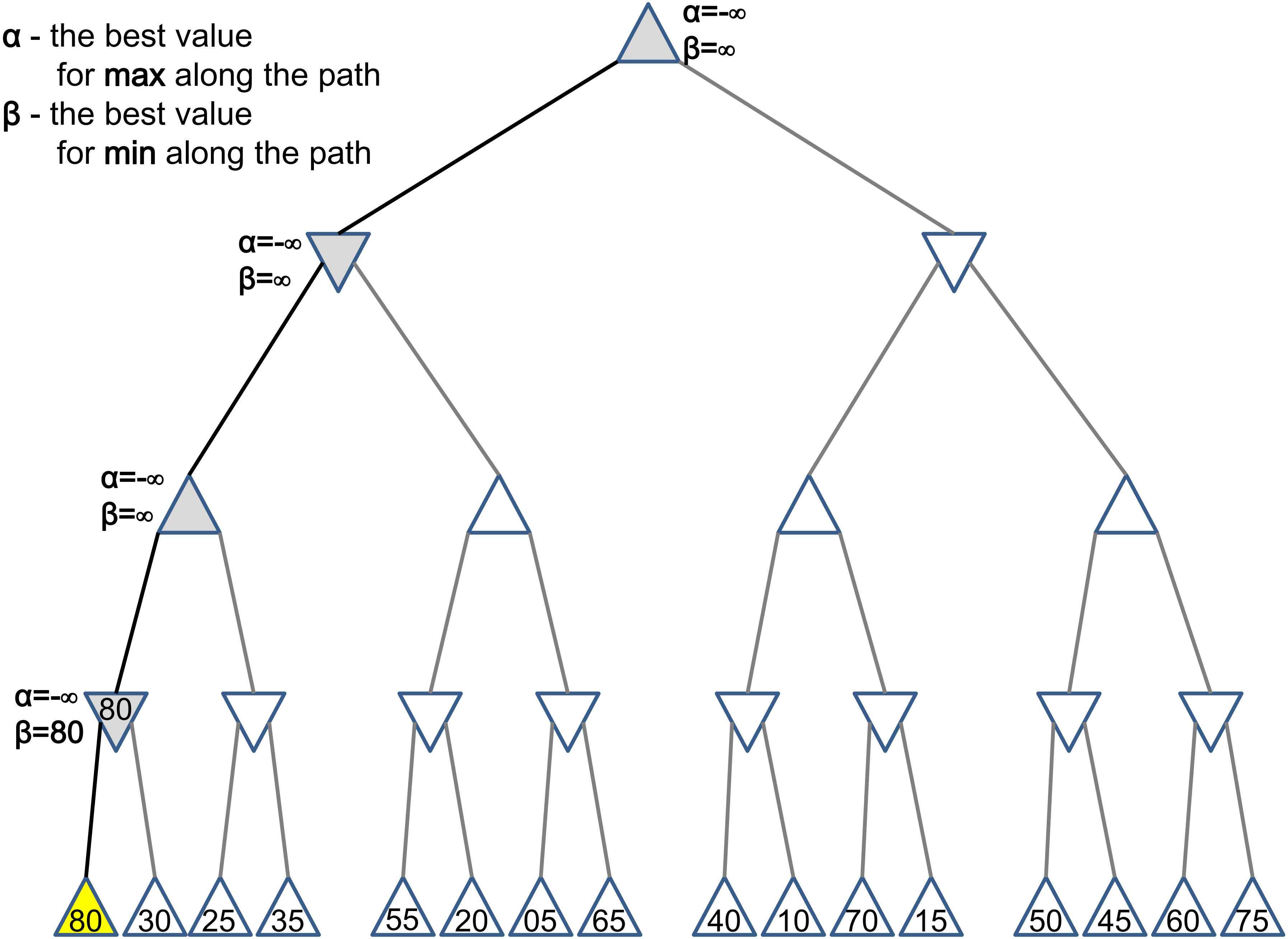
beta = the **lowest** value for **MIN** along the path

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



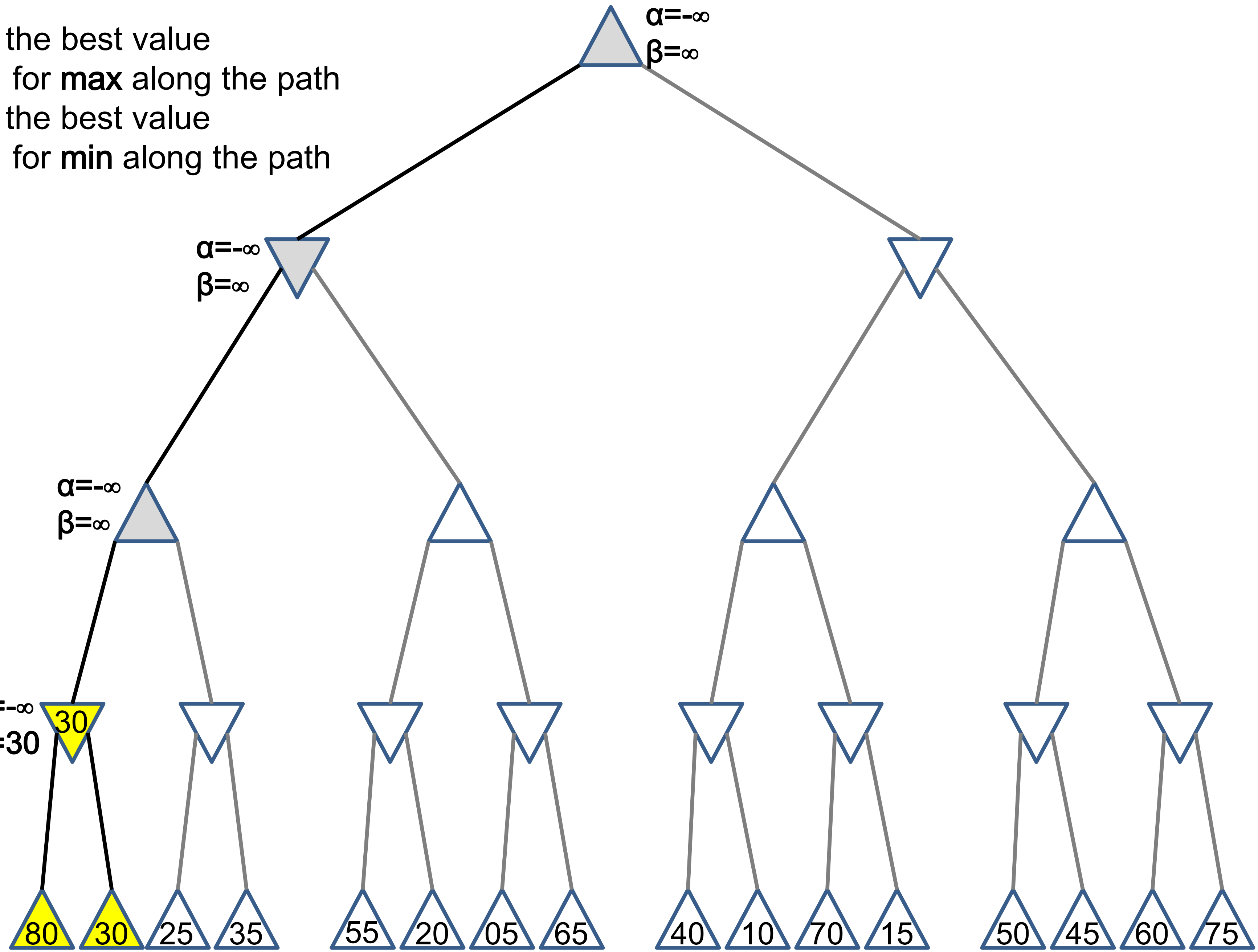
α - the best value
for **max** along the path

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



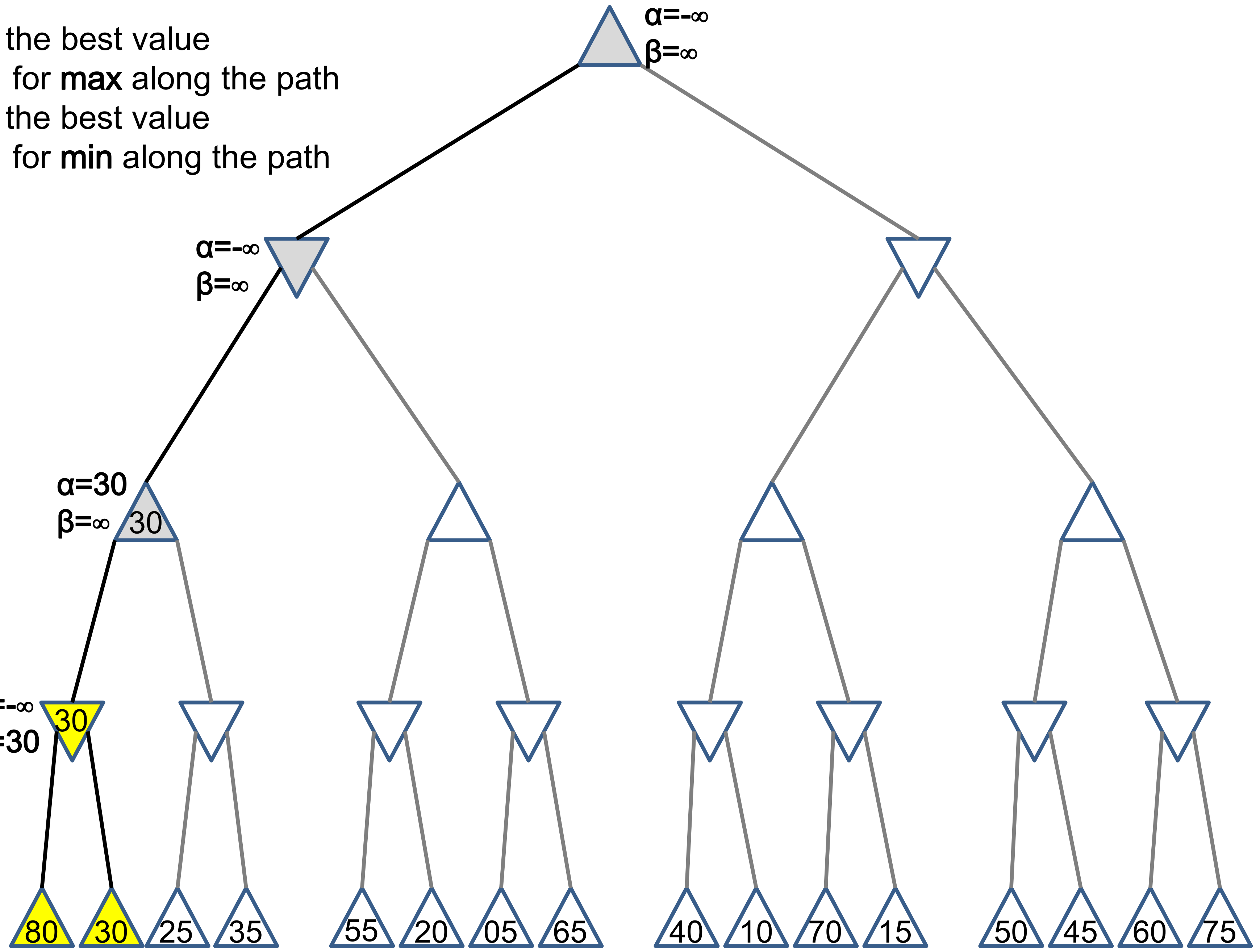
α - the best value
for **max** along the path

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



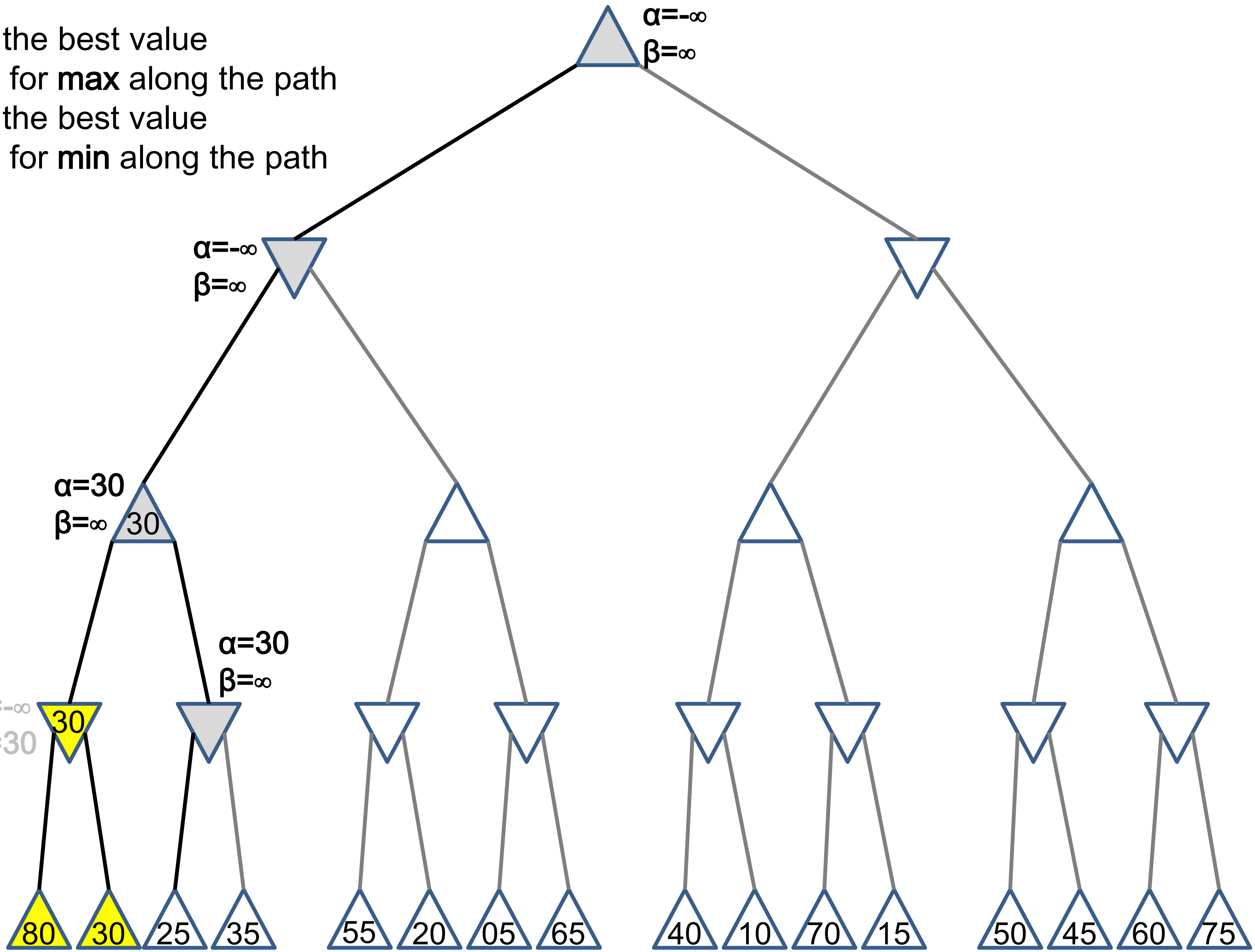
α - the best value
for **max** along the path

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



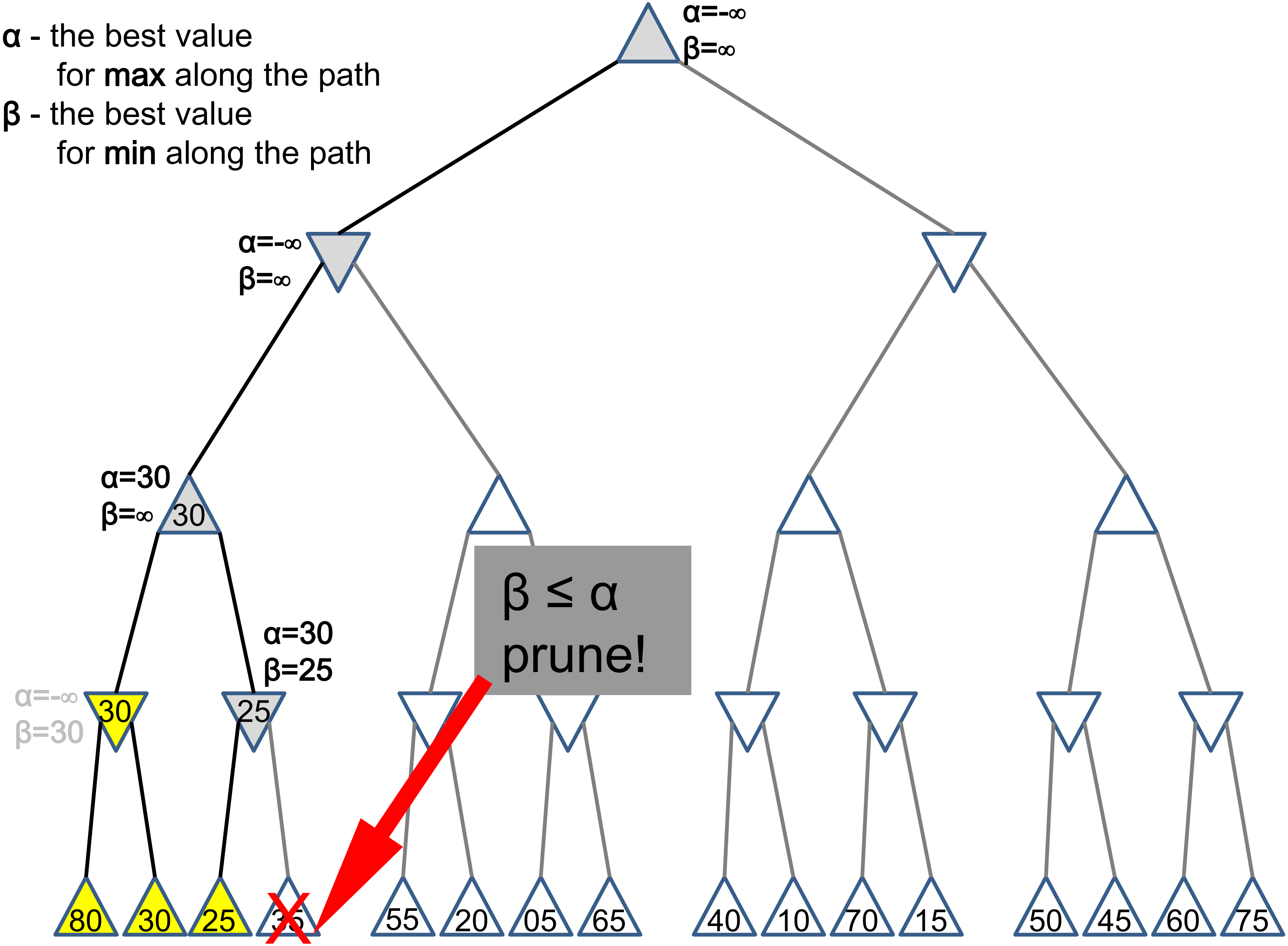
α - the best value
for **max** along the path

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

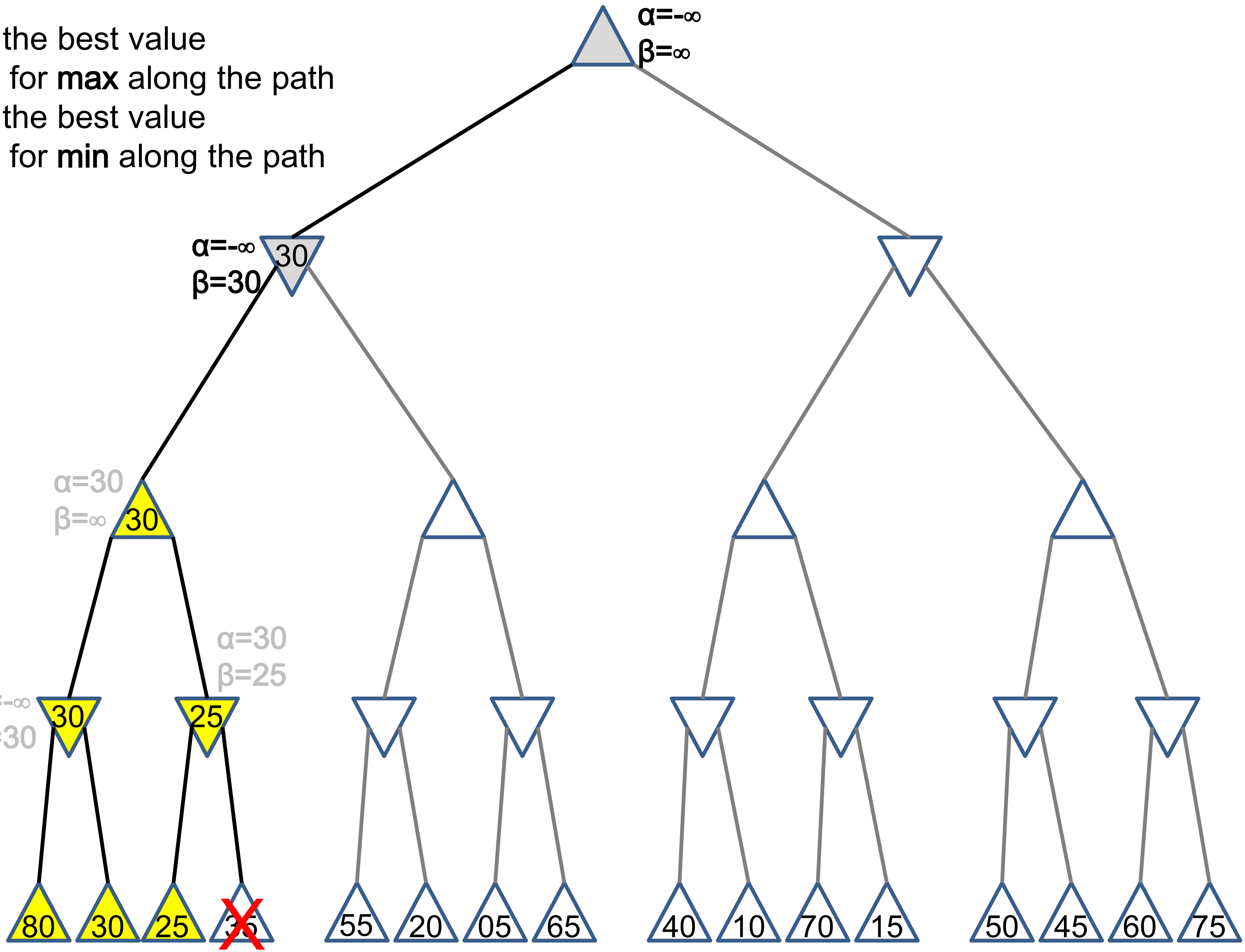


α - the best value
for **max** along the path

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

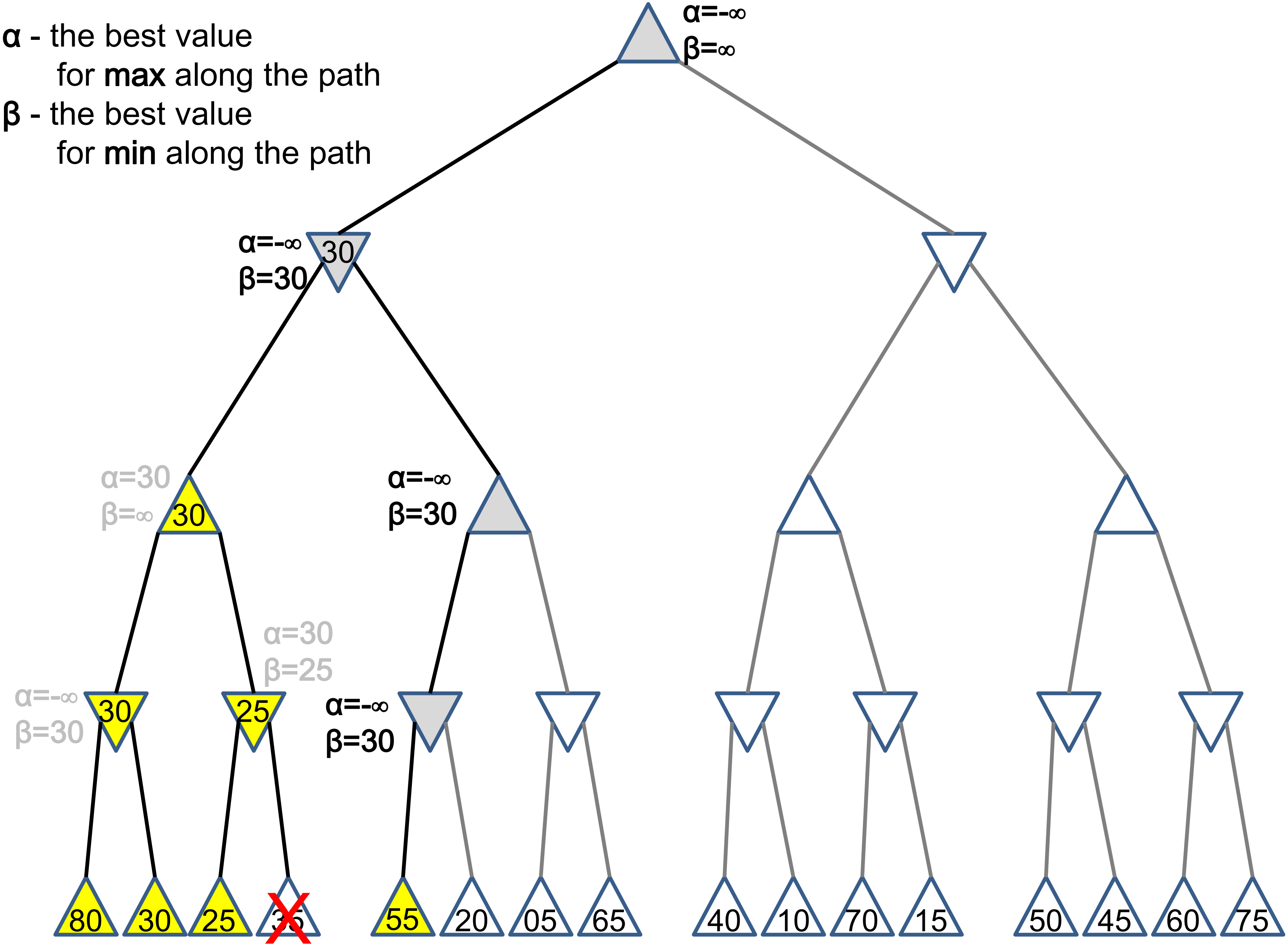


α - the best value
for **max** along the path
β - the best value
for **min** along the path

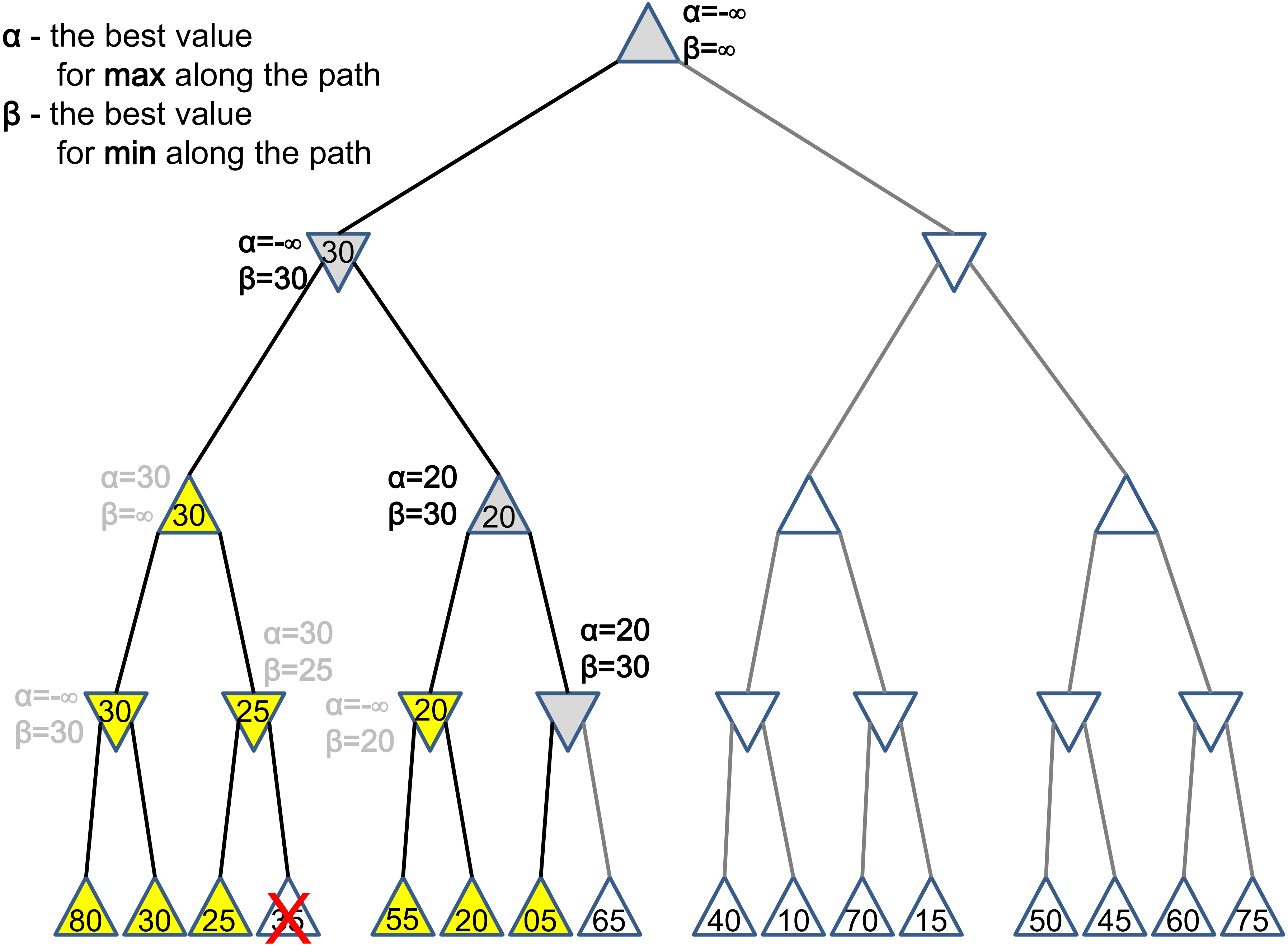


α - the best value
for **max** along the path

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

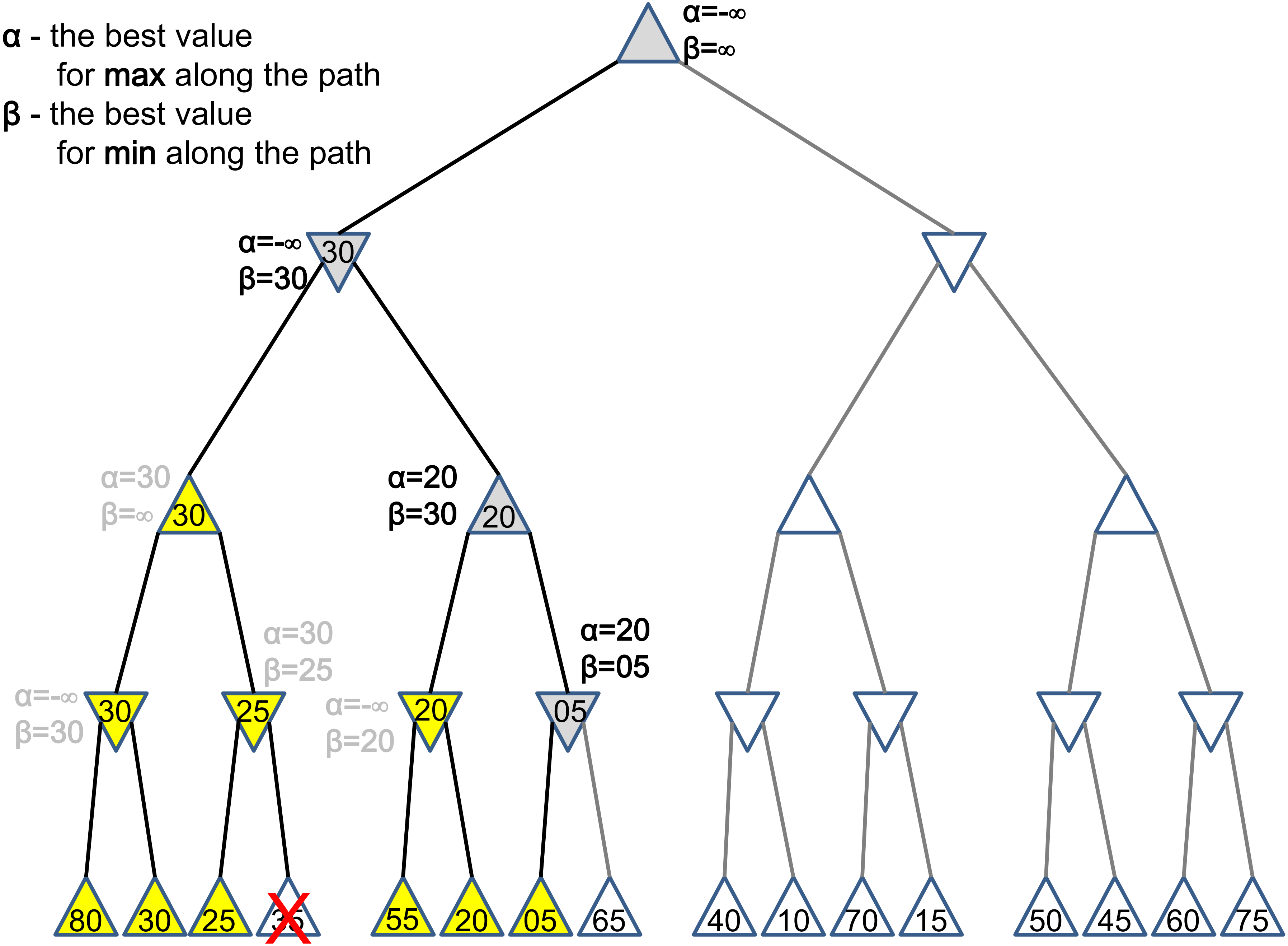


α - the best value
for **max** along the path
β - the best value
for **min** along the path

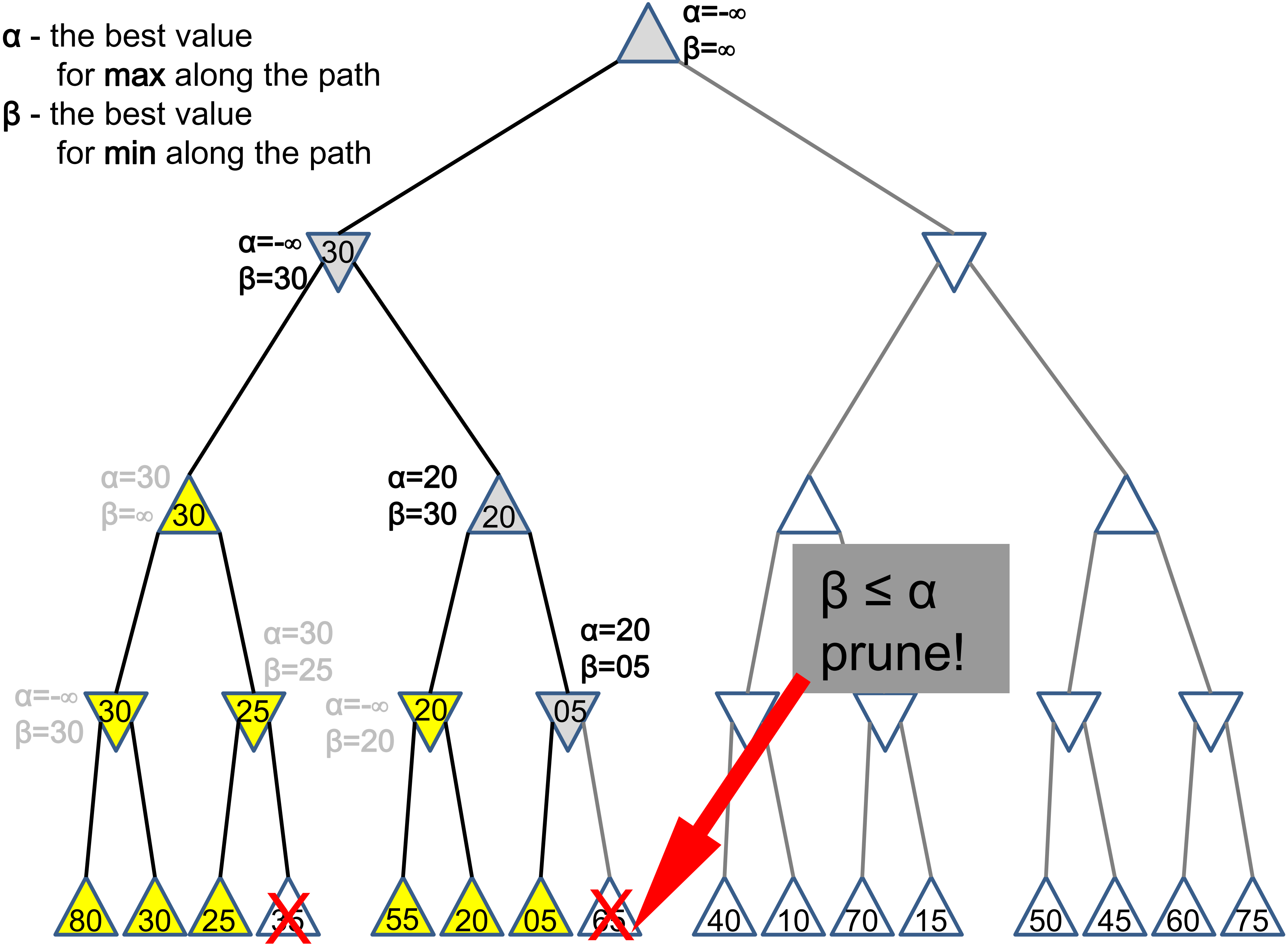


α - the best value
for **max** along the path

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

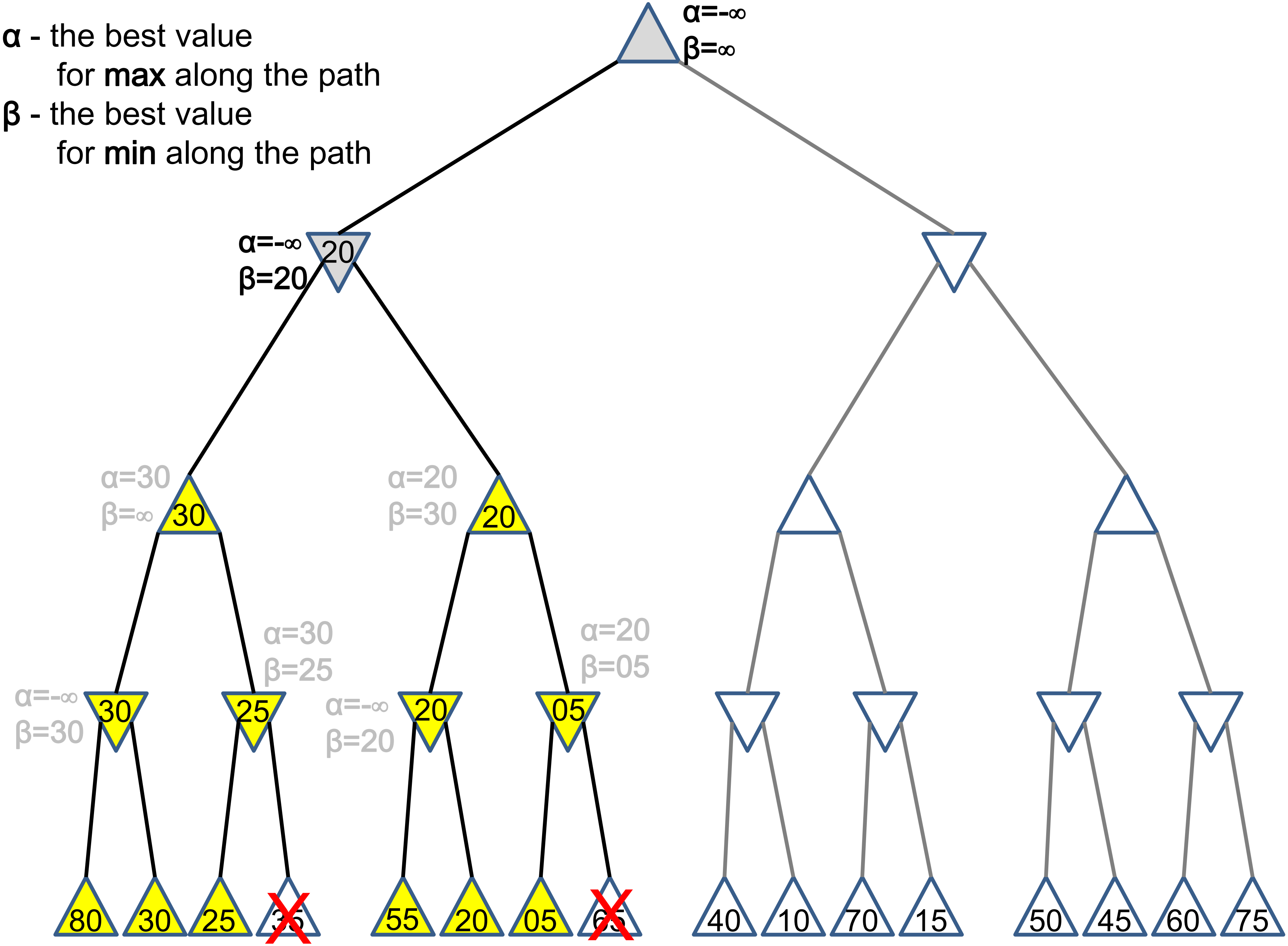


α - the best value
for **max** along the path
β - the best value
for **min** along the path



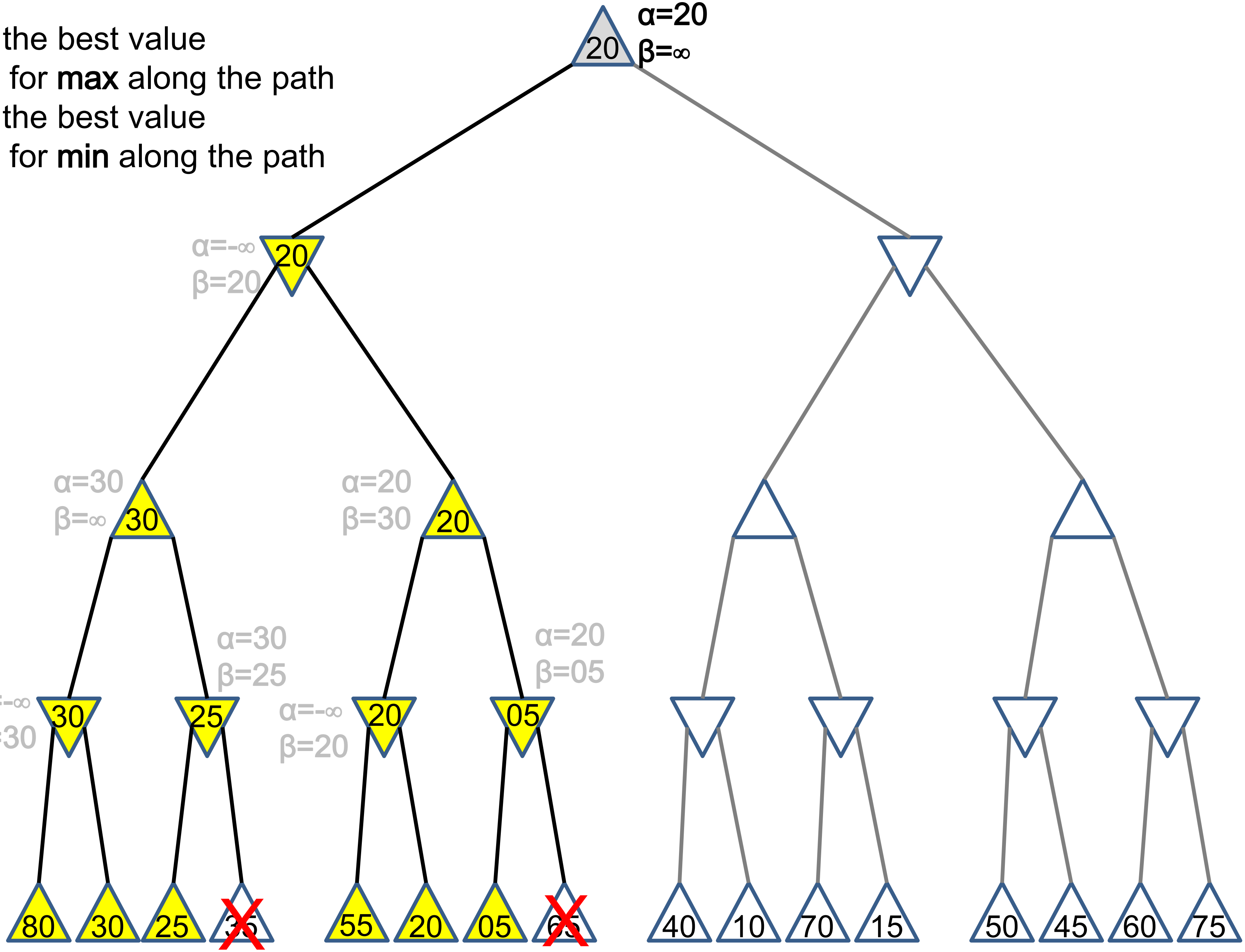
α - the best value
for **max** along the path

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

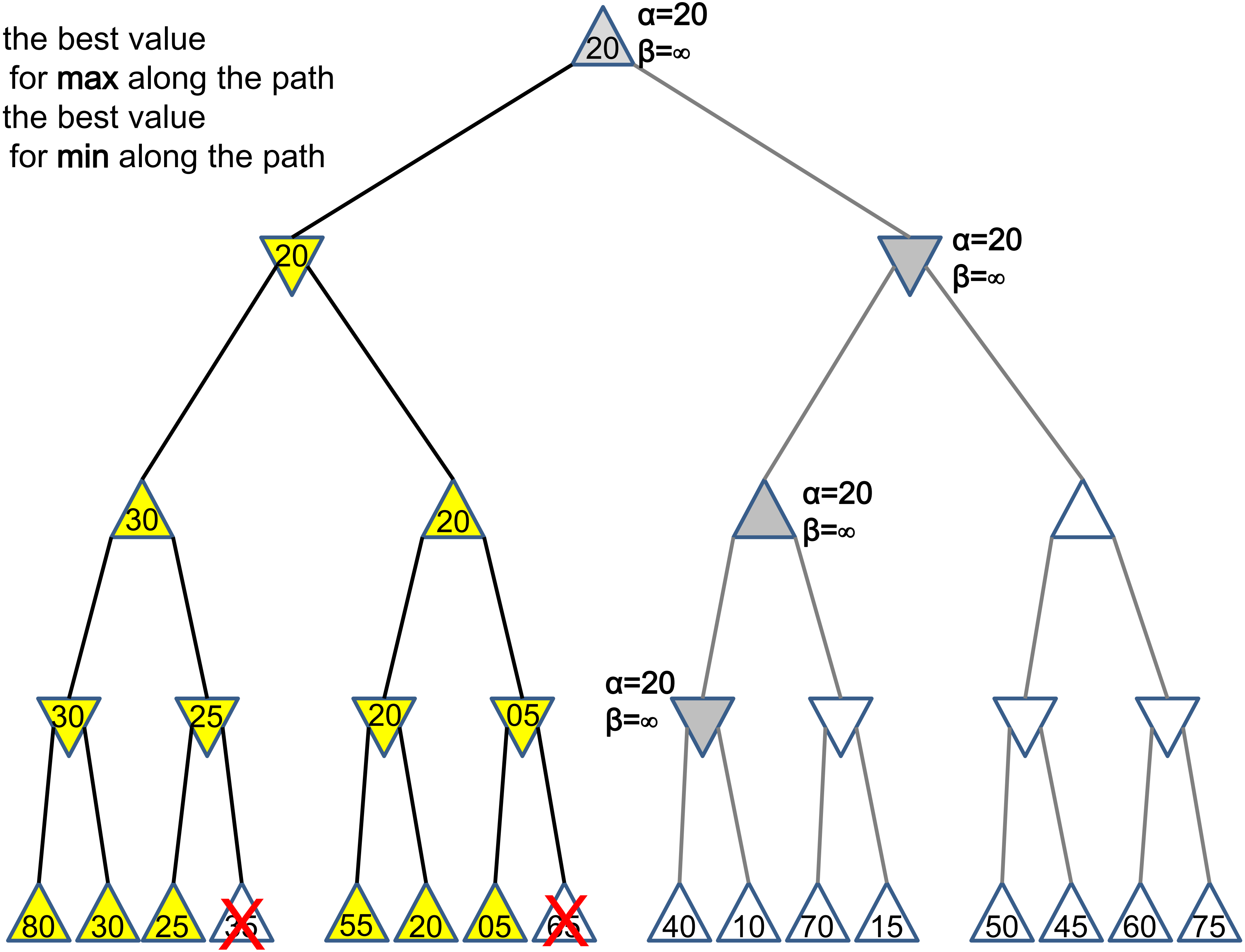


α - the best value
for **max** along the path

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

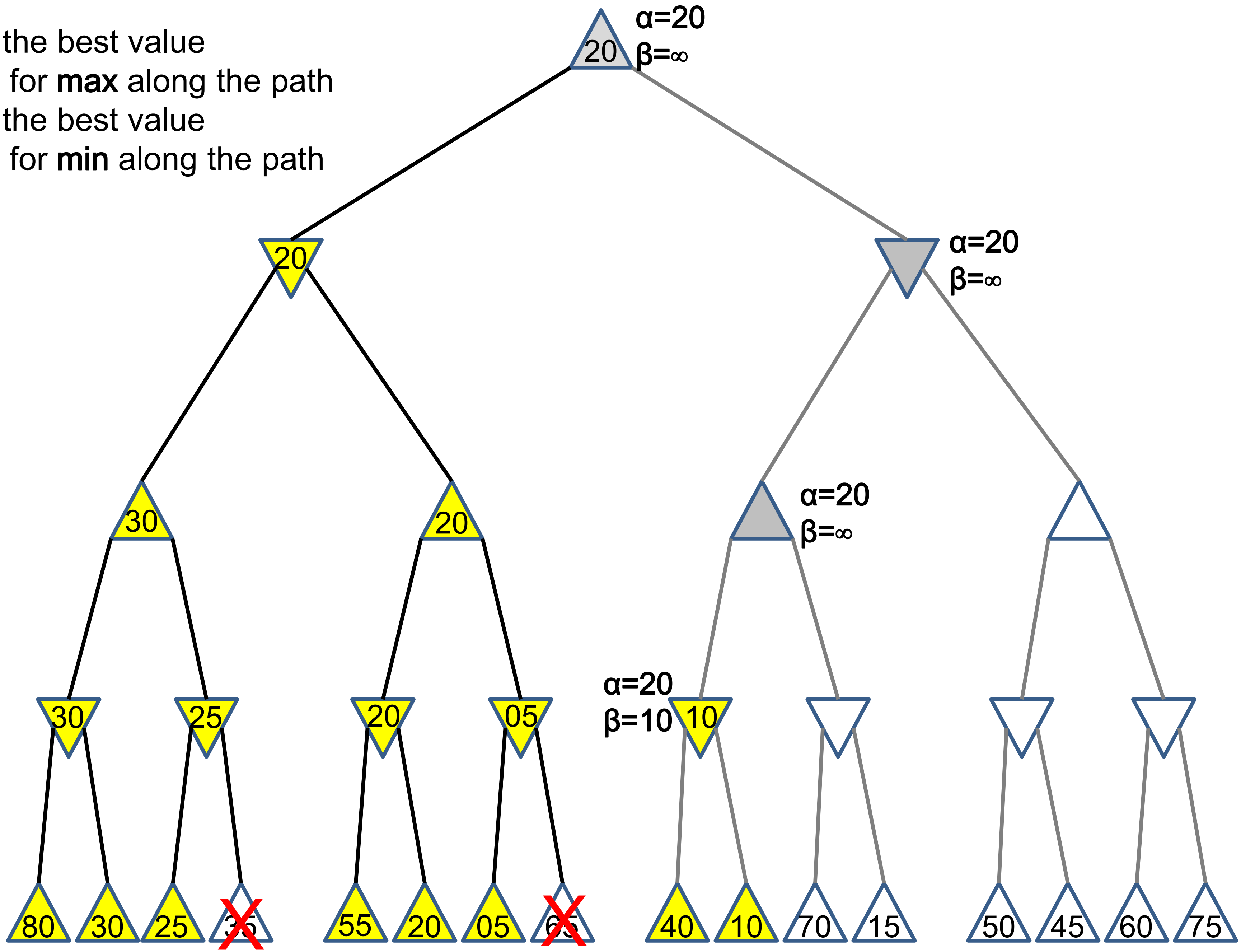


α - the best value
for **max** along the path
β - the best value
for **min** along the path



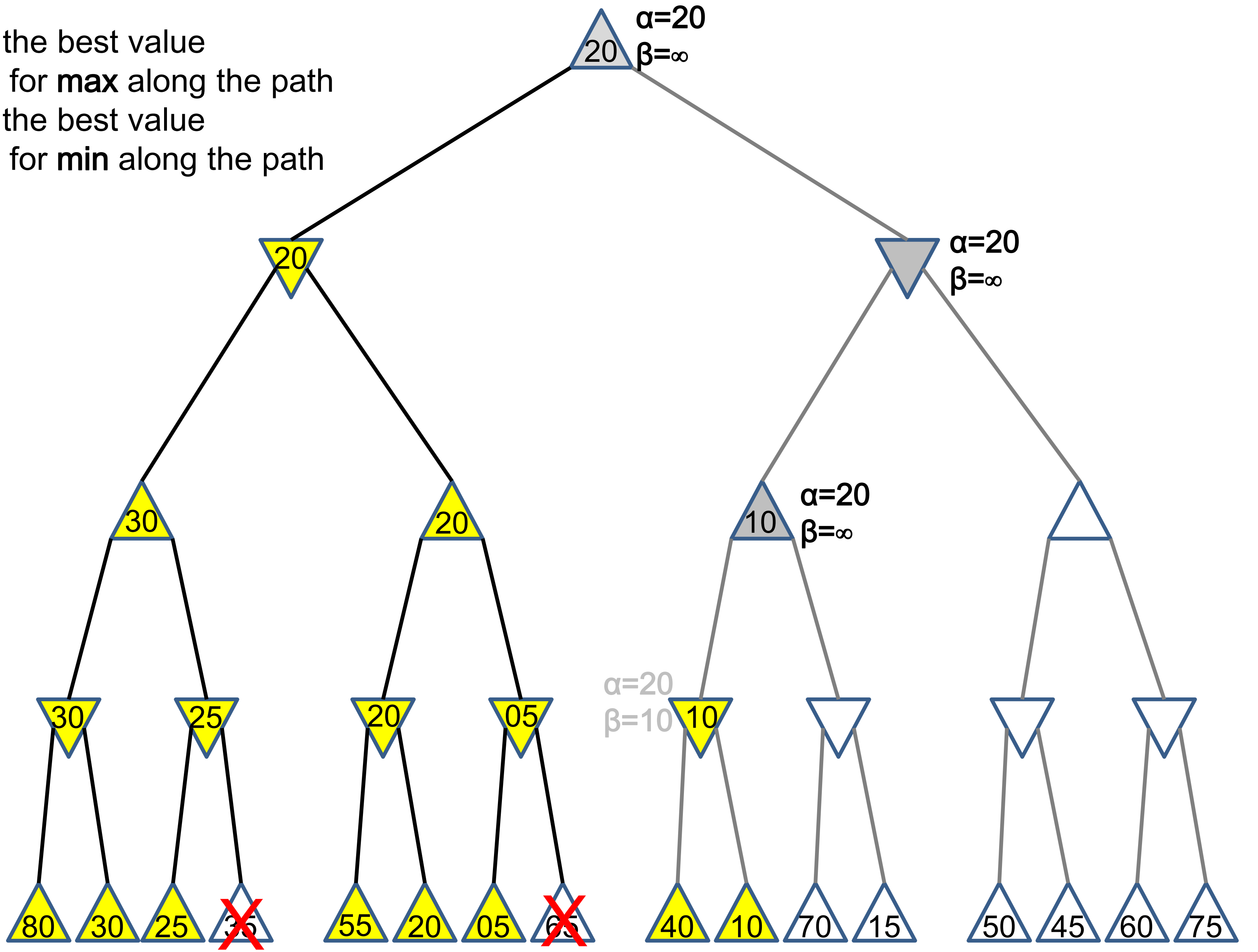
α - the best value
for **max** along the path

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



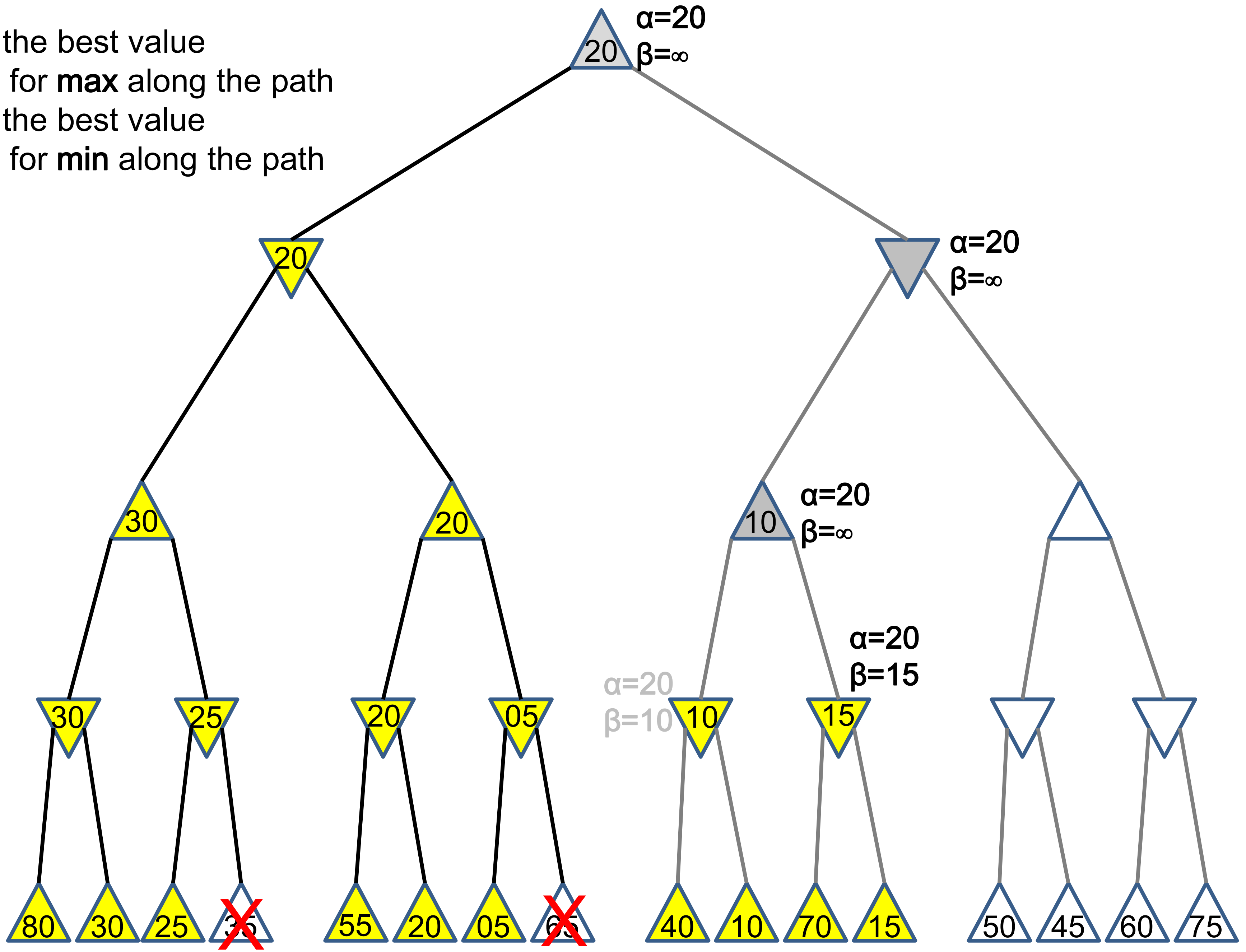
α - the best value
for **max** along the path

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



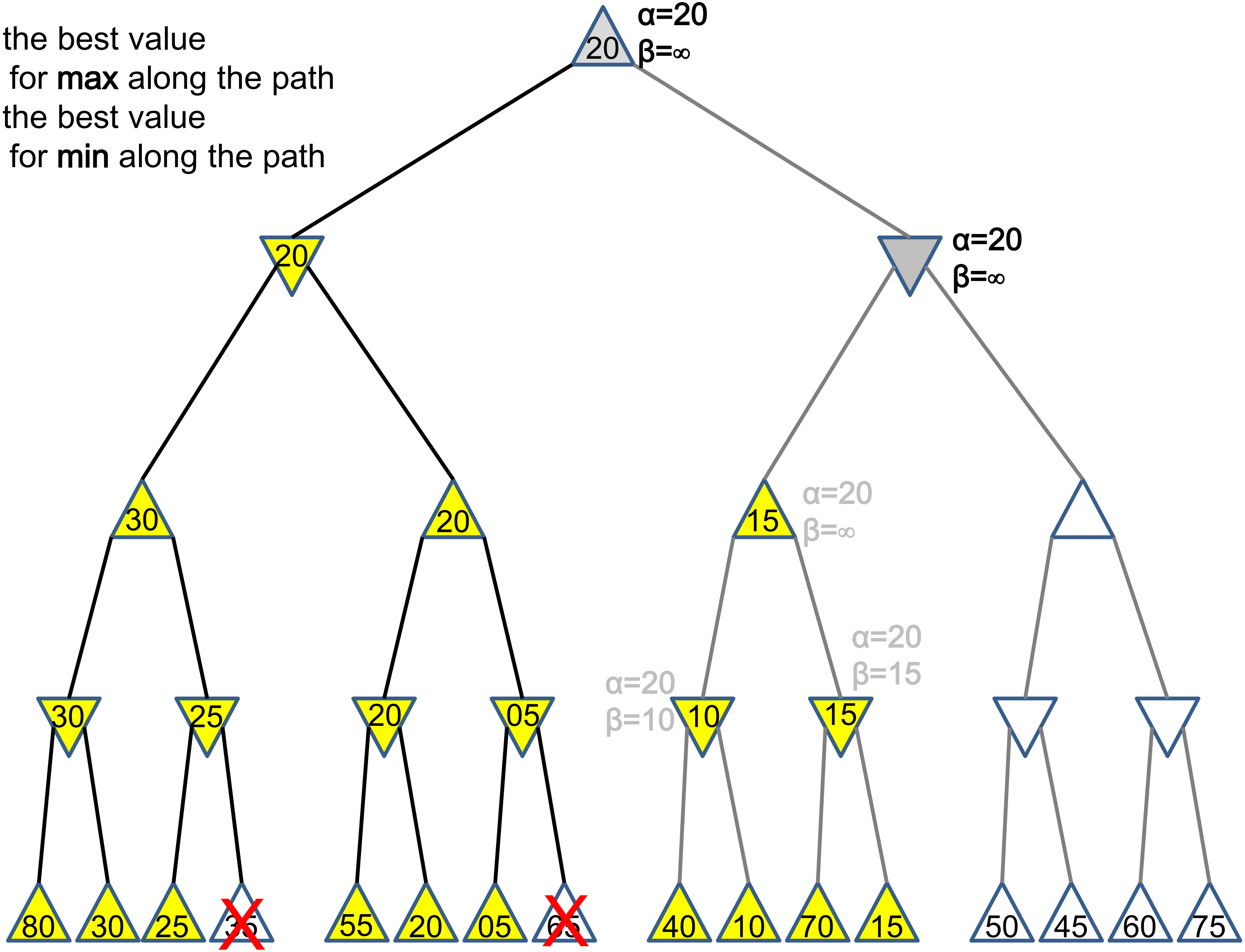
α - the best value
for **max** along the path

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



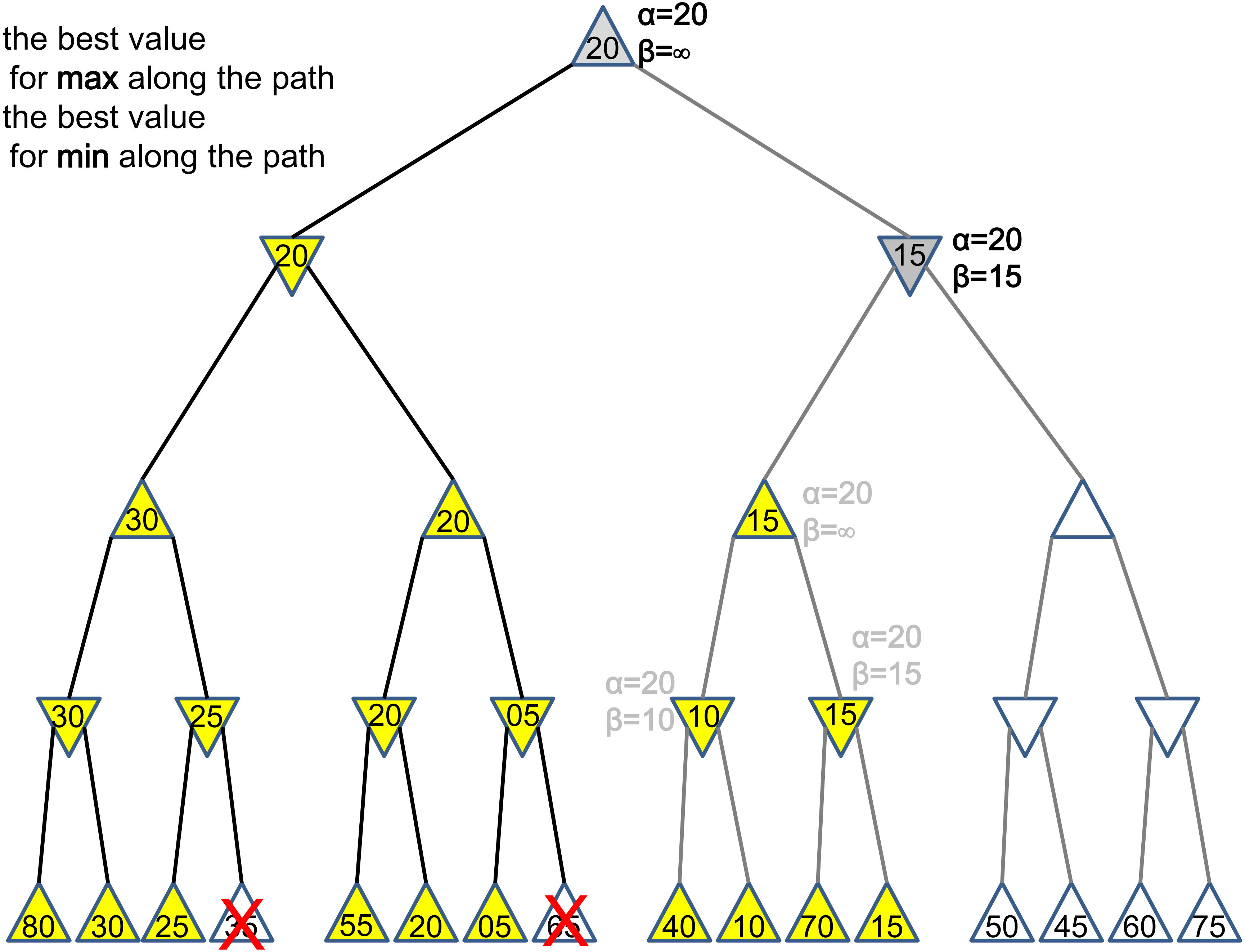
α - the best value
for **max** along the path

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

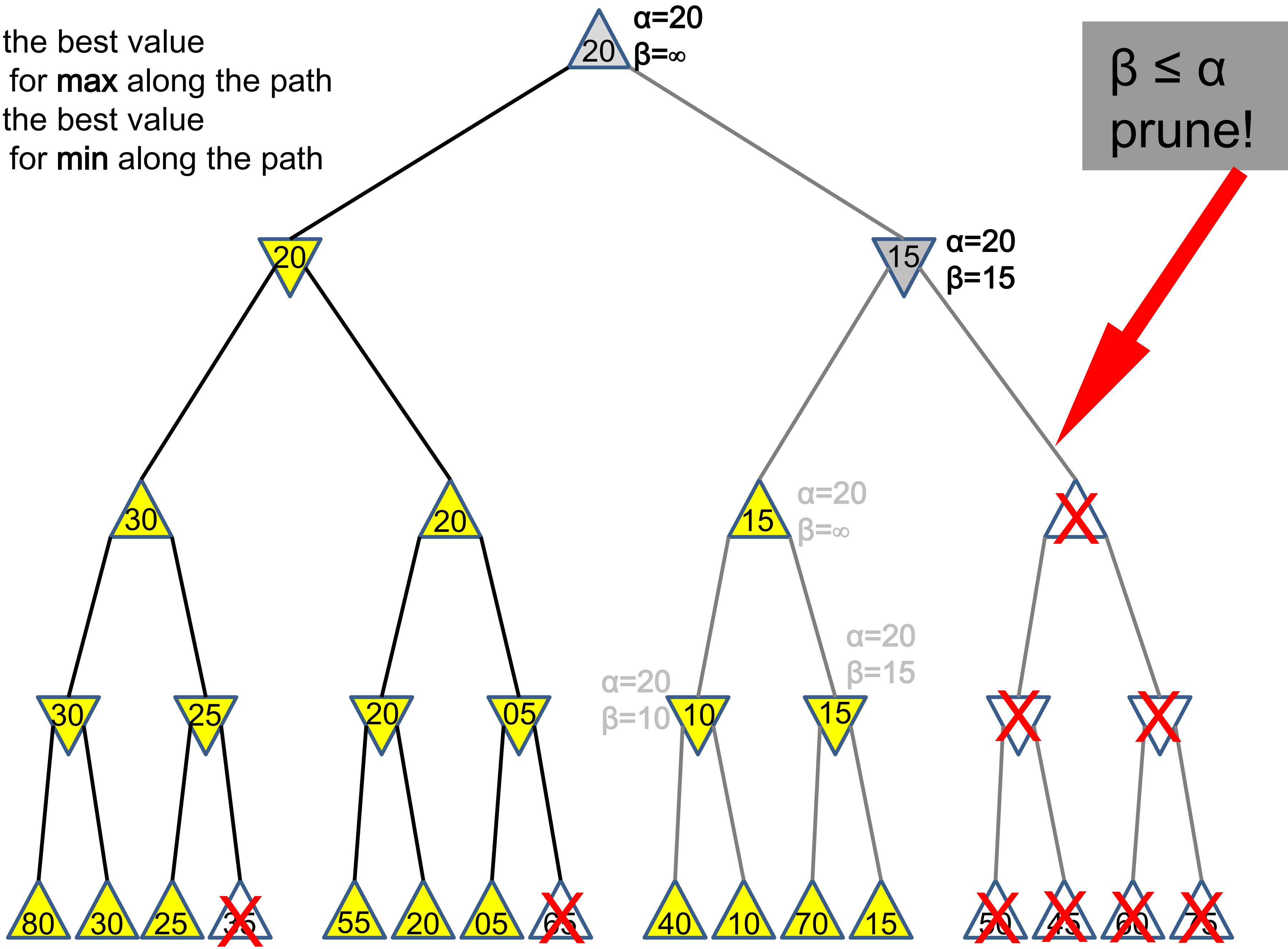


α - the best value
for **max** along the path

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

```
                      4                       MAX
            +--------------+--------------+
            2              3              4         MIN
       +----+----+     +----+----+     +----+----+
       6    4    2     7    5    3     8    6    4       MAX
     +--+  +--+  +--+ +-+-+ +--+ +--+ +--+  +--+ +--+--+
     6  5  4  3  2  1 1 3 7  4  5  2  3  8  2  1  6 1  2  4
```

- Good: Good moves ordered first

```
                      4                       MAX
            +--------------+--------------+
            4              3              2         MIN
       +----+----+     +----+----+     +----+----+
       4    6    8    3    x    x    2    x    x       MAX
     +--+  +--+  +--+ +--+          +-+-+
     4  2  6  x  8  x 3  2          1 2 1
```

- 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