18CSC305J
ARTIFICIAL INTELLIGENCE
UNIT – 1

CLR1 : Provide a broad understanding of the basic techniques for building intelligent computer systems and an understanding of how AI is applied to problems.

CLO1 : Formulate a problem and build intelligent agents

Unit 1 List of Topics

Adversarial Search Methods (Game Theory) - Mini max algorithm - Alpha beta pruning - Constraint satisfactory problems – Constraints – Crypt Arithmetic Puzzles – Constraint Domain – CSP as a search problem (Room colouring). Intelligent Agent – Rationality and Rational Agent – Performance Measures – Rationality and Performance – Flexibility and Intelligent Agents – Task environment and its properties – Types of agents

Definition of adversarial search

- **Searches in which two or more players with conflicting goals are trying to explore the same search space for the solution, are called adversarial searches, often known as Games**.
- Games are modeled as a Search problem and heuristic evaluation function, and these are the two main factors which help to model and solve games in AI
- **What Kinds of Games?**

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 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)

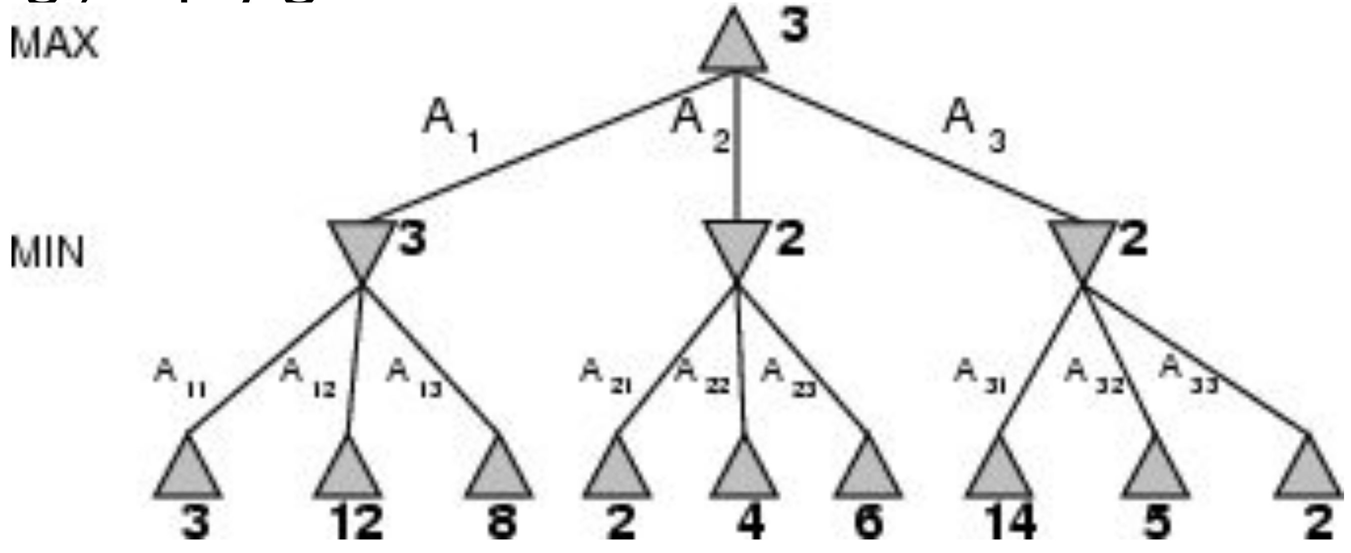
Mini-Max Terminology

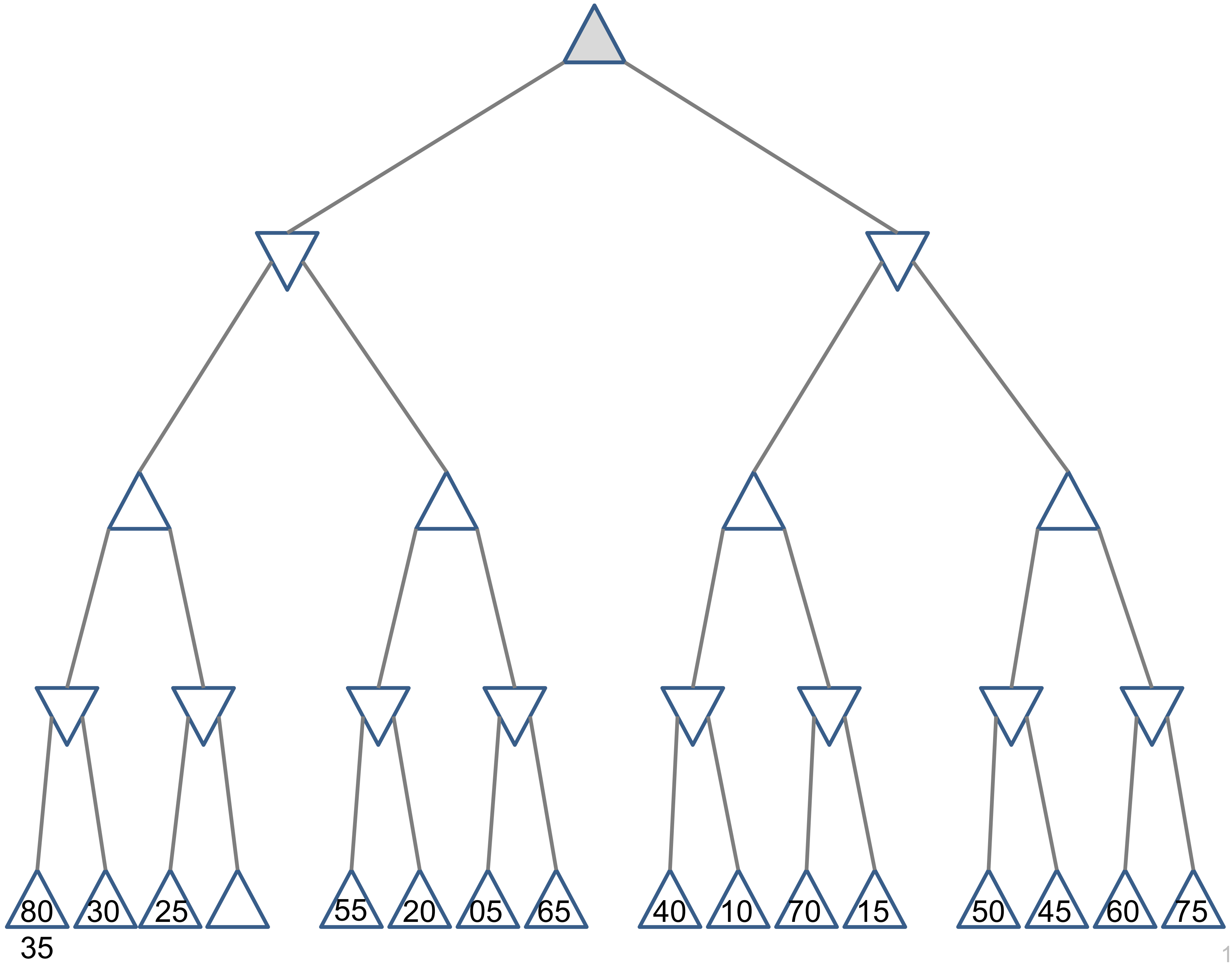
- move: a move by both players
- ply: a half-move
- utility function: the function applied to leaf nodes
- backed-up value
  - of a max-position: the value of its largest successor
  - of a min-position: the value of its smallest successor
- minimax procedure: search down several levels; at the bottom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

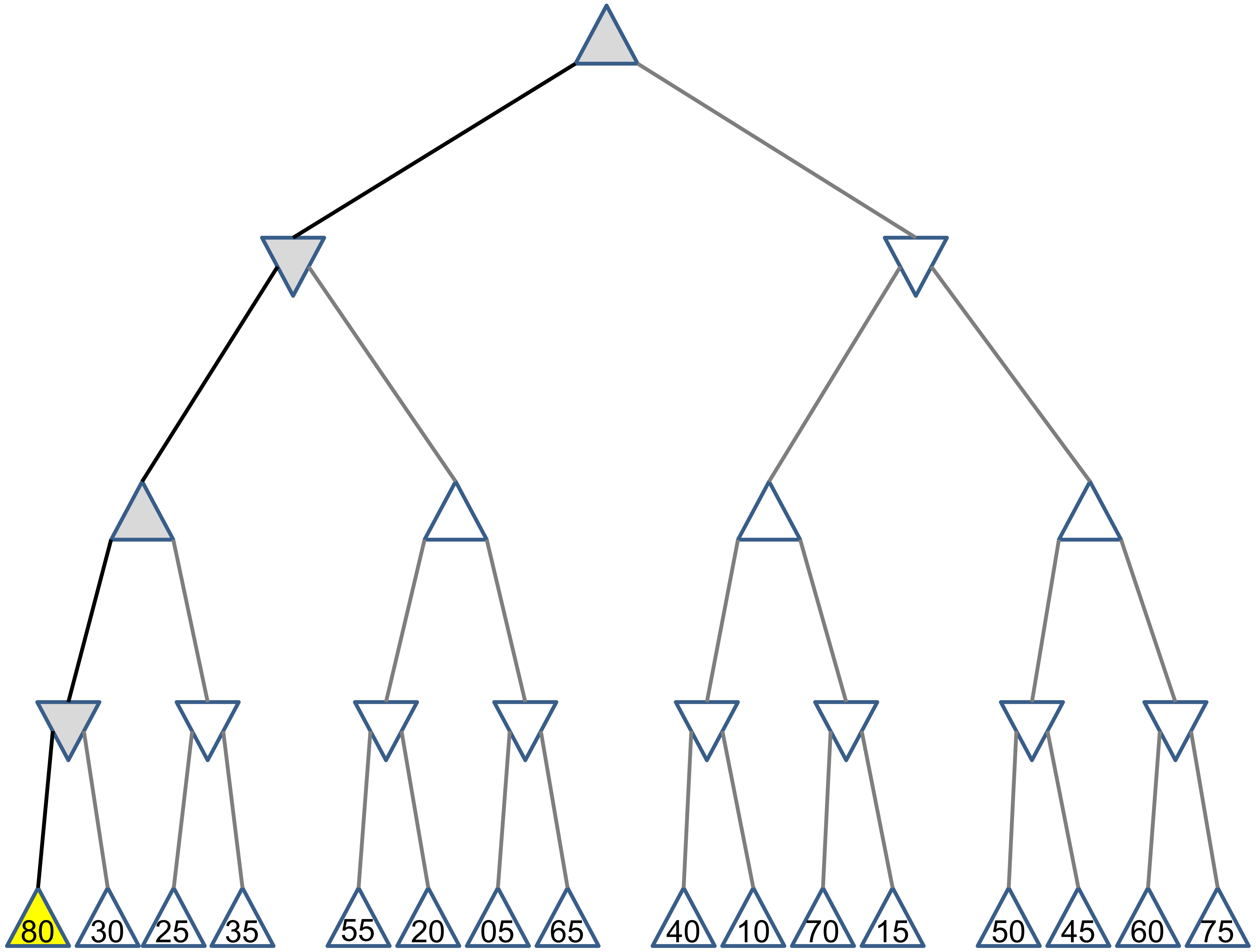
Mini-Max Terminology

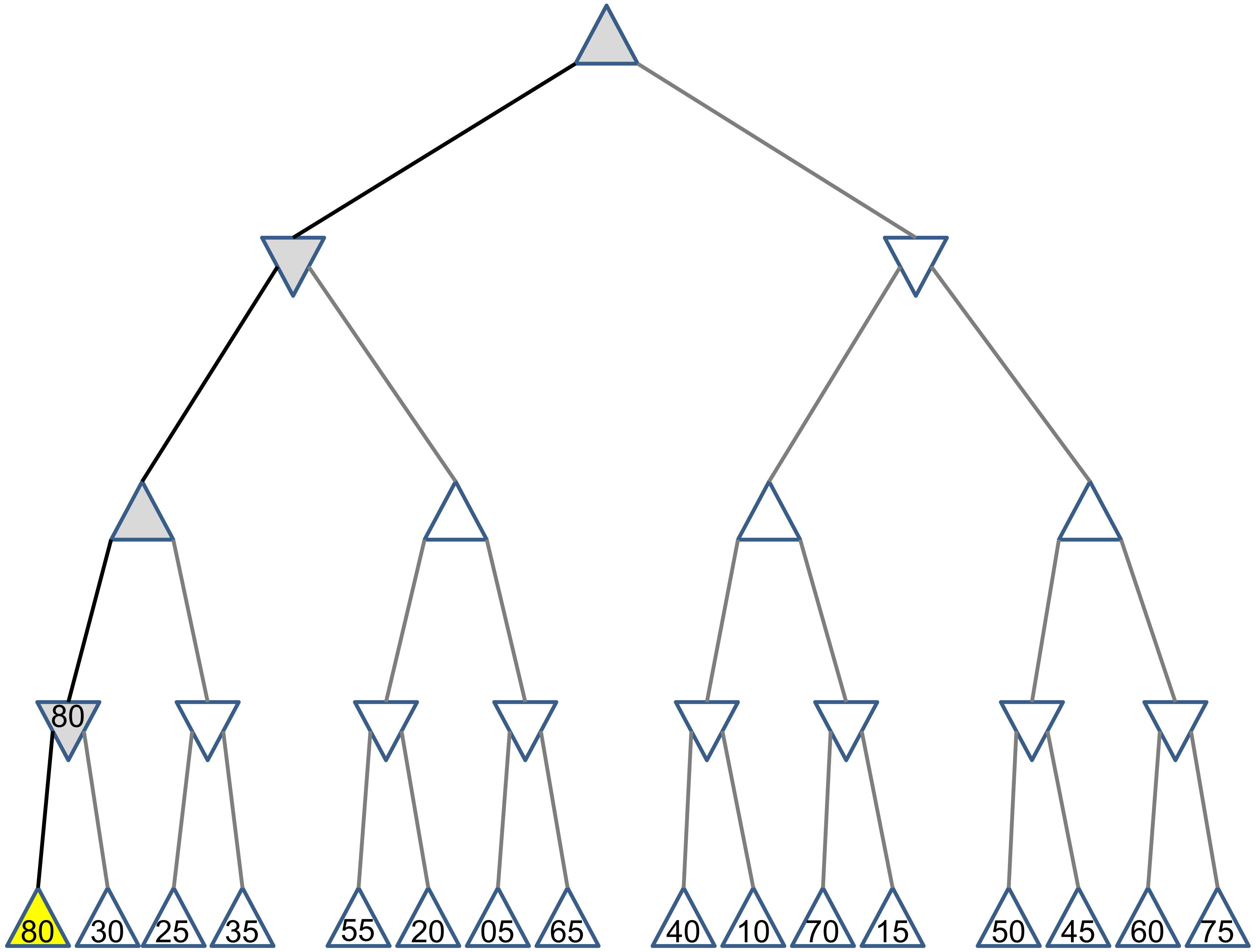
- Perfect play for deterministic games
- Idea: choose move to position with highest minimax value

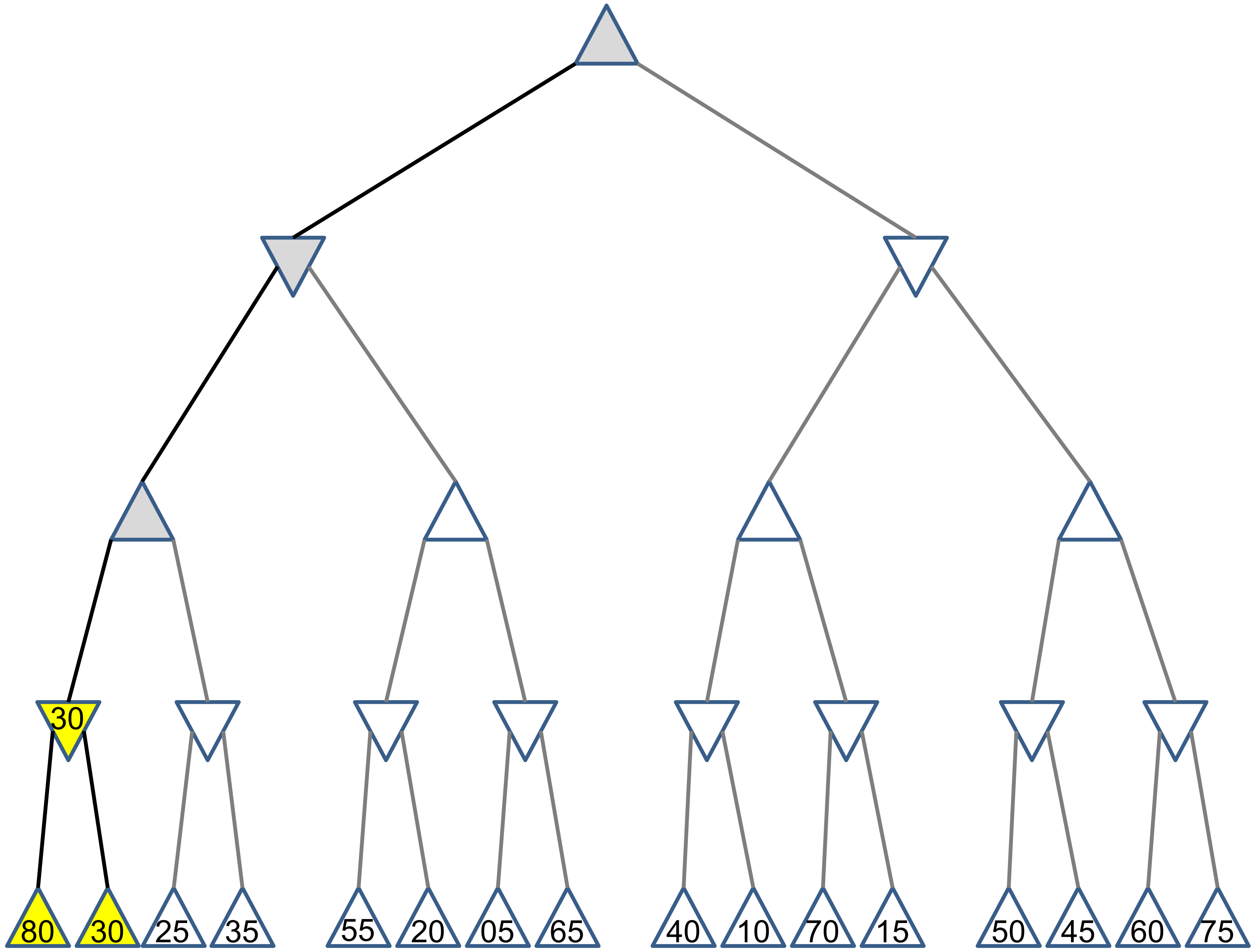
  = best achievable payoff against best play
- E.g., 2-ply game

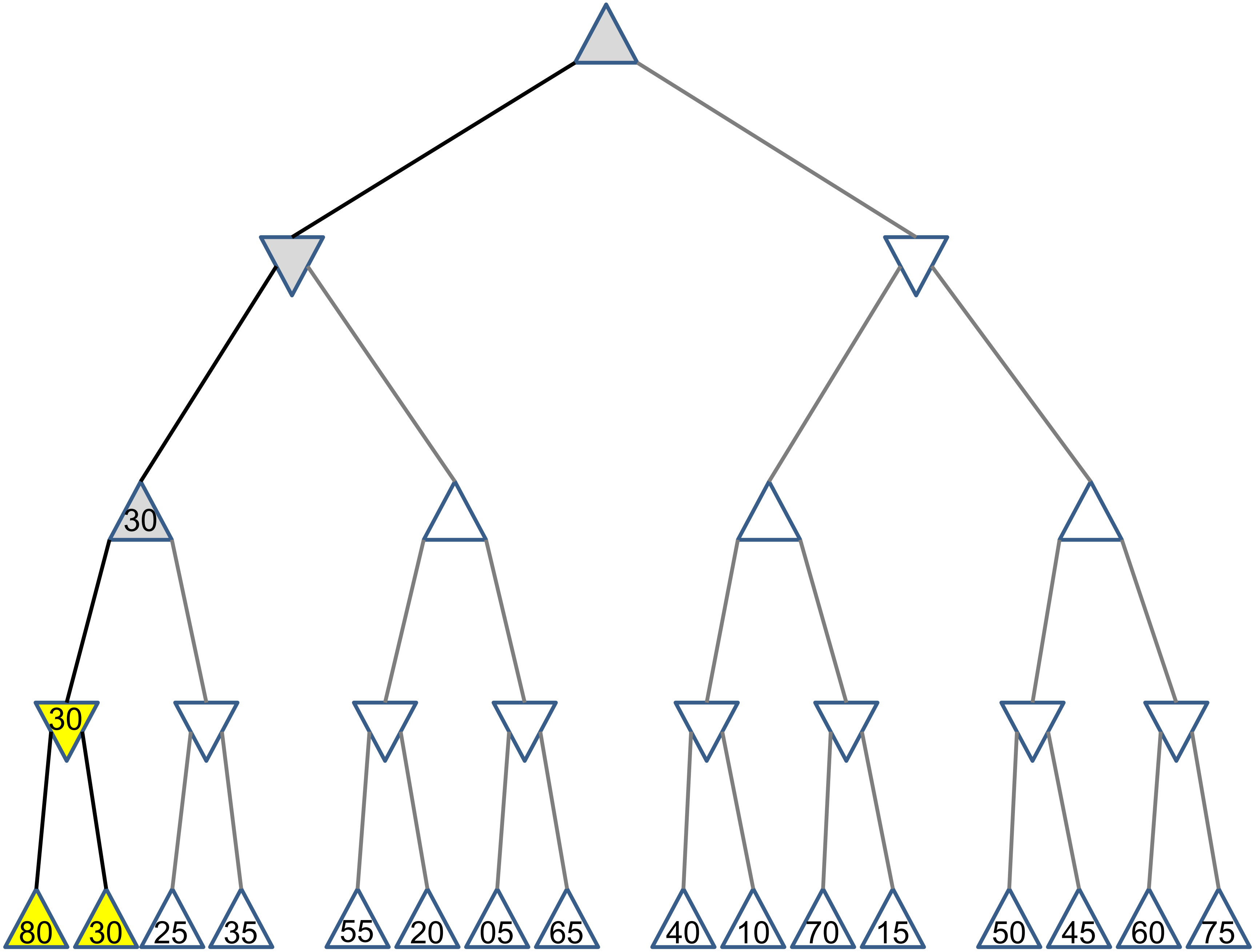


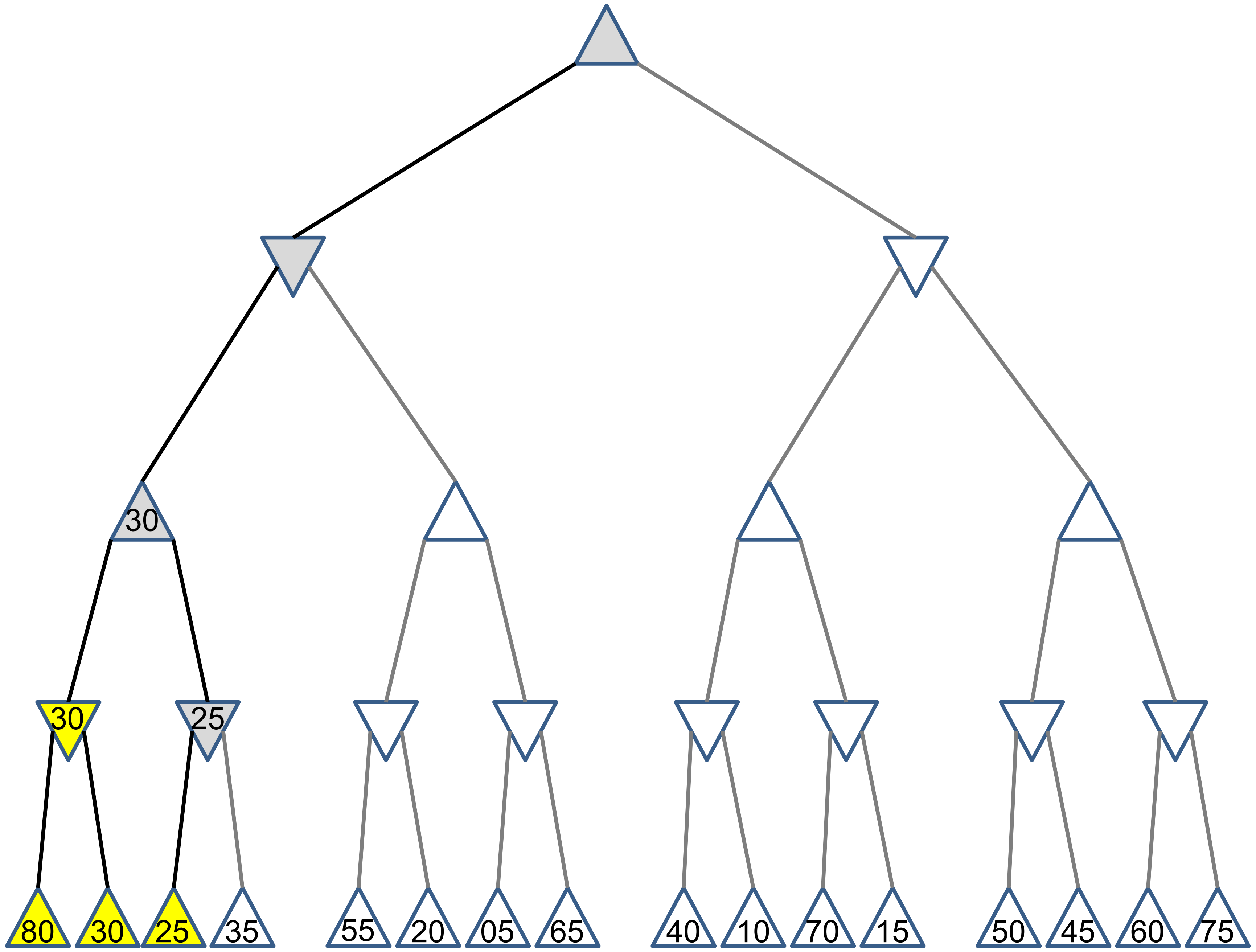

80
35
30
25
55
20
05
65
40
10
70
15
50
45
60
75

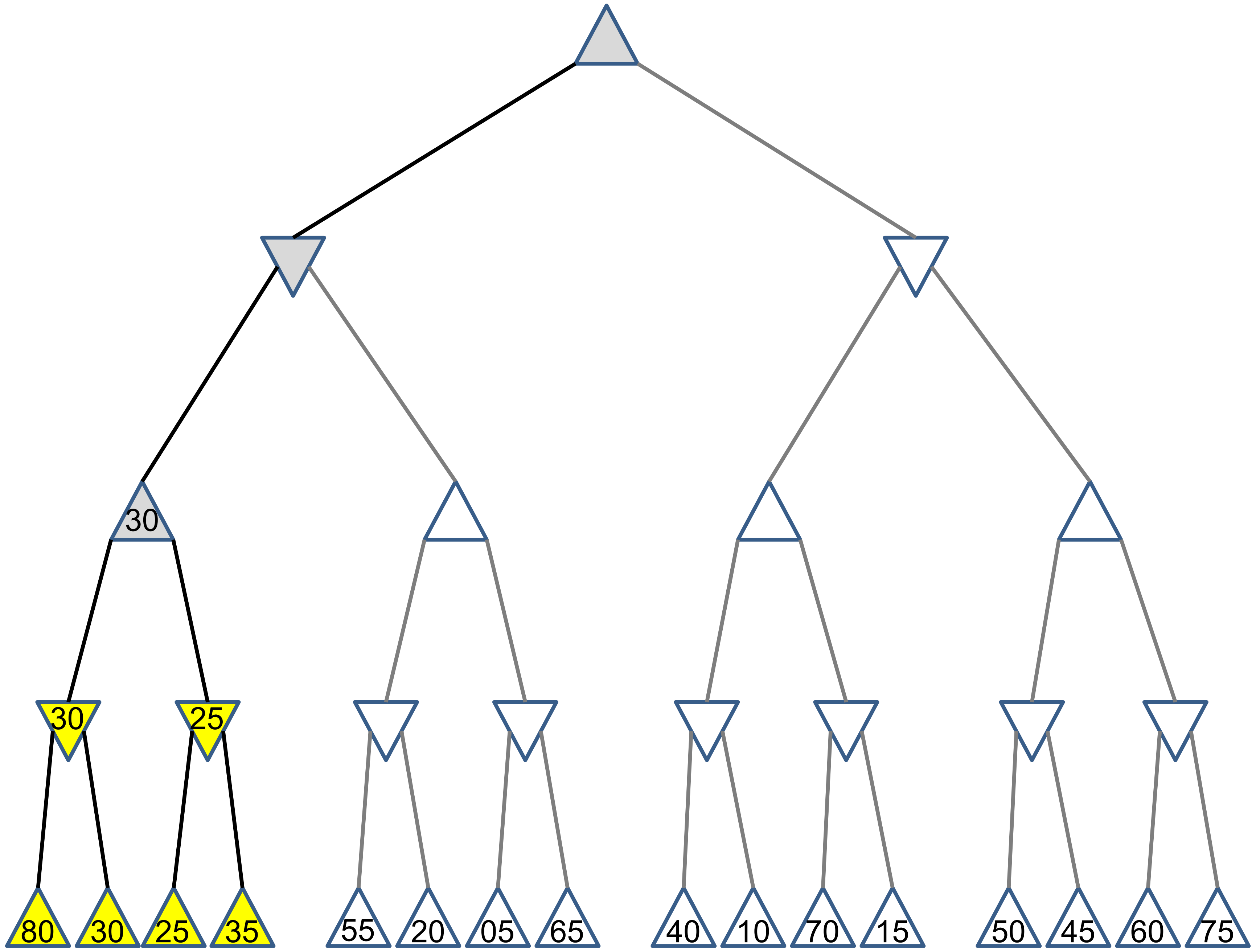

80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

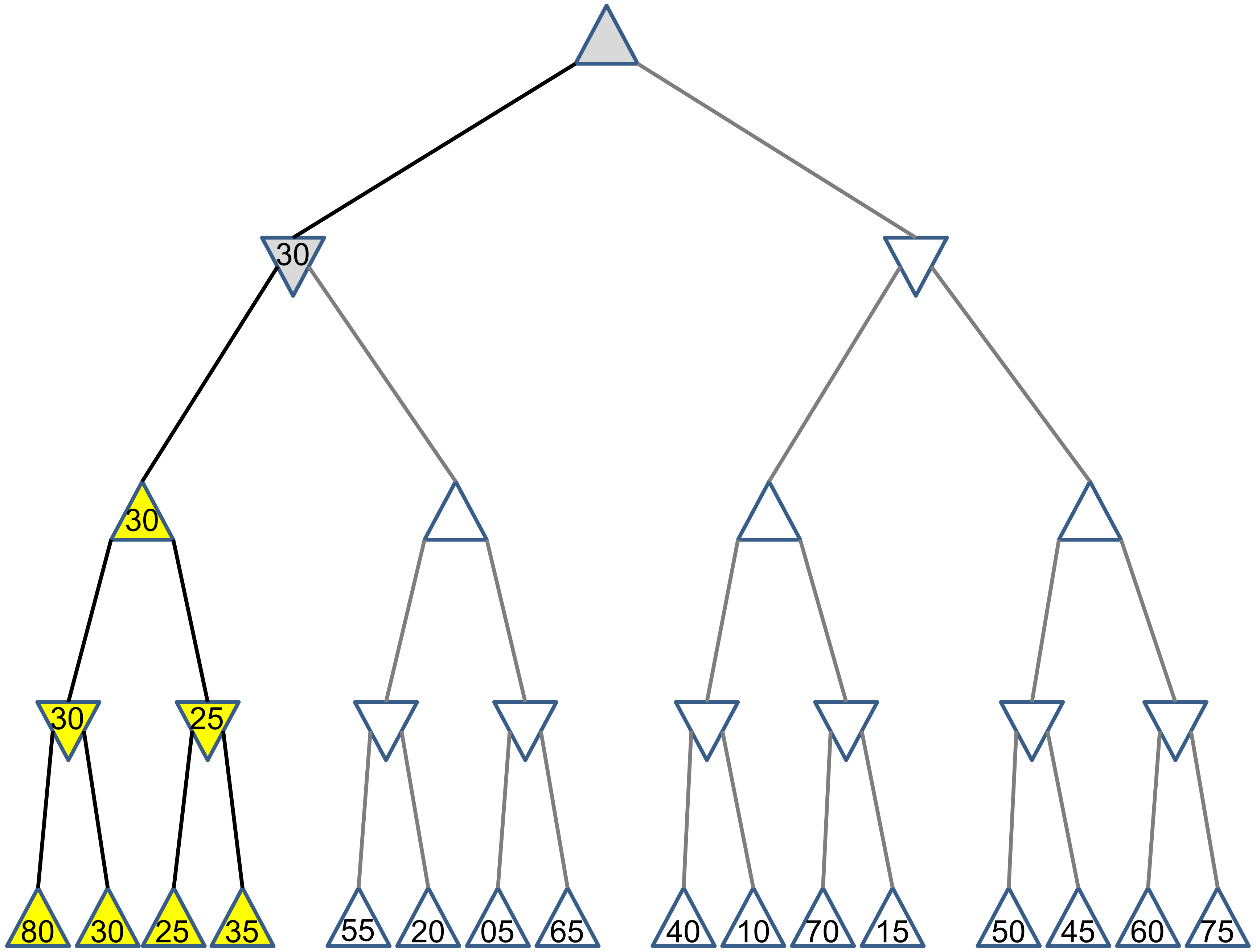

80
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

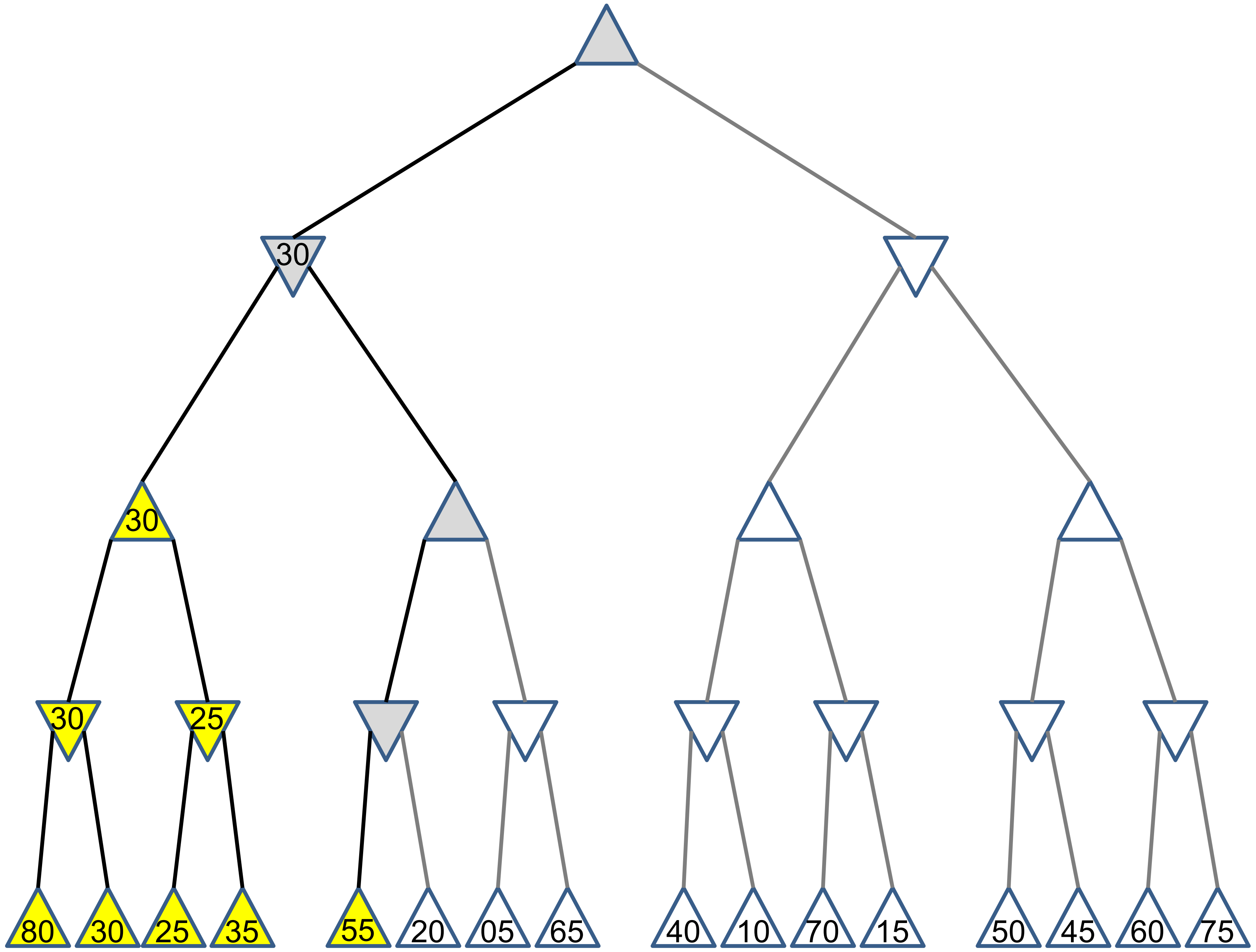

30
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

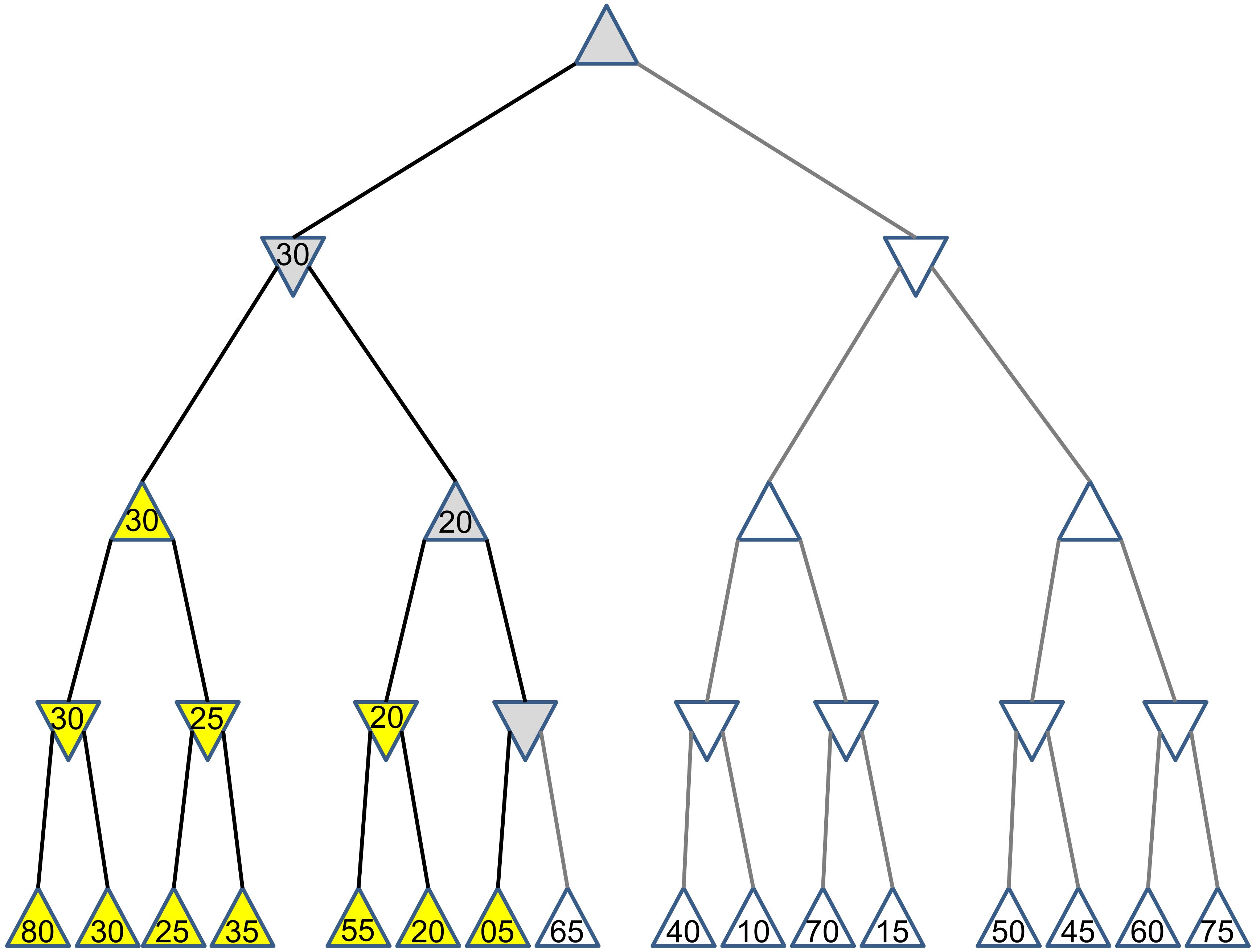

30
30
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

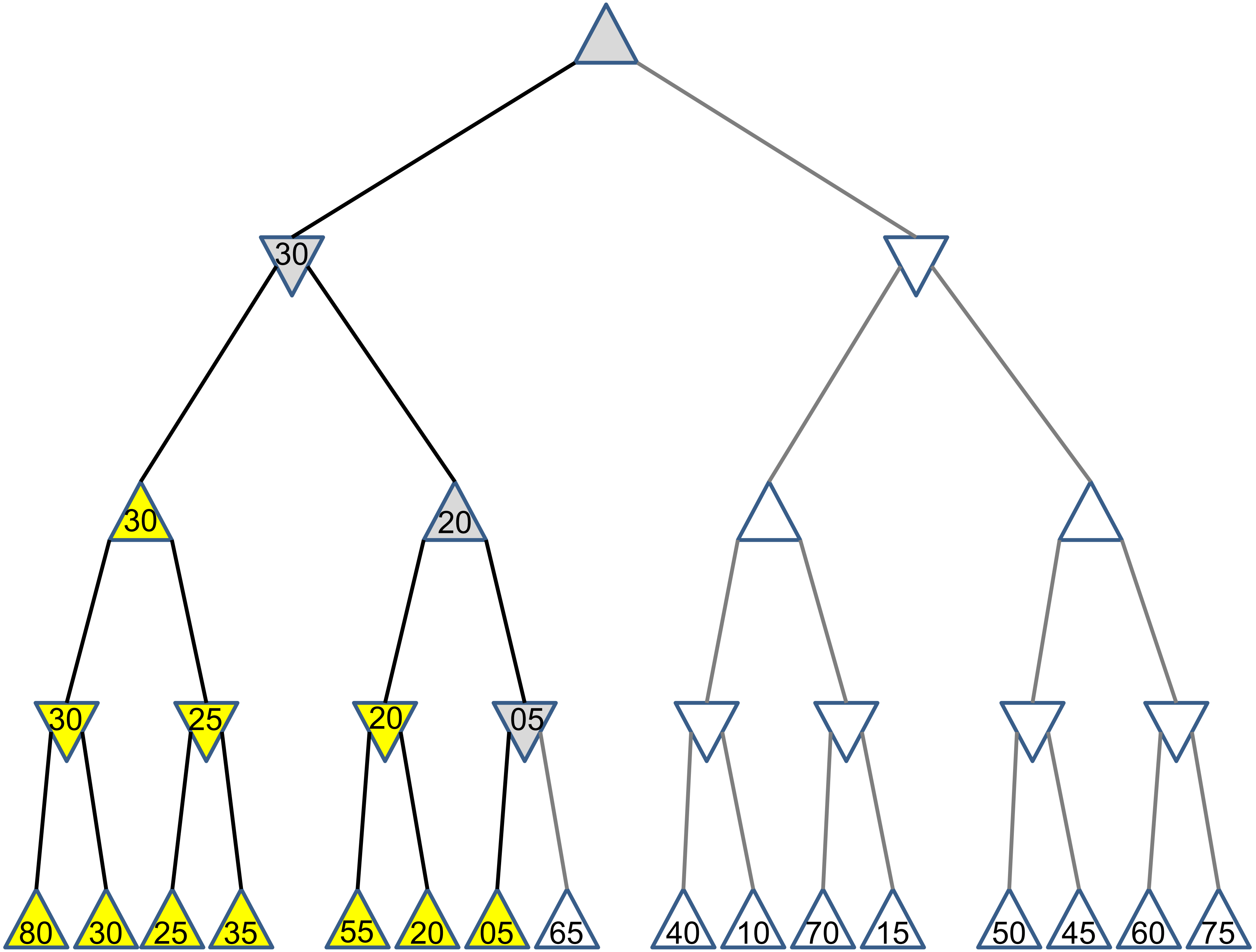

30
30
25
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

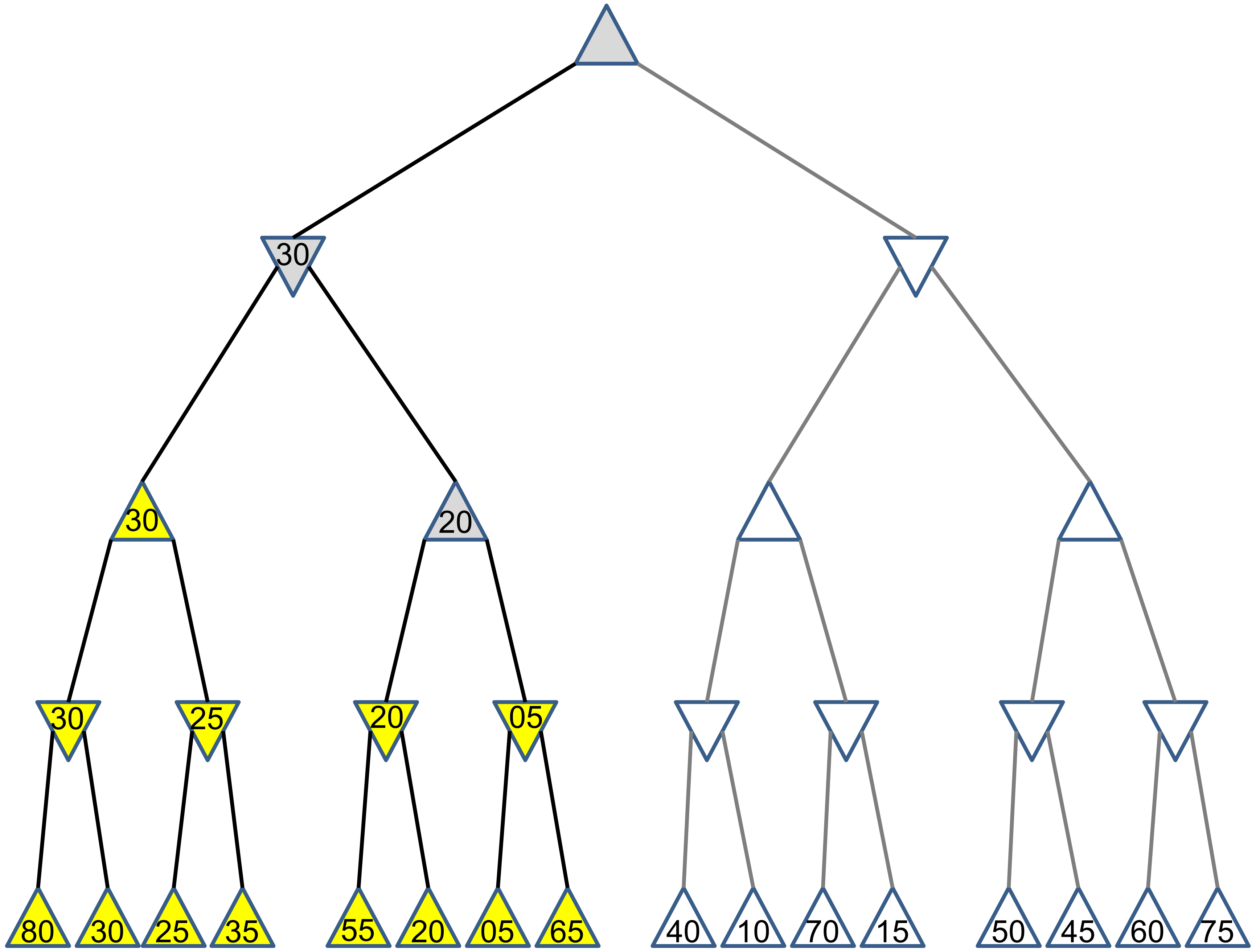

30
30
25
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

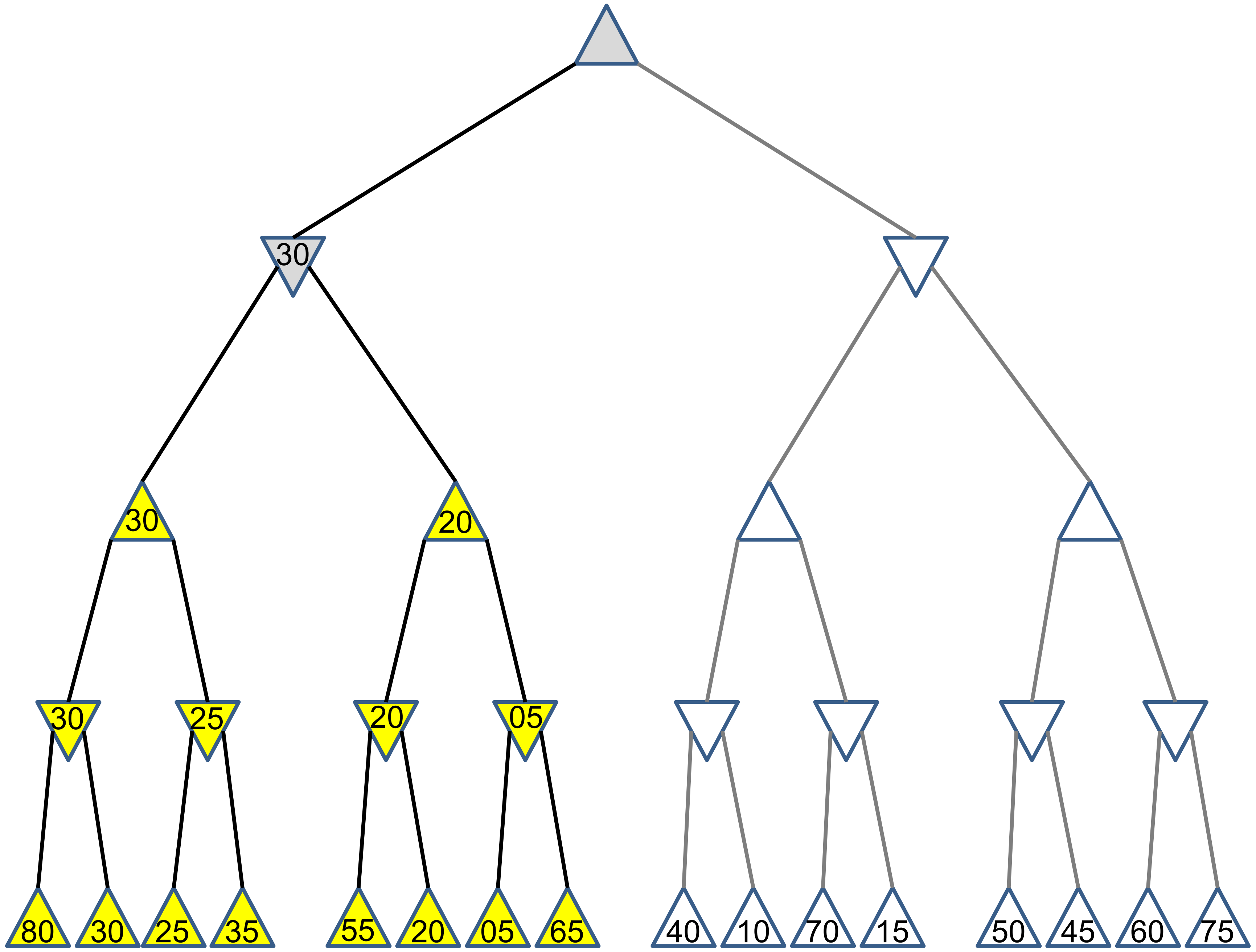

30
30
30
25
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

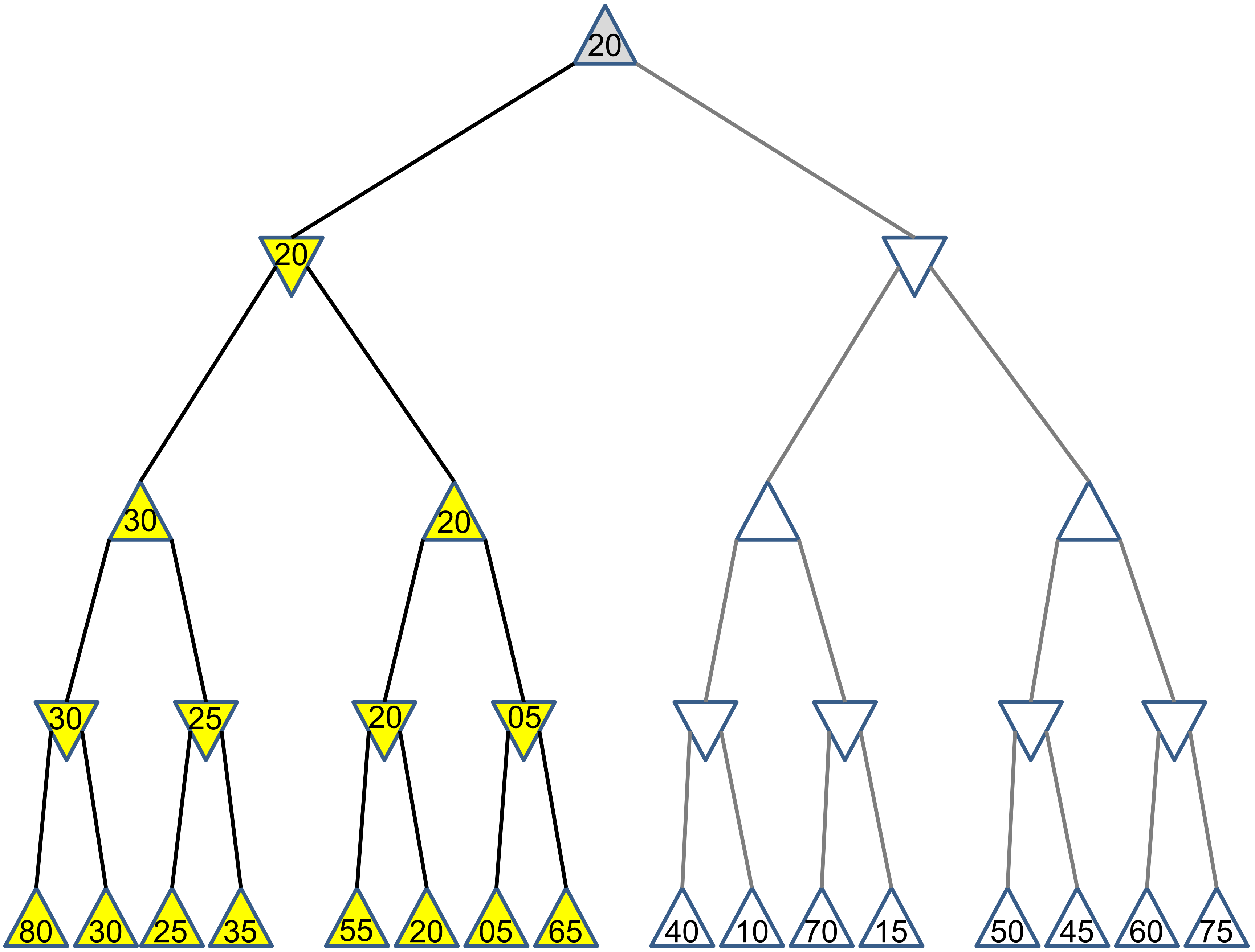

30
30
30
25
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

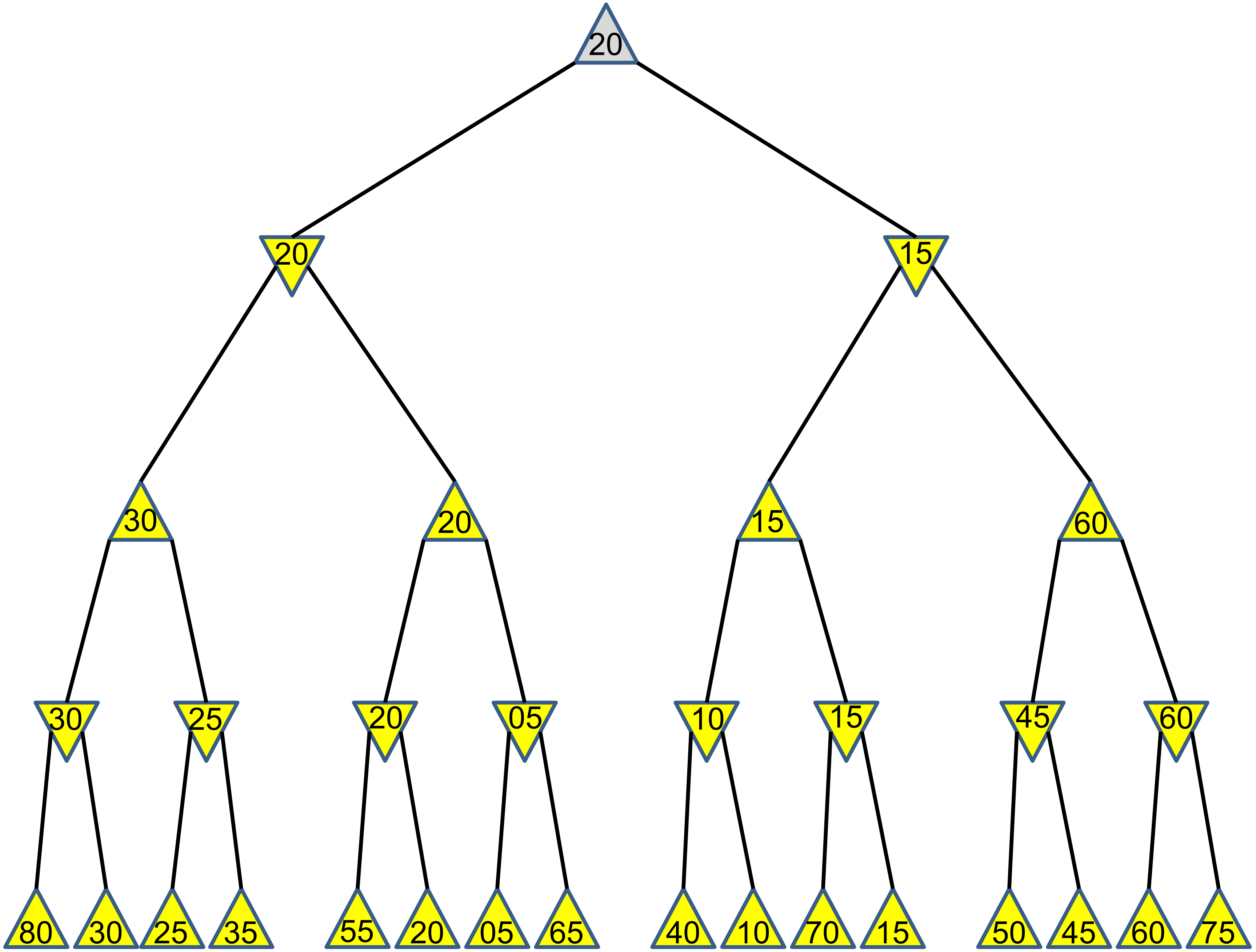

30
30
20
30
25
20
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

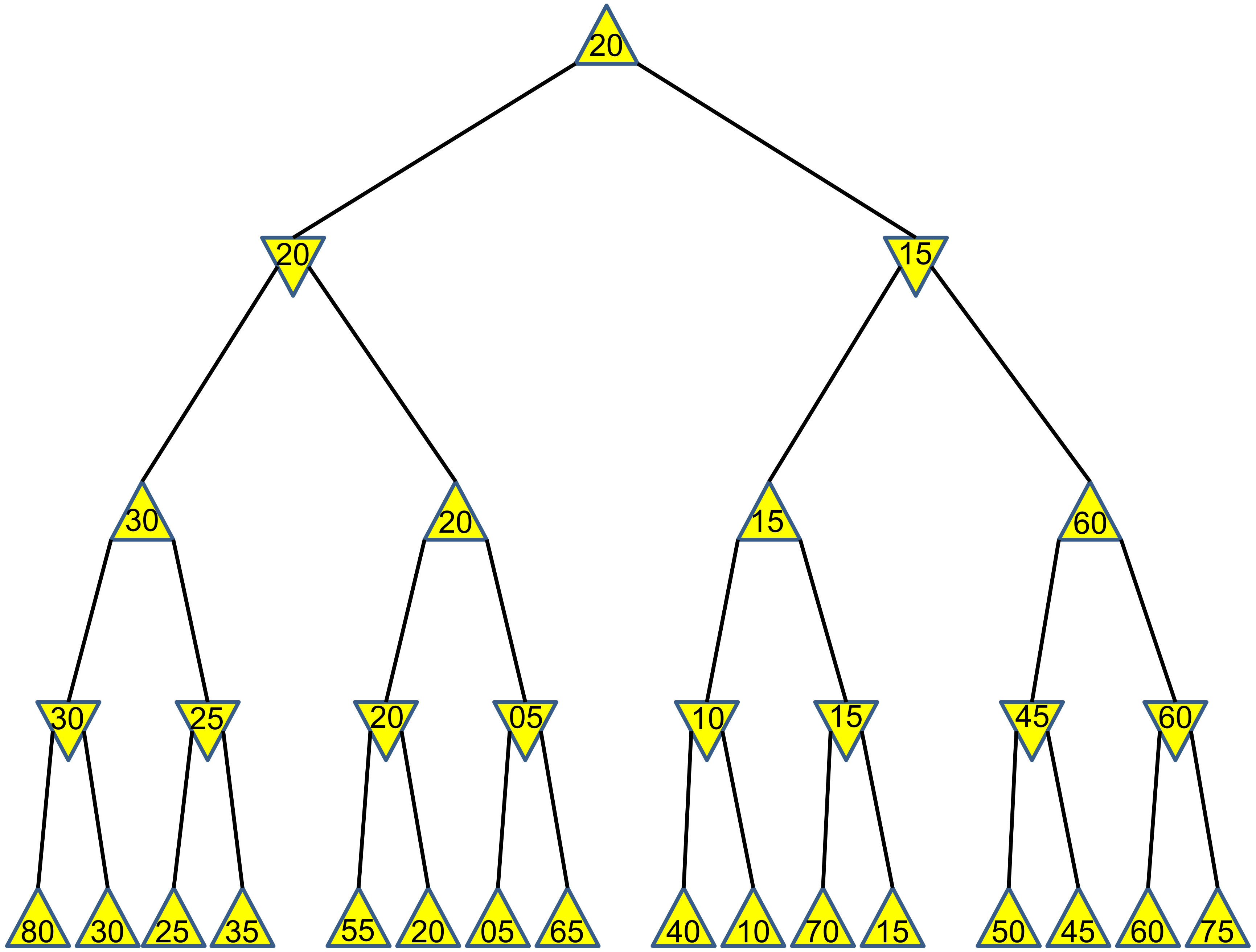

30
30
20
30
25
20
05
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75

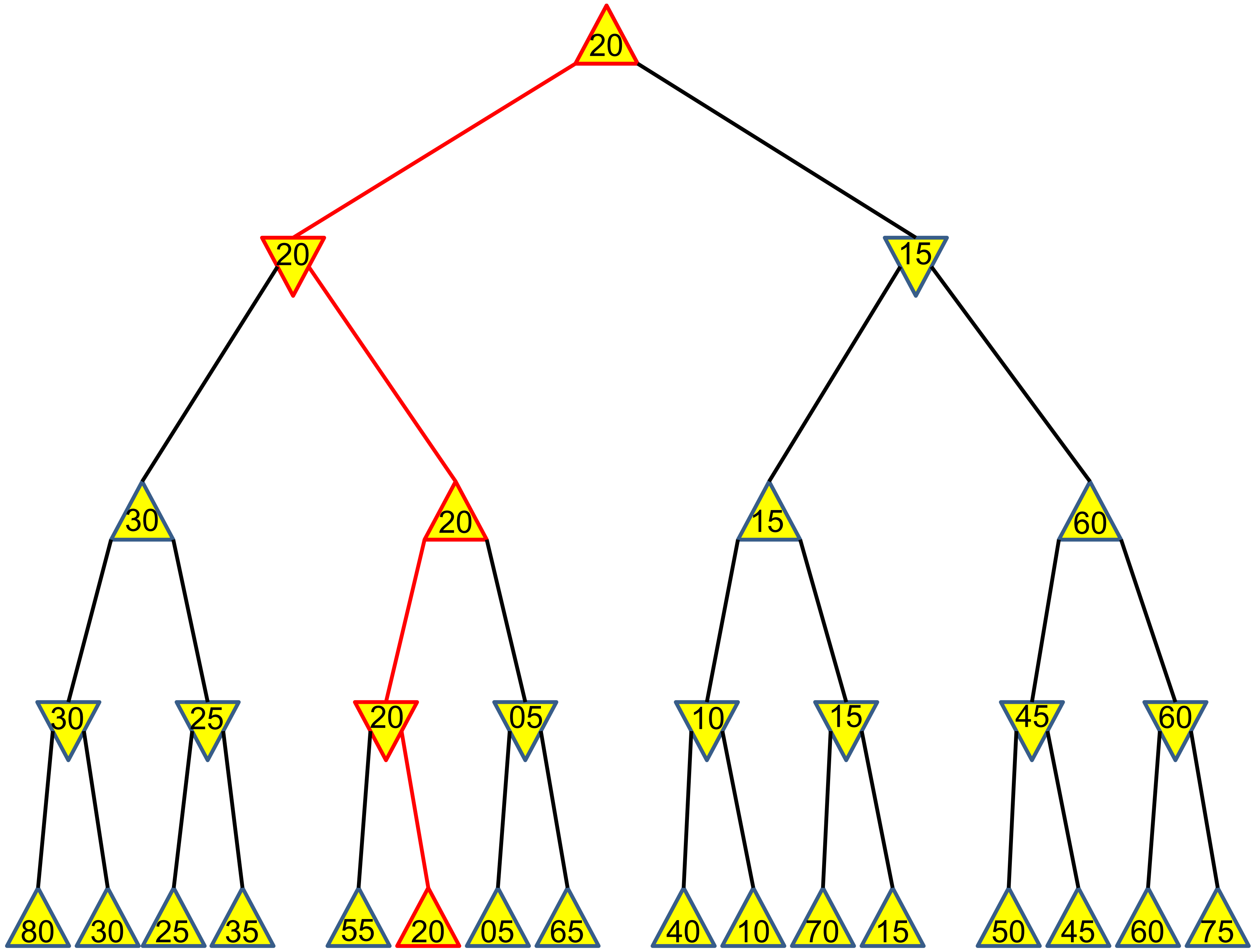

30
30
20
30
25
20
05
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75


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

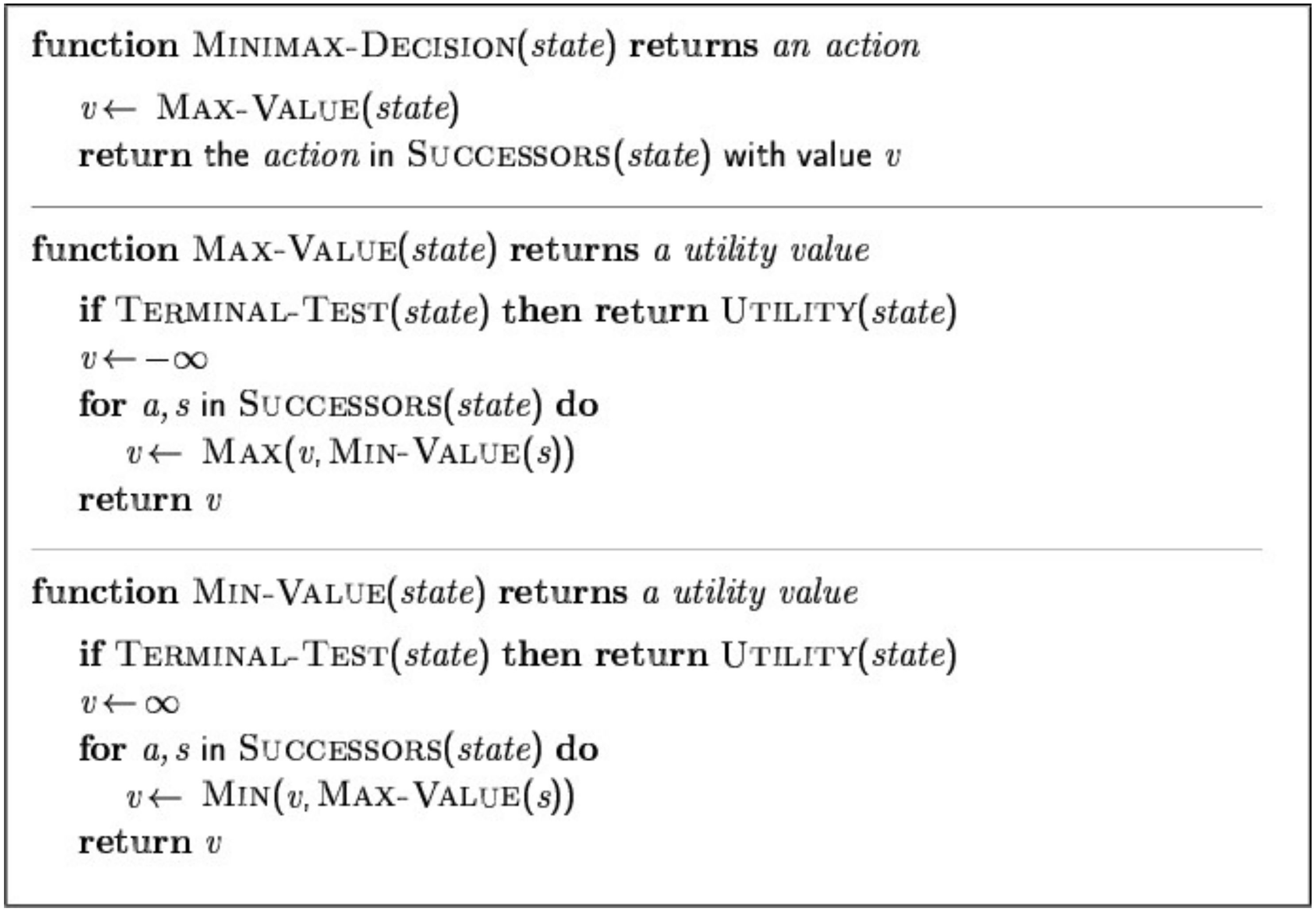

20
20
15
30
20
15
60
30
25
20
05
10
15
45
60
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75


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



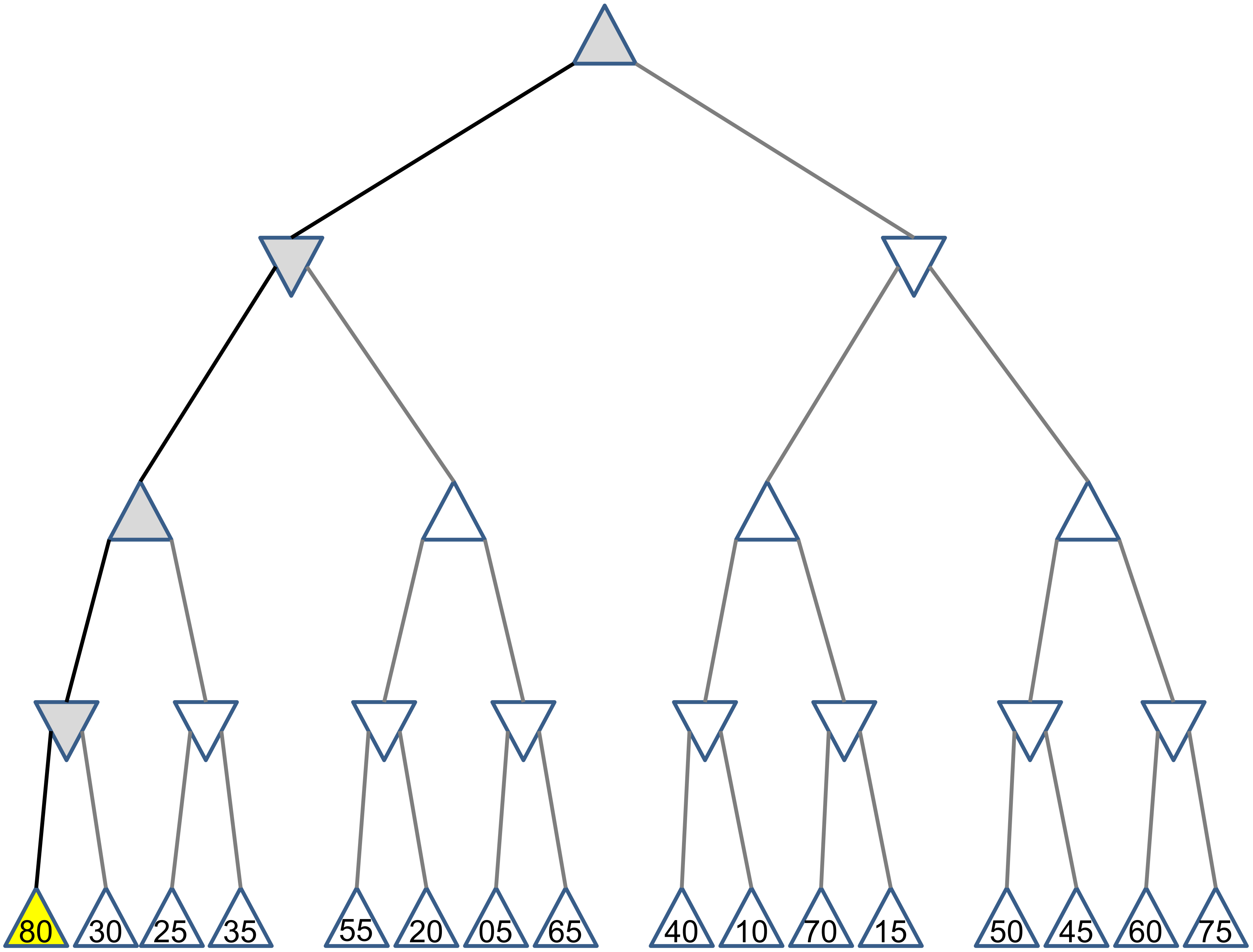
```
function MINIMAX-DECISION(state) returns an action
    $v \leftarrow$ MAX-VALUE(state)
    return the action in SUCCESSORS(state) with value $v$

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

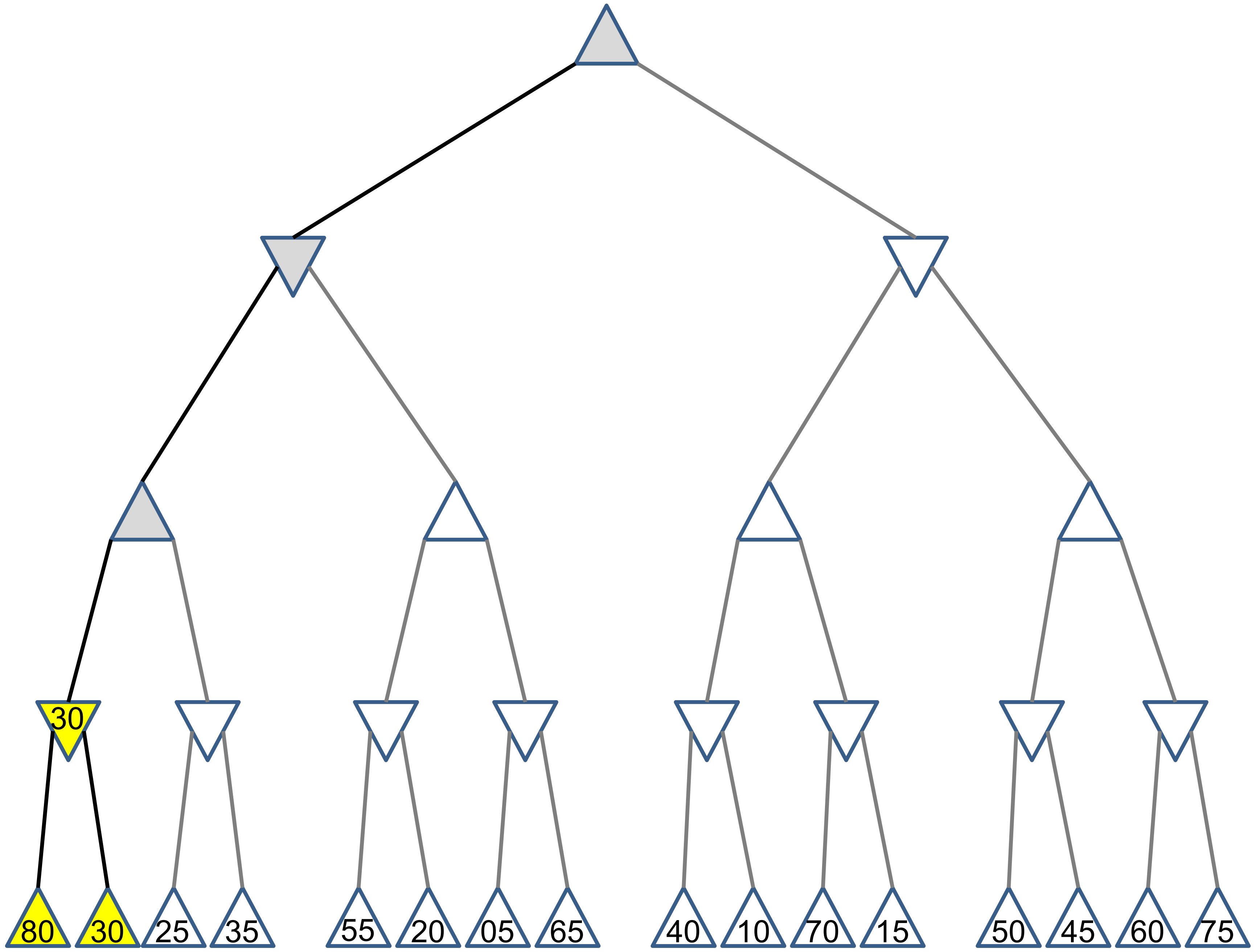
Properties of Minimax

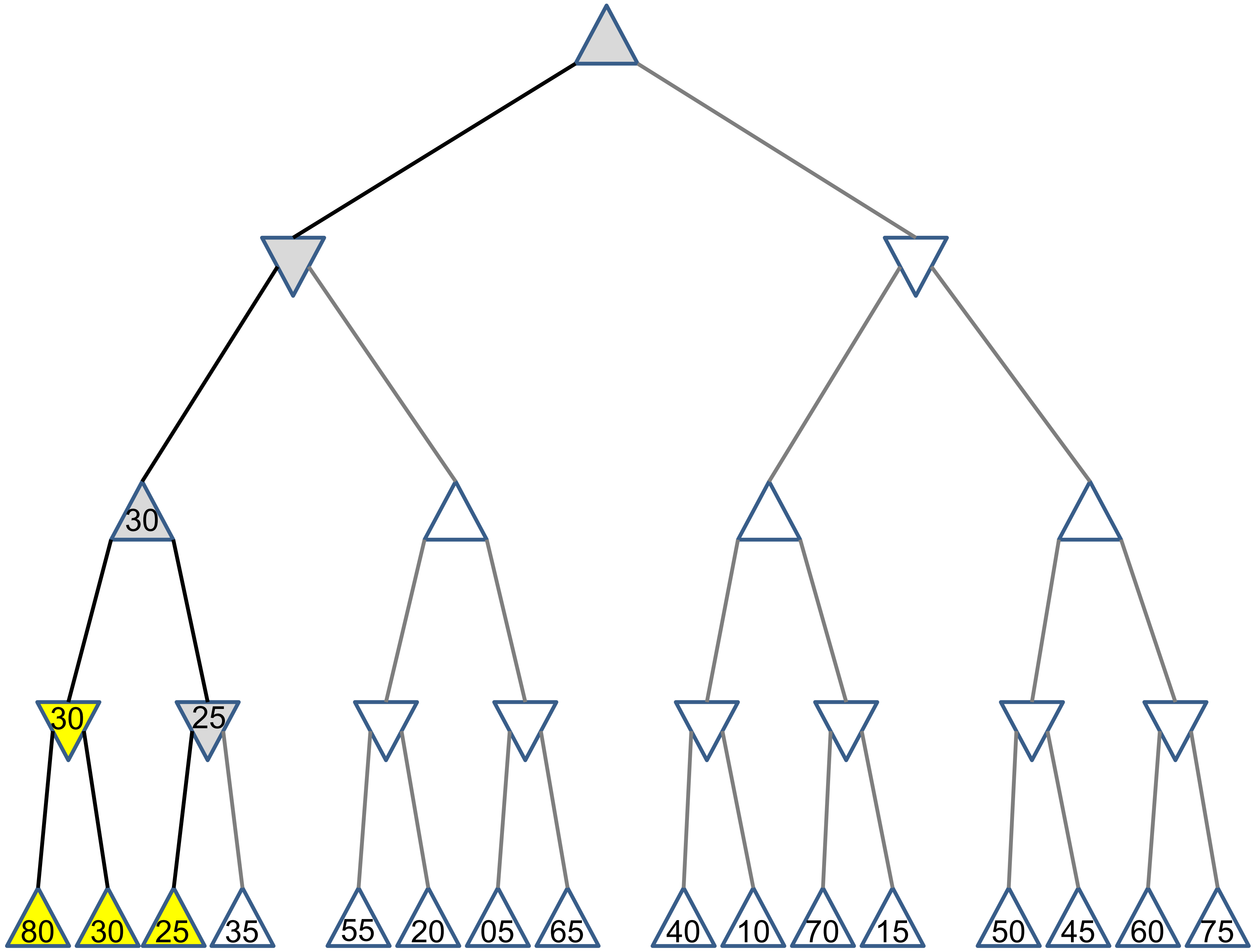
- Complete?
  - Yes (if tree is finite)
- Optimal?
  - Yes (against an optimal opponent)
  - No (does not exploit opponent weakness against suboptimal opponent)
- Time complexity?
  - O(b^m)
- Space complexity?
  - O(bm) (depth-first exploration)

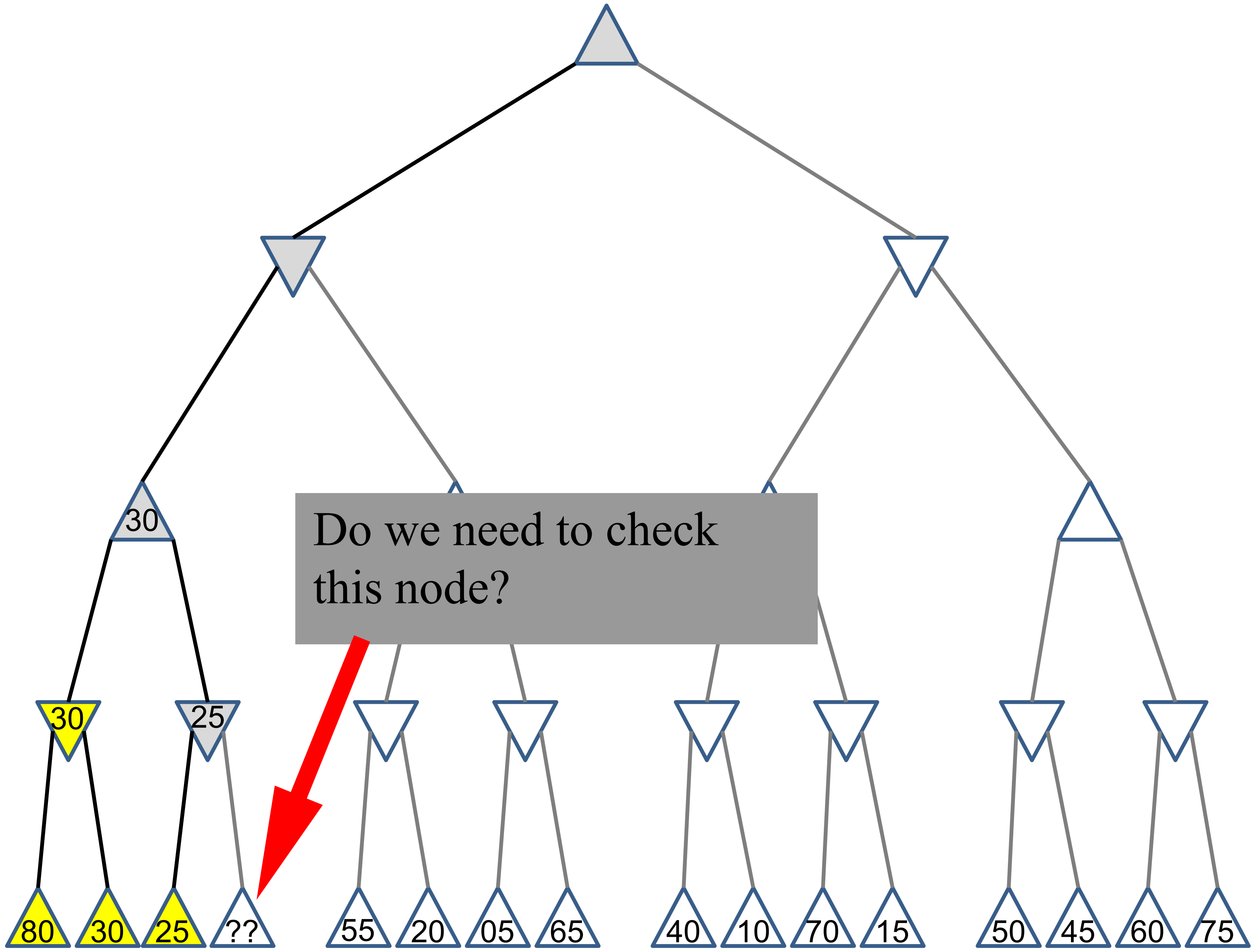
Good Enough?

- 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

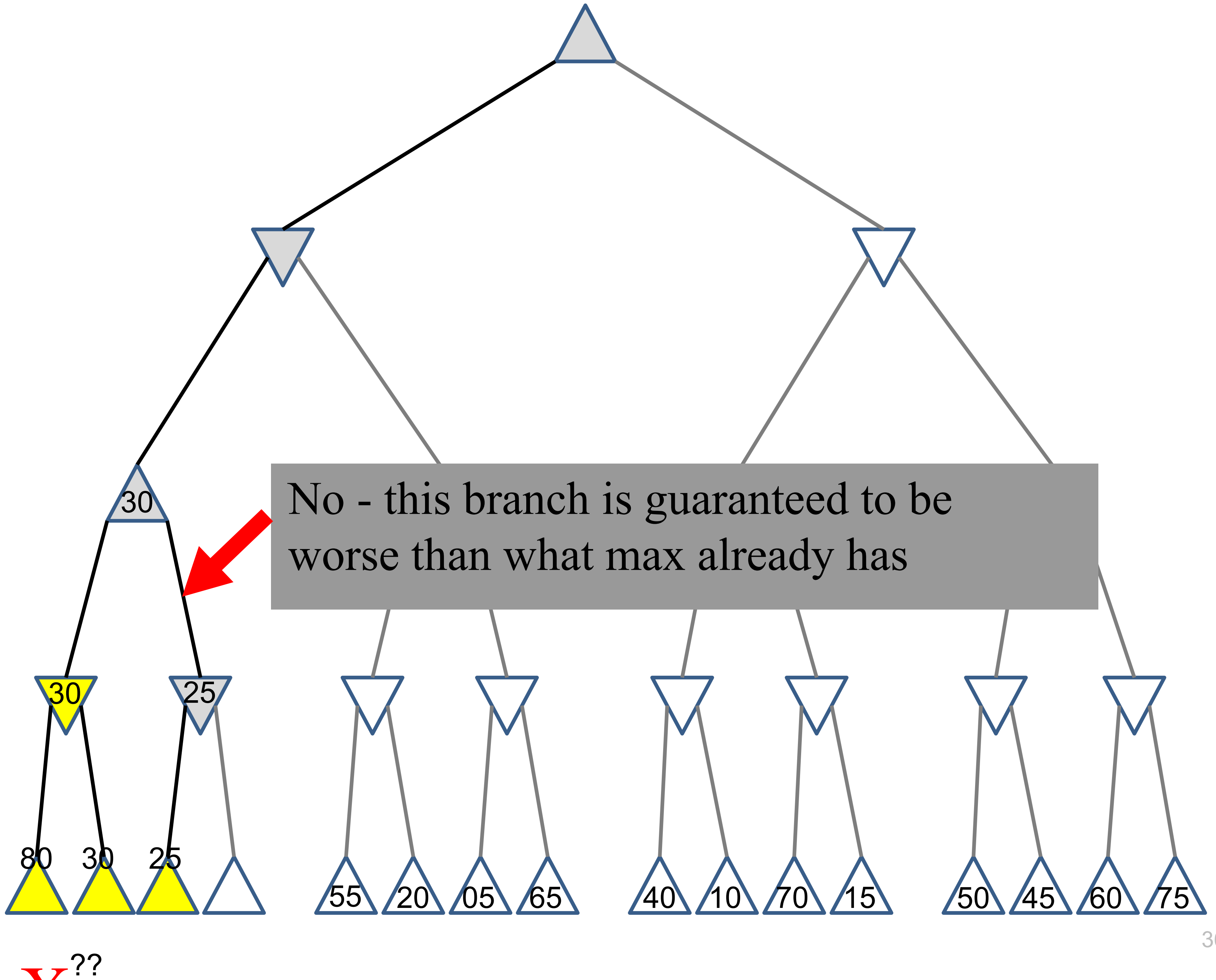

80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75


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

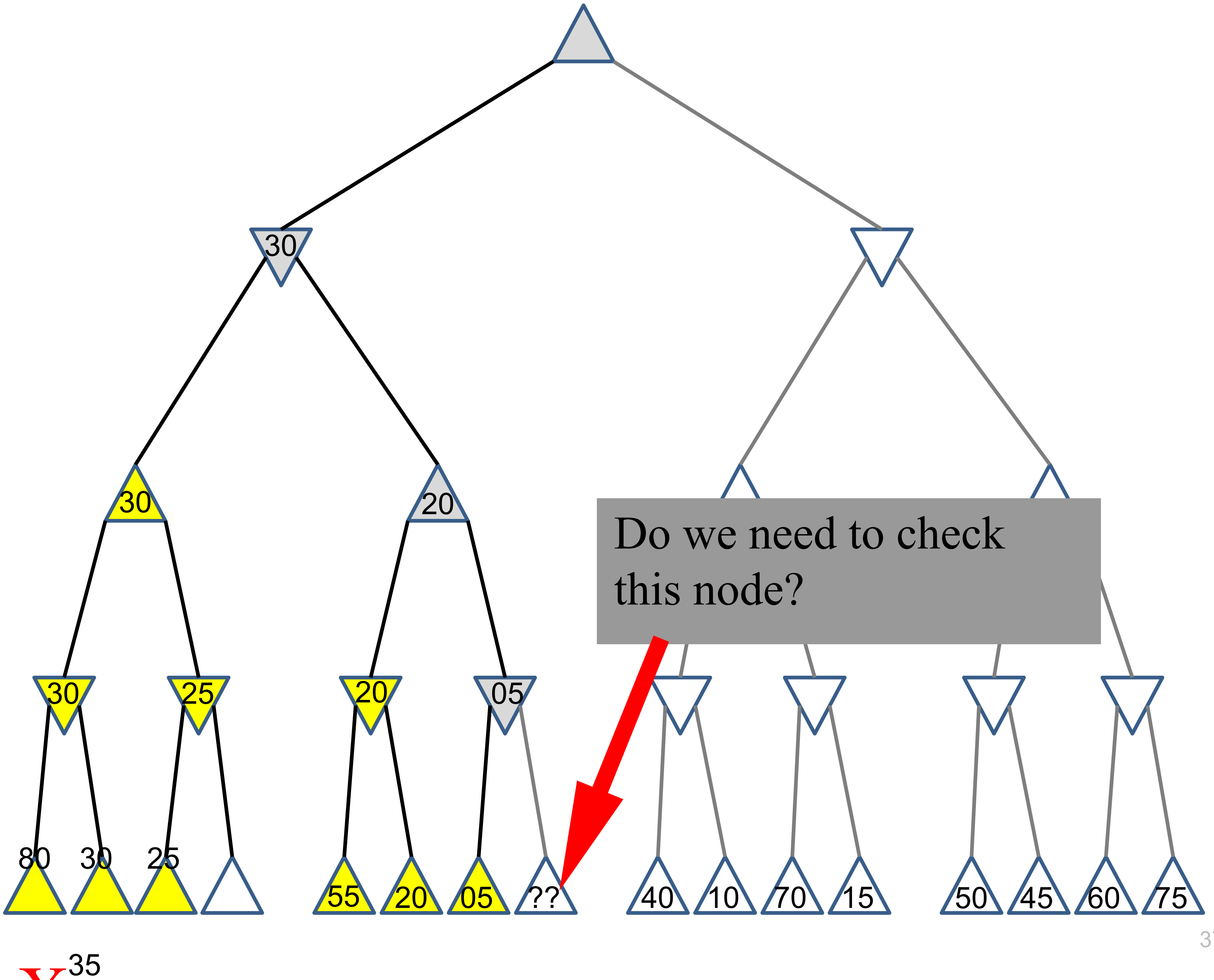

30
30
25
80
30
25
35
55
20
05
65
40
10
70
15
50
45
60
75



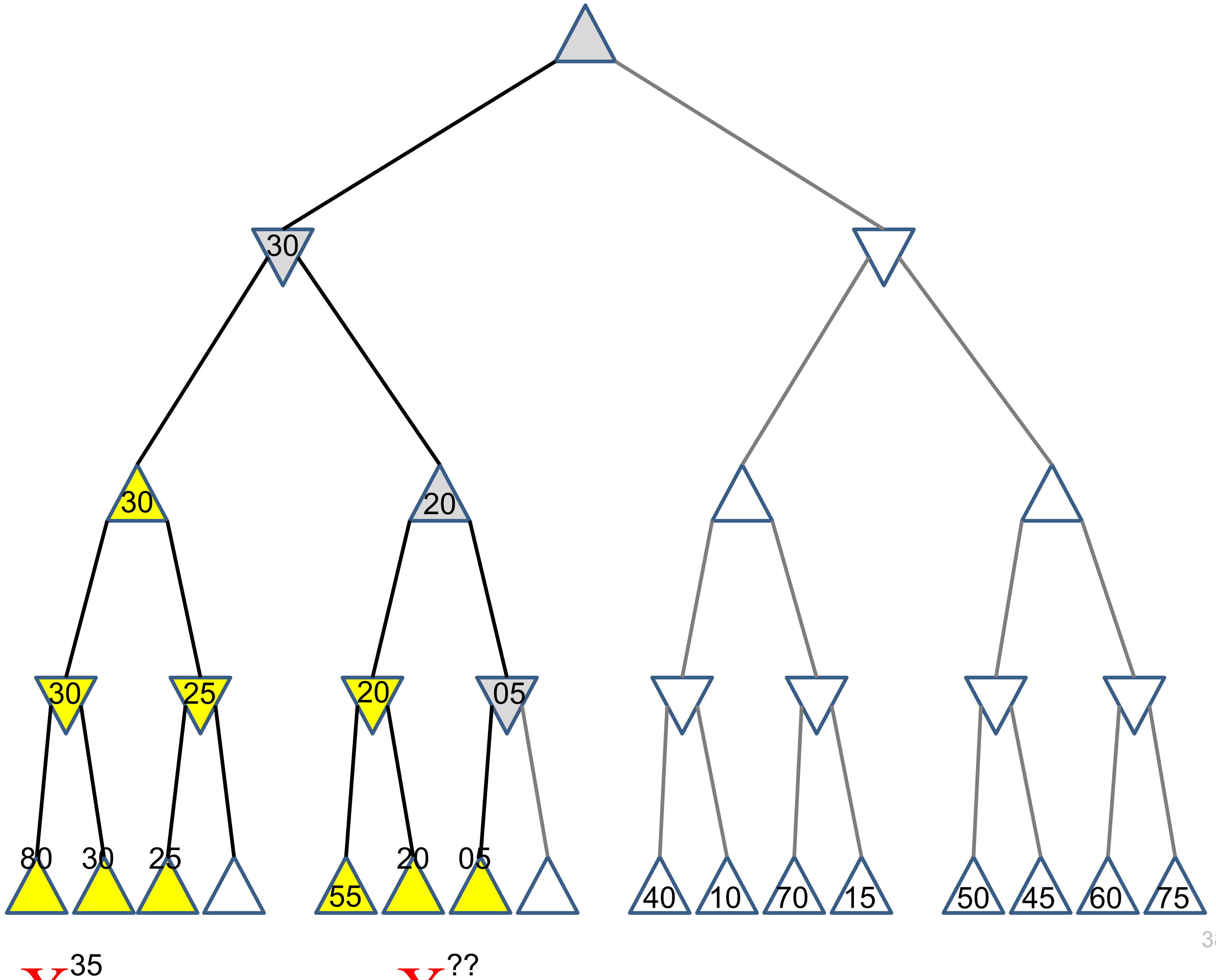
Do we need to check this node?



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



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
35
??

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

  $v \geq$ □

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

  $v \leq$ □

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

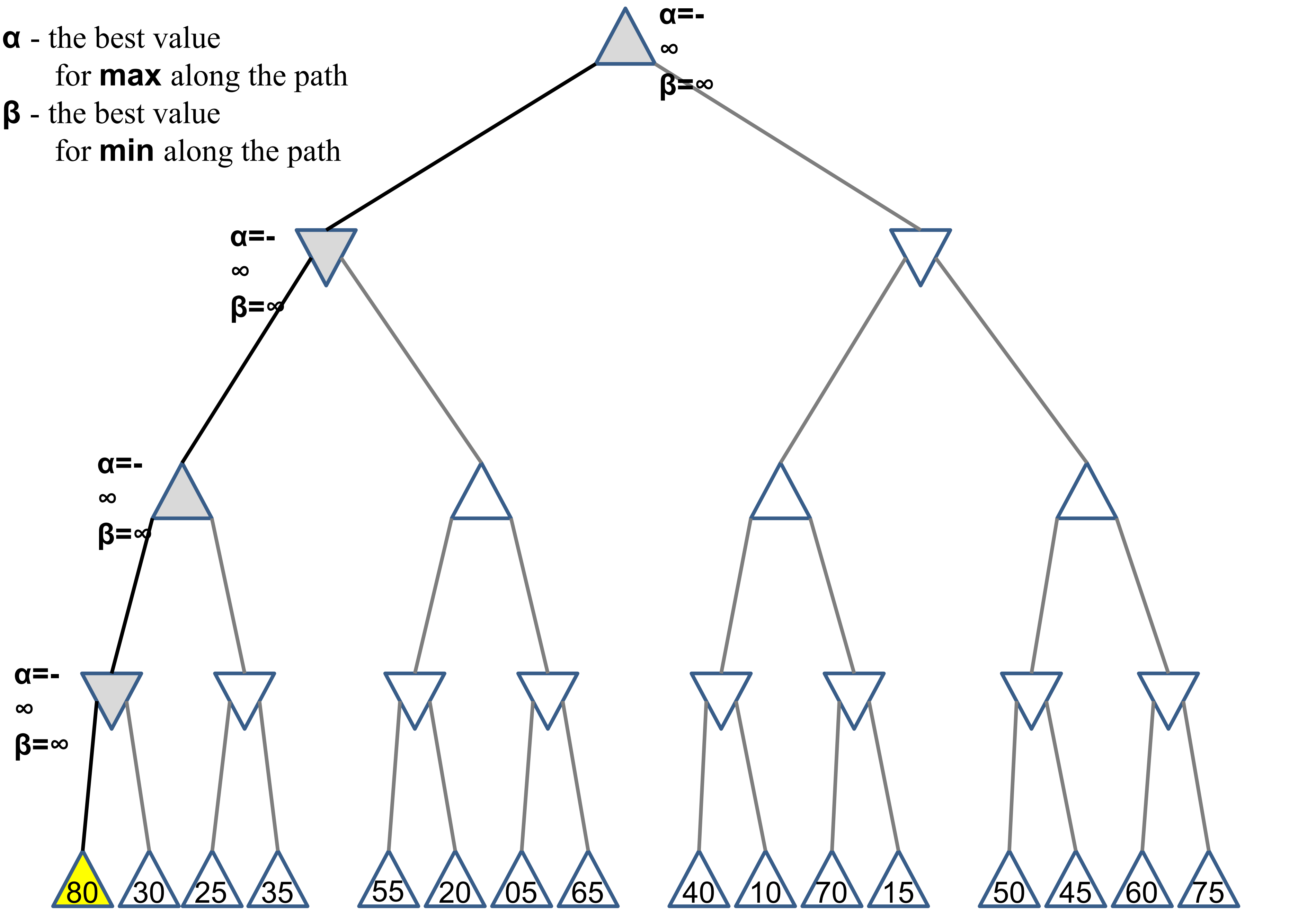
Alpha-Beta

```
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); }
```

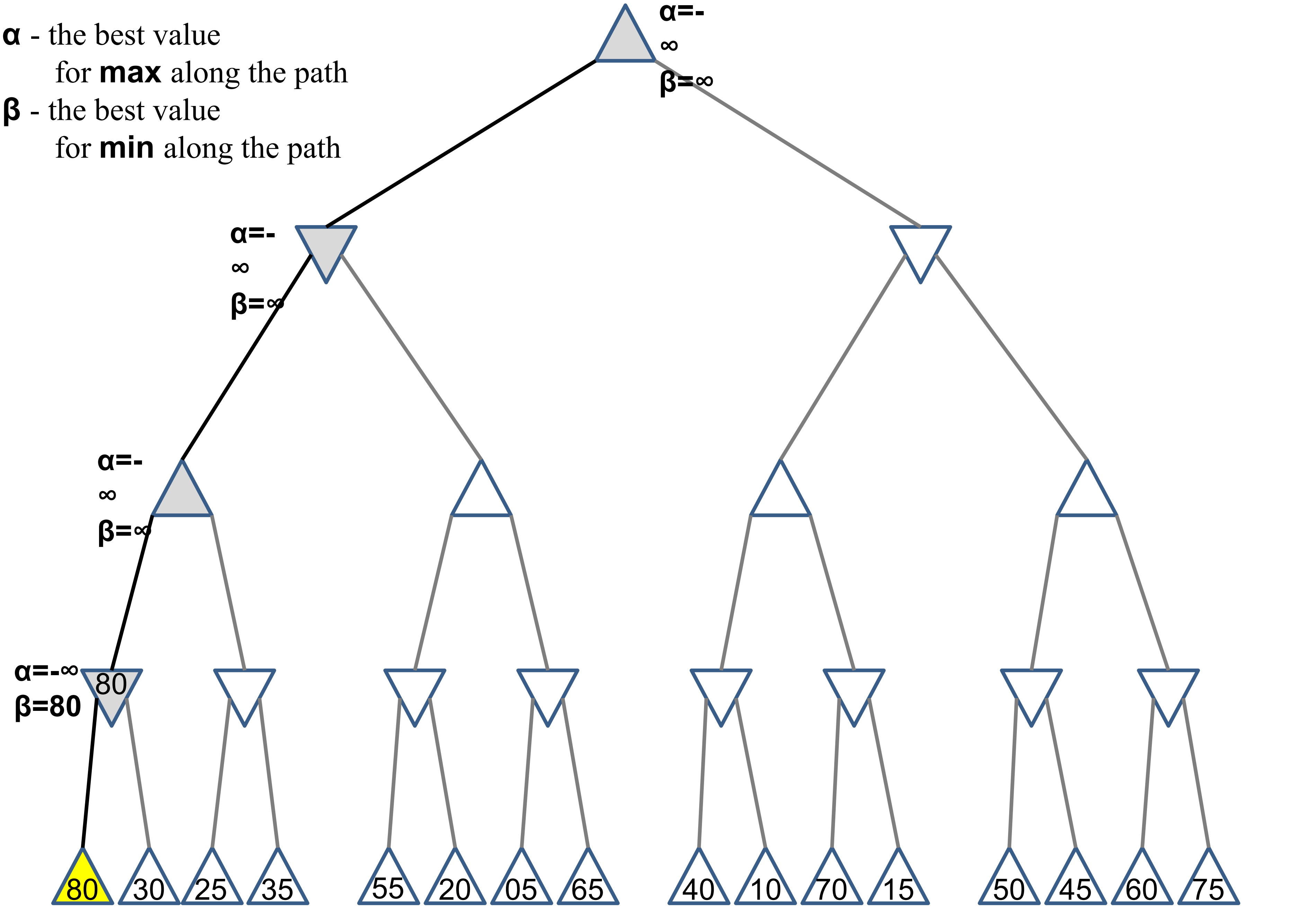
alpha = the **highest** value for **MAX** along the path

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

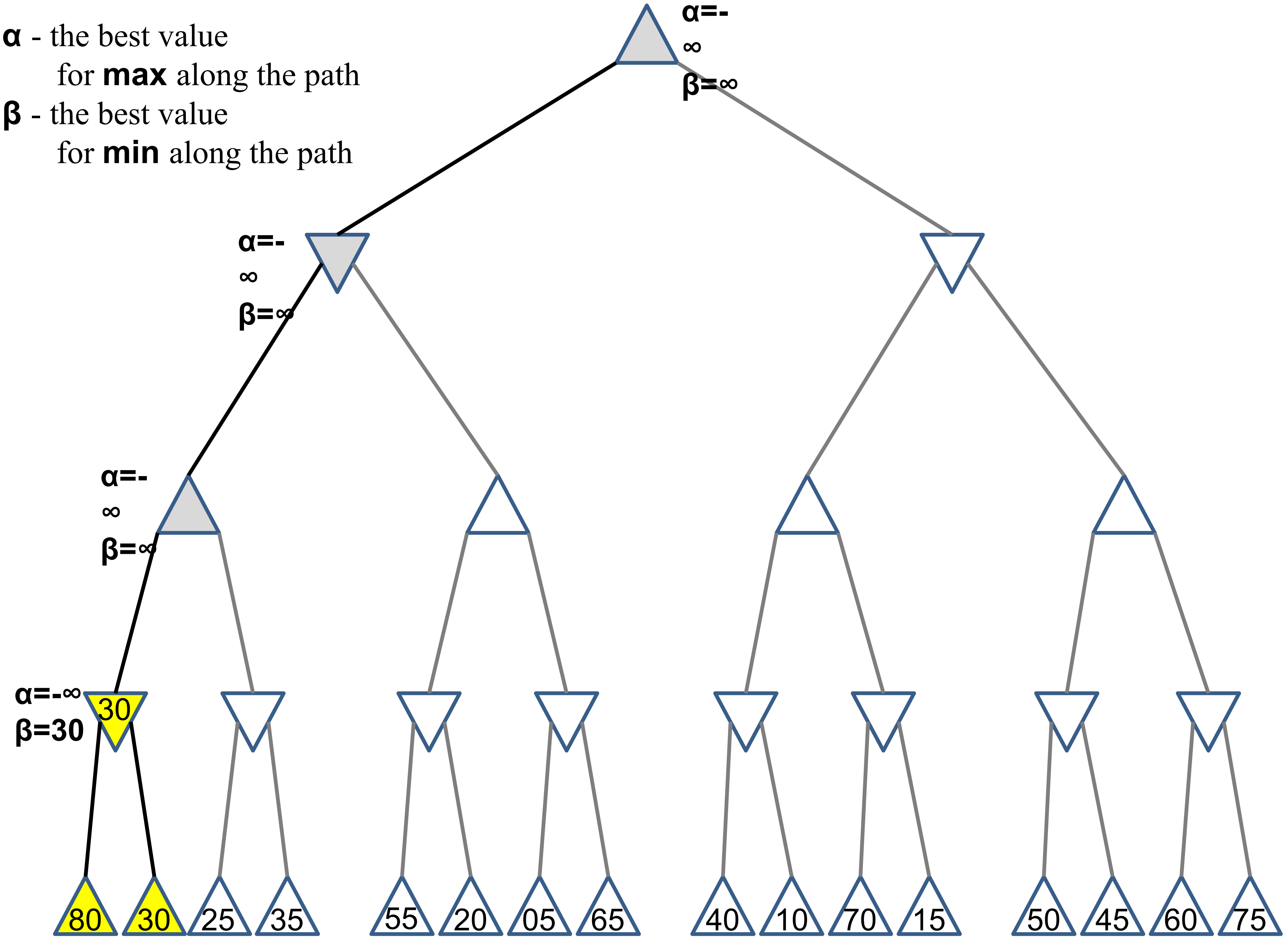
α - the best value
for **max** along the path
β - the best value
for **min** along the path



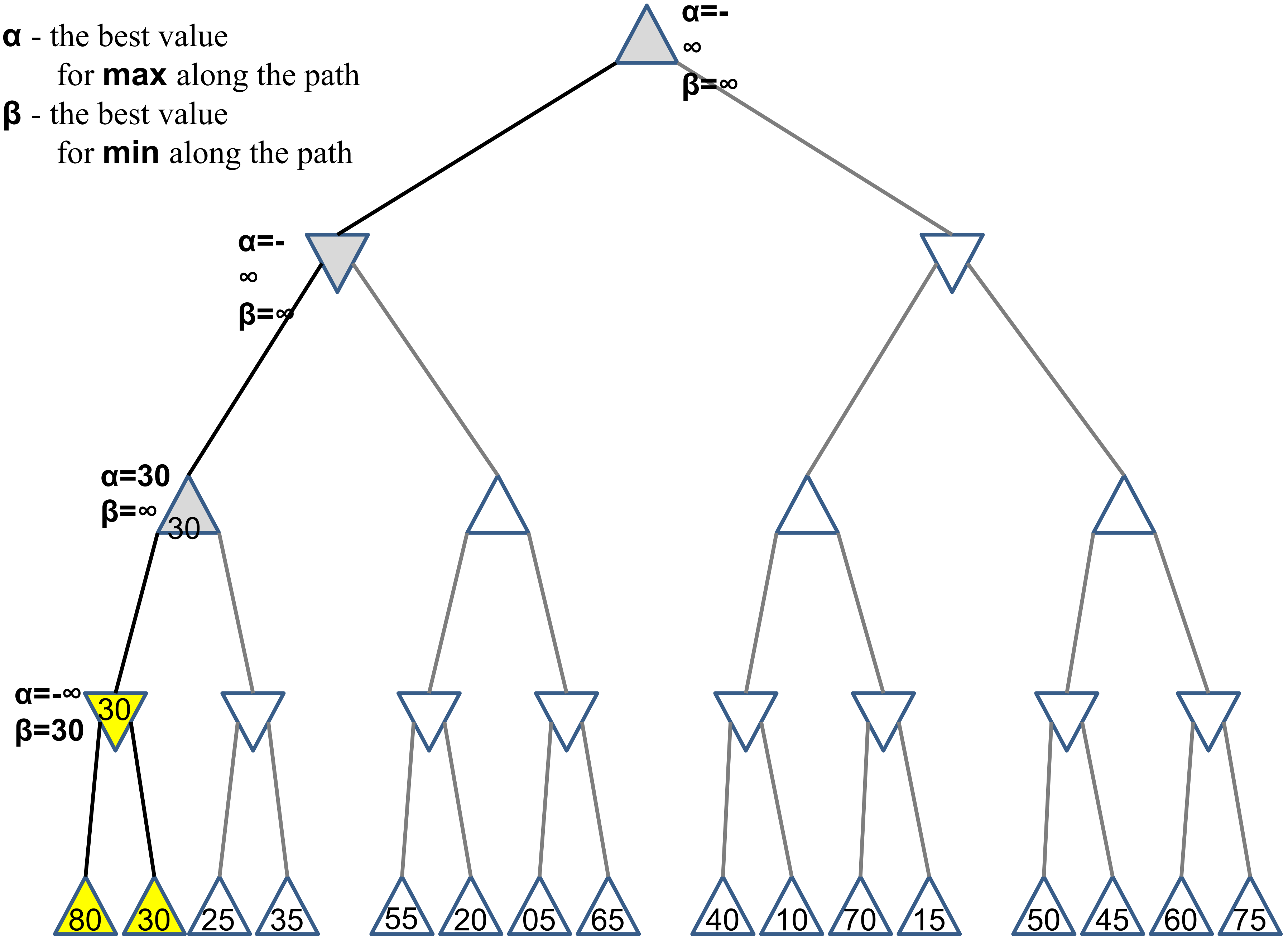
α - the best value
for **max** along the path
β - the best value
for **min** along the path



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

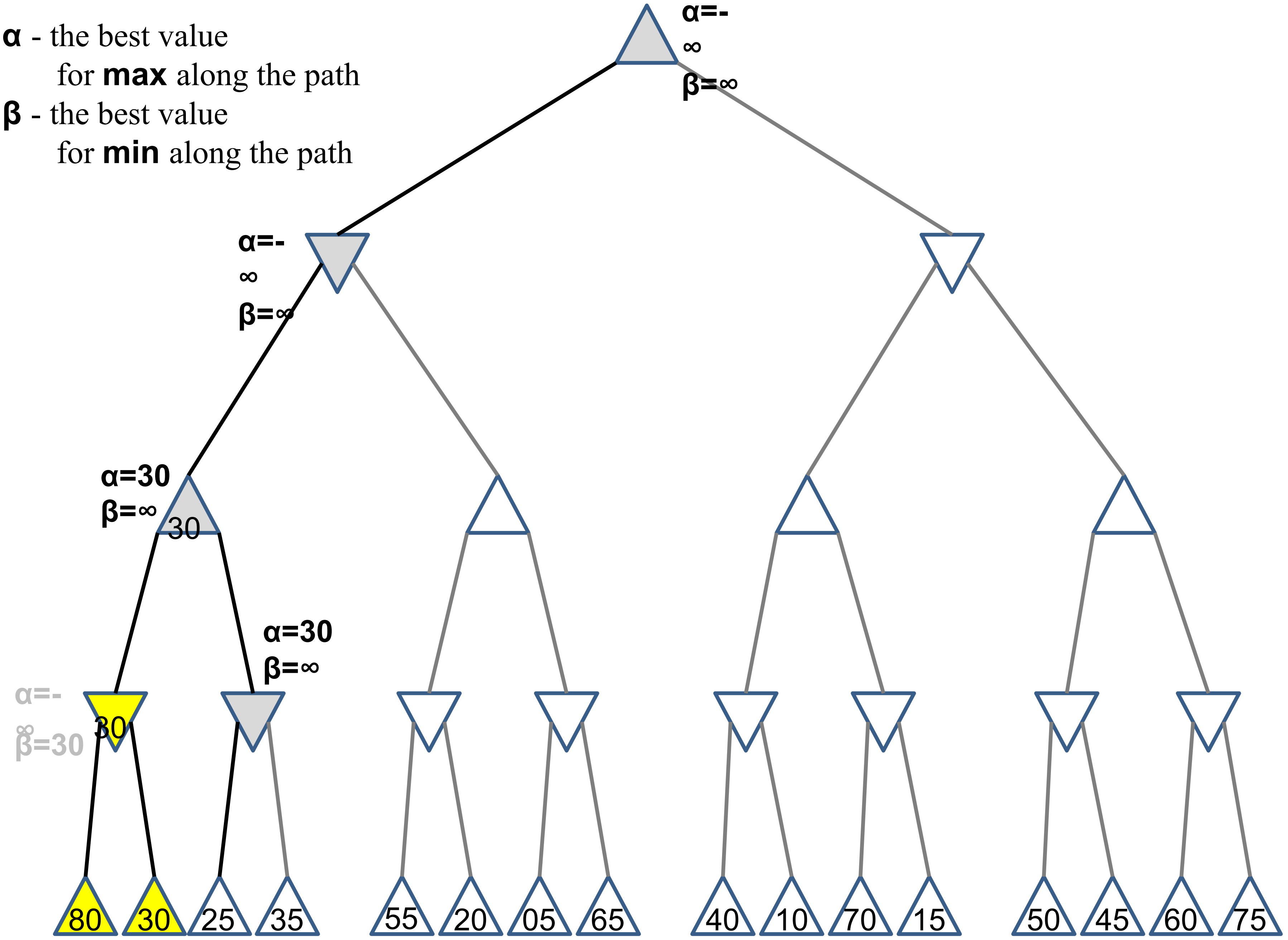


α - the best value
for **max** along the path
β - the best value
for **min** along the path

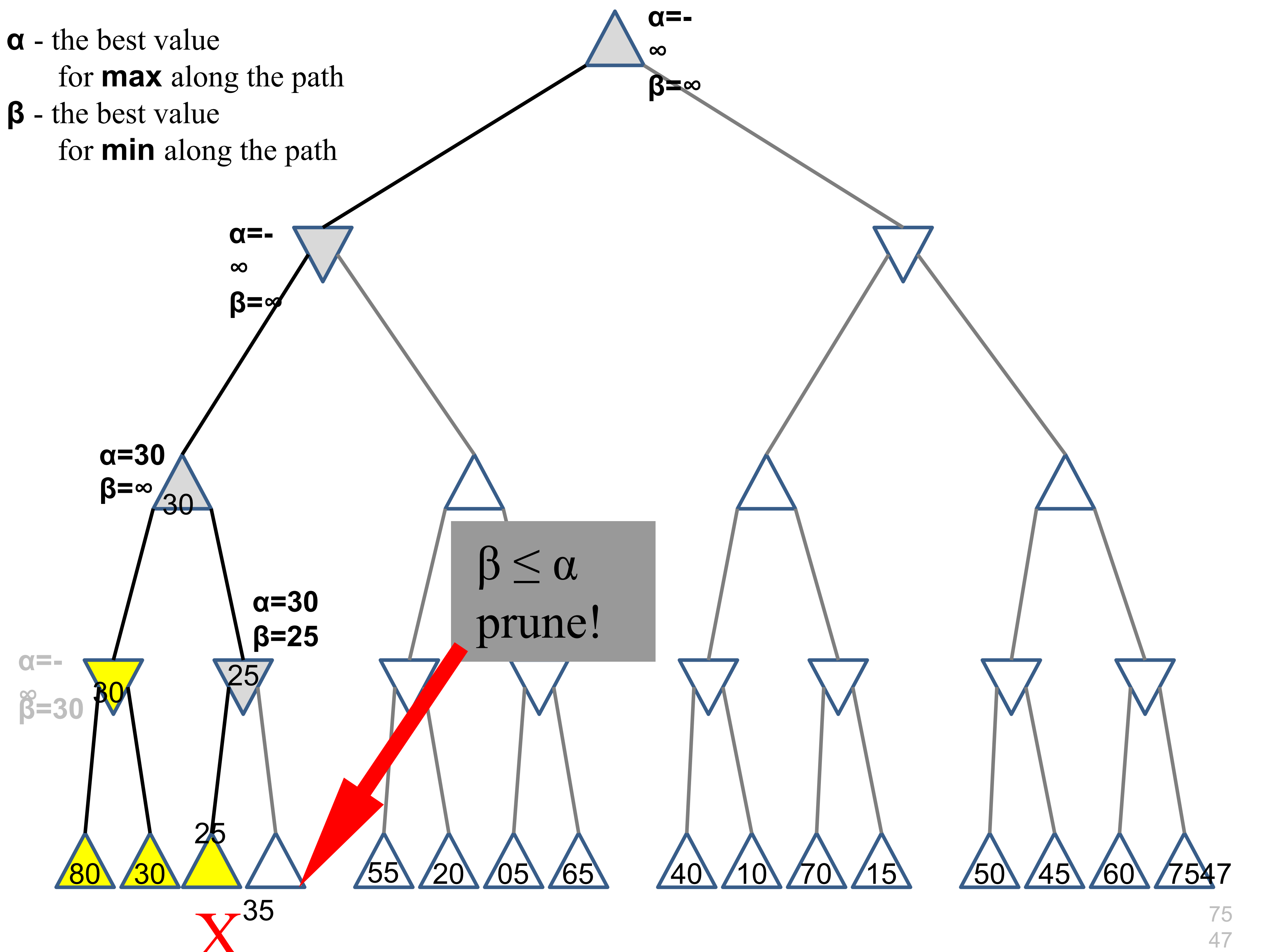


α - the best value
for **max** along the path

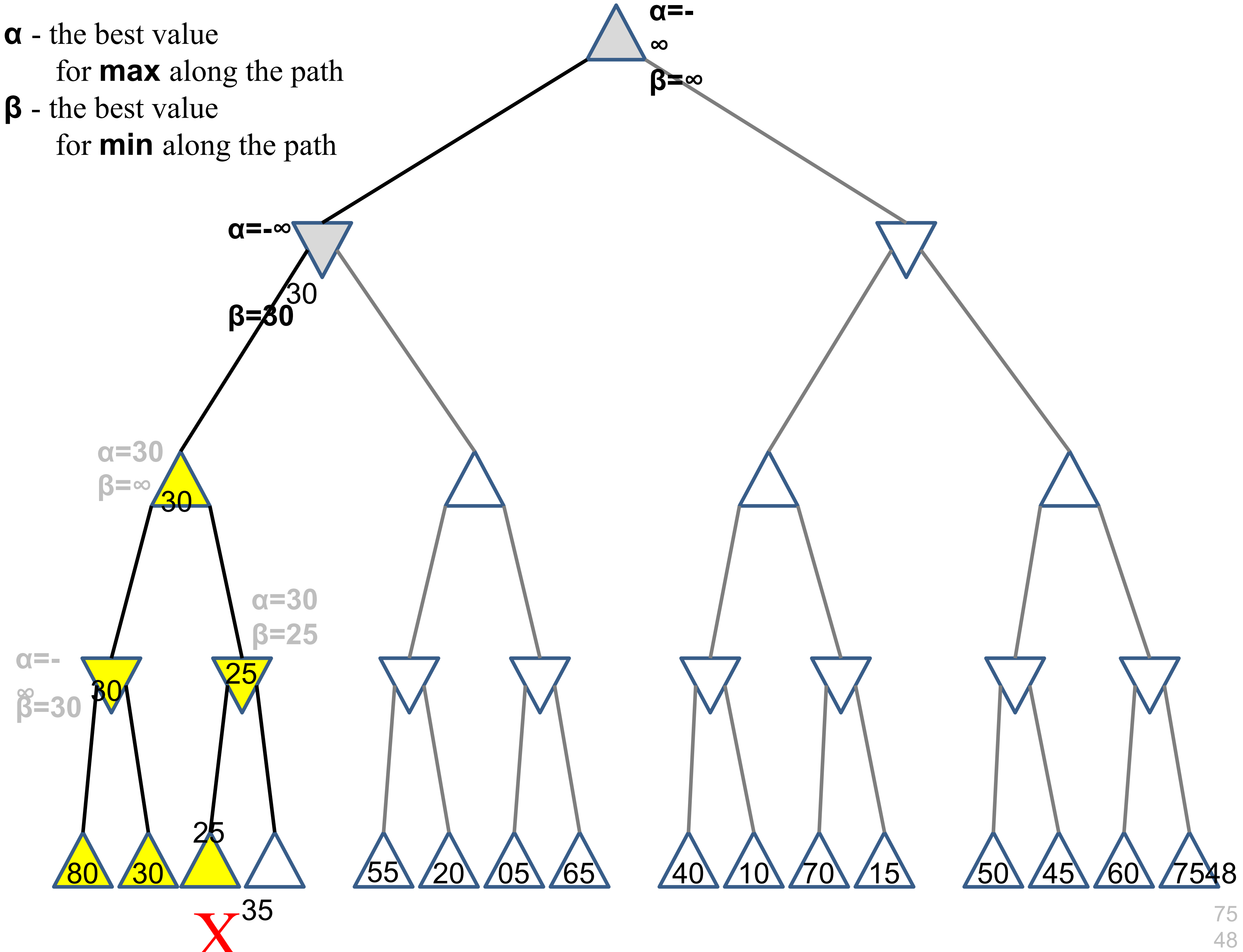
β - the best value
for **min** along the path



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

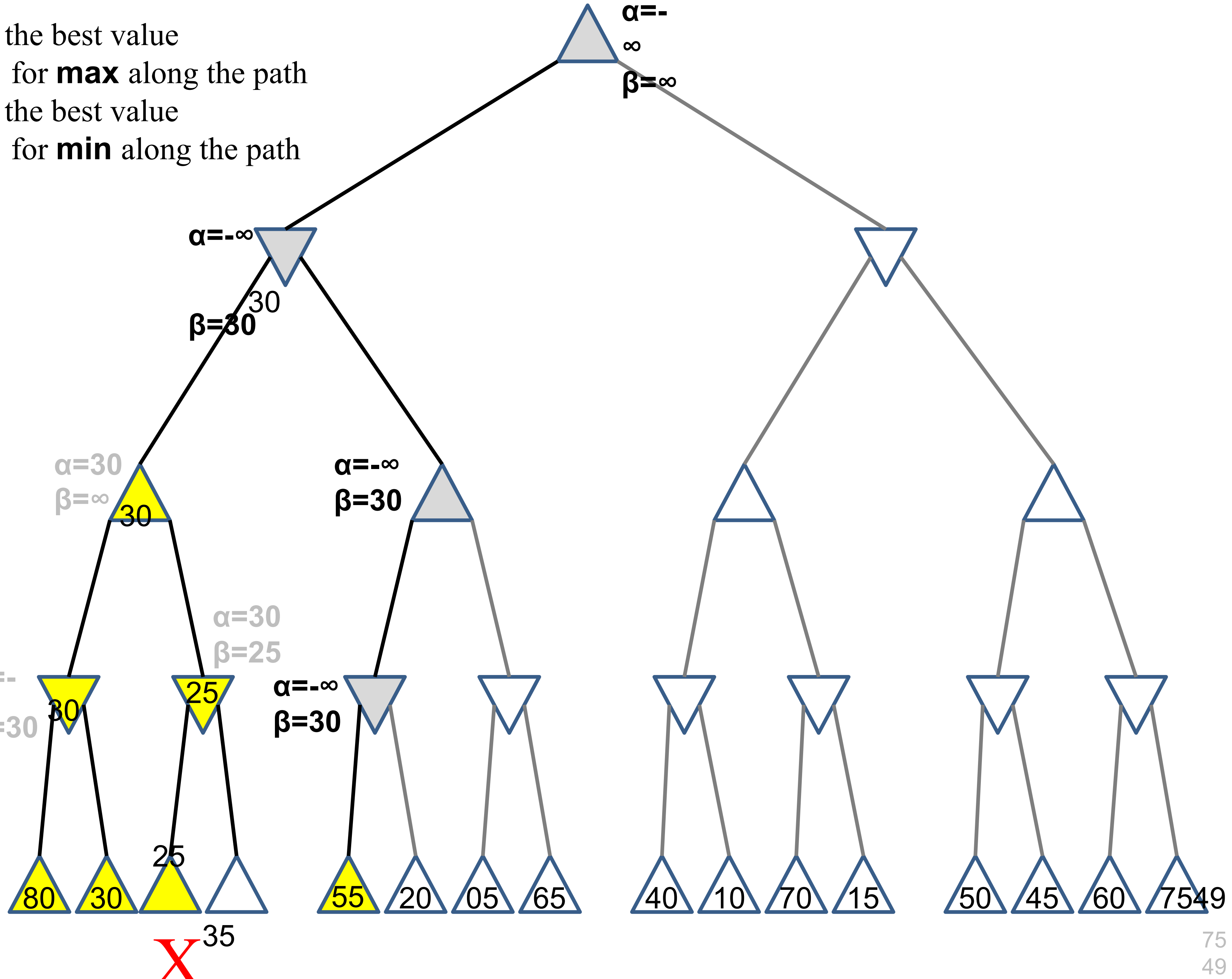


α - the best value
for **max** along the path
β - the best value
for **min** along the path



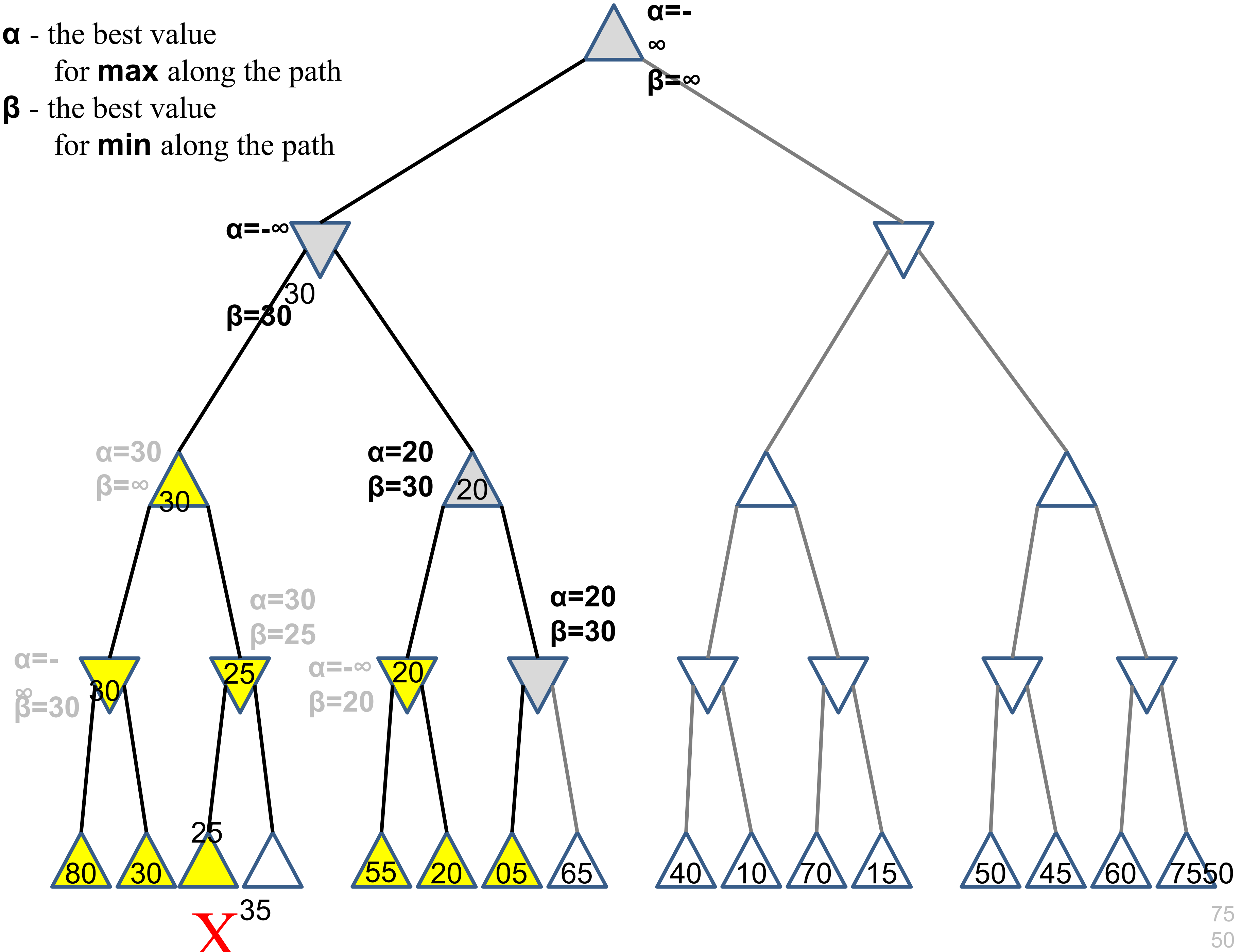
α - the best value
for **max** along the path

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

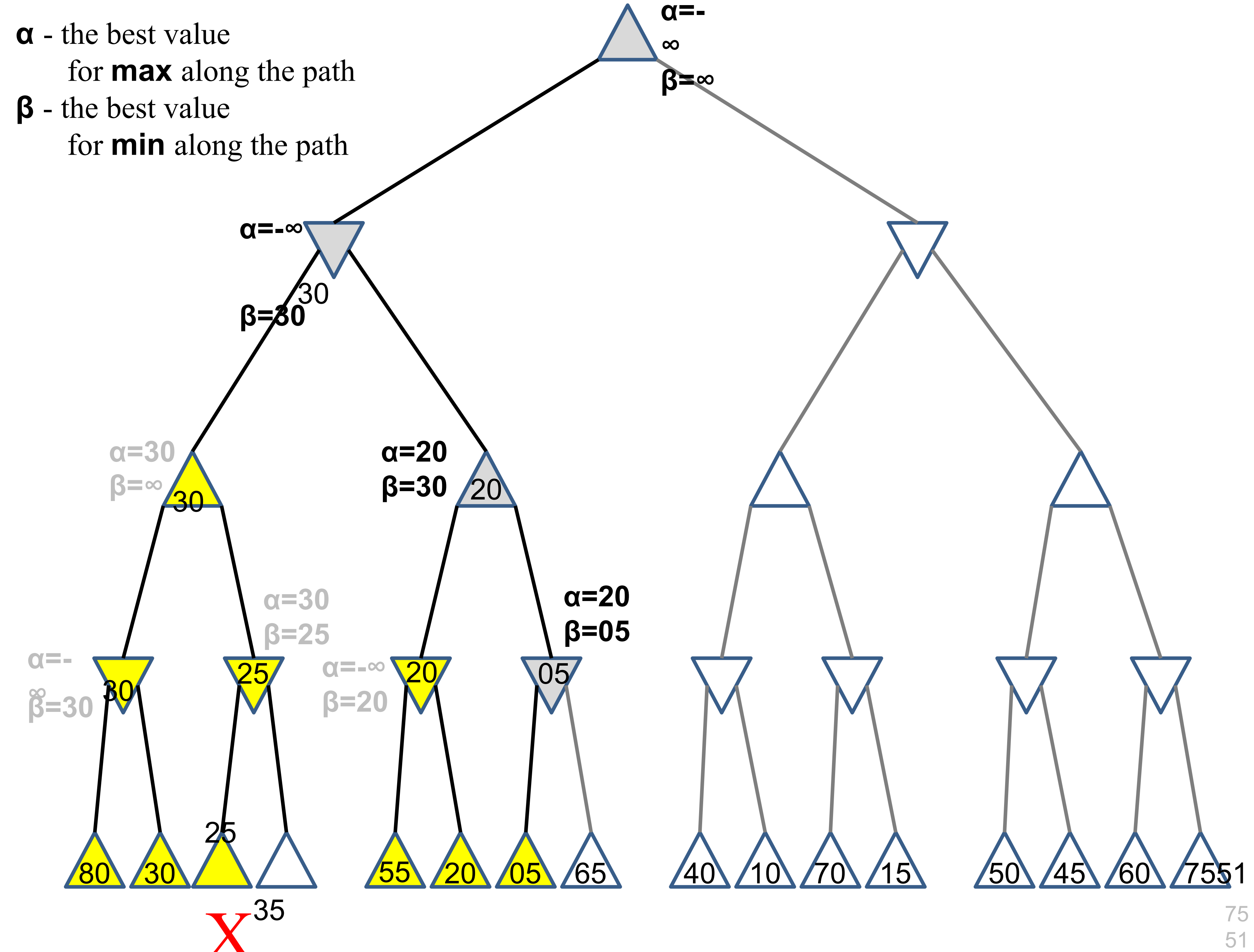


α - the best value
for **max** along the path

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

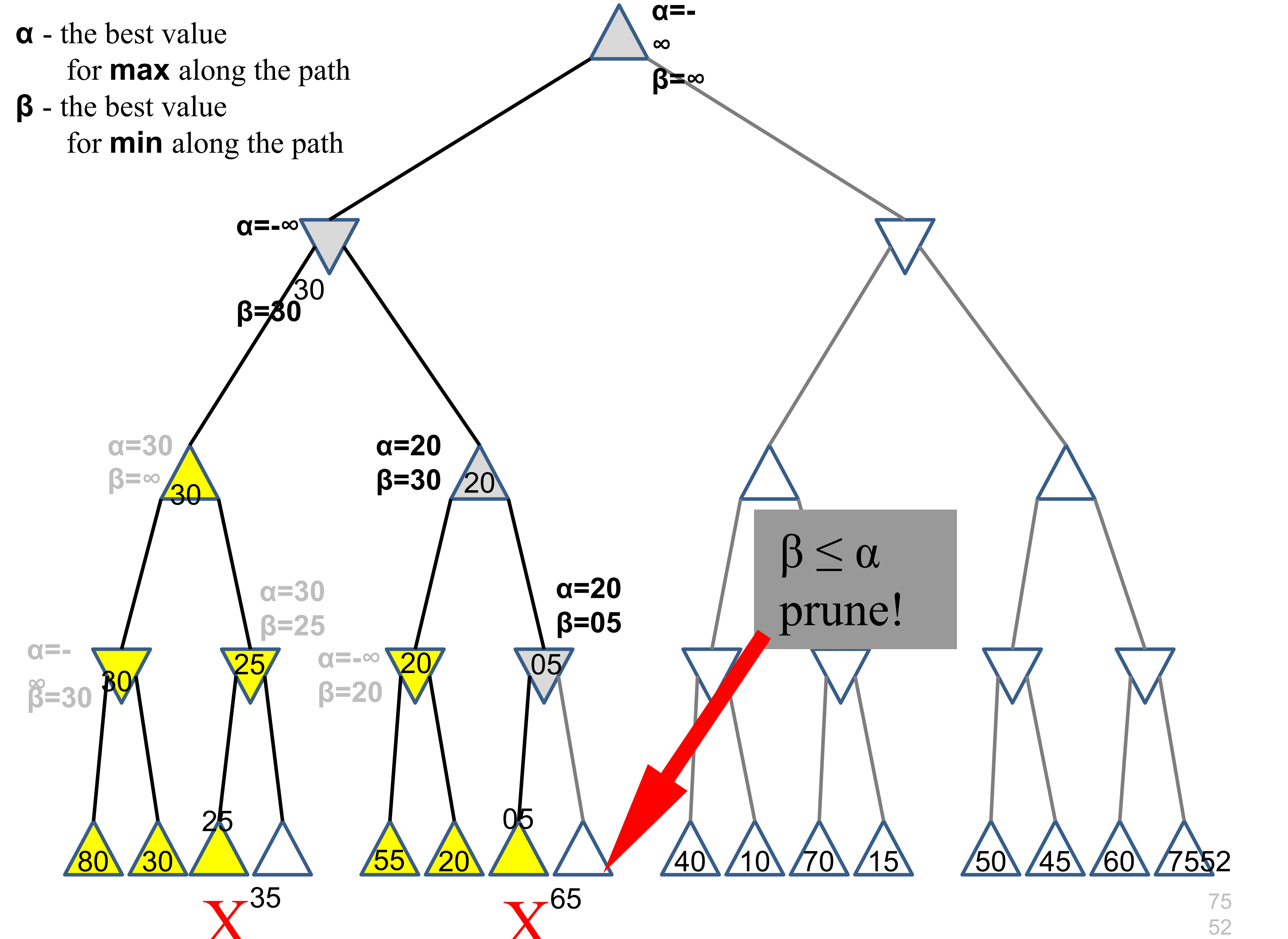


α - the best value
for **max** along the path
β - the best value
for **min** along the path

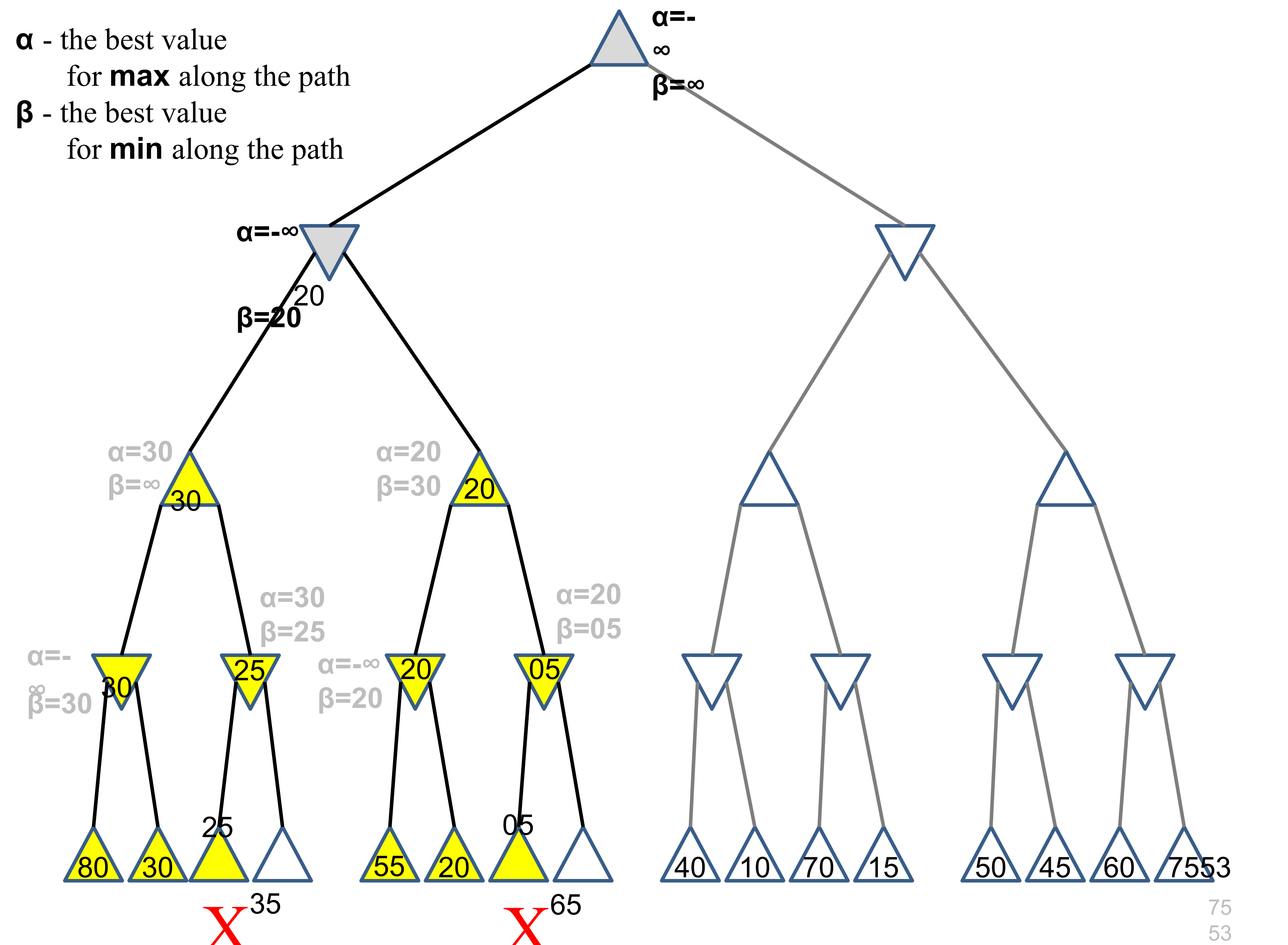


α - the best value
for **max** along the path

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

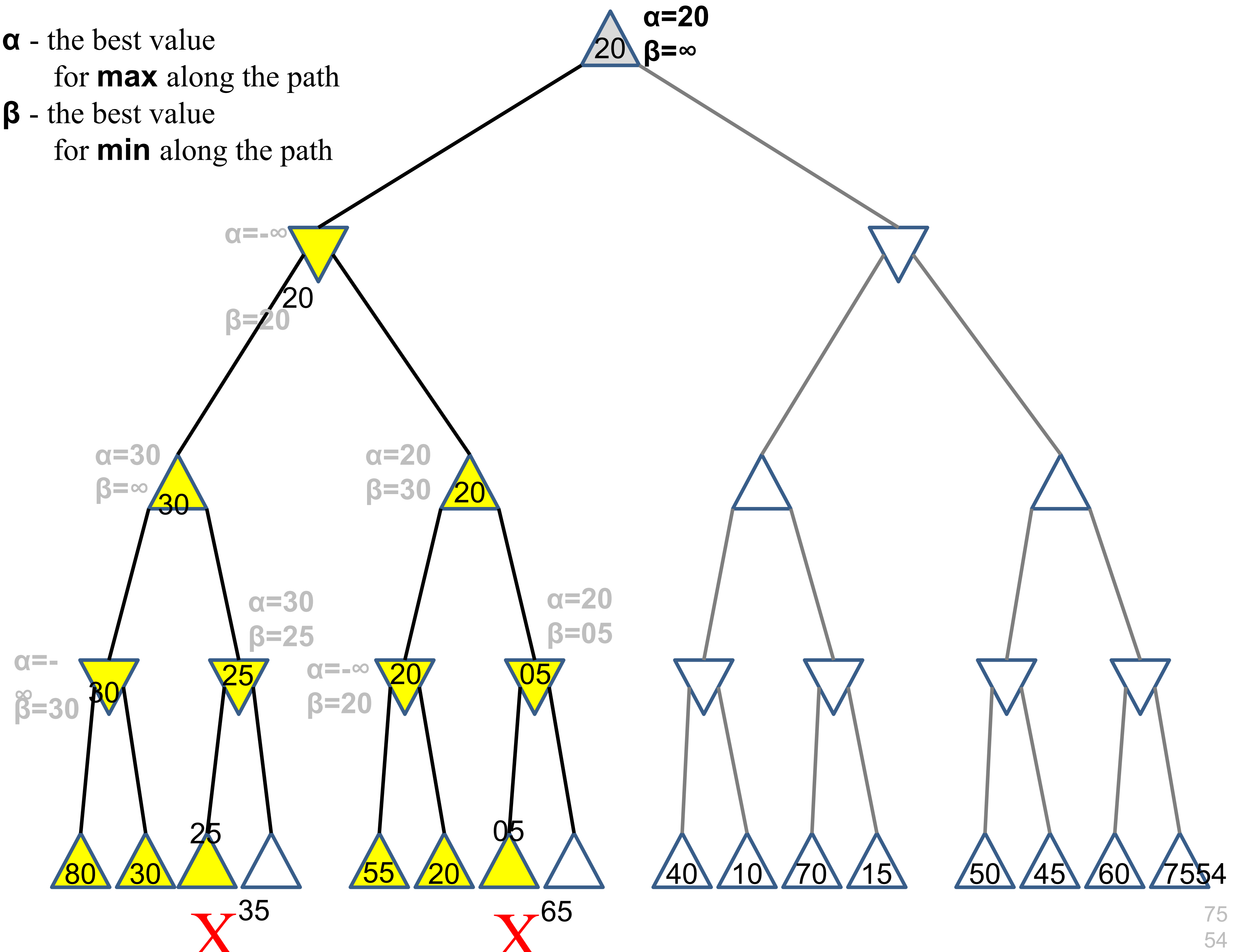


α - the best value
for **max** along the path
β - the best value
for **min** along the path



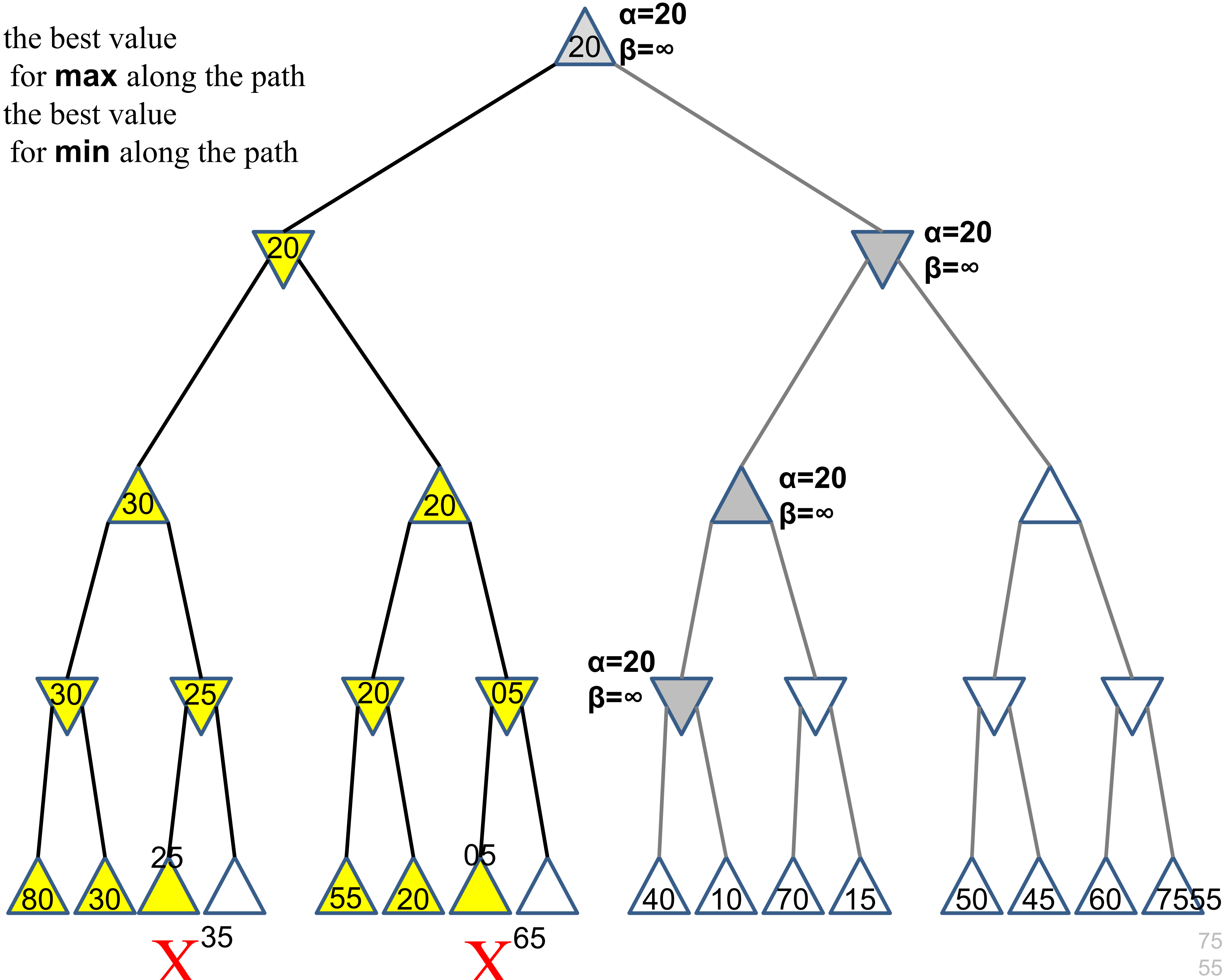
α - the best value
for **max** along the path

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



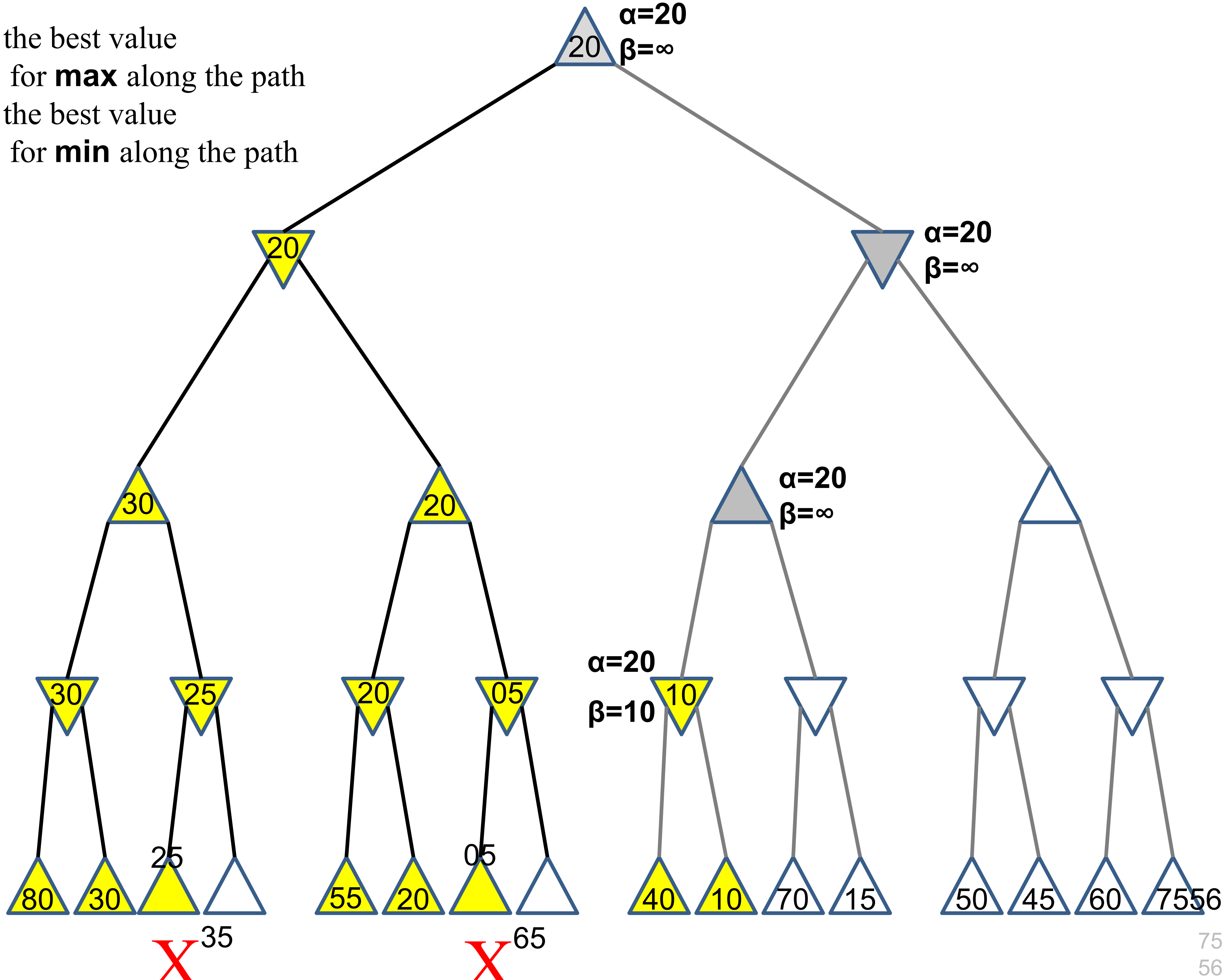
α - the best value
for **max** along the path

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



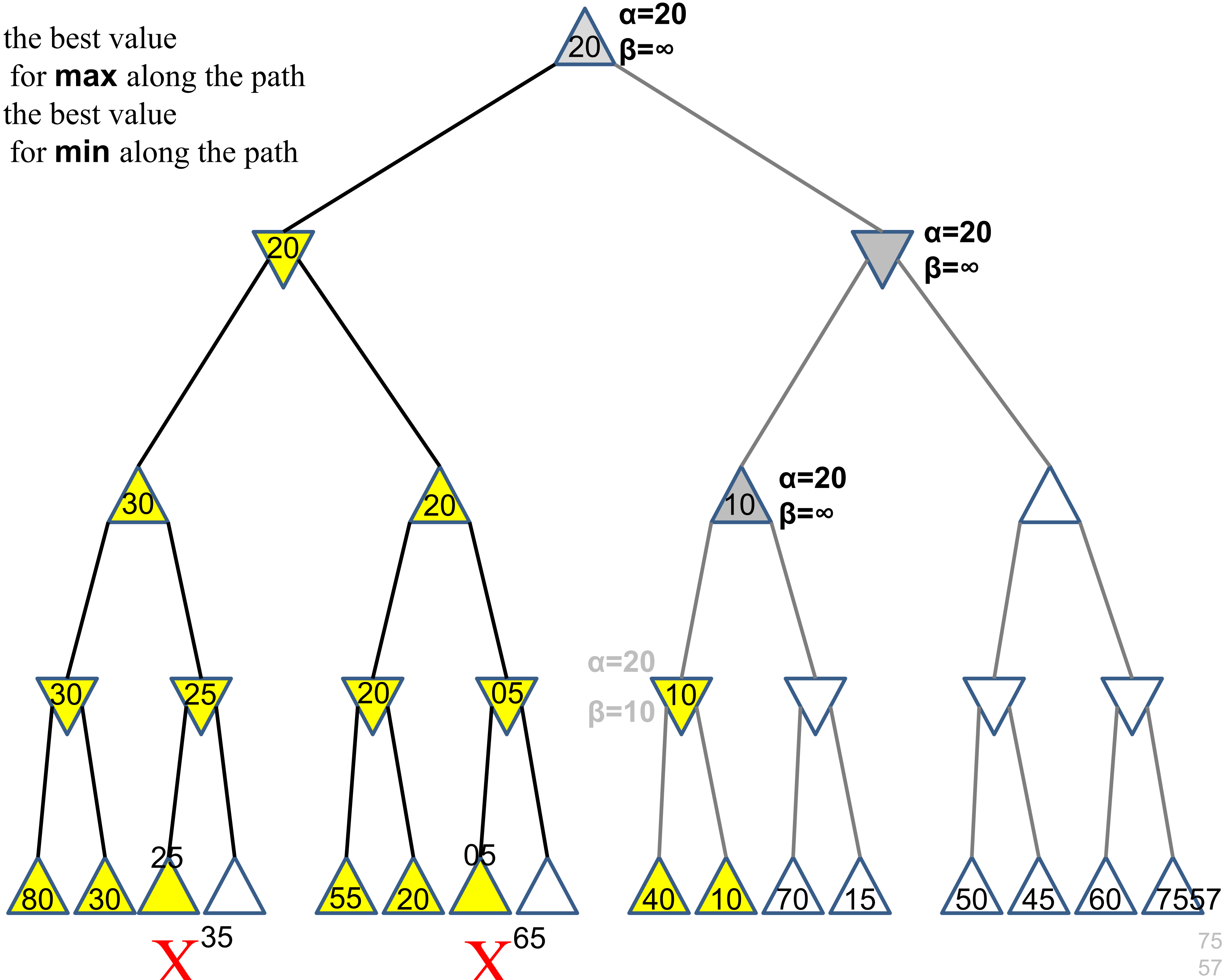
α - the best value
for **max** along the path

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

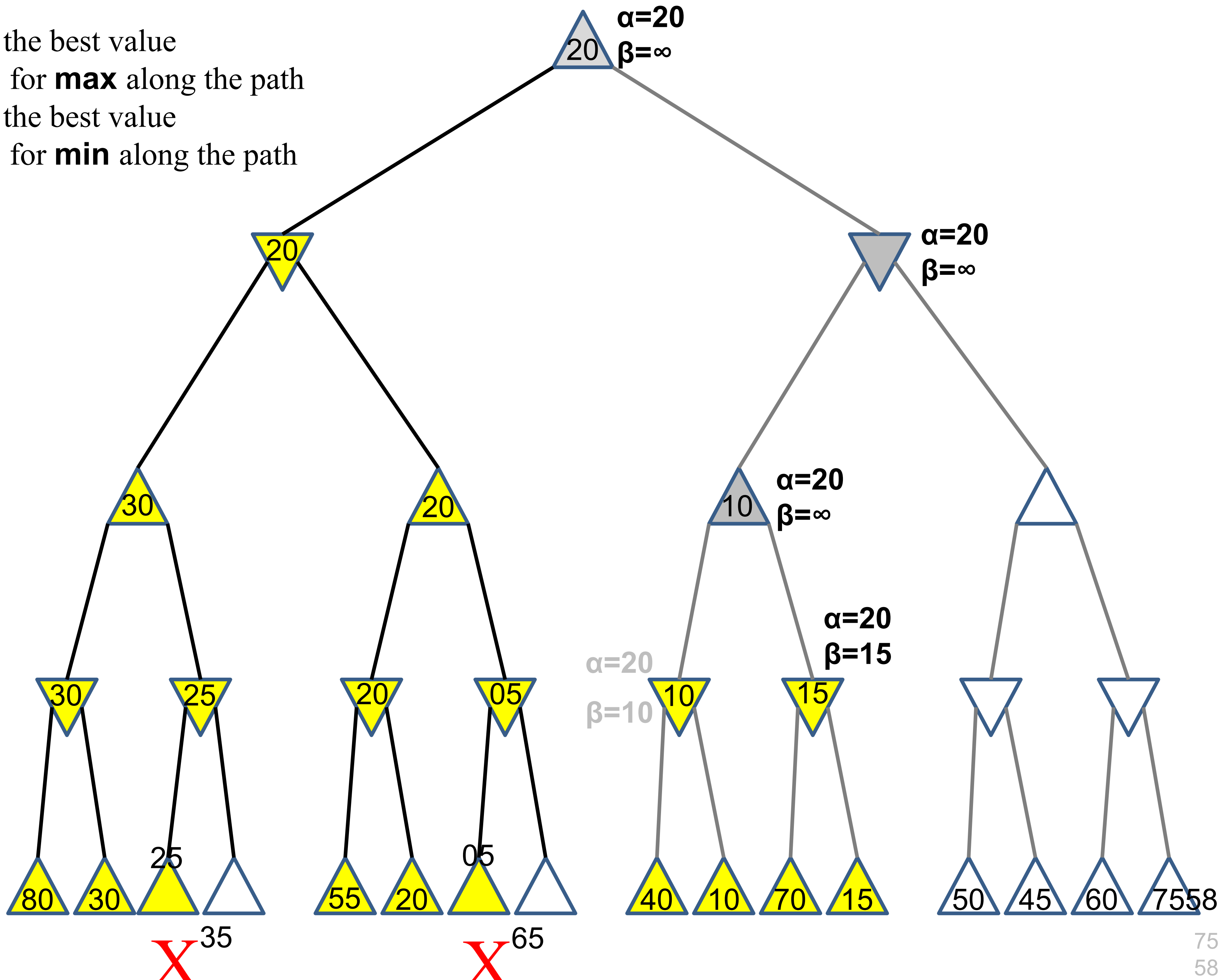


α - the best value
for **max** along the path

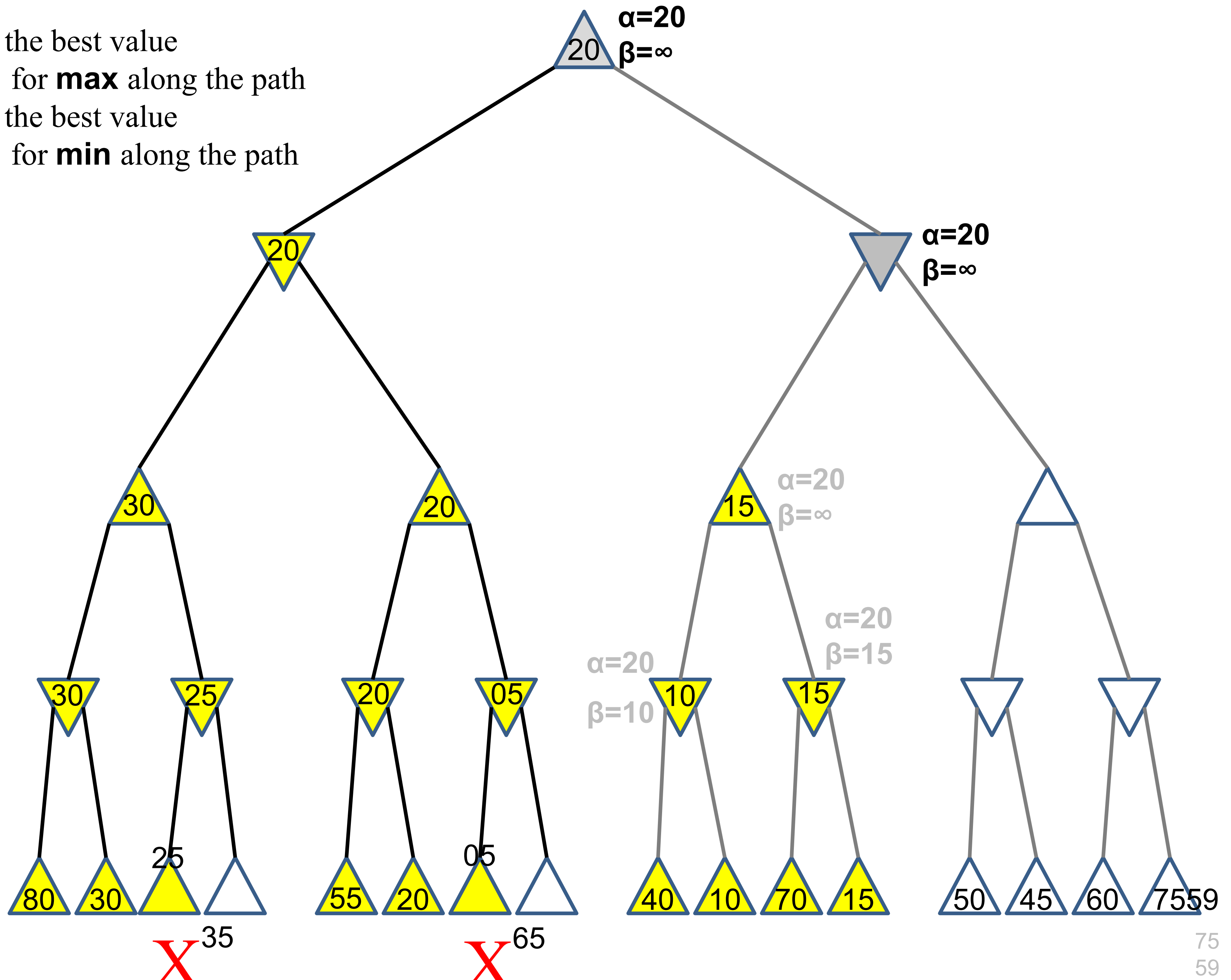
β - the best value
for **min** along the path



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

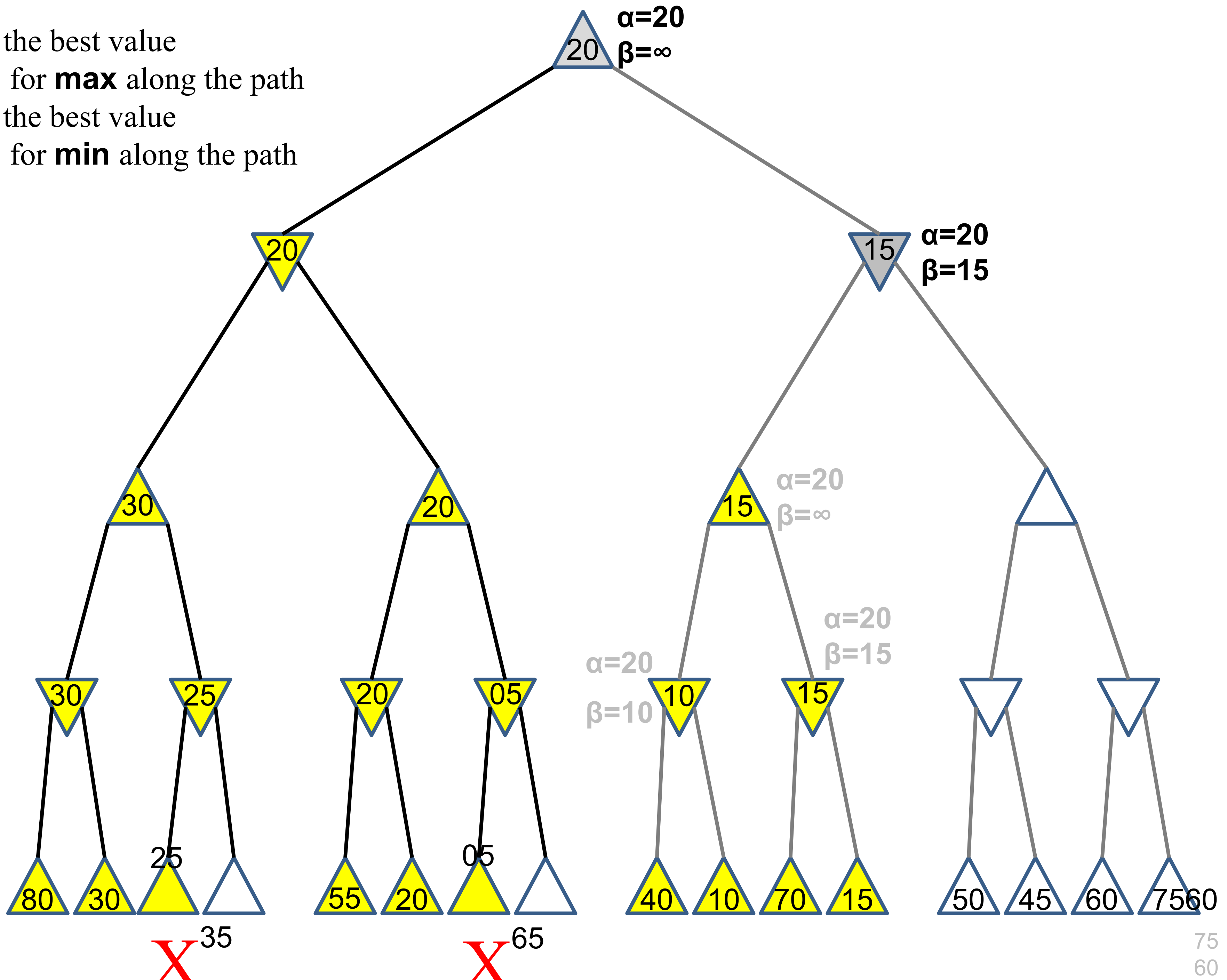


α - the best value
for **max** along the path
β - the best value
for **min** along the path



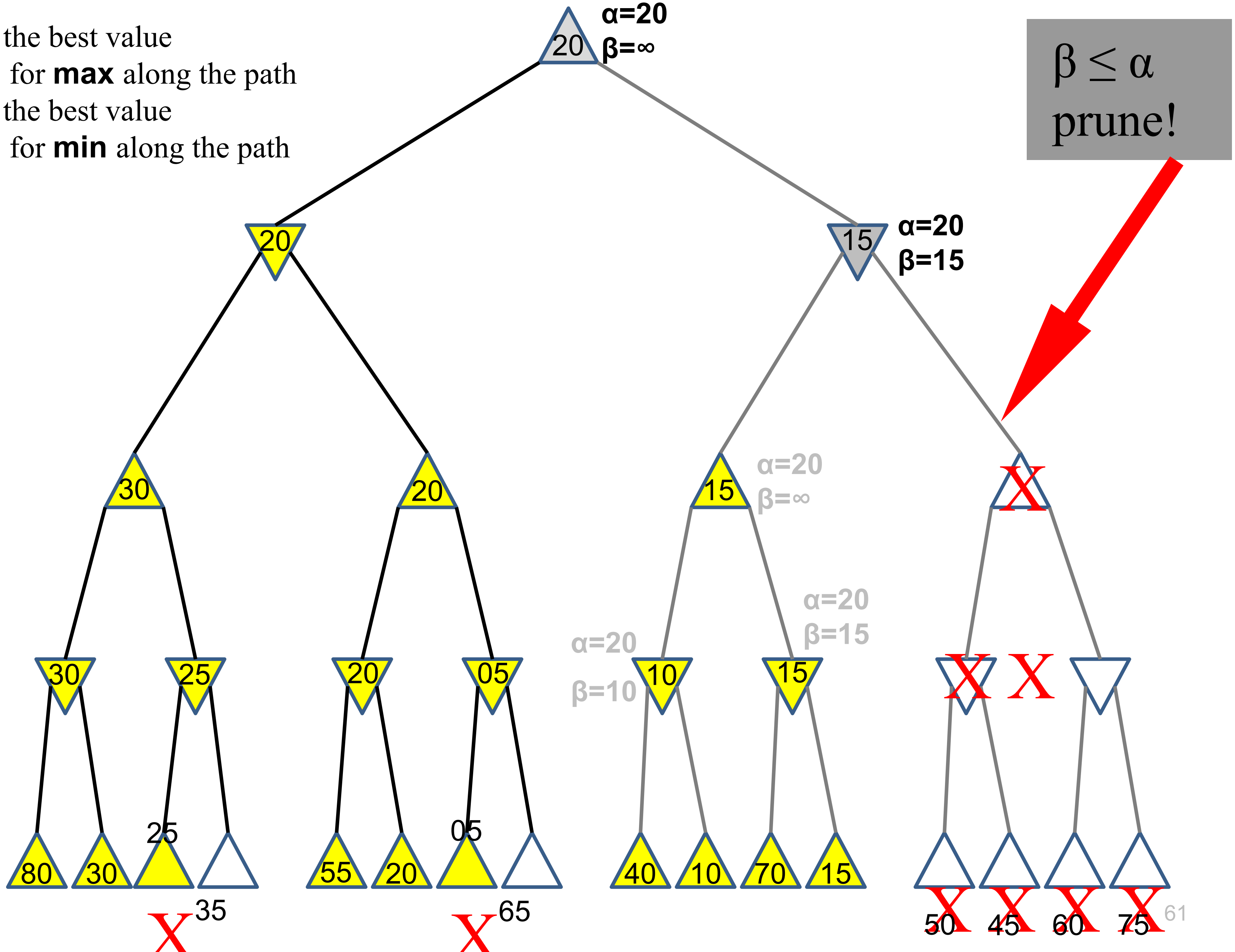
α - the best value
for **max** along the path

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                 34                        MIN
         +----+----+       +----+----+        +----+----+
         6     4           7     5     3     8     6     4       MAX
    +--+ +-2+ +--+ +-+-+ +--+ +--+ +--+ +--+ +--+--+  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       MAX
   +--+ +-x+ +--+ +--+
   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

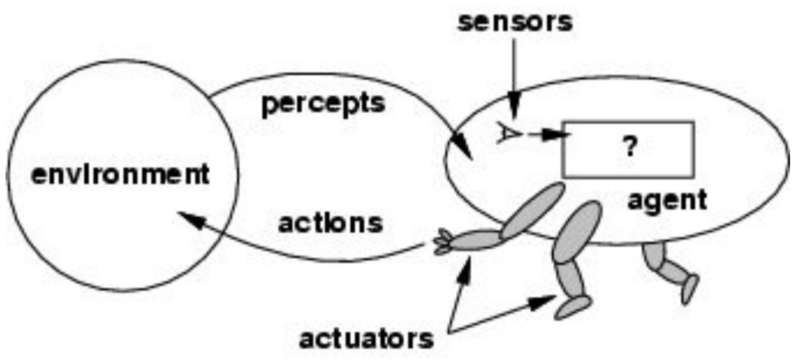
- Pruning does not affect final result. This means that it gets the exact same result as does full minimax.
- Good move ordering improves effectiveness of pruning
- With "perfect ordering," time complexity = $O(b^{m/2})$
   doubles depth of search
- A simple example of reasoning about 'which computations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

Intelligent Agents

- An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators

- **Human agent:** eyes, ears, and other organs for sensors;
- hands,legs, mouth, and other body parts for actuators
- **Robotic agent:** cameras and infrared range finders for sensors;
- various motors for actuators

Agents and Environment

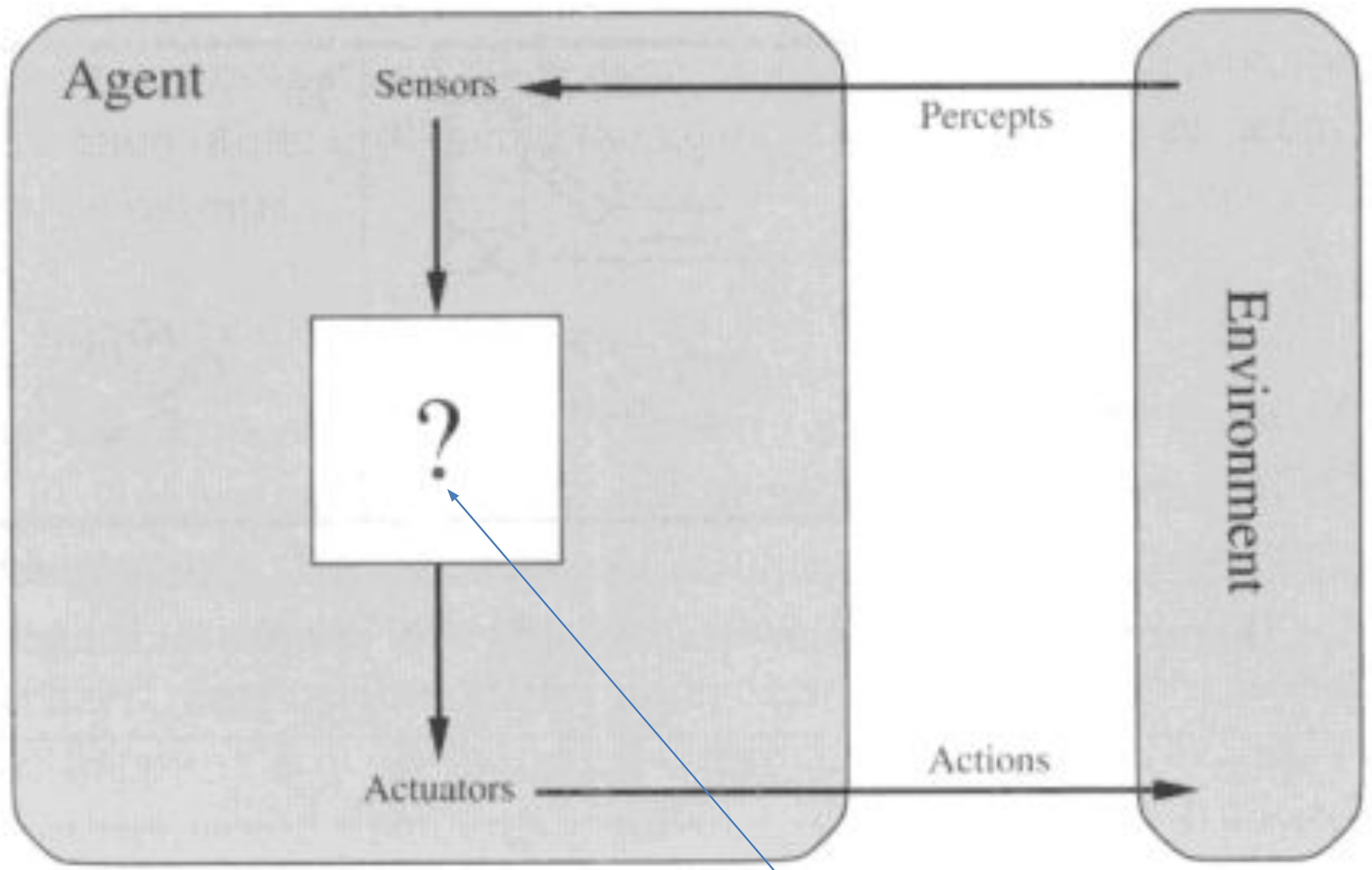


- The agent function maps from percept histories to actions:

$$[f: \mathcal{P}^\star \rightarrow \mathcal{A}]$$

- The agent program runs on the physical architecture to produce *f*
- agent = architecture + program

Diagram of an Agent



What AI should fill

Simple Terms

Percept

- Agent’s perceptual inputs at any given instant

Percept sequence

- Complete history of everything that the agent has ever perceived

Unit 1 List of Topics

- Introduction to AI-AI techniques
- Problem solving with AI
- AI Models, Data acquisition and learning aspects in AI
- Problem solving- Problem solving process, Formulating problems
- Problem types and characteristics
- Problem space and search
- Intelligent agent
- Rationality and Rational agent with performance measures
- Flexibility and Intelligent agents
- Task environment and its properties

- Types of agents
- Other aspects of agents
- Constraint satisfaction problems(CSP)
- Crypto arithmetic puzzles
- CSP as a search problem-constrains and representation
- CSP-Backtracking, Role of heuristic
- CSP-Forward checking and constraint propagation
- CSP-Intelligent backtracking

Rationality

- **Rationality**. **Rationality** is nothing but status of being reasonable, sensible, and having good sense of judgment. **Rationality** is concerned with expected actions and results depending upon what the agent has perceived. Performing actions with the aim of obtaining useful information is an important part of **rationality**.

What is Ideal Rational Agent?

- An ideal rational agent is the one, which is capable of doing expected actions to maximize its performance measure, on the basis of −
- Its percept sequence
- Its built-in knowledge base
- Rationality of an agent depends on the following −
- The **performance measures**, which determine the degree of success.
- Agent's **Percept Sequence** till now.
- The agent's **prior knowledge about the environment**.
- The **actions** that the agent can carry out.
- A rational agent always performs right action, where the right action means the action that causes the agent to be most successful in the given percept sequence. **The problem the agent solves is characterized by Performance Measure, Environment, Actuators, and Sensors (PEAS).**

Rational Agents

- Rational Agent: one that does the right thing
- Criteria: Performance measure
- Performance measures for
  - Web search engine?
  - Tic-tac-toe player? Chess player?
- How does when performance is measured play a role?
  - short vs. long term

Rational Agents

- Omniscient agent
  - Knows the actual outcome of its actions
  - What information would a chess player need to have to be omniscient?
- Omniscience is (generally) impossible
  - A rational agent should do the right thing based on the knowledge it has

Rational Agents

- What is rational depends on four things:
  - Performance measure
  - Percept sequence: everything agent has seen so far
  - Knowledge agent has about environment
  - Actions agent is capable of performing
- Ideal Rational Agent
  - Does whatever action is expected to maximize its performance measure, based on percept sequence and built-in knowledge

Ideal Mapping

- Percept sequence is only varying element of rationality
  - Percept Sequence Action
- In theory, can build a table mapping percept sequence to action
  - Sample table for chess board
- Ideal mapping describes behavior of ideal agents

The Mapping Table

- In most cases, mapping is explosively too large to write down explicitly
- In some cases, mapping can be defined via a specification
  - Example: agent to sort a list of numbers
  - Sample table for such an agent
  - Lisp code

Autonomy

- "Independence"
- A system is autonomous if its behavior is determined by its own experience
  - An alarm that goes off at a prespecified time is not autonomous
  - An alarm that goes off when smoke is sensed is autonomous
- A system without autonomy lacks flexibility

Structure of Intelligent Agents

- AI designes the agent program
- The program runs on some kind of architecture
- To design an agent program, need to understand
  - Percepts
  - Actions
  - Goals
  - Environment

Some Sample Agents

- What are percepts, actions, goals, and environment for:
  - Chess Player?
  - Web Search Tool?
  - Matchmaker?
  - Musical performer?
- Environments can be real (web search tool) or artificial (Turing test)
  - distinction is about whether it is a simulation for agent or the “real thing”

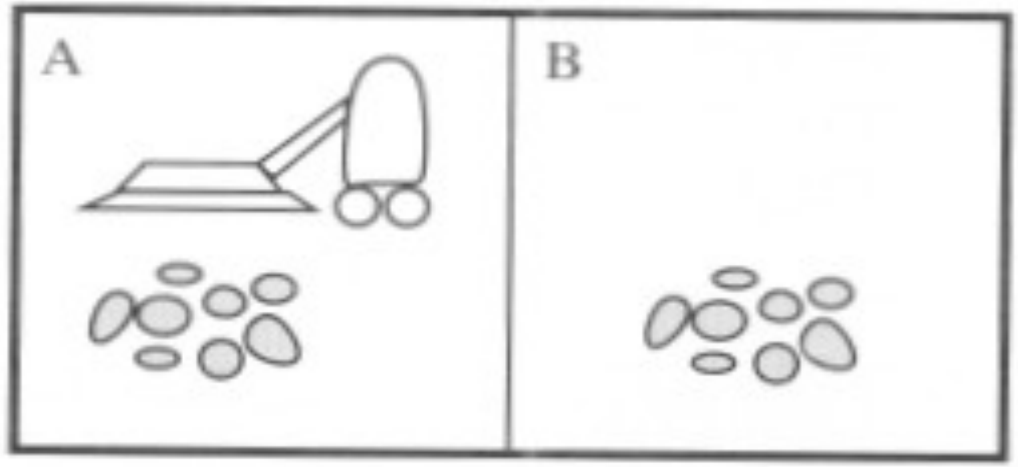
Agent Programs

- Some extra Lisp: Persistence of state (static variables)
- Allows a function to keep track of a variable over repeated calls.
  - Put functions inside a let block

  -
    ```
    (let ((sum 0))
       (defun myfun (x)
          (setf sum (+ sum x)))
       (defun report ()
           sum)
    )
    ```

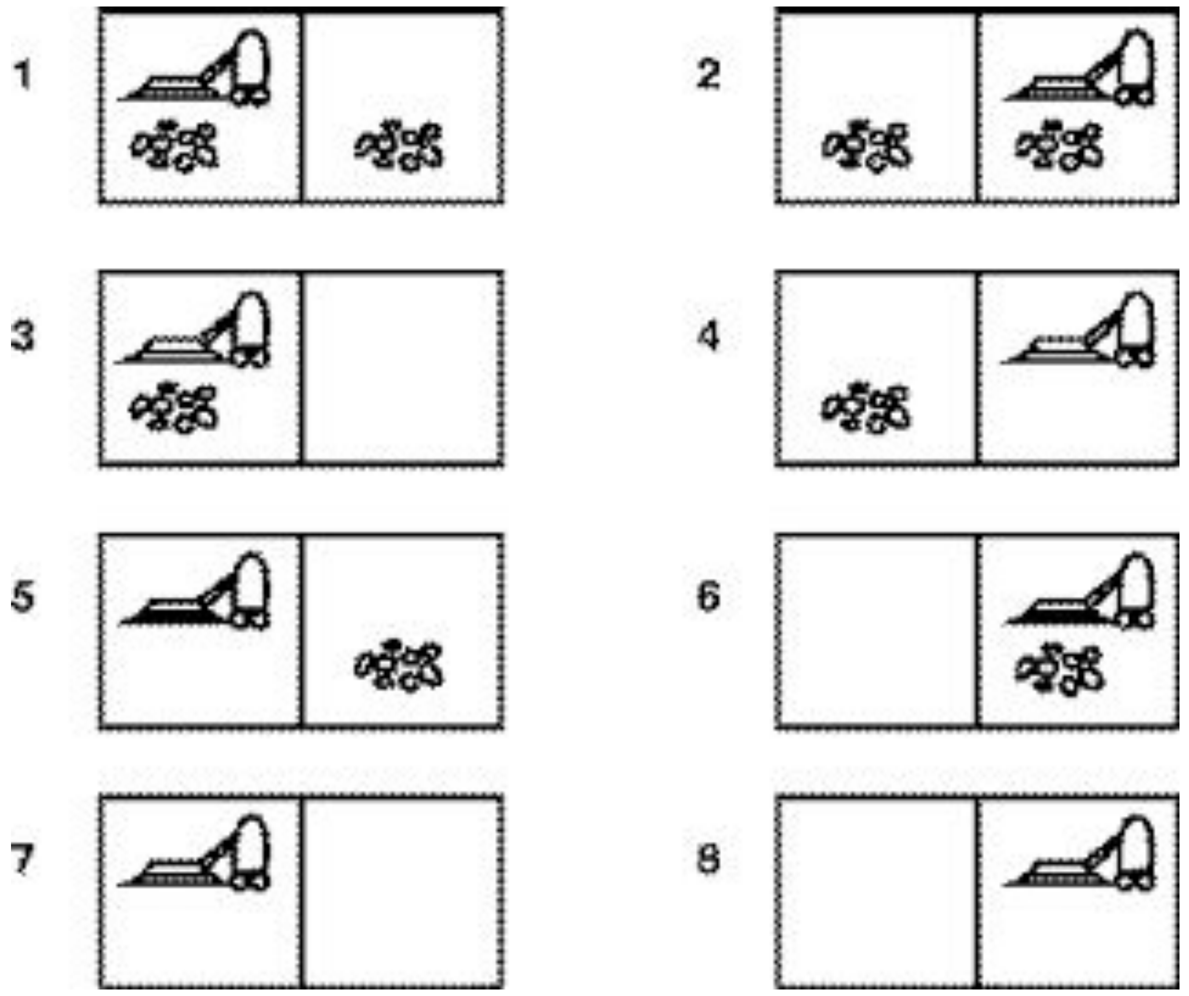
Vacuum-cleaner

Consider a Vacuum cleaner world



- Imagine that our intelligent agent is a robot vacuum cleaner.
  Let's suppose that the world has just two rooms.
- The robot can be in either room and there can be dirt in zero, one, or two rooms.

Vacuum-cleaner World



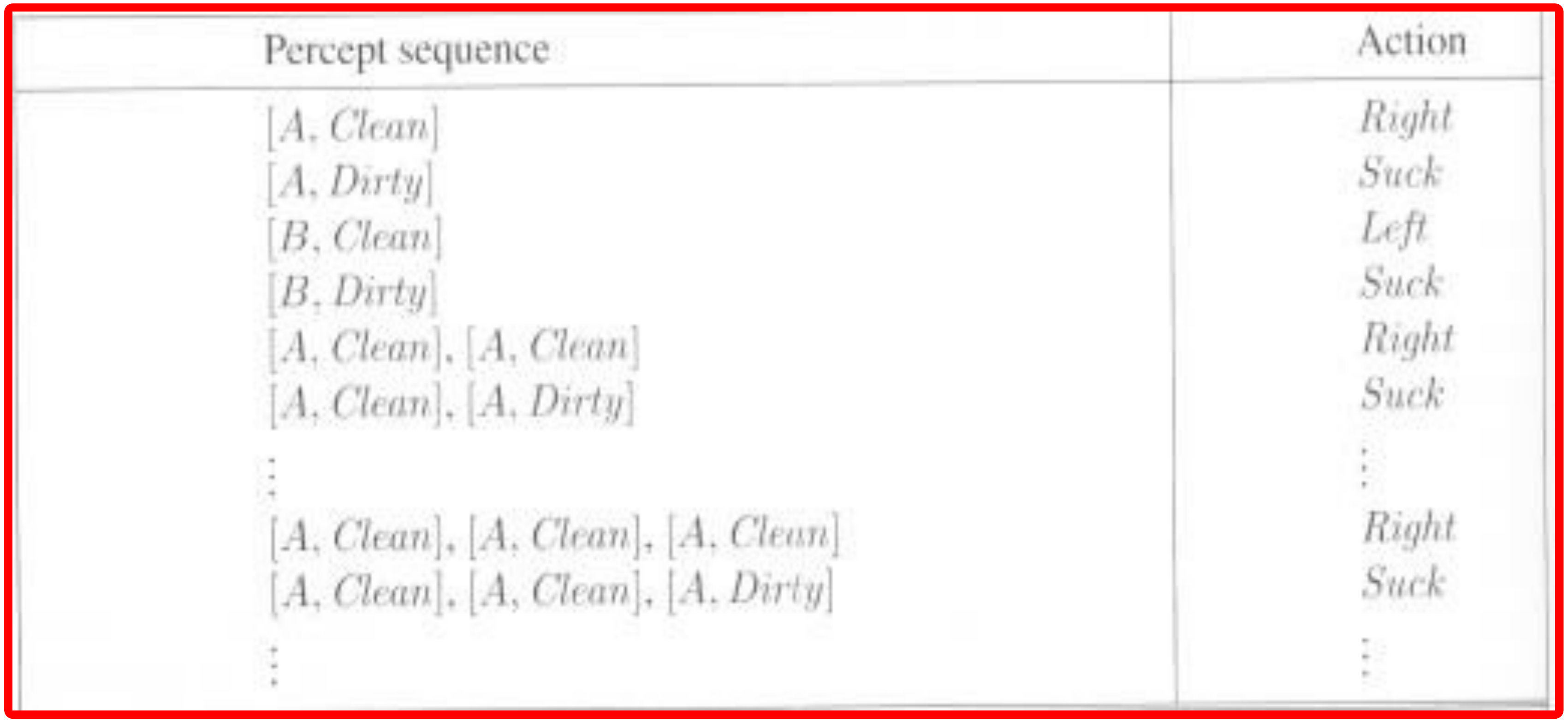
Vacuum-cleaner world

Perception: Clean or Dirty? where it is in?

Goal formulation: intuitively, we want all the dirt cleaned up. Formally, the goal is
{ state 7, state 8 }.

Problem formulation(Actions):
Move Left, Move Right, Suck, NoOp(do nothing)

Vacuum-cleaner world



Percept sequence	Action
[*A*, *Clean*]	*Right*
[*A*, *Dirty*]	*Suck*
[*B*, *Clean*]	*Left*
[*B*, *Dirty*]	*Suck*
[*A*, *Clean*], [*A*, *Clean*]	*Right*
[*A*, *Clean*], [*A*, *Dirty*]	*Suck*
⋮	⋮
[*A*, *Clean*], [*A*, *Clean*], [*A*, *Clean*]	*Right*
[*A*, *Clean*], [*A*, *Clean*], [*A*, *Dirty*]	*Suck*
⋮	⋮

Partial tabulation of a simple agent function for the vacuum-cleaner world

Vacuum-cleaner world

Program implements the agent function tabulated

```
**Function** Reflex-Vacuum-Agent([*location,status*]) return
   an action
  **If** *status* = *Dirty* **then return** *Suck*
  **else if** *location* = *A* **then return** *Right*
  **else if** *location* = *B* **then return** *left*
```

Unit 1 List of Topics

- Introduction to AI-AI techniques
- Problem solving with AI
- AI Models, Data acquisition and learning aspects in AI
- Problem solving- Problem solving process, Formulating problems
- Problem types and characteristics
- Problem space and search
- Intelligent agent
- Rationality and Rational agent with performance measures
- Flexibility and Intelligent agents
- Task environment and its properties

- Types of agents
- Other aspects of agents
- Constraint satisfaction problems(CSP)
- Crypto arithmetic puzzles
- CSP as a search problem-constrains and representation
- CSP-Backtracking, Role of heuristic
- CSP-Forward checking and constraint propagation
- CSP-Intelligent backtracking

Performance Measuring

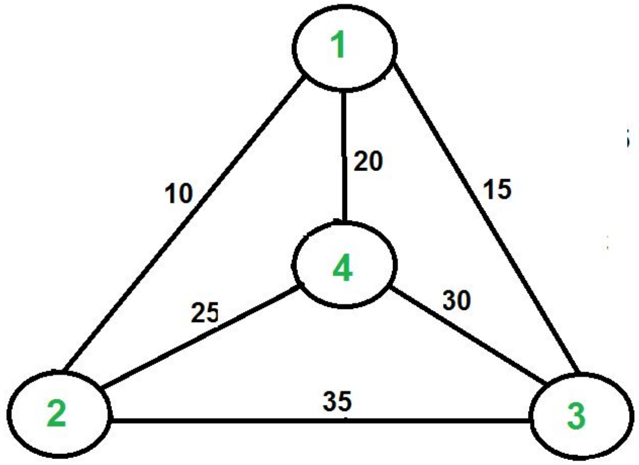
With any intelligent agent, we want it to find a (good) solution and not spend forever doing it.
The interesting quantities are, therefore,

- the **search cost**--how long the agent takes to come up with the solution to the problem, and
- the **path cost**--how expensive the actions in the solution are.

The **total cost** of the solution is the sum of the above two quantities.

Performance Measures

STATES,
OPERATORS,
GOAL TEST &
PATH COST



- **TSP - Naive Solution revisited:**

  1)Consider city 1 as the starting and ending point.

  2) Generate all (n-1)! Permutations of cities.

  3)Calculate cost of every permutation and keep track of minimum cost permutation.

  4) Return the permutation with minimum cost.

- Solution is complete, optimal, time and space complexity = n!

Measuring problem-solving performance

- Completeness: Is the algorithm guaranteed to find a solution when there is one?
- Optimality: Does the strategy find the optimal solution (i.e., lowest path cost among all solutions)
- Time complexity: How long does it take to find a solution?
- Space complexity: How much memory is needed to perform the search?

Real-world Problems

Example of rational action performed by any intelligent agent:

Automated Taxi Driver:

Performance Measure: Safe, fast, legal, comfortable trip, maximize profits.

Environment: Roads, other traffic, customers.

Actuators: Steering wheel, accelerator, brake, signal, horn.

Sensors: Cameras, sonar, speedometer, GPS, odometer, engine sensors, keyboard.

Unit 1 List of Topics

- Introduction to AI-AI techniques
- Problem solving with AI
- AI Models, Data acquisition and learning aspects in AI
- Problem solving- Problem solving process, Formulating problems
- Problem types and characteristics
- Problem space and search
- Intelligent agent
- Rationality and Rational agent with performance measures

- Flexibility and Intelligent agents
- Task environment and its properties
- Types of agents
- Other aspects of agents
- Constraint satisfaction problems(CSP)
- Crypto arithmetic puzzles
- CSP as a search problem-constrains and representation
- CSP-Backtracking, Role of heuristic
- CSP-Forward checking and constraint propagation
- CSP-Intelligent backtracking

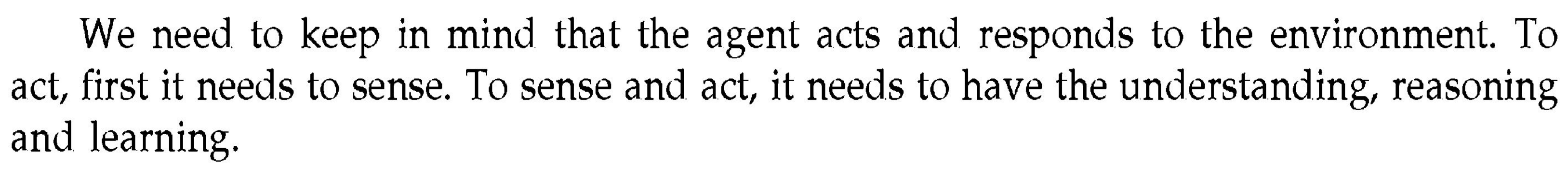
Flexibility and Intelligent agents

- Agents need to be Flexible.
- Handle dynamic Scenarios
- Flexibility Agent should need to,
  1. Responsive – timely respond
  2. Pro-Active - goal-directed behavior
  3. Social – should interact with other Artificial Agents and humans to complete problem-solving.

Other Properties of Intelligent Agent

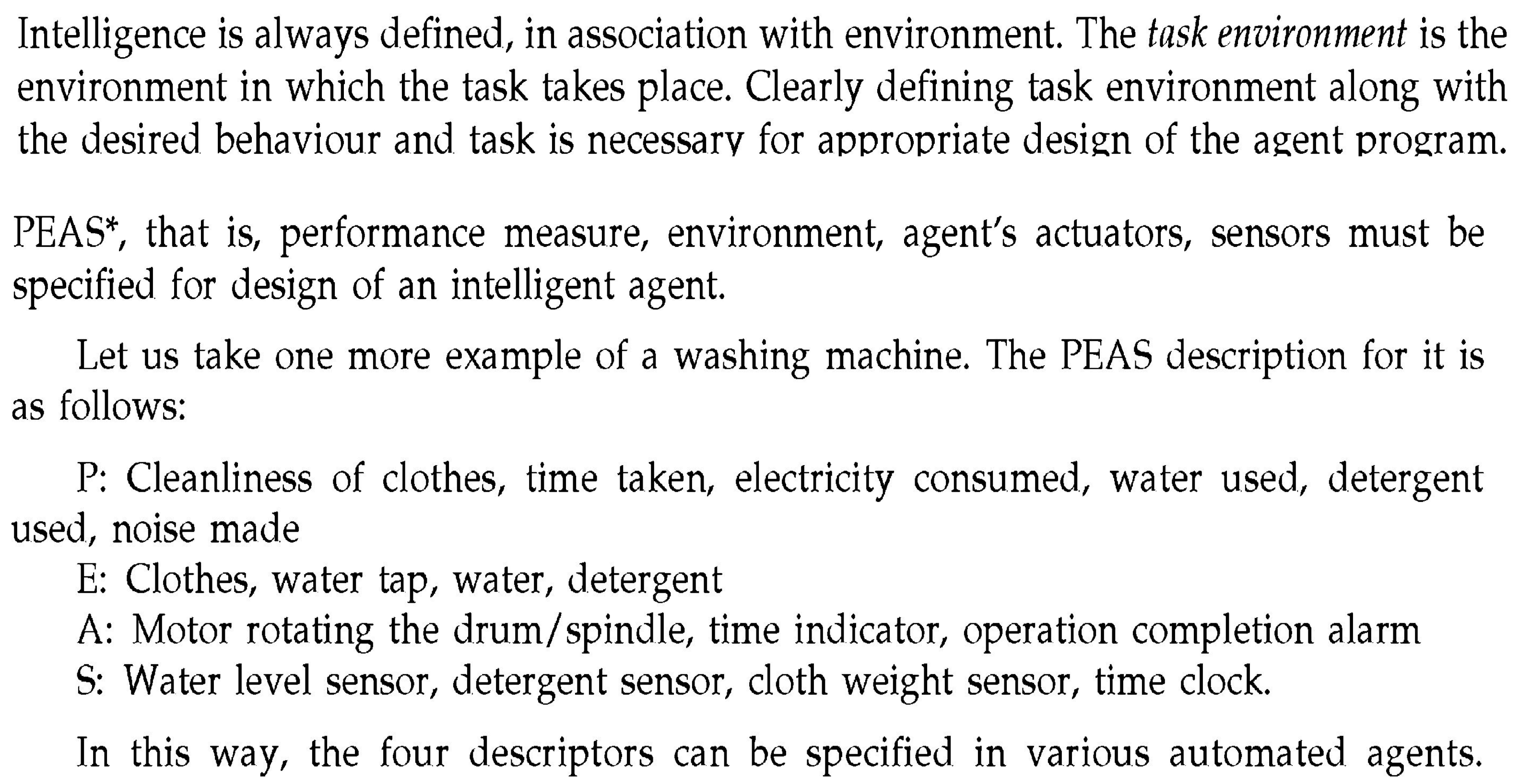
1. Mobility – Should be mobile to accumulate knowledge (decision making)
2. Veracity – truthful and should not hide information.
3. Benevolence – Should avoid conflict and should do what is told
4. Rationality – should act to maximize the expected performance
5. Learning – Performance is increased with learning, and you should have learning ability.

Note :



We need to keep in mind that the agent acts and responds to the environment. To act, first it needs to sense. To sense and act, it needs to have the understanding, reasoning and learning.

Task environment and its properties



Intelligence is always defined, in association with environment. The *task environment* is the environment in which the task takes place. Clearly defining task environment along with the desired behaviour and task is necessary for appropriate design of the agent program.

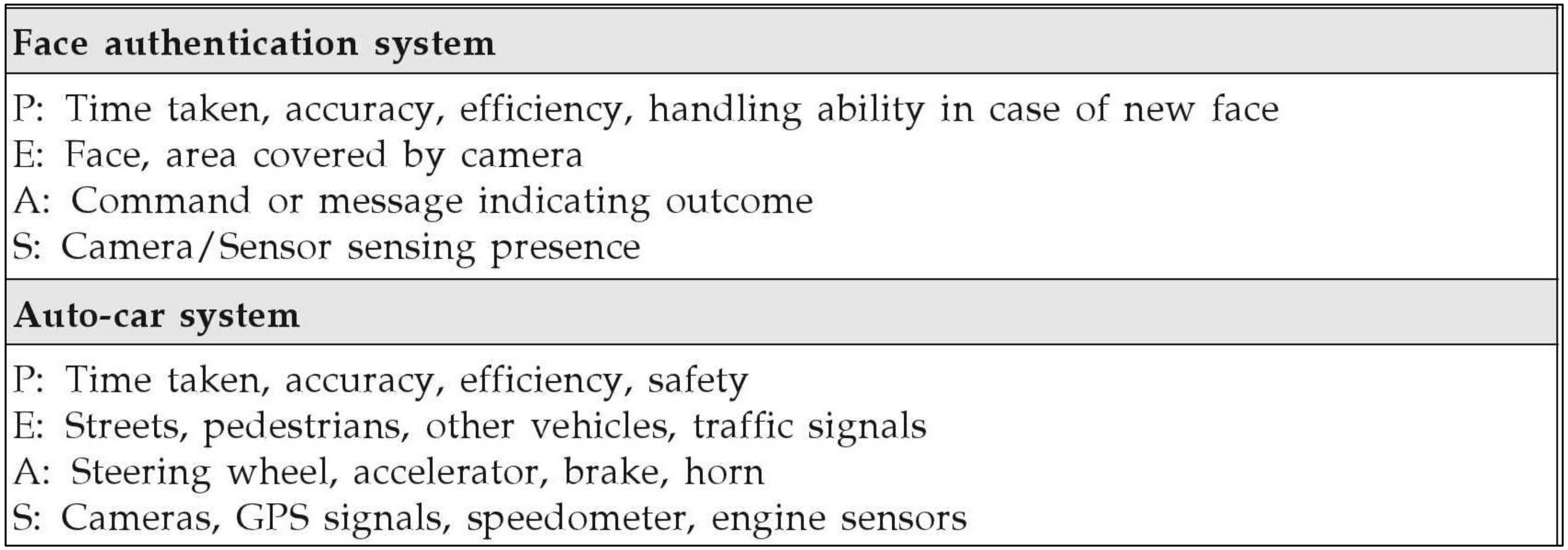
PEAS*, that is, performance measure, environment, agent's actuators, sensors must be specified for design of an intelligent agent.

Let us take one more example of a washing machine. The PEAS description for it is as follows:

P: Cleanliness of clothes, time taken, electricity consumed, water used, detergent used, noise made
E: Clothes, water tap, water, detergent
A: Motor rotating the drum/spindle, time indicator, operation completion alarm
S: Water level sensor, detergent sensor, cloth weight sensor, time clock.

In this way, the four descriptors can be specified in various automated agents.

EXAMPLE: Agent Types and their PEAS Description



Face authentication system
P: Time taken, accuracy, efficiency, handling ability in case of new face E: Face, area covered by camera A: Command or message indicating outcome S: Camera/Sensor sensing presence
Auto-car system
P: Time taken, accuracy, efficiency, safety E: Streets, pedestrians, other vehicles, traffic signals A: Steering wheel, accelerator, brake, horn S: Cameras, GPS signals, speedometer, engine sensors

PEAS Description is the core part of designing an agent.

Agent can be

Single and Multi-Agent

- Chess
- Crossword
- Football

Types of Agent

- Table Driven agents
- Simple Reflex agents
- Model – based reflex agents
- Goal–based agents
- Utility–based agents

Table Driven agents

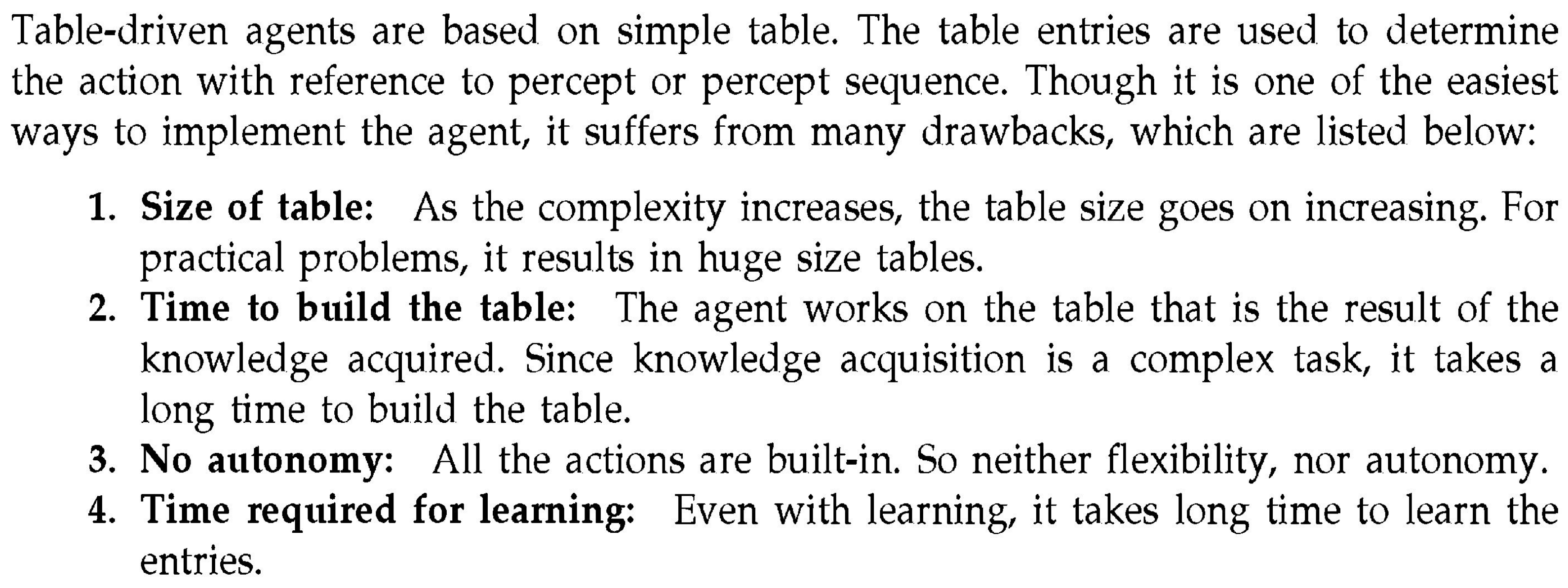


Table-driven agents are based on simple table. The table entries are used to determine the action with reference to percept or percept sequence. Though it is one of the easiest ways to implement the agent, it suffers from many drawbacks, which are listed below:

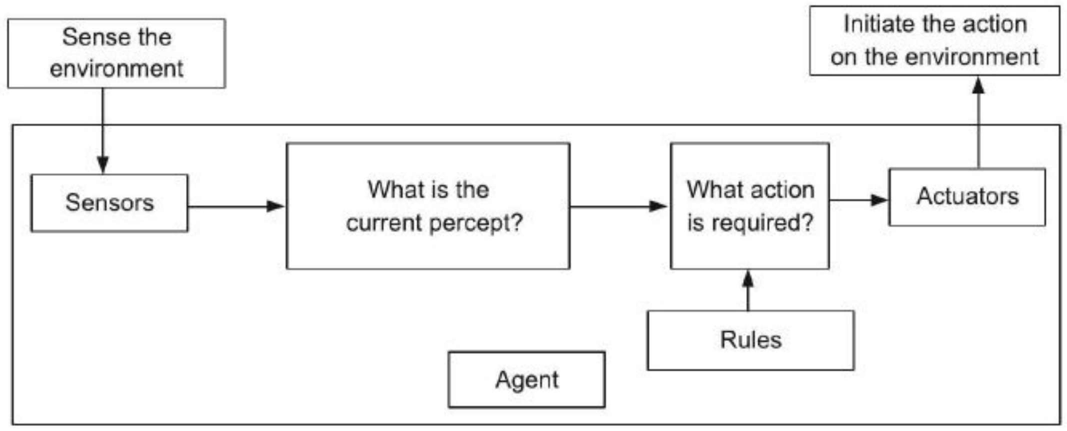
1. **Size of table:** As the complexity increases, the table size goes on increasing. For practical problems, it results in huge size tables.
2. **Time to build the table:** The agent works on the table that is the result of the knowledge acquired. Since knowledge acquisition is a complex task, it takes a long time to build the table.
3. **No autonomy:** All the actions are built-in. So neither flexibility, nor autonomy.
4. **Time required for learning:** Even with learning, it takes long time to learn the entries.

Simple Reflex agents

- Its one of the simple rule based agent that takes into account only the current percept.

Defn:

Action decided by condition

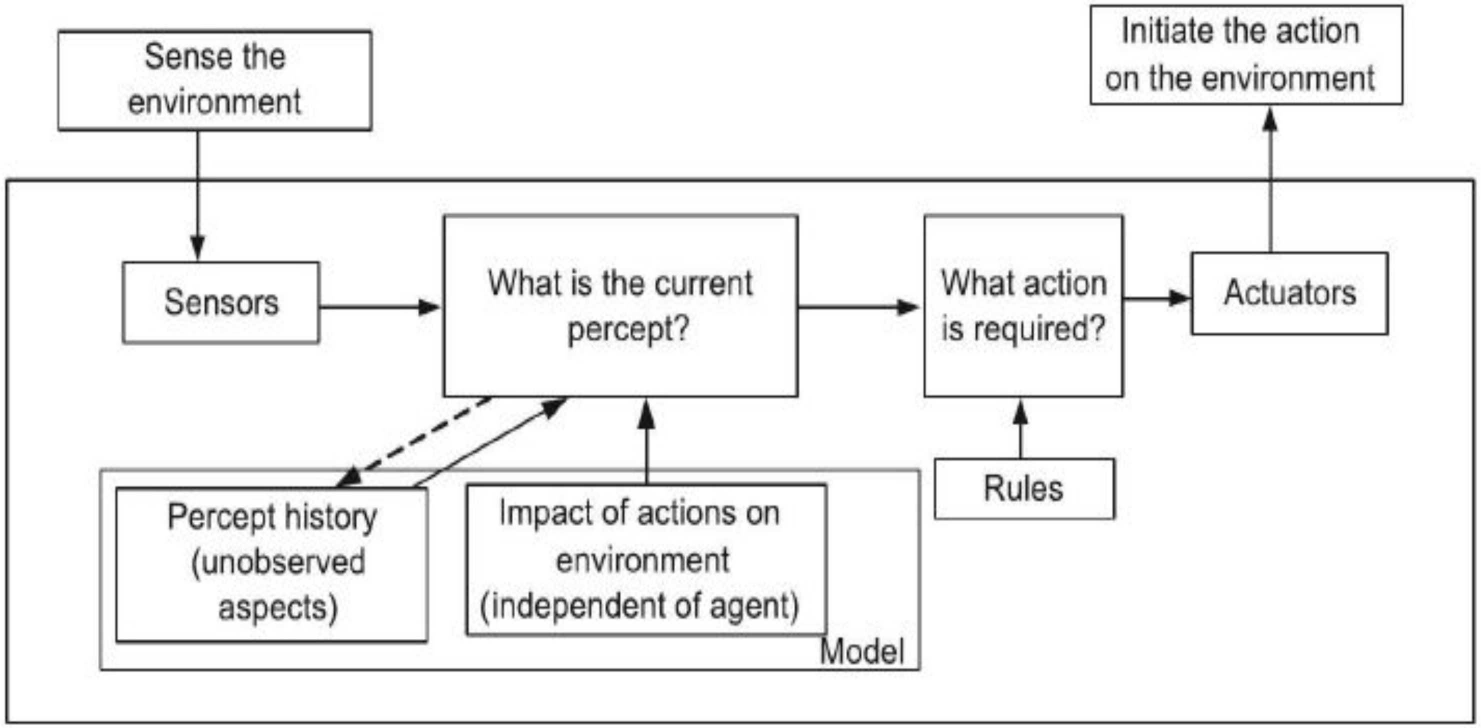


Model – based Reflex agents

- In Real life, problems face issues with a partially observable environment without keeping track of previous states.
- This agent has maintained some of the unobserved aspects of the current state. (reflects)
- This model need to know
  - How has the world evolved independent of agents?
  - How does the agent’s action affect the world?

  By incorporating these two things to determine the model.

Model – based Reflex Agents

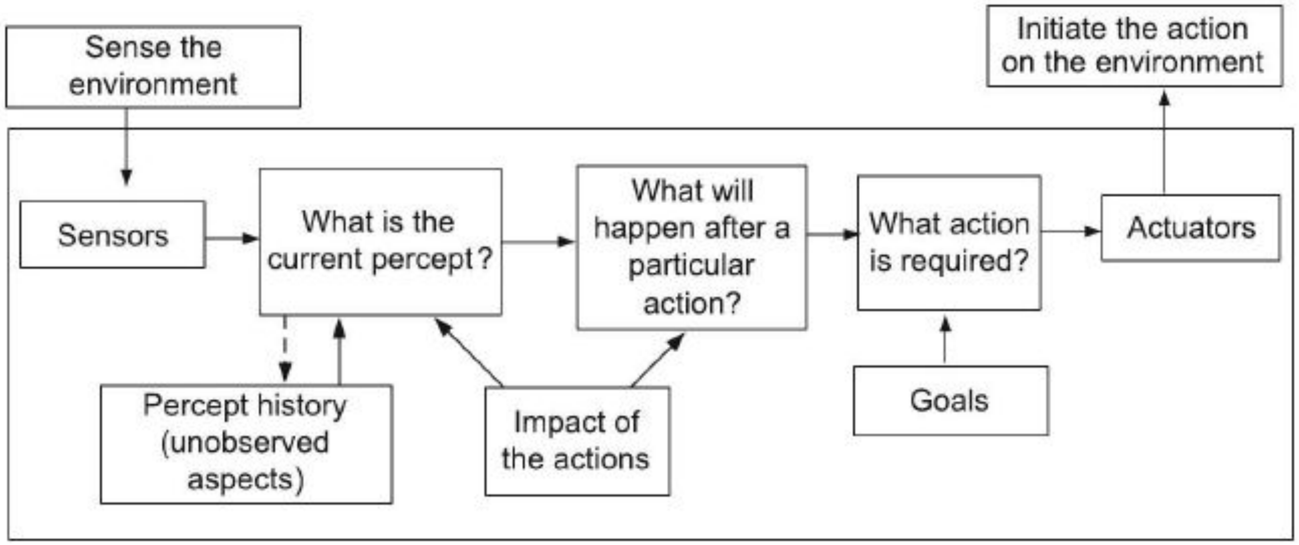

Sense the environment
Initiate the action on the environment
Sensors
What is the current percept?
What action is required?
Actuators
Rules
Percept history (unobserved aspects)
Impact of actions on environment (independent of agent)
Model

Goal–based agents

Example

Car driving (Landmark)

- Building knowledge with reference to goal.
- More amount of time is required for the reasoning process as it is goal-oriented.

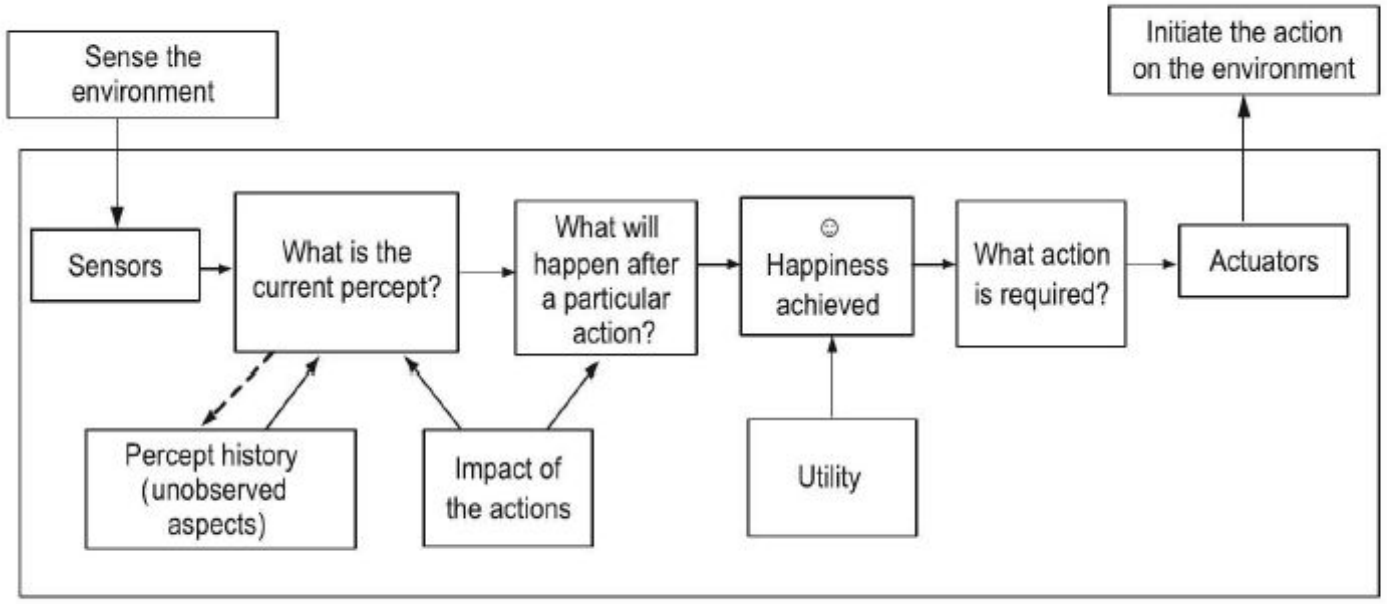

Sense the environment
Initiate the action on the environment
Sensors
What is the current percept?
What will happen after a particular action?
What action is required?
Actuators
Percept history (unobserved aspects)
Impact of the actions
Goals

The goal-based agents

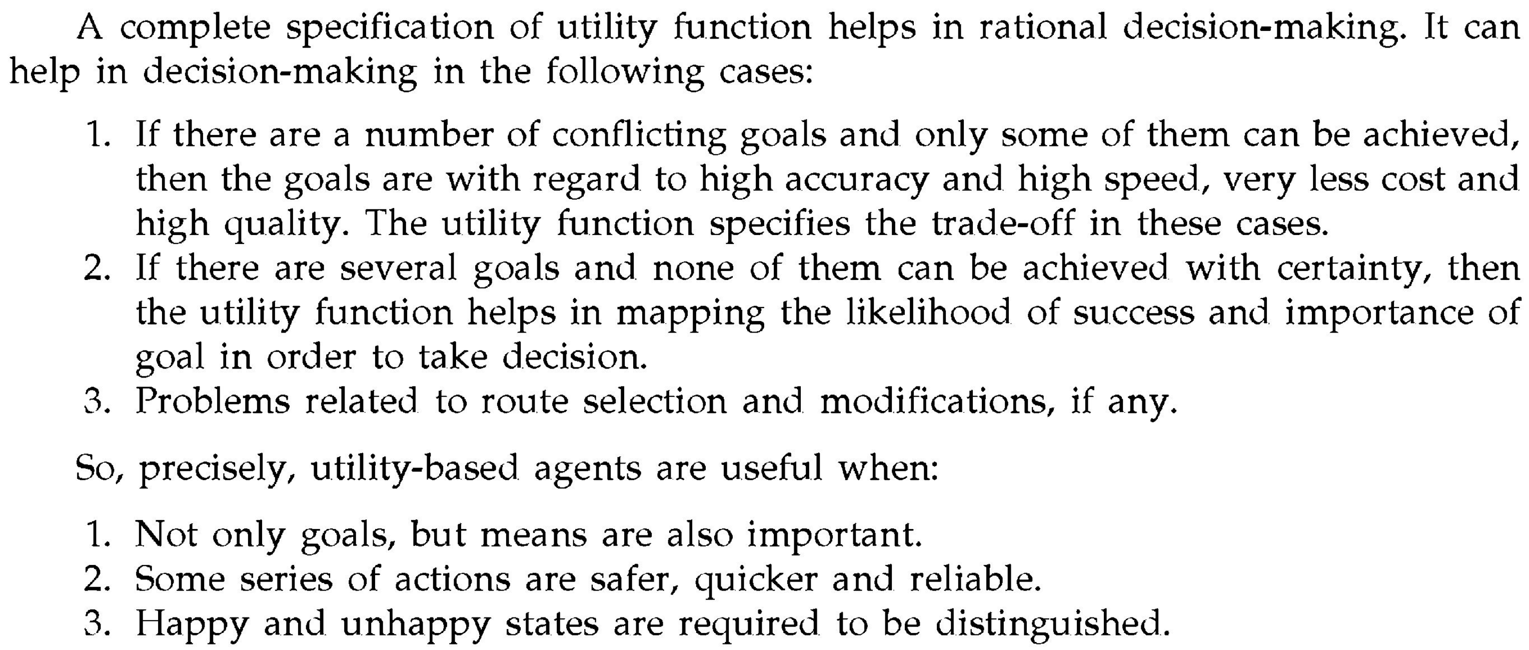
1. Require search and planning to drive the goal.
2. Need to be more flexible, as knowledge supports actions.

Utility–based agents

- Just a goal is not sufficient to describe what we want to achieve out of the agent.
- Utility function is most often a real number that maps to the degree of happiness.


Sense the environment
Initiate the action on the environment
Sensors
What is the current percept?
What will happen after a particular action?
☺ Happiness achieved
What action is required?
Actuators
Percept history (unobserved aspects)
Impact of the actions
Utility

Utility–based agents



A complete specification of utility function helps in rational decision-making. It can help in decision-making in the following cases:

1. If there are a number of conflicting goals and only some of them can be achieved, then the goals are with regard to high accuracy and high speed, very less cost and high quality. The utility function specifies the trade-off in these cases.
2. If there are several goals and none of them can be achieved with certainty, then the utility function helps in mapping the likelihood of success and importance of goal in order to take decision.
3. Problems related to route selection and modifications, if any.

So, precisely, utility-based agents are useful when:

1. Not only goals, but means are also important.
2. Some series of actions are safer, quicker and reliable.
3. Happy and unhappy states are required to be distinguished.

Learning Agents

- Important Element of Learning Agents

  1. Performance Element

  2. Learning Element

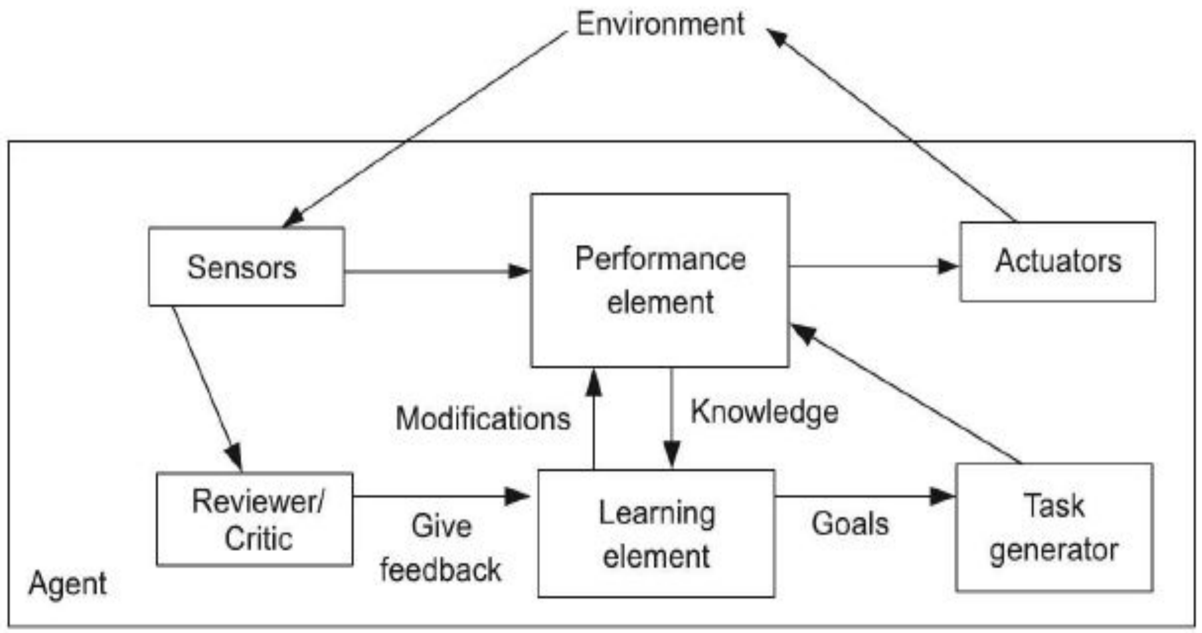
Performance Element:

Responsible for making improvements

Learning Element

Responsible for the selection of external actions.

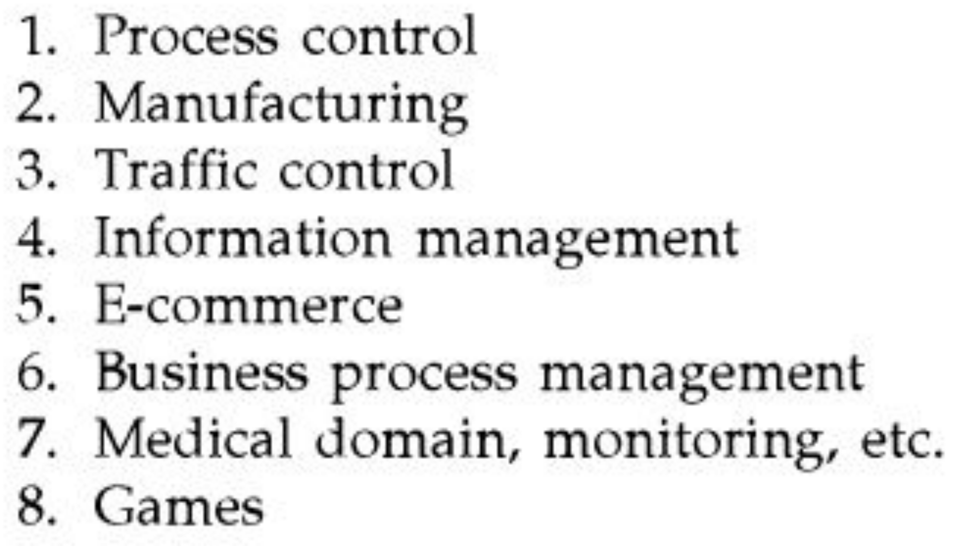
Learning Agent



Other Aspects of Intelligent Agents

- Dynamic Learning Capability

Application domains of Intelligent agents

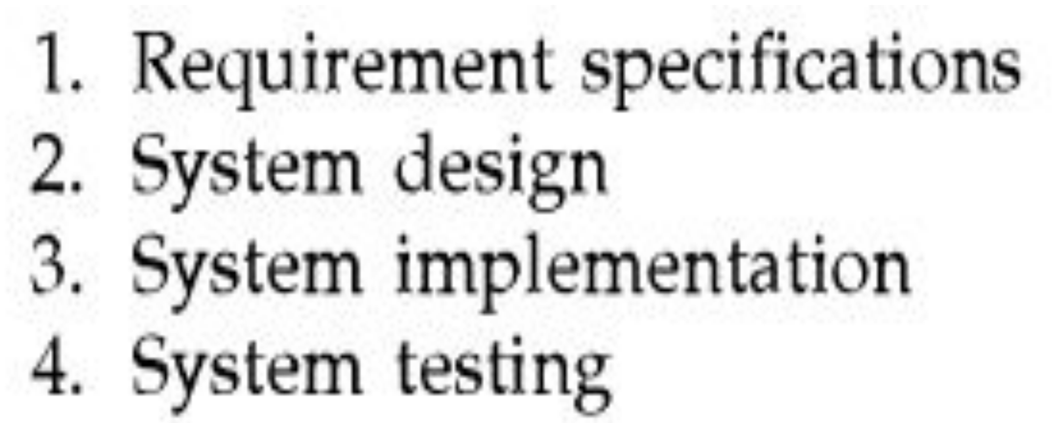


1. Process control
2. Manufacturing
3. Traffic control
4. Information management
5. E-commerce
6. Business process management
7. Medical domain, monitoring, etc.
8. Games

Draw backs of Intelligent Agents

- No overall system controllers
- No global perspective

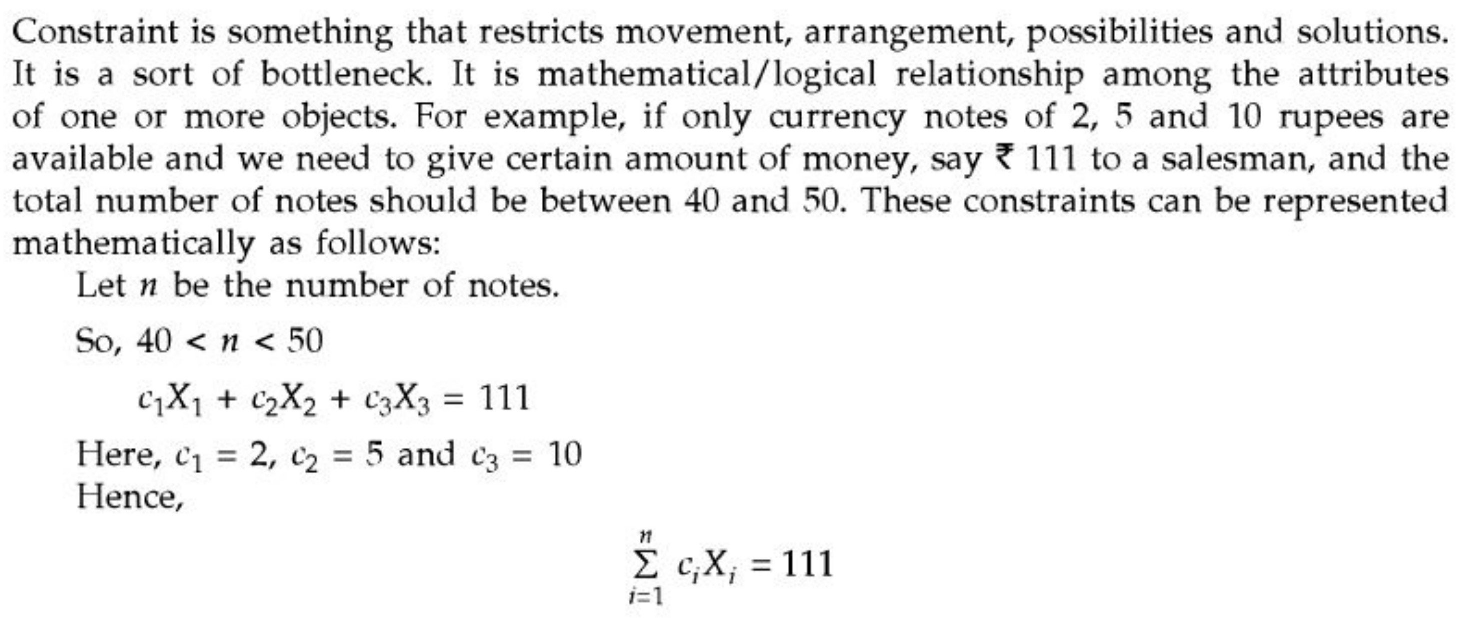
Bottlenecks in agent development



1. Requirement specifications
2. System design
3. System implementation
4. System testing

Constraint Satisfaction Problem(CSP)

- Constraint – Condition



Constraint is something that restricts movement, arrangement, possibilities and solutions. It is a sort of bottleneck. It is mathematical/logical relationship among the attributes of one or more objects. For example, if only currency notes of 2, 5 and 10 rupees are available and we need to give certain amount of money, say ₹ 111 to a salesman, and the total number of notes should be between 40 and 50. These constraints can be represented mathematically as follows:

Let n be the number of notes.

So, $40 < n < 50$

$$c_1X_1 + c_2X_2 + c_3X_3 = 111$$

Here, $c_1 = 2$, $c_2 = 5$ and $c_3 = 10$

Hence,

$$\sum_{i=1}^{n} c_iX_i = 111$$

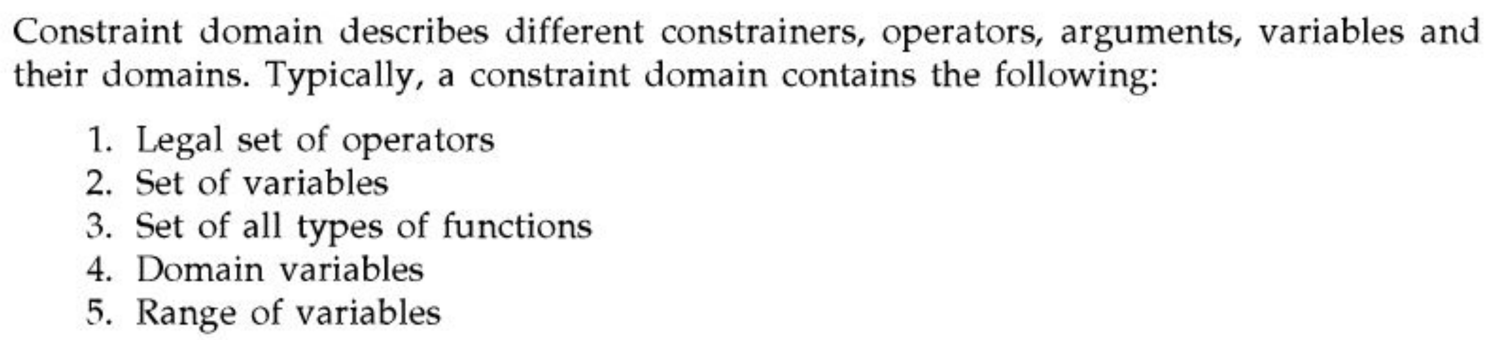
Types of Constraint

- Unary Constraint
- Binary Constraint
- Higher order Constraint

Example for Constraint Satisfaction and Constraint Programming

1. School, college or any time table with limited classrooms and teachers
2. Management of employees to achieve certain task
3. Course planning and scheduling
4. Bus route planning
5. Resource allocation
6. Network configuration

Constraint Domain



Constraint domain describes different constrainers, operators, arguments, variables and their domains. Typically, a constraint domain contains the following:

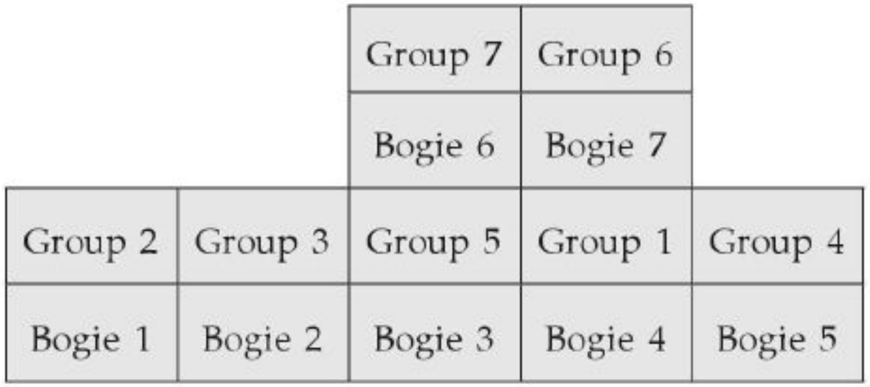
1. Legal set of operators
2. Set of variables
3. Set of all types of functions
4. Domain variables
5. Range of variables

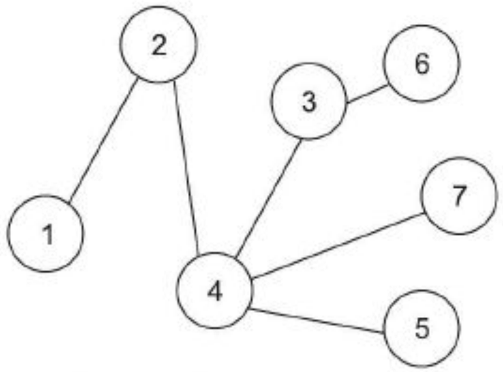
Group Assignment Problem

There are seven groups and seven bogies.

Group 1 can speak English, Marathi and Hindi.
Group 2 can speak Tamil and Kannad.
Group 3 can speak English, Kannad and Punjabi.
Group 4 can speak Gujarati and Hindi.
Group 5 can speak Kannad and Hindi.
Group 6 can speak Marathi and Bangla.
Group 7 can speak Hindi and Bangla.

We need to put groups in bogies so that every group should be able to speak to a group in adjacent bogies (Figure 6.2).



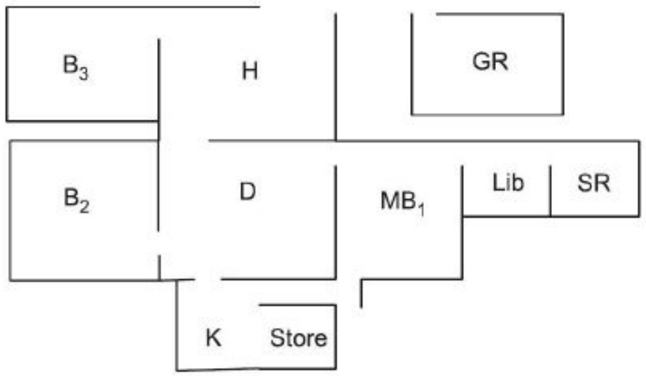


CSP As a SEARCH PROBLEM

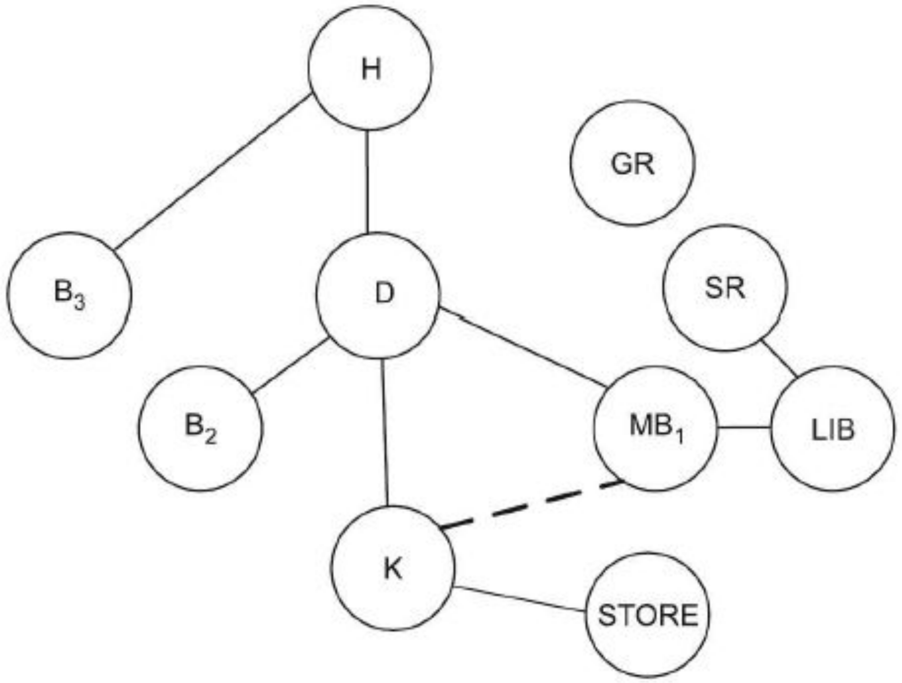
Here, K is kitchen, H is hall, D is dining room, Store is storeroom, B_2, B_3 are bedrooms 2 and 3, MB_1 is the master bedroom, SR is the study room, GR is the guest Room, Lib is library.

Constraints

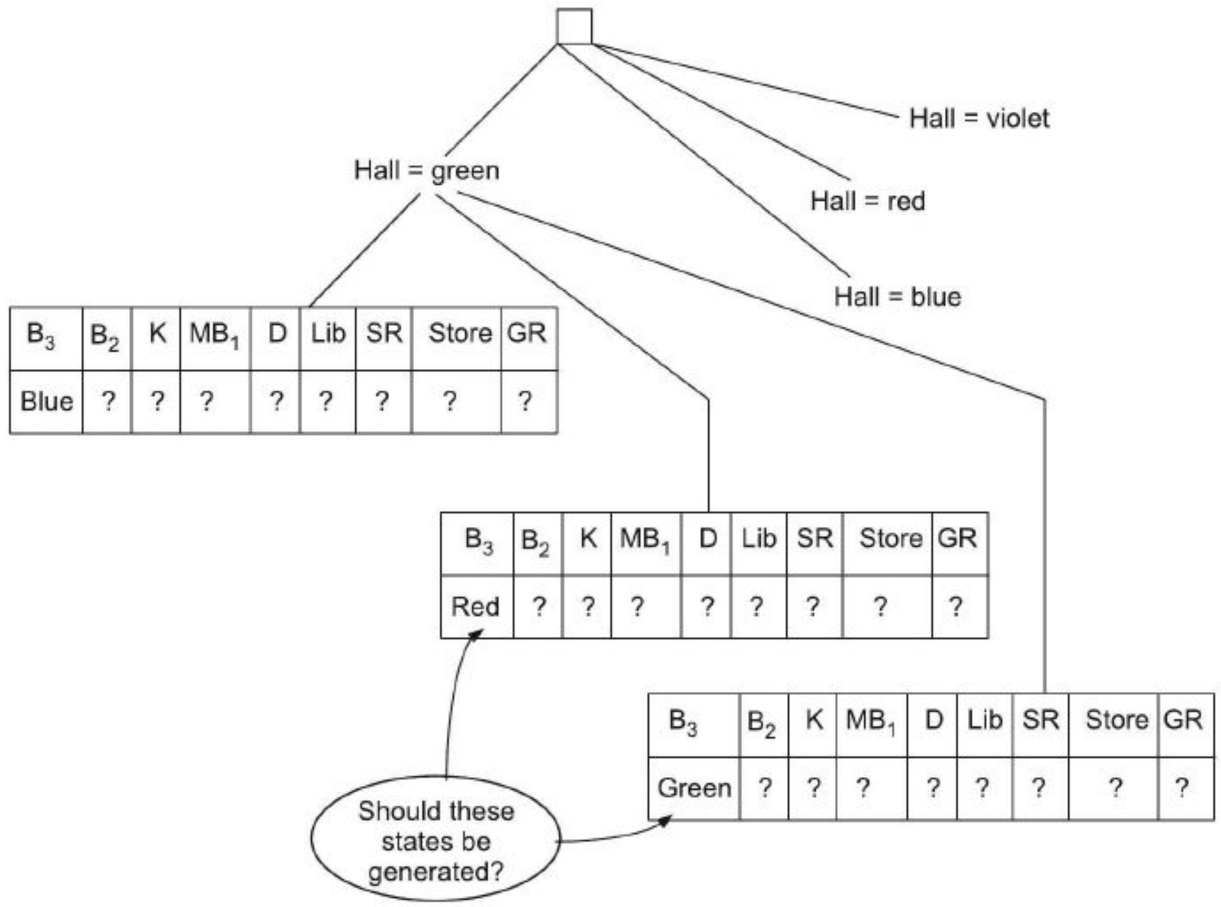
1. All bedrooms should not be coloured in red, only one can.
2. No two adjacent rooms can have the same colour.
3. The colours available are blue, red, green and violet.
4. Kitchen should not be coloured in green.
5. It is recommended that the kitchen should be coloured blue.
6. Dining room should not have violet colour.



Representation as a Search Problem



Search Tree Generation

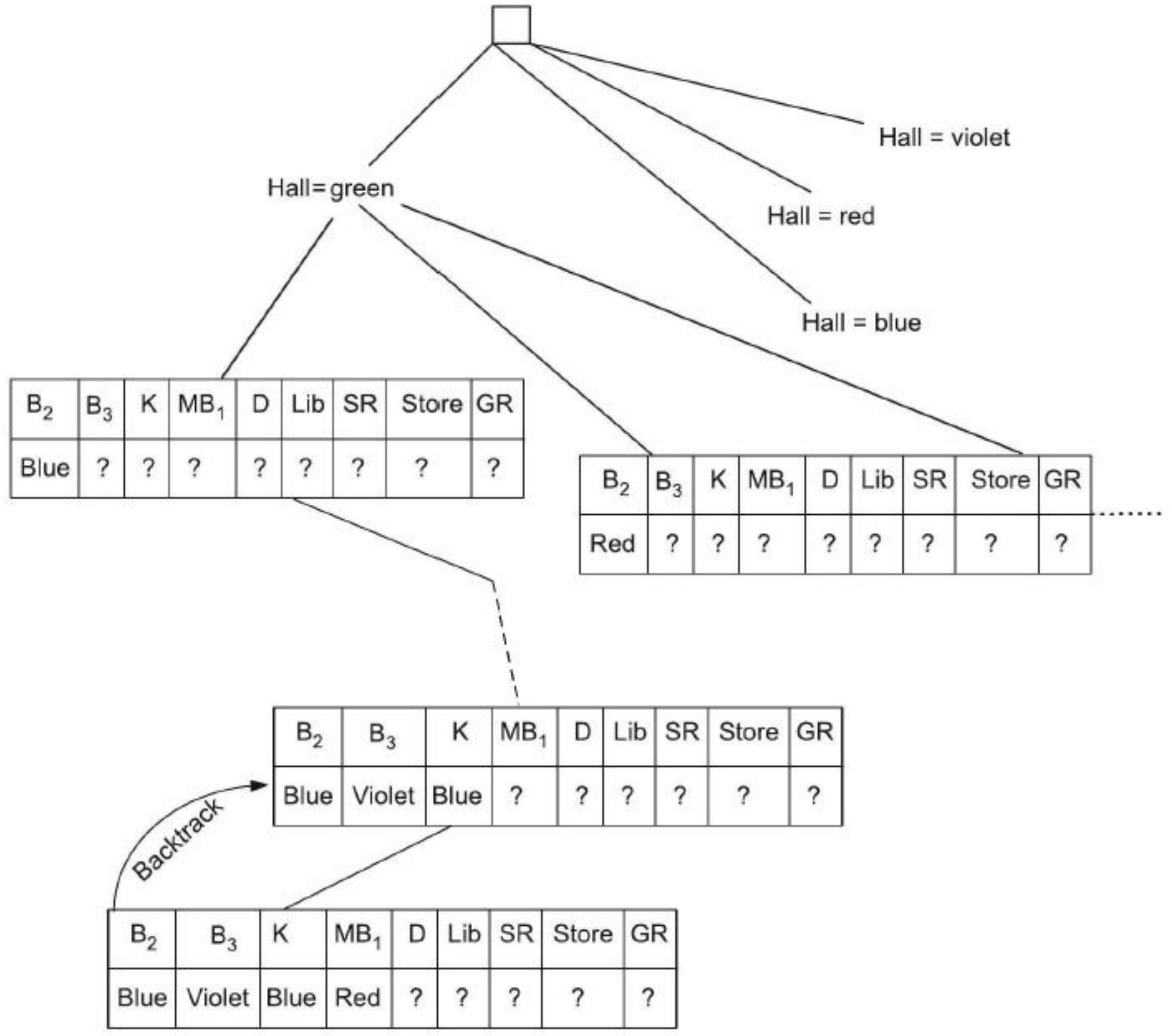


B3	B2	K	MB1	D	Lib	SR	Store	GR
Blue	?	?	?	?	?	?	?	?

B3	B2	K	MB1	D	Lib	SR	Store	GR
Red	?	?	?	?	?	?	?	?

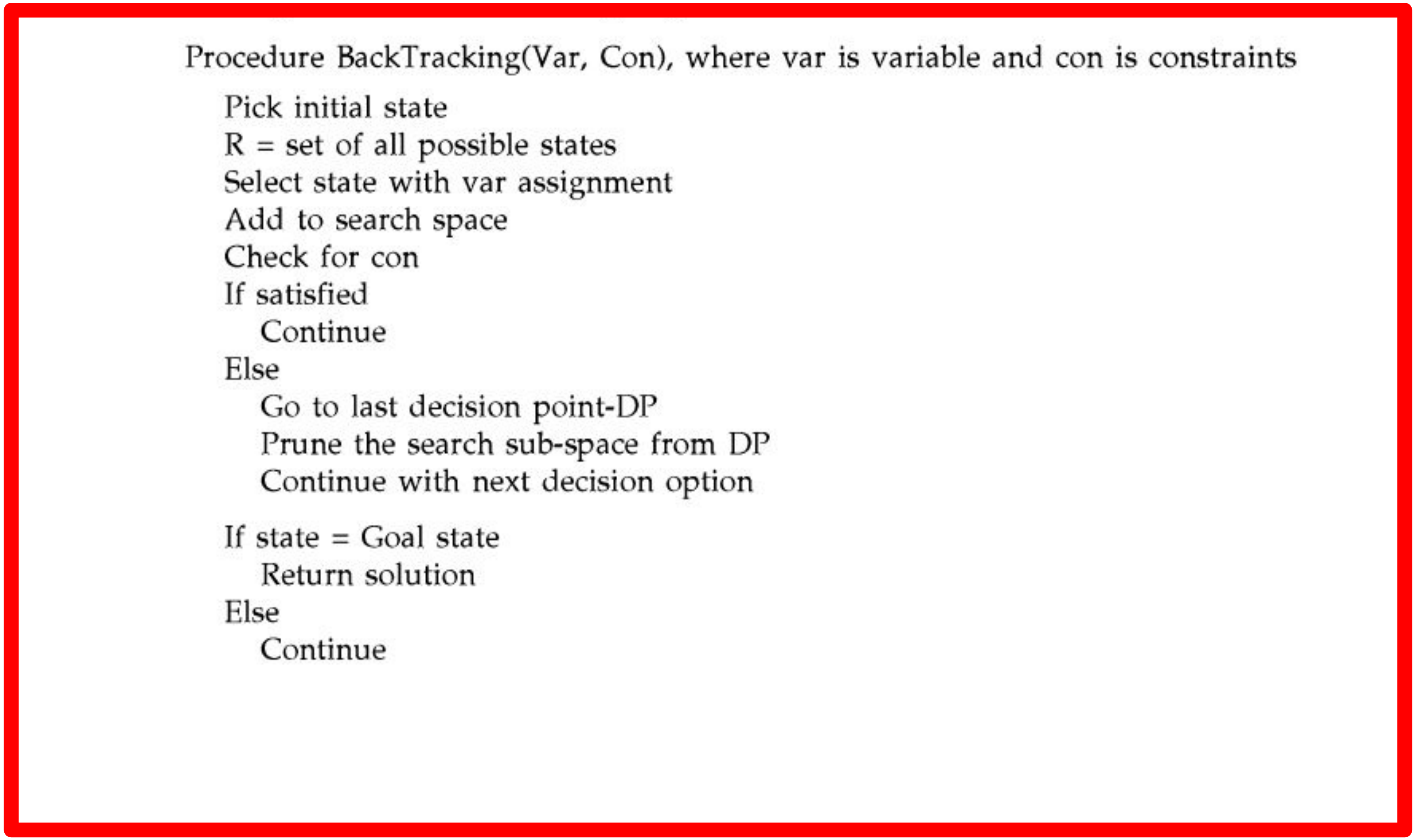
B3	B2	K	MB1	D	Lib	SR	Store	GR
Green	?	?	?	?	?	?	?	?

Backtracking Search for CSP



Hall = violet
Hall = red
Hall = blue
Hall=green
B2 B3 K MB1 D Lib SR Store GR
Blue ? ? ? ? ? ? ? ?
B2 B3 K MB1 D Lib SR Store GR
Red ? ? ? ? ? ? ? ?
B2 B3 K MB1 D Lib SR Store GR
Blue Violet Blue ? ? ? ? ? ?
Backtrack
B2 B3 K MB1 D Lib SR Store GR
Blue Violet Blue Red ? ? ? ? ?

Procedure Backtracking Algorithm



```
Procedure BackTracking(Var, Con), where var is variable and con is constraints
    Pick initial state
    R = set of all possible states
    Select state with var assignment
    Add to search space
    Check for con
    If satisfied
        Continue
    Else
        Go to last decision point-DP
        Prune the search sub-space from DP
        Continue with next decision option

    If state = Goal state
        Return solution
    Else
        Continue
```

Role of Heuristic

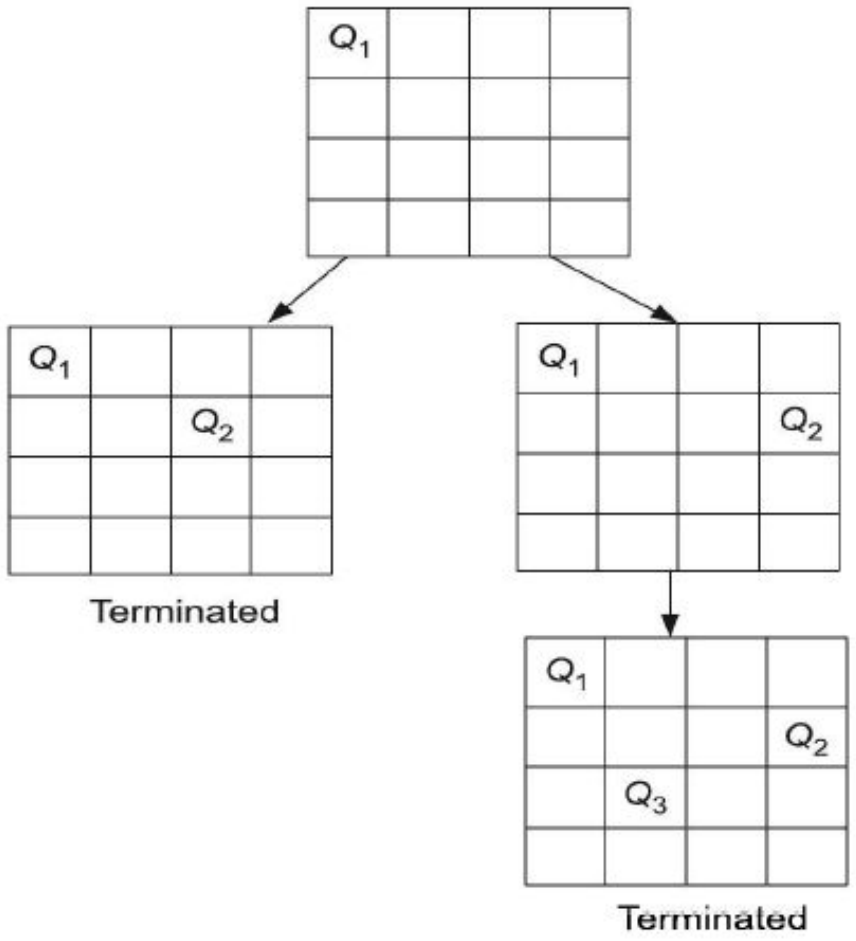
- Deciding the initial state as well as guiding the selection of subsequent states.

- Selection should be a minimum number of possible values that leads to a simplified search

  * Referred as Minimum Remaining Values heuristic(MRV)/ Most Constraint Variable.

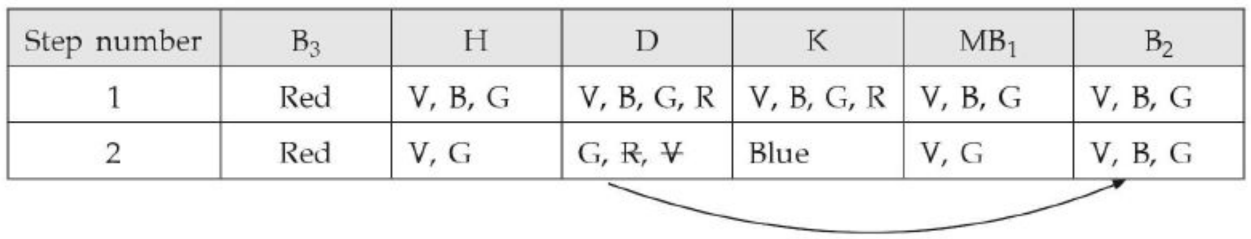
*Detect failure at an early stage.

Forward Checking



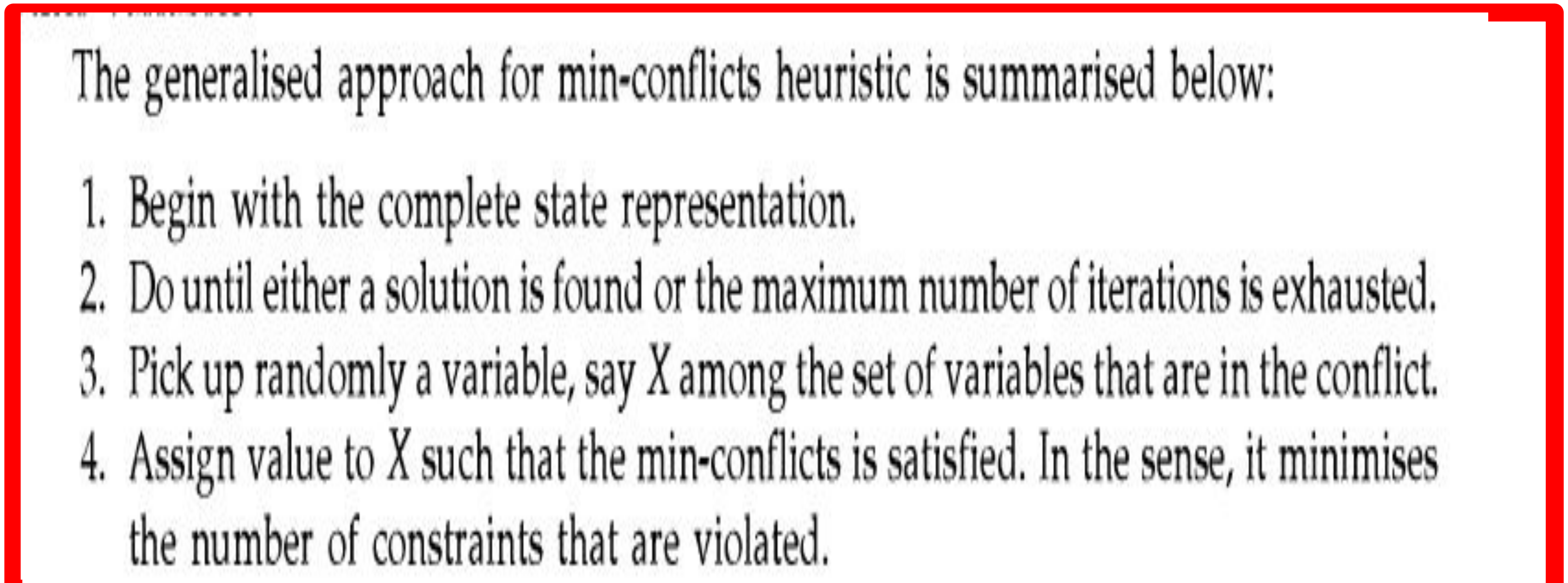
Constraint Propagation

Using the constraints to reduce the number of legal values for a variable, which in turn can reduce the legal values for another variable, and so on.



Step number	B_3	H	D	K	MB_1	B_2
1	Red	V, B, G	V, B, G, R	V, B, G, R	V, B, G	V, B, G
2	Red	V, G	G, ~~R~~, ~~V~~	Blue	V, G	V, B, G

Local Search for Constraint Satisfaction Problem



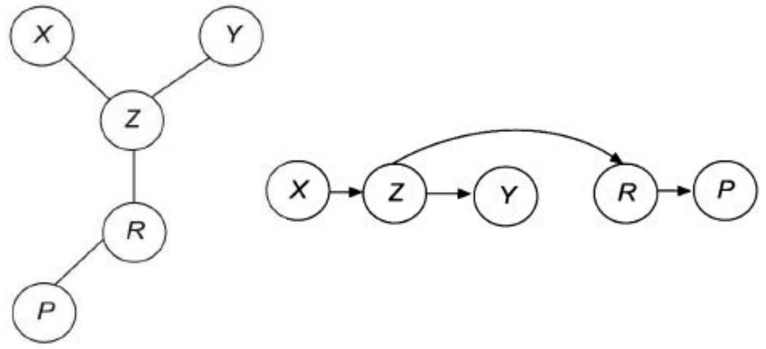
The generalised approach for min-conflicts heuristic is summarised below:

1. Begin with the complete state representation.
2. Do until either a solution is found or the maximum number of iterations is exhausted.
3. Pick up randomly a variable, say X among the set of variables that are in the conflict.
4. Assign value to X such that the min-conflicts is satisfied. In the sense, it minimises the number of constraints that are violated.

Formulating Problem Structure

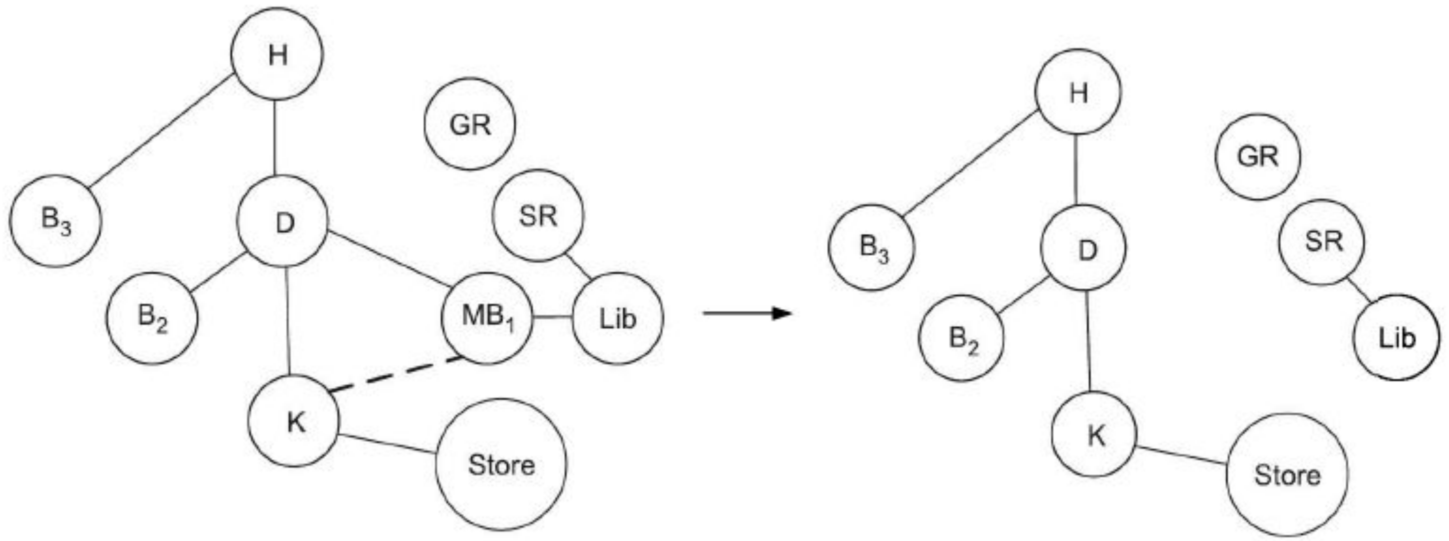
1. Select a variable or unassigned node as root.
2. Once selection is done, perform ordering such that the parent node precedes in the ordering.
3. For i = number of nodes(n) down to 2, perform arc consistency to remove the inconsistency.
4. For $i = 1$ to number of nodes, assign value to variable(node) X that is consistent with $Y \mid Y$ is parent of the node X.

So, here, we are actually converting or reducing the graph to tree structure. So, is it possible to convert the generalised graphs to trees? Yes. It is possible by two means—by removal of variables/nodes and by collapsing variables/nodes.



$$\begin{tikzcd} X \arrow[r] & Z \arrow[r] \arrow[rr, bend left] & Y & R \arrow[r] & P \end{tikzcd}$$

Tree Structured CSP example



A generalised algorithm for the same is given below:

1. Remove a subset from the set of variables such that after removal of this subset, the structure turns out to be a tree. Now, let us say that the subset has some set of variables, say V', whereas the other variables are V''.
2. To assign values to V' | all the constraints in subset are met or satisfied
   (a) Remove from V'' the other values that happen to be inconsistent with the ones assigned in V'.
   (b) There is a possibility that we get an answer in the V'' and return the solution with the values assigned in V'.