

Adversarial Search and Game Playing

(Where making good decisions
requires respecting your opponent)

R&N: Chap. 5

- Games like Chess or Go are compact settings that mimic the uncertainty of interacting with the natural world
- For centuries humans have used them to exert their intelligence
- Recently, there has been great success in building game programs that challenge human supremacy

Games

- In this chapter we cover **competitive environments**
 - The agents' goals are in conflict
 - Giving rise to adversarial search problems
- Mathematical **game theory**, a branch of economics, views any multiagent environment as a game, provided that the impact of each agent on the others is "significant," regardless of whether the agents are **cooperative** or **competitive**.

Games

- In AI, the most common games are of a rather specialized kind what **game theorists** call **deterministic, turn-taking, two-player, zero-sum** games of **perfect information** (such as chess).
- In **our terminology**, this means **deterministic, fully observable** environments in which two agents act alternately and in which the utility values at the end of the game are always equal and opposite.
 - If one player wins a game of chess, the other player necessarily loses.

Specific Setting

Two-player, turn-taking, deterministic, fully observable, zero-sum, time-constrained game

- **State space**
- **Initial state**
- **Successor function ($\text{RESULT}(s, a)$):** it tells which actions can be executed in each state and gives the successor state for each action
- **MAX's and MIN's actions alternate,** with MAX playing first in the initial state
- **Terminal test:** it tells if a state is terminal and, if yes, if it's a win or a loss for MAX, or a draw
- **All states are fully observable**

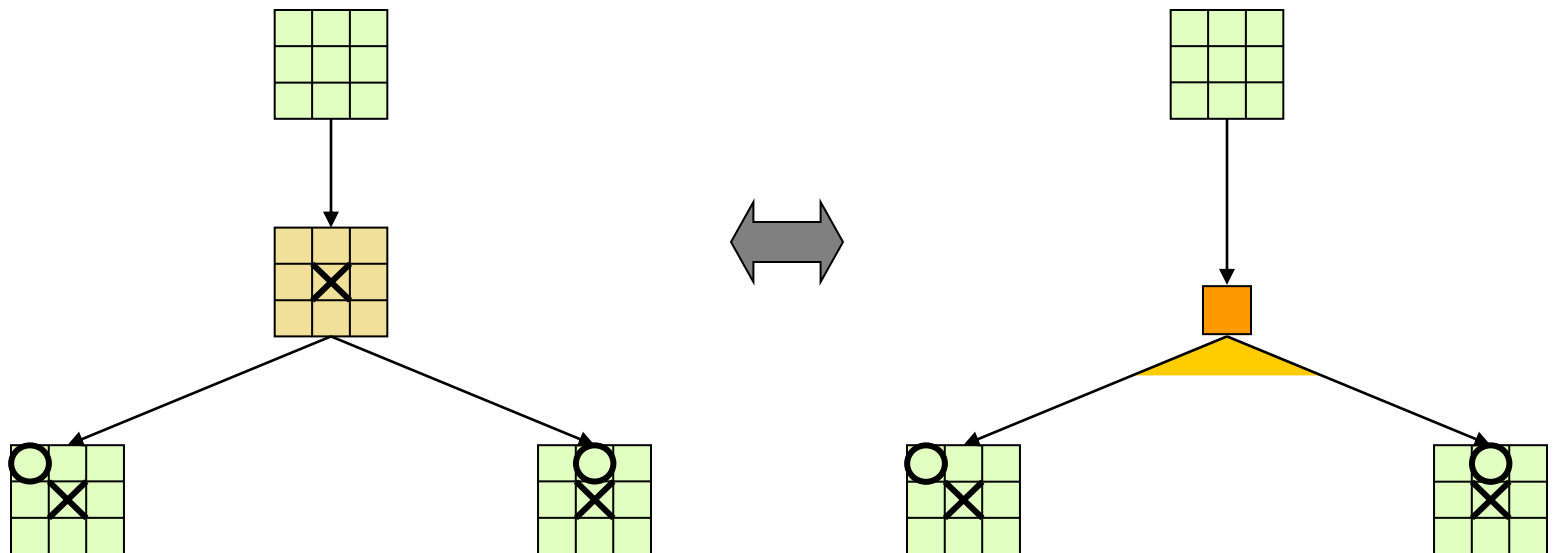
Specific Setting

Two-player, turn-taking, deterministic, fully observable, zero-sum, time-constrained game

- **PLAYER(s)**
 - Defines which player has the move in a state
- **ACTIONS(s)**
 - Returns the set of legal moves in a state.
- **UTILITY(s, p)**
 - A utility function, defines the final numeric value for a game that ends in terminal state **s** for a player **p**.
- A **zero-sum game** is (confusingly) defined as one where **the total payoff to all players** is the same for every instance of the game.
 - Chess is zero-sum because every game has payoff of either $0 + 1$, $1 + 0$ or $1/2 + 1/2$

Relation to Previous Lecture

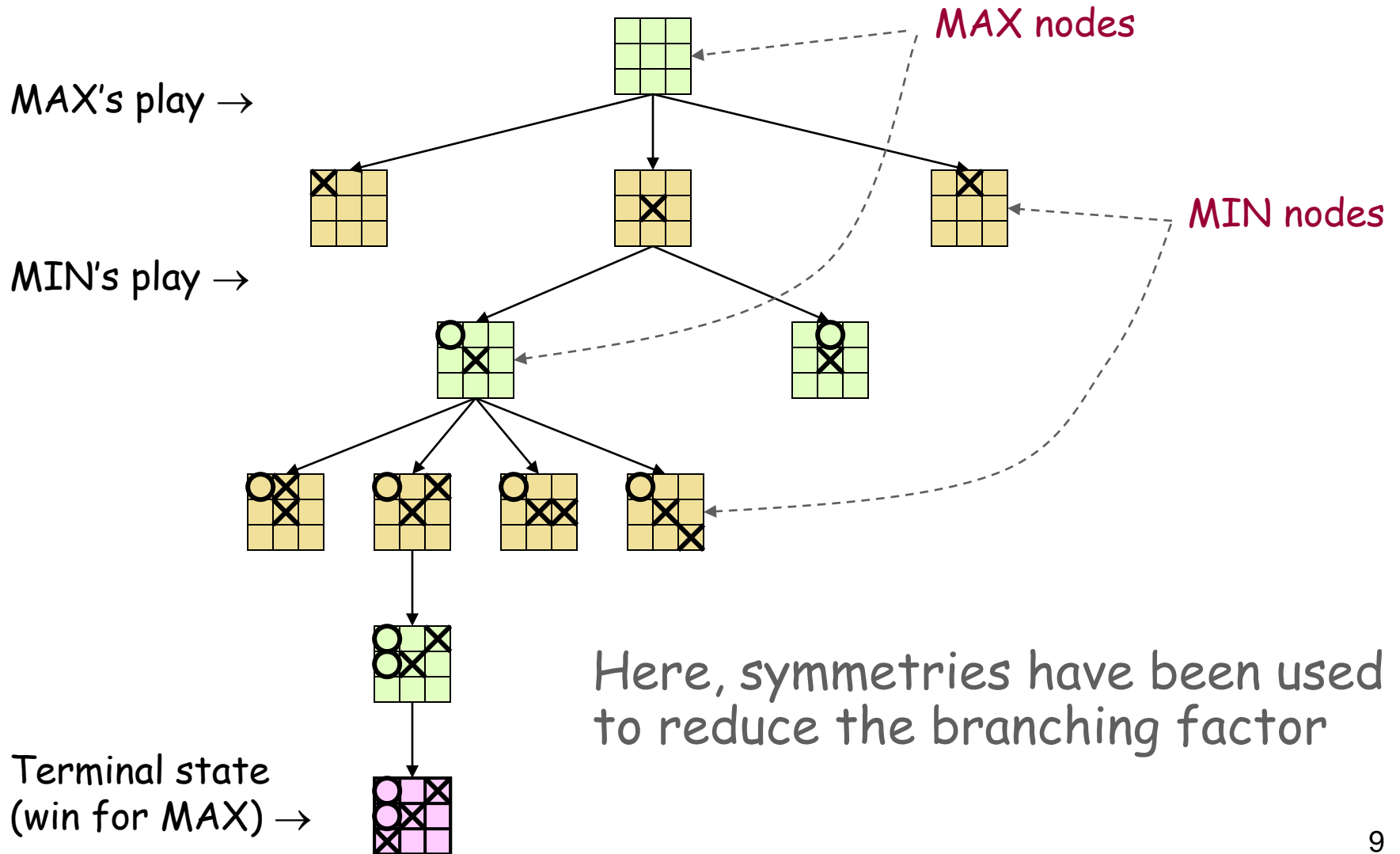
- Here, uncertainty is caused by the actions of another agent (MIN), who competes with our agent (MAX)



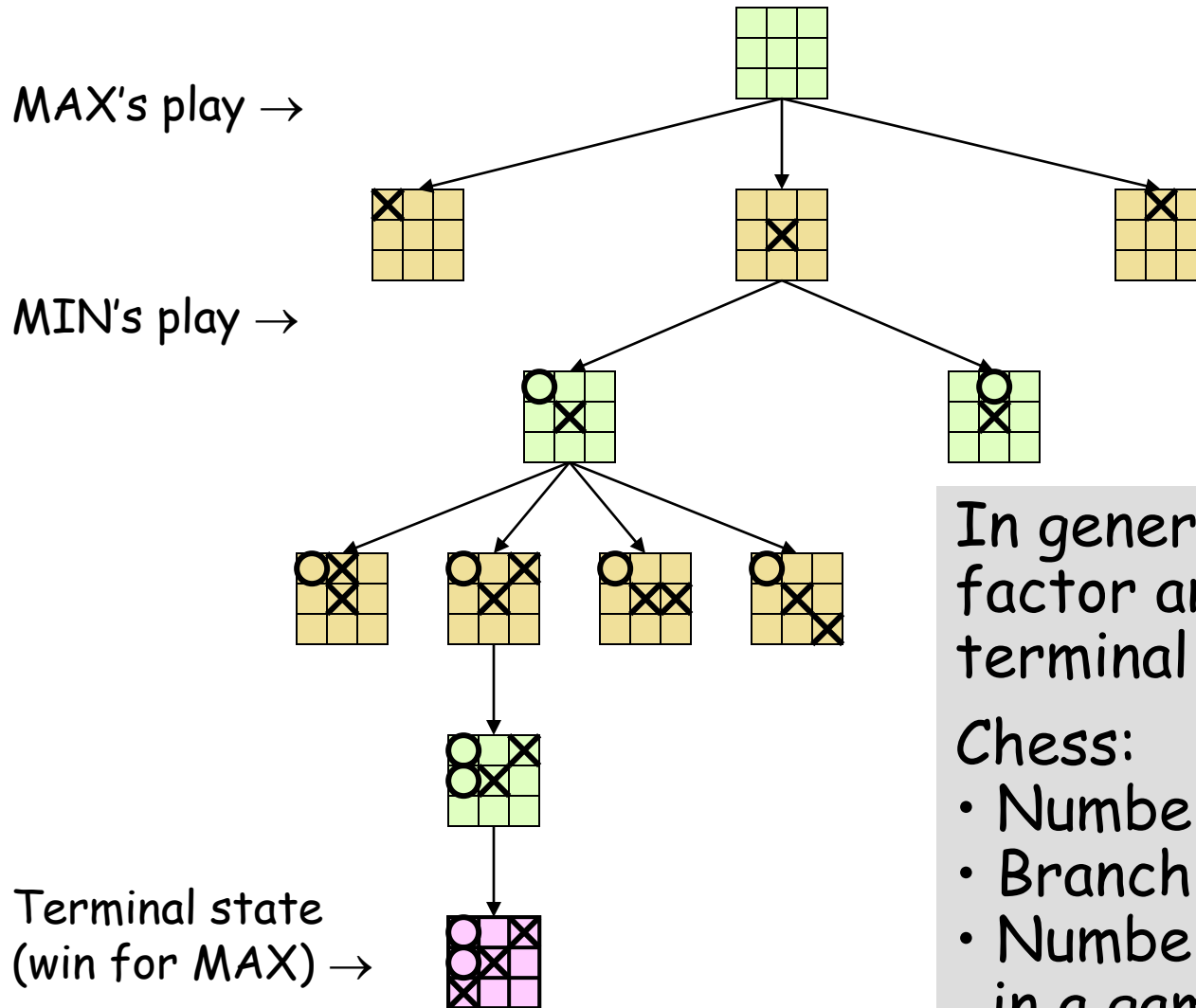
Relation to Previous Lecture

- Here, uncertainty is caused by the actions of another agent (MIN), who competes with our agent (MAX)
- MIN wants MAX to lose (and vice versa)
- No plan exists that guarantees MAX's success regardless of which actions MIN executes (the same is true for MIN)
- At each turn, the choice of which action to perform must be made within a specified **time limit**
- The state space is enormous: only a tiny fraction of this space can be explored within the time limit

Game Tree



Game Tree



In general, the branching factor and the depth of terminal states are large

Chess:

- Number of states: $\sim 10^{40}$
- Branching factor: ~ 35
- Number of total moves in a game: ~ 100

Choosing an Action: Basic Idea

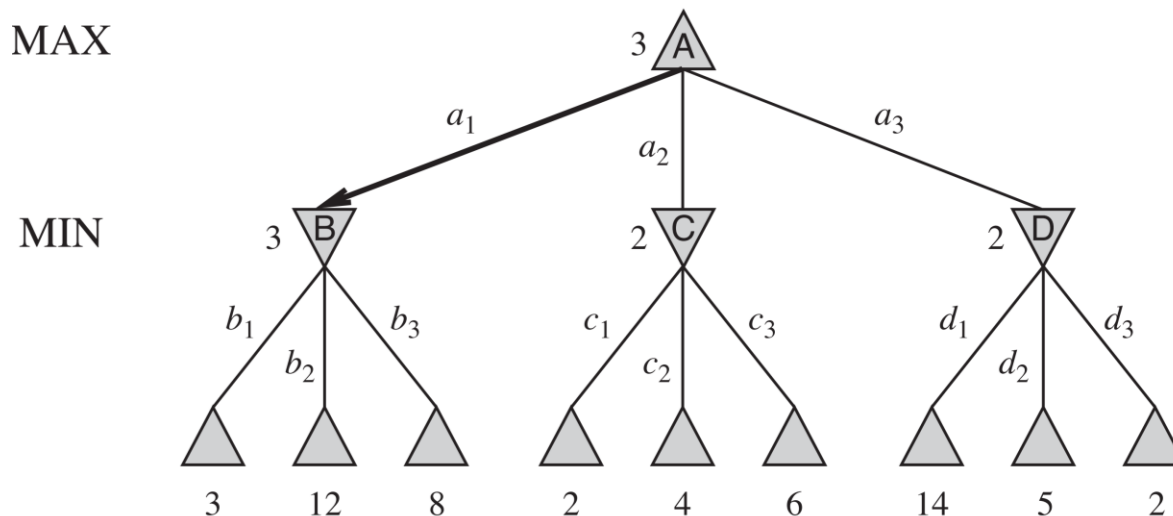
- 1) Using the current state as the initial state, build the game tree uniformly to the maximal depth **h** (called **horizon**) feasible within the time limit
 - 2) **Evaluate** the states of the leaf nodes
 - 3) **Back up** the results from the leaves to the root and pick the best action assuming the worst from **MIN**
- **Minimax algorithm**

Minimax Algorithm

MINIMAX(s) =

$$\begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \end{cases}$$

- MINIMAX(s) shows the best achievable outcome of being in state (assumption: optimal opponent)
 - Maximizes the worst-case outcome for MAX



Minimax Algorithm

```
function MINIMAX-DECISION(state) returns an action  
  return  $\operatorname{argmax}_{a \in \text{ACTIONS}(s)} \text{MIN-VALUE}(\text{RESULT}(\text{state}, a))$ 
```

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

```
function MIN-VALUE(state) returns a utility value  
  if TERMINAL-TEST(state) then return UTILITY(state)  
   $v \leftarrow +\infty$   
  for each  $a$  in ACTIONS(state) do  
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(s, a)))$   
  return  $v$ 
```

Computational time limit (example)

- 100 secs is allowed for each move (game rule)
- 10^4 nodes/sec (processor speed)
- We can explore just 10^6 nodes for each move
 - $b^m = 10^6, b=35 \Rightarrow m=4$
- We must make a decision even when finding the optimal move is infeasible.

Computational time limit (Solution)

- Cut off the search and apply a heuristic evaluation function
 - **cutoff test**: turns non-terminal nodes into terminal leaves
 - Cut off test instead of terminal test (e.g., depth limit)
 - **evaluation function**: estimated desirability of a state
 - Heuristic function evaluation instead of utility function
- This approach does not guarantee optimality.

H-MINIMAX(s, d) =

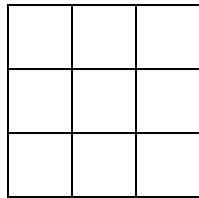
$$\begin{cases} \text{EVAL}(s) & \text{if CUTOFF-TEST}(s, d) \\ \max_{a \in \text{Actions}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{Actions}(s)} \text{H-MINIMAX}(\text{RESULT}(s, a), d + 1) & \text{if PLAYER}(s) = \text{MIN}. \end{cases}$$

Evaluation Function

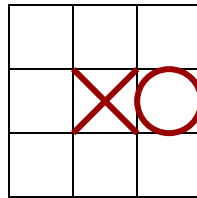
- Function e : state $s \rightarrow$ number $e(s)$
- $e(s)$ is a **heuristics** that estimates how favorable s is for MAX
- $e(s) > 0$ means that s is favorable to MAX (the larger the better)
- $e(s) < 0$ means that s is favorable to MIN
- $e(s) = 0$ means that s is neutral

Example: Tic-tac-Toe

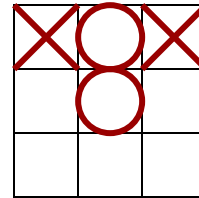
$e(s)$ = number of rows, columns,
and diagonals open for MAX
– number of rows, columns,
and diagonals open for MIN



$$8 - 8 = 0$$



$$6 - 4 = 2$$



$$3 - 3 = 0$$

Construction of an Evaluation Function

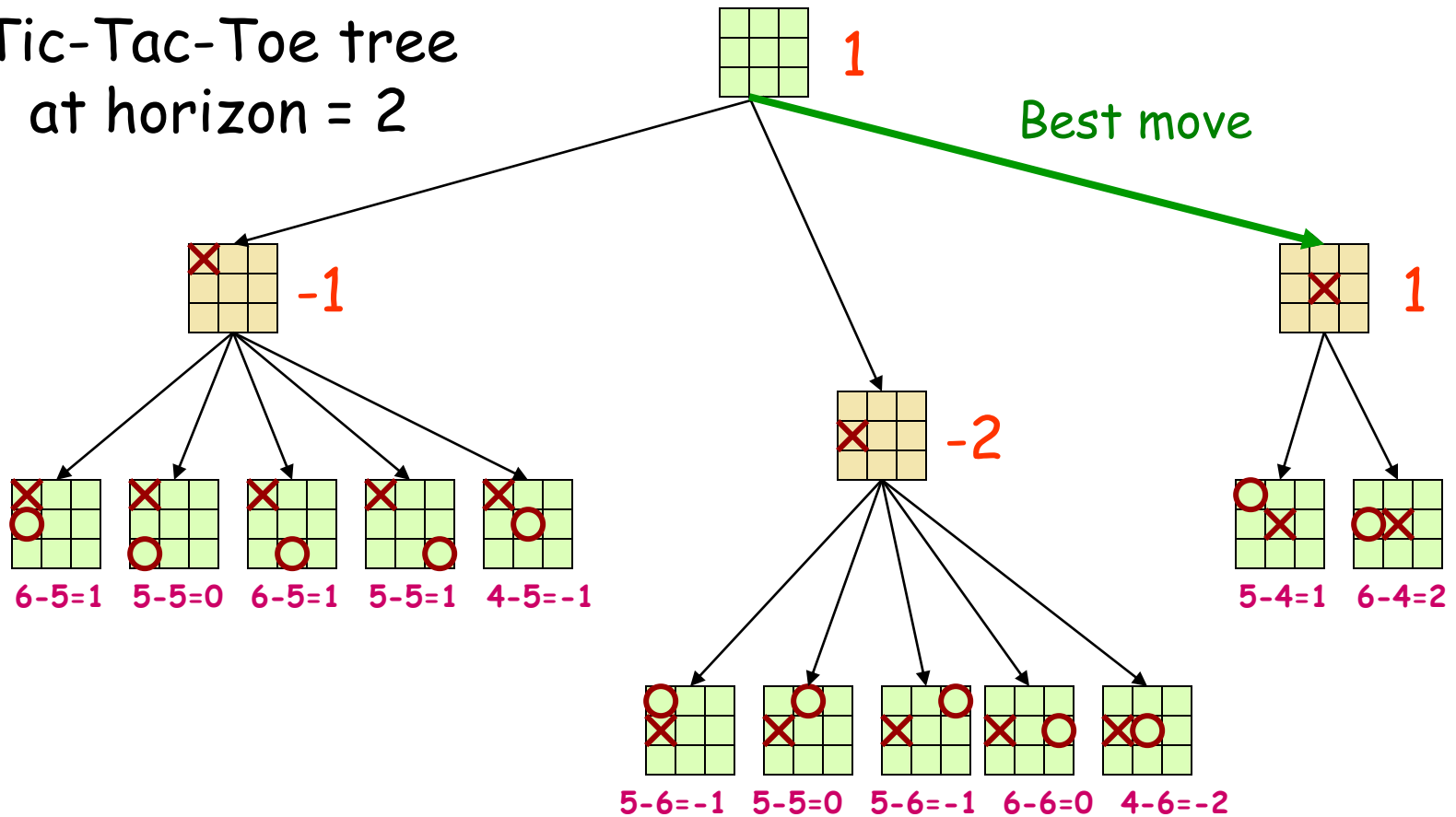
- Usually a weighted sum of “features”:

$$e(s) = \sum_{i=1}^n w_i f_i(s)$$

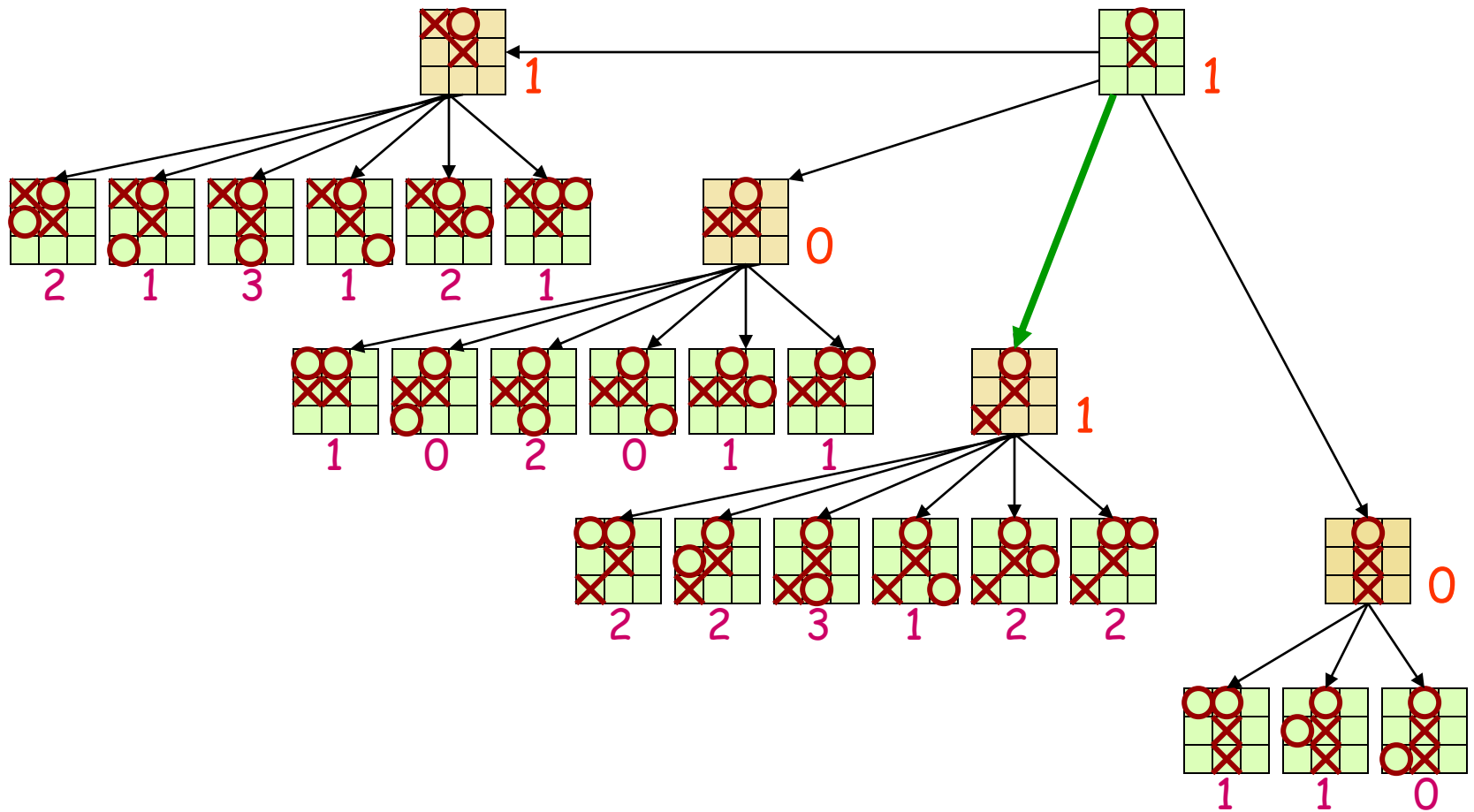
- Features may include
 - Number of pieces of each type
 - Number of possible moves
 - Number of squares controlled
- Weights can be assigned based on the human experience or machine learning methods.

Backing up Values

Tic-Tac-Toe tree
at horizon = 2



Continuation



Why using backed-up values?

- At each non-leaf node N , the backed-up value is the value of the best state that MAX can reach at depth h if MIN plays well (by the same criterion as MAX applies to itself)
- If e is to be trusted in the first place, then the backed-up value is a better estimate of how favorable $STATE(N)$ is than $e(STATE(N))$

Minimax Algorithm

1. **Expand the game tree** uniformly from the current state (where it is MAX's turn to play) to depth **h**
2. **Compute the evaluation function** at every leaf of the tree
3. **Back-up the values** from the leaves to the root of the tree as follows:
 - a. A MAX node gets the maximum of the evaluation of its successors
 - b. A MIN node gets the minimum of the evaluation of its successors
4. **Select the move** toward a MIN node that has the largest backed-up value

Minimax Algorithm

1. Expand the game tree uniformly from the current state (where it is MAX's turn to play) to depth **h**
2. Compute the evaluation function at every leaf of the tree
3. Back-up the values from the leaves to the root of the tree as
 - a. A MAX node gets the maximum of the evaluation of its successors
 - b. A MIN node gets the minimum of the evaluation of its successors
4. Select the move toward a MIN node that has the largest backed-up value

Horizon: Needed to return a decision within allowed time

Repeated States

Left as an exercise

[Distinguish between states on the same path and states on different paths]

Game Playing (for MAX)

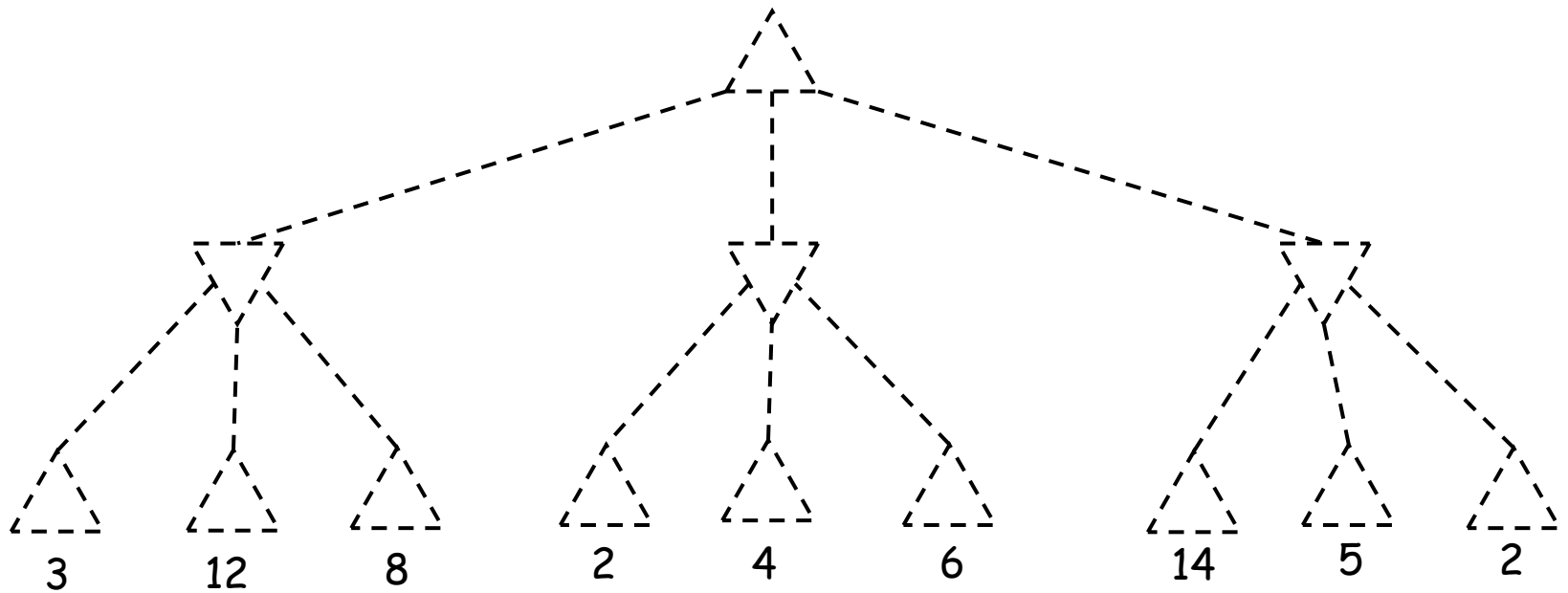
Repeat until a terminal state is reached

1. Select move using Minimax
2. Execute move
3. Observe MIN's move

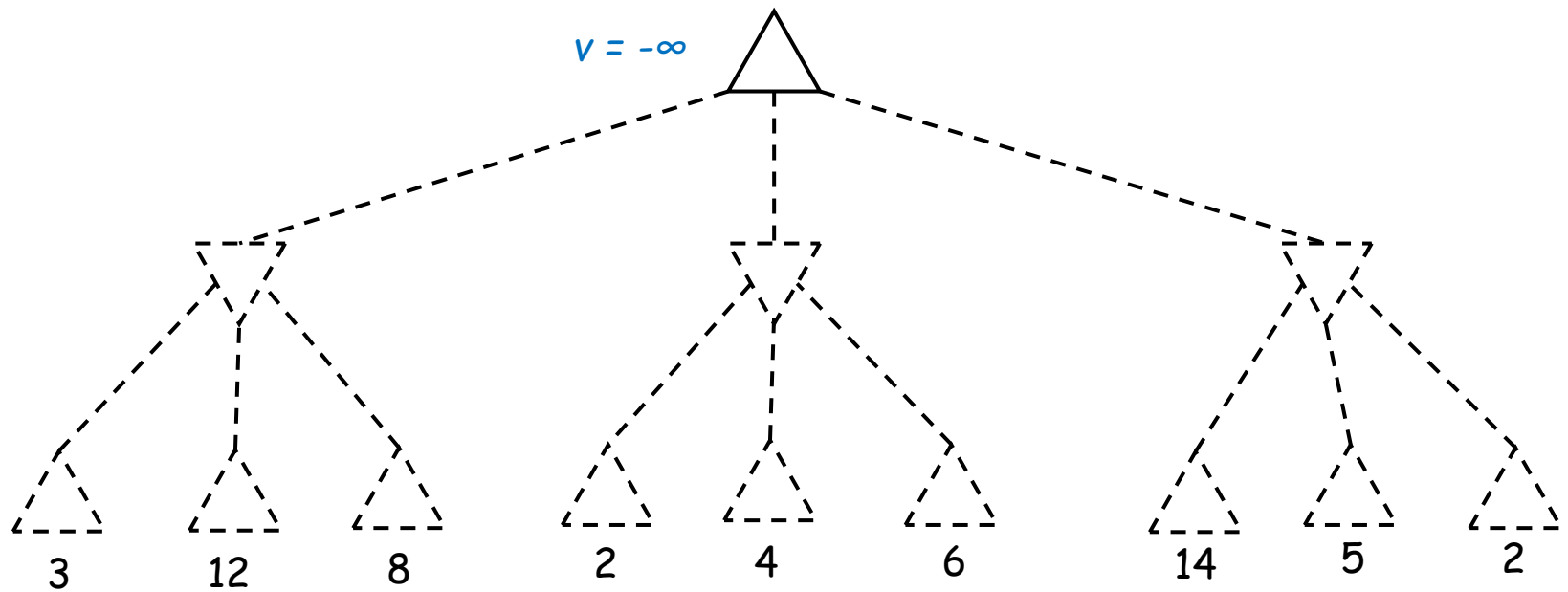
Note that at each cycle the large game tree built to horizon h is used to select only one move

All is repeated again at the next cycle (a sub-tree of depth $h-2$ can be re-used)

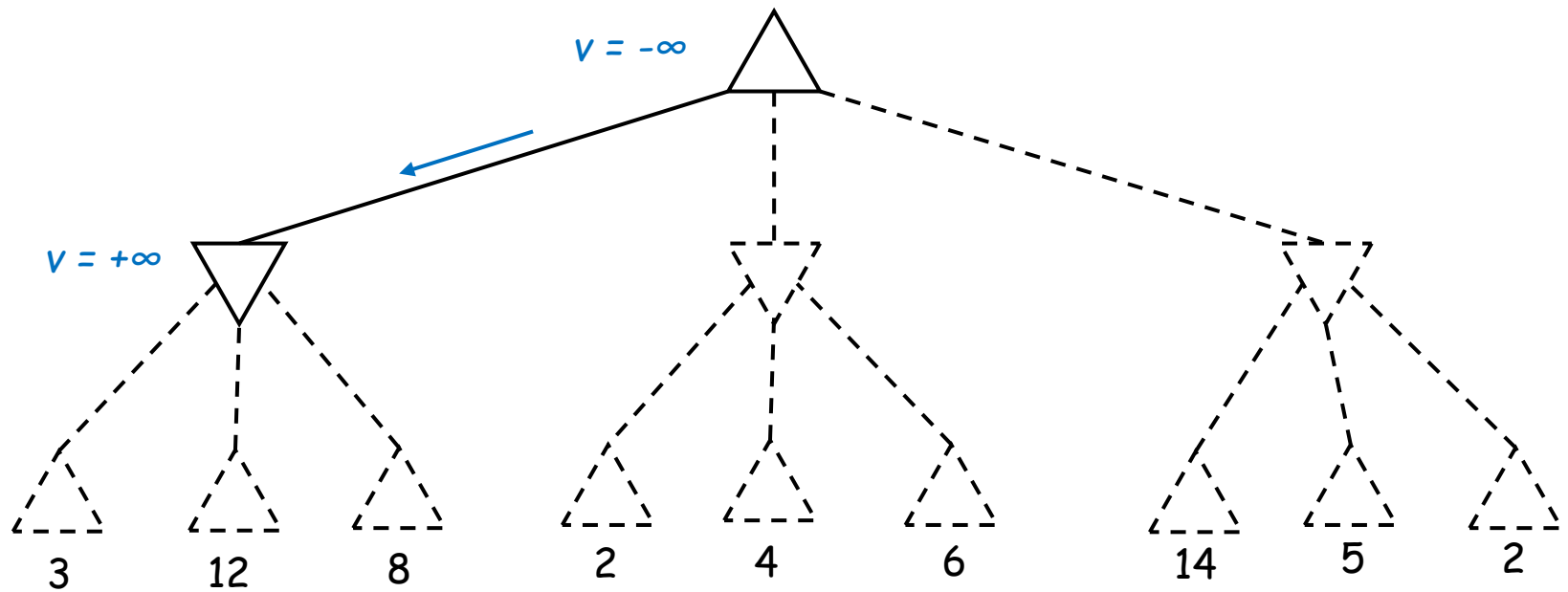
Example



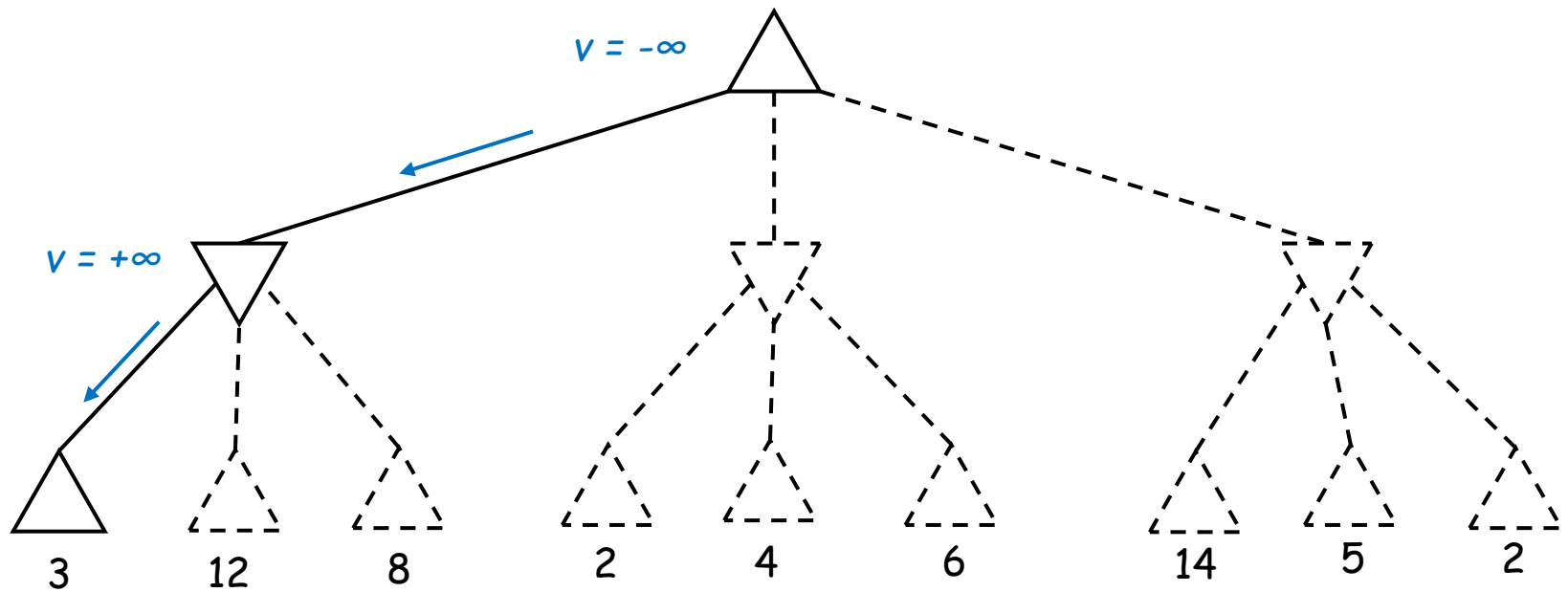
Example



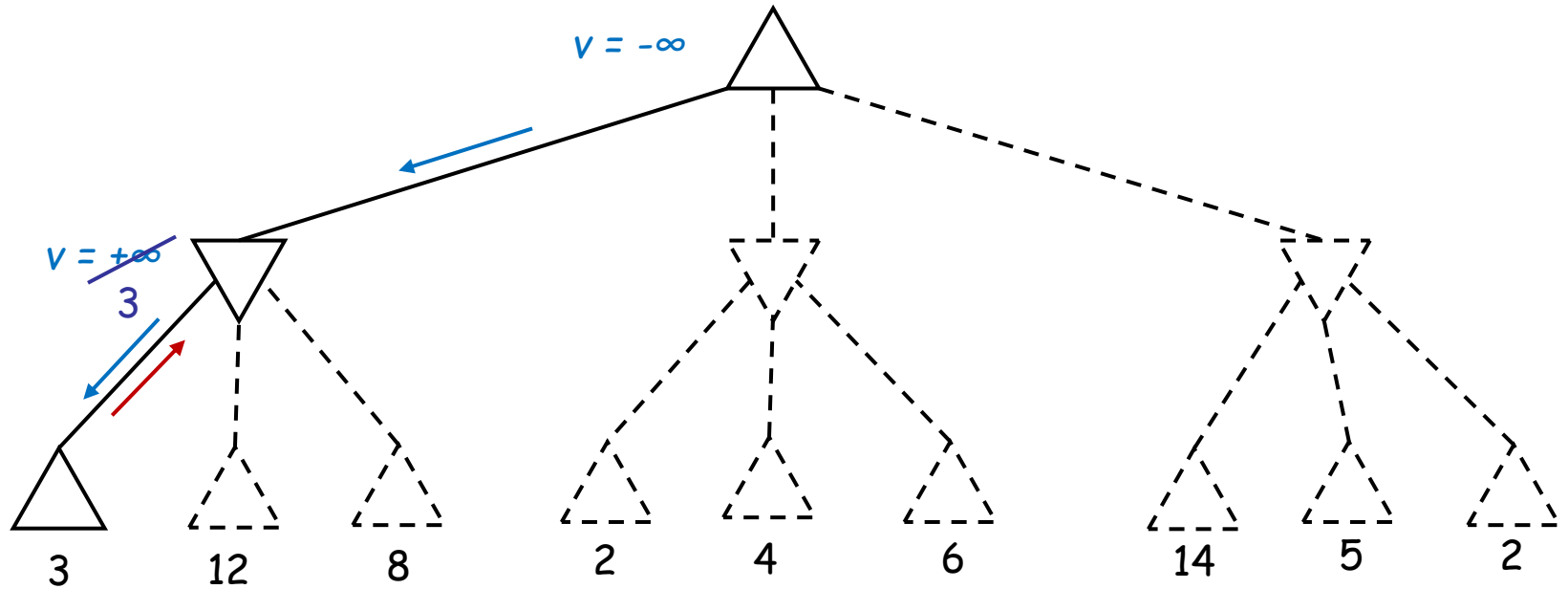
Example



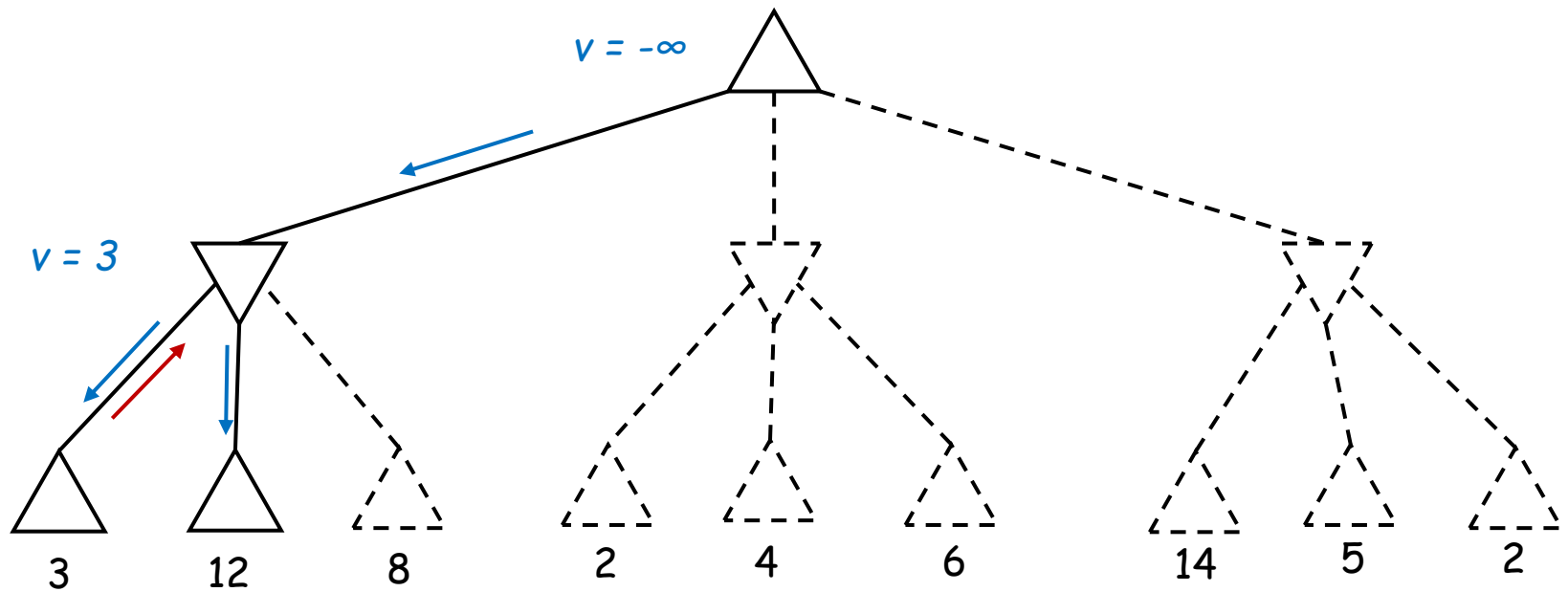
Example



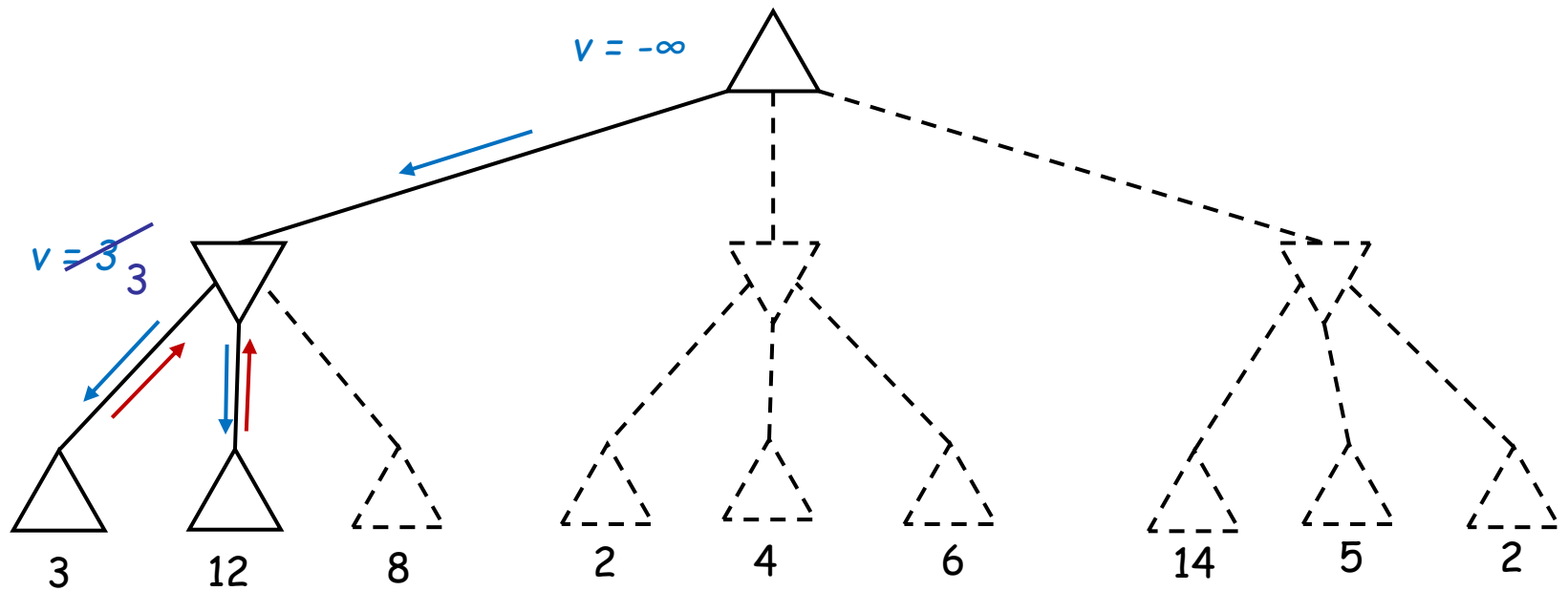
Example



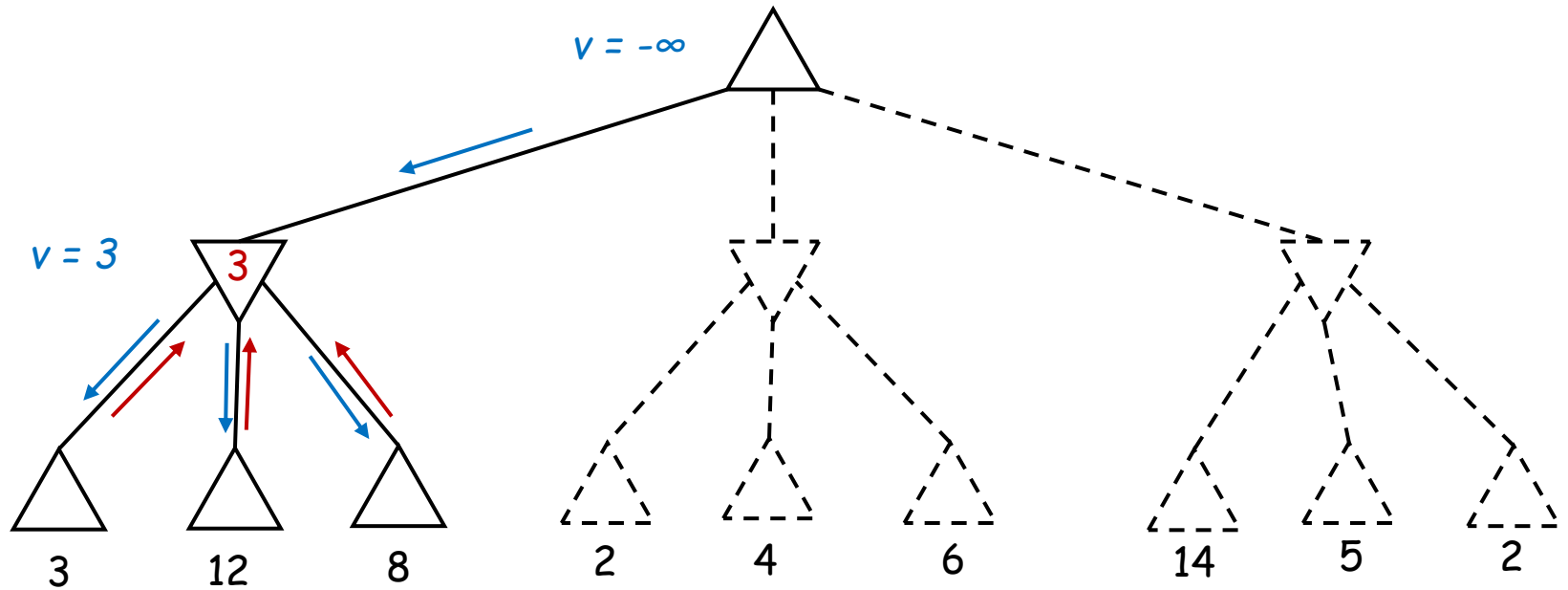
Example



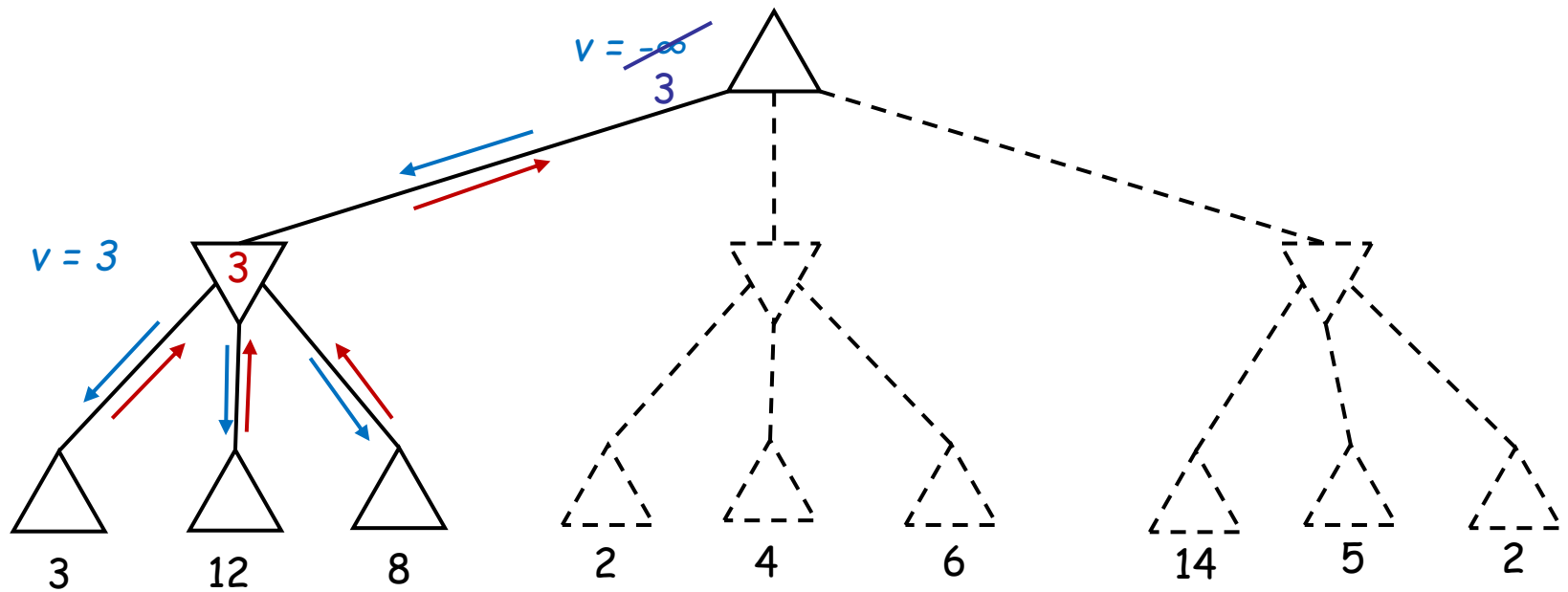
Example



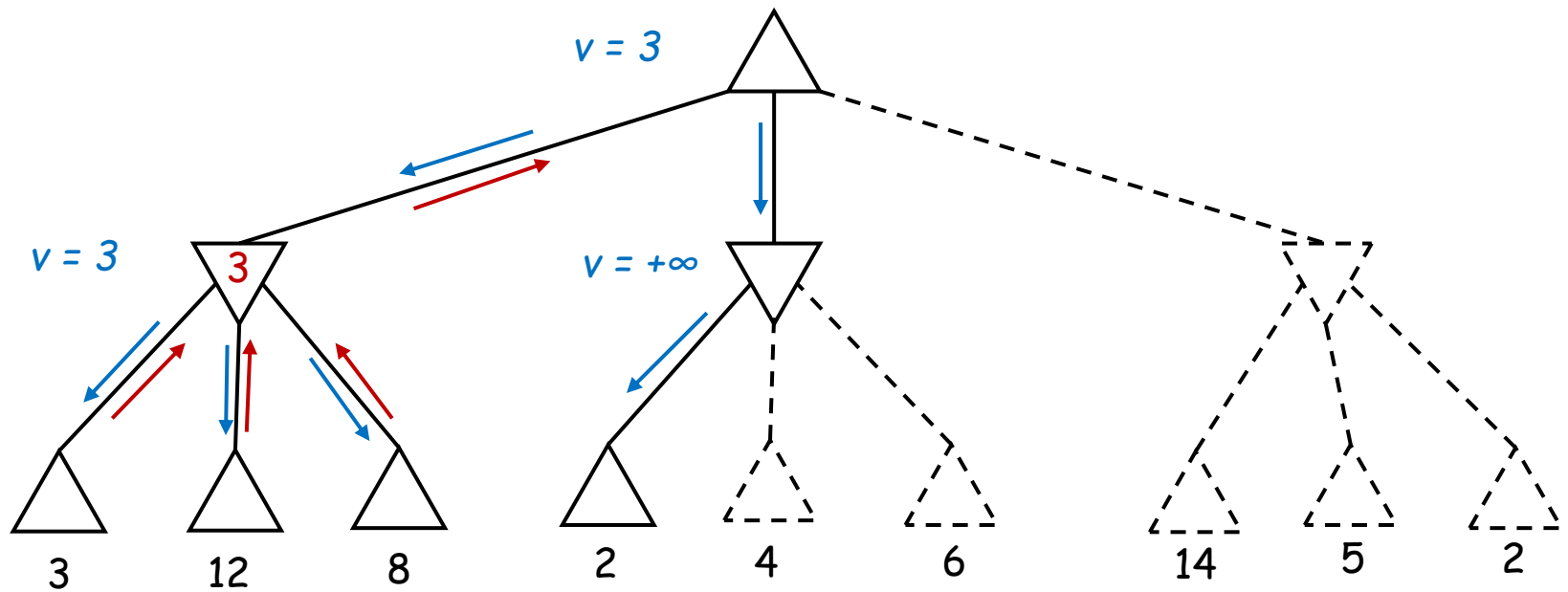
Example



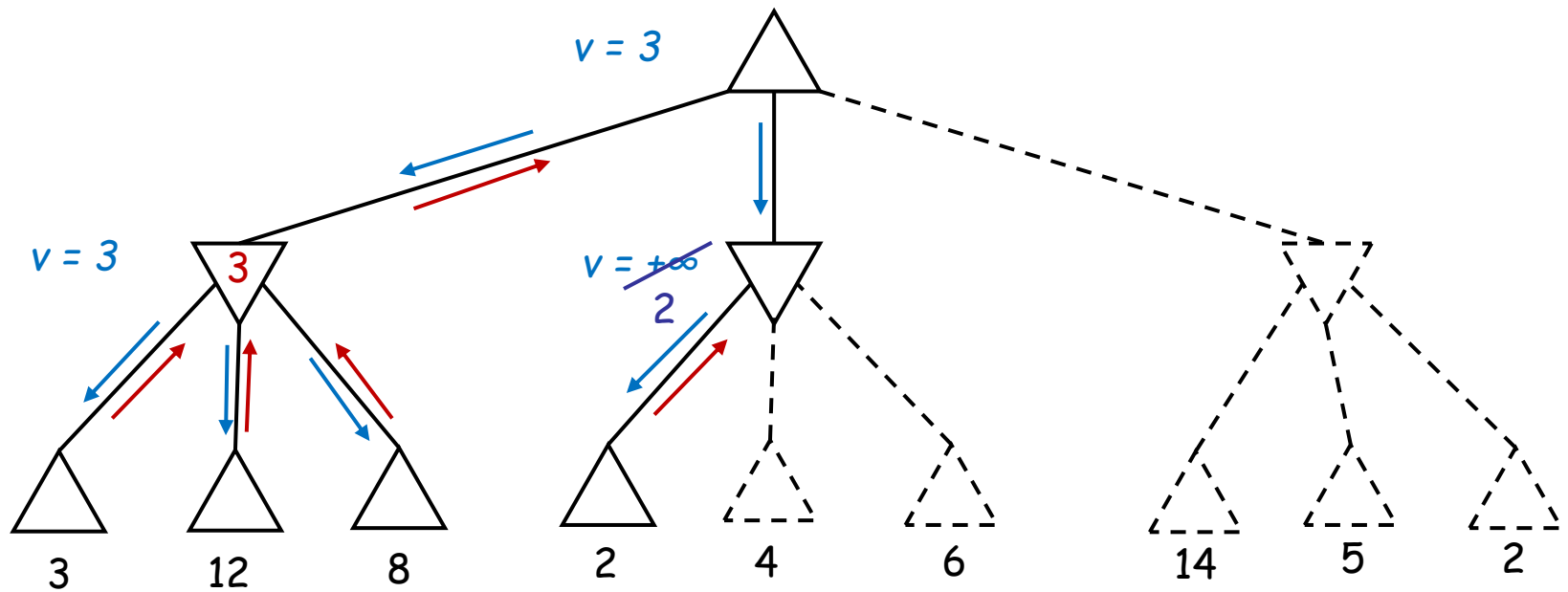
Example



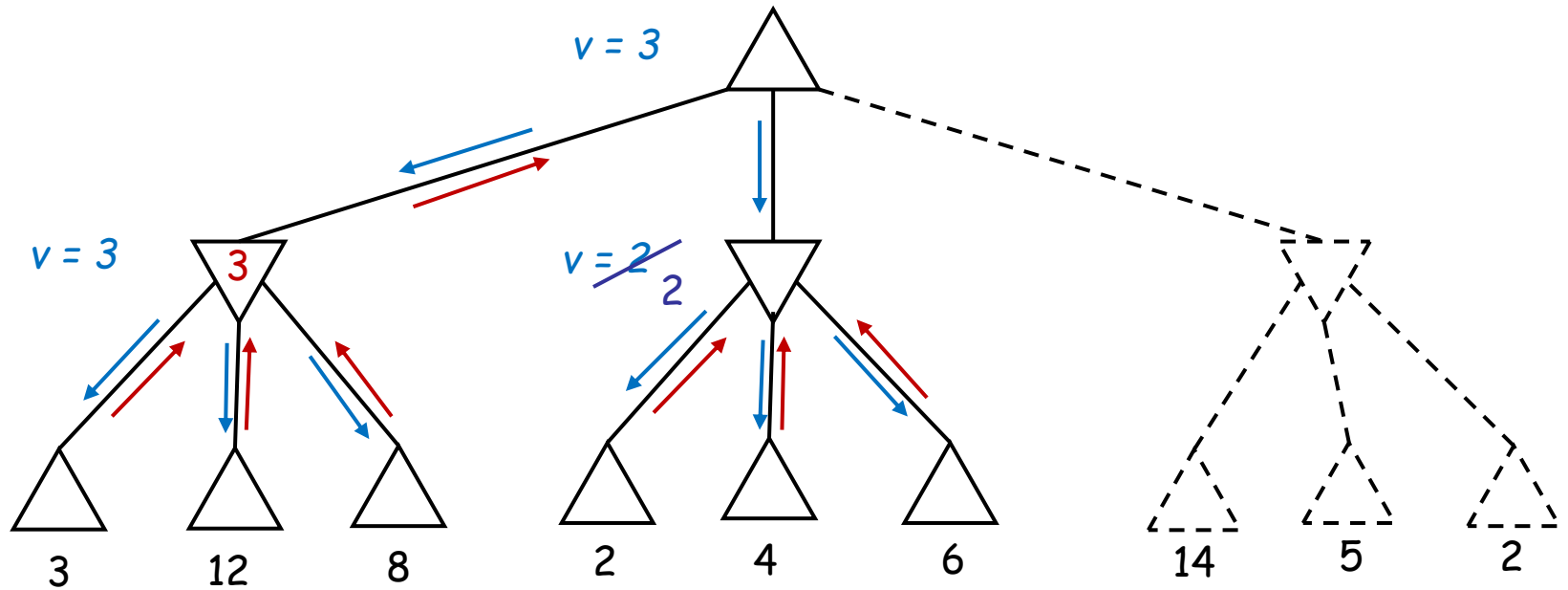
Example



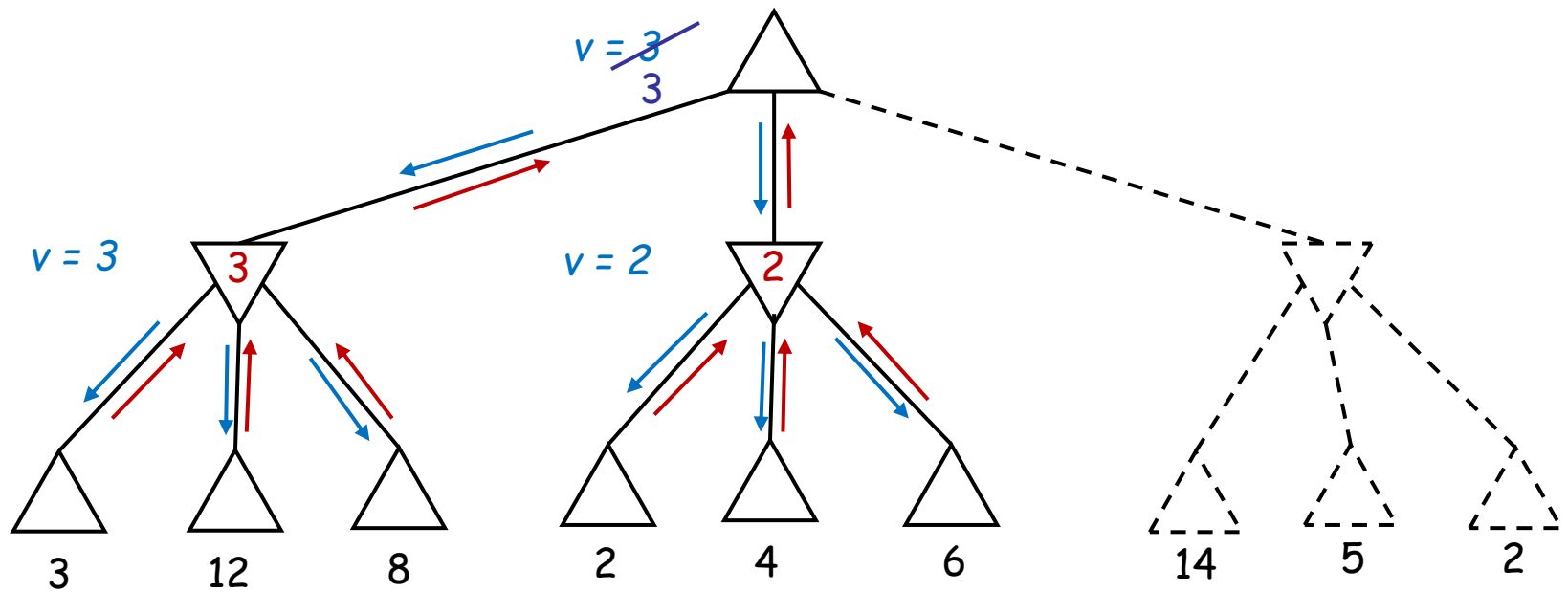
Example



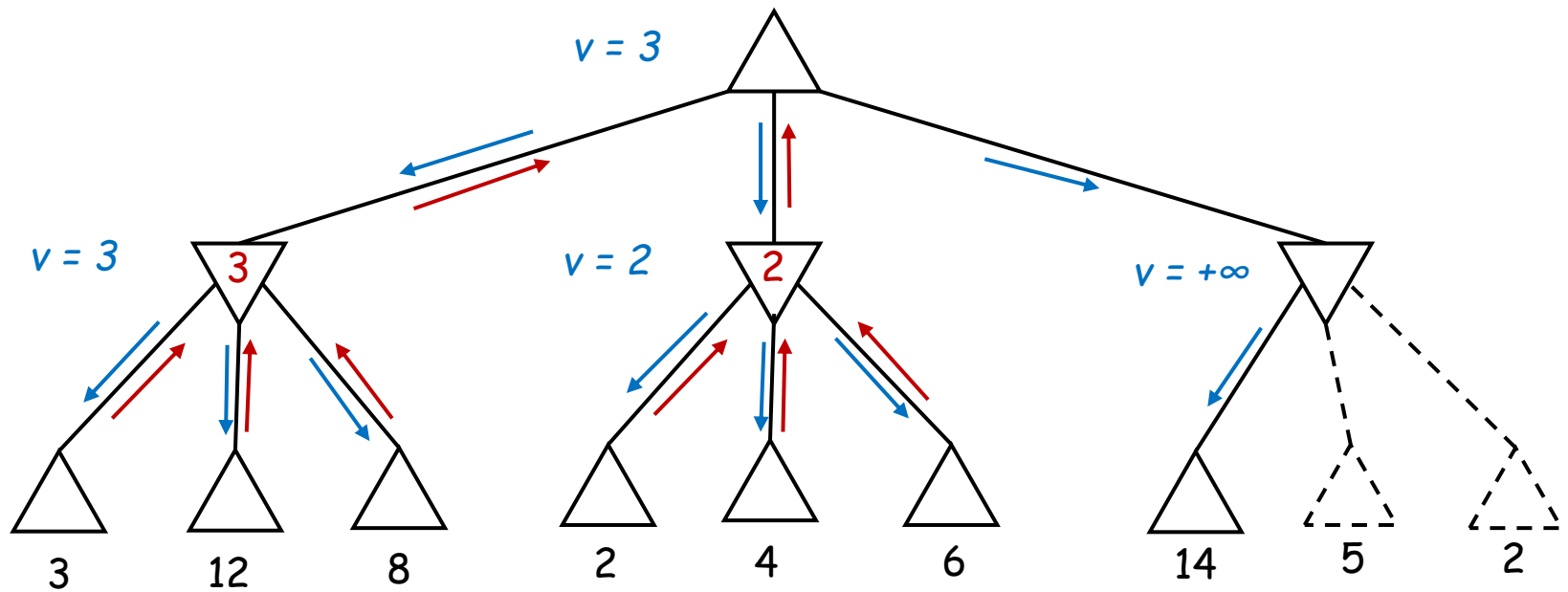
Example



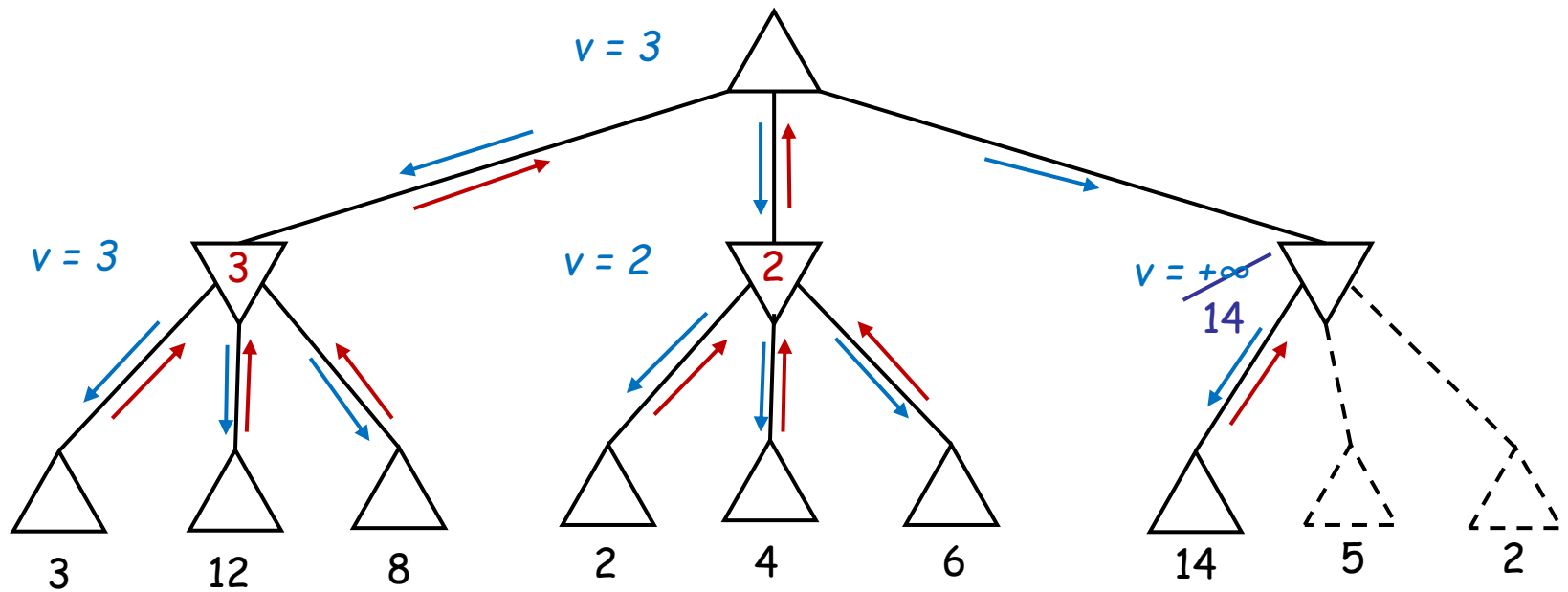
Example



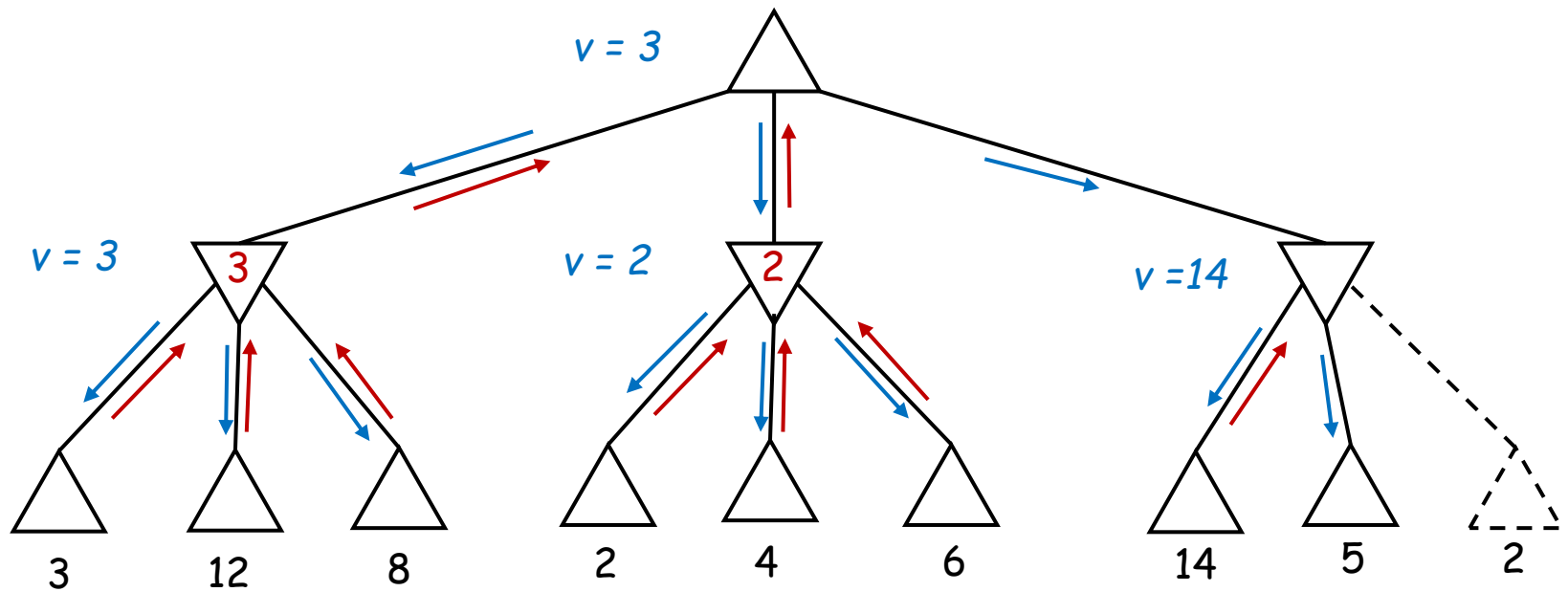
Example



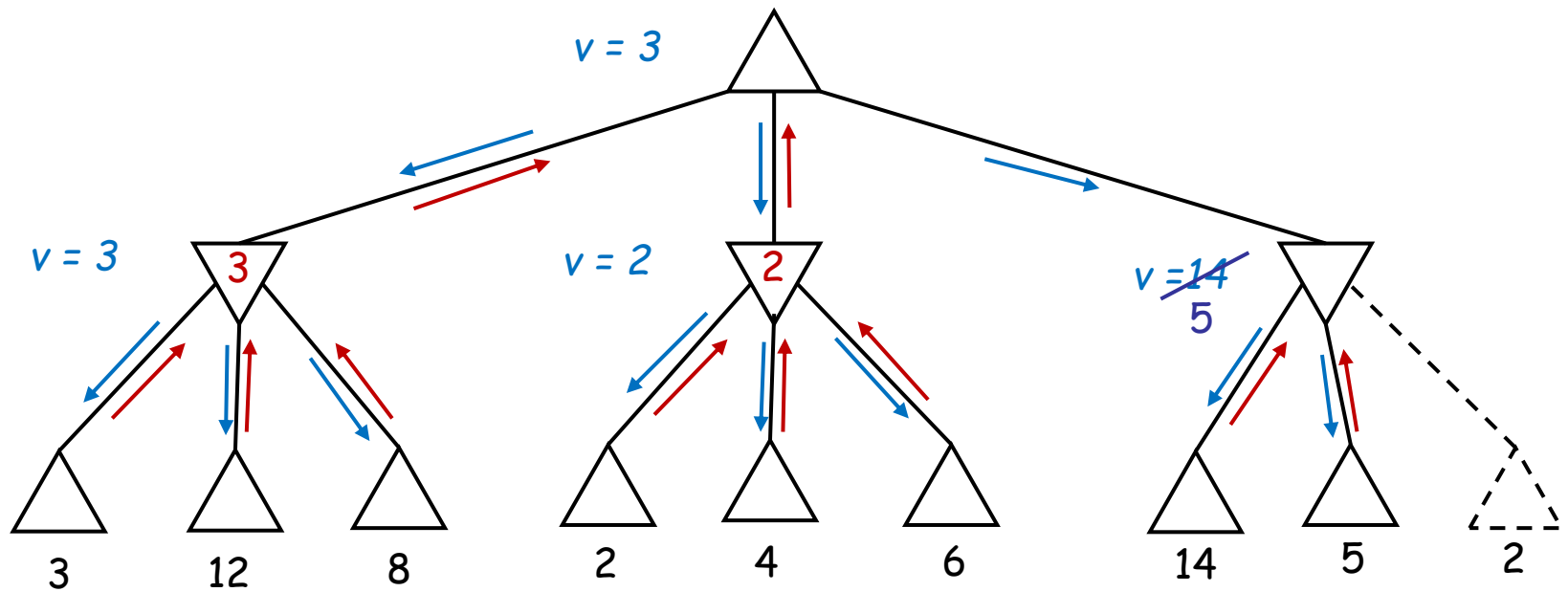
Example



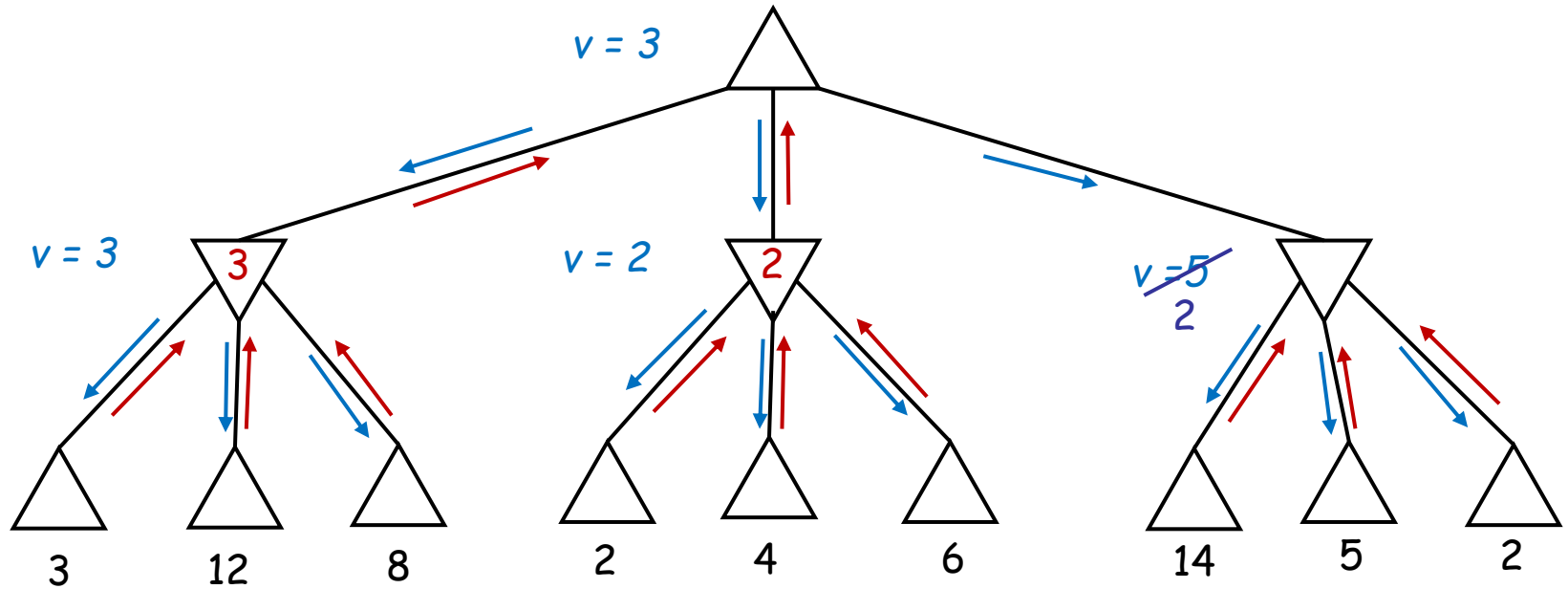
Example



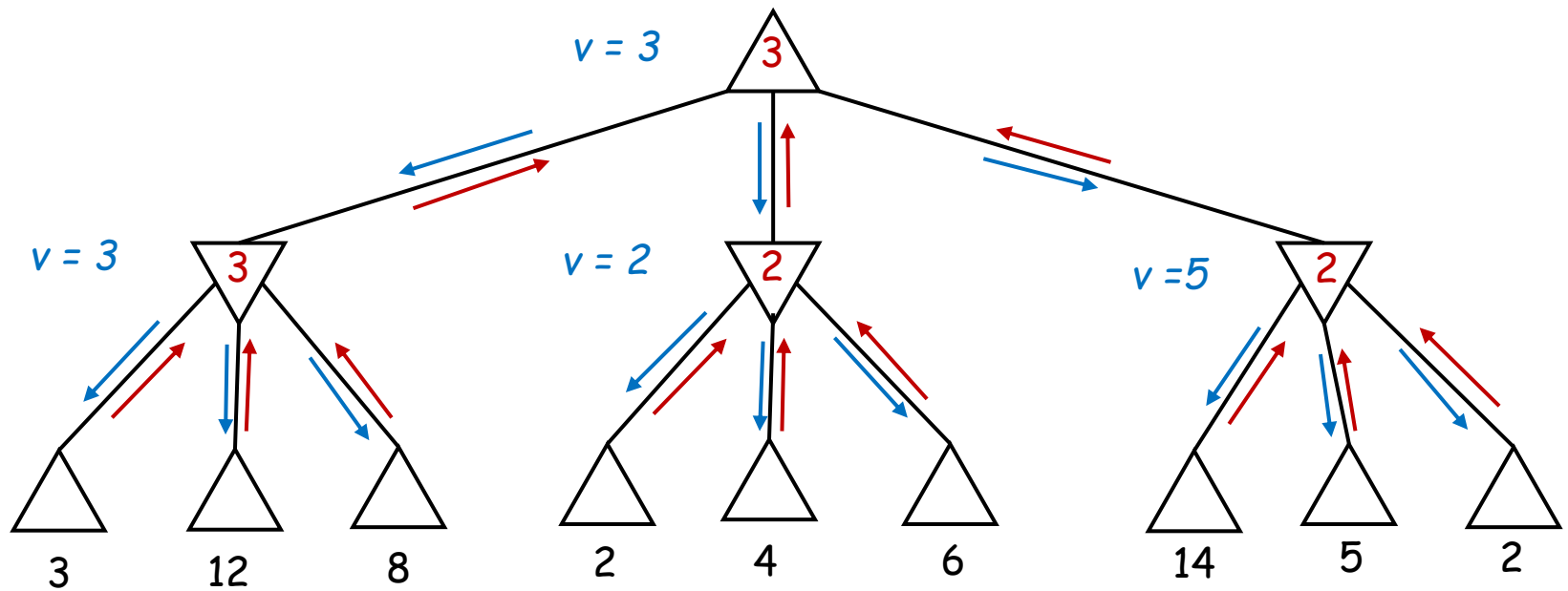
Example



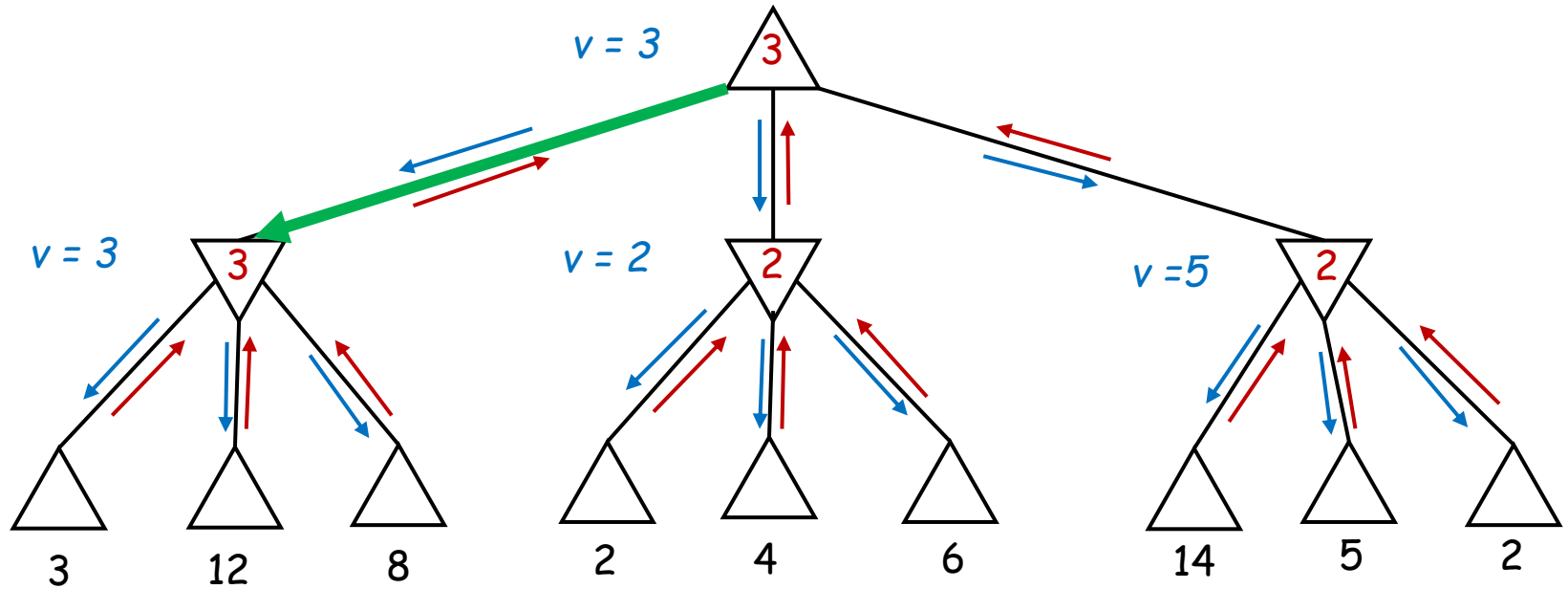
Example



Example



Example



Properties of minimax

- Complete?
 - Yes (when tree is finite)
- Optimal?
 - Yes (against an optimal opponent)
- Time complexity:
 - $O(b^m)$
- Space complexity:
 - $O(bm)$ (depth-first exploration)

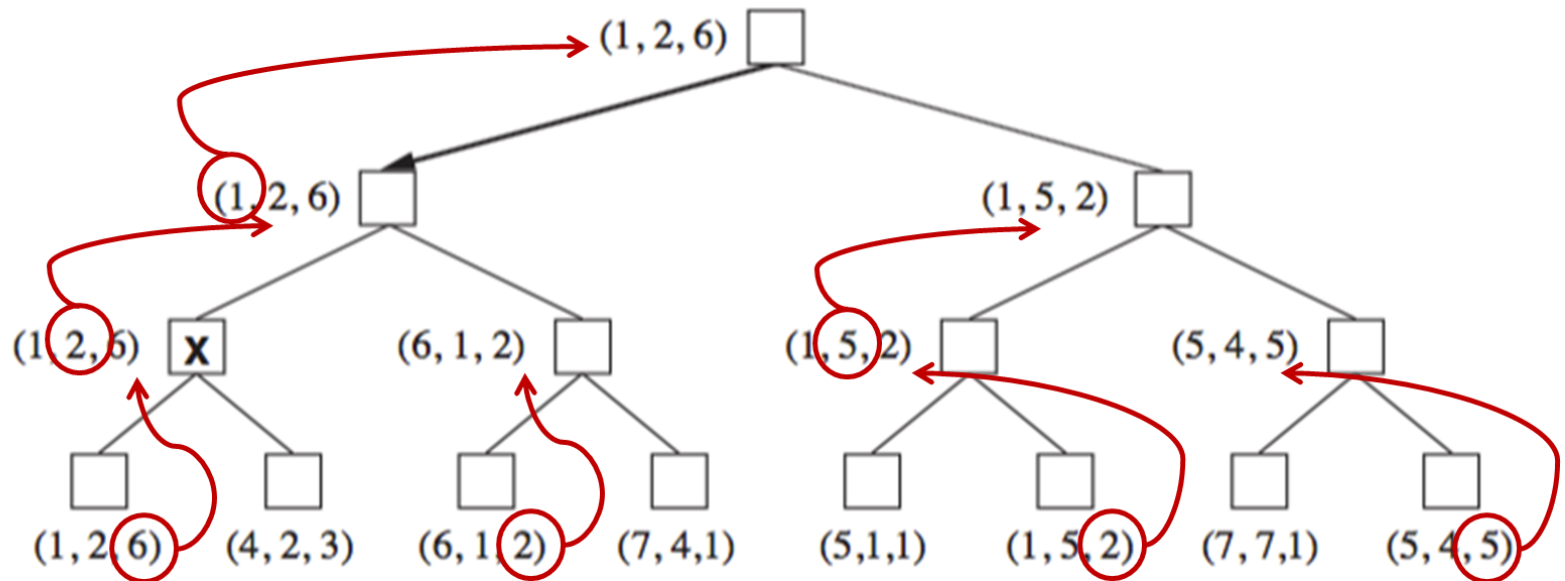
Multiplayer games

to move
A

B

C

A

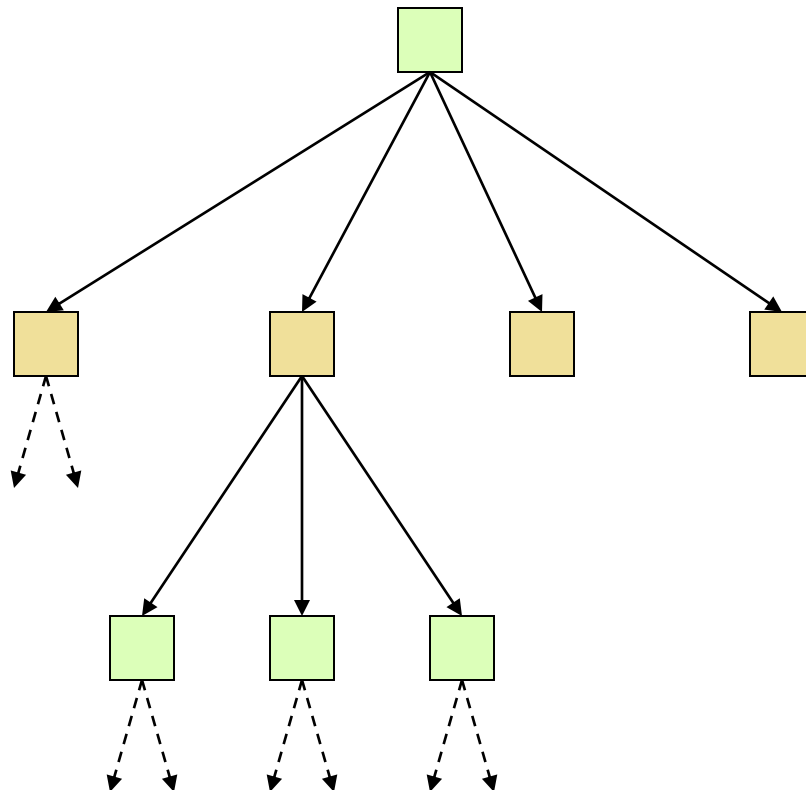


Can we do better?

Yes ! Much better !

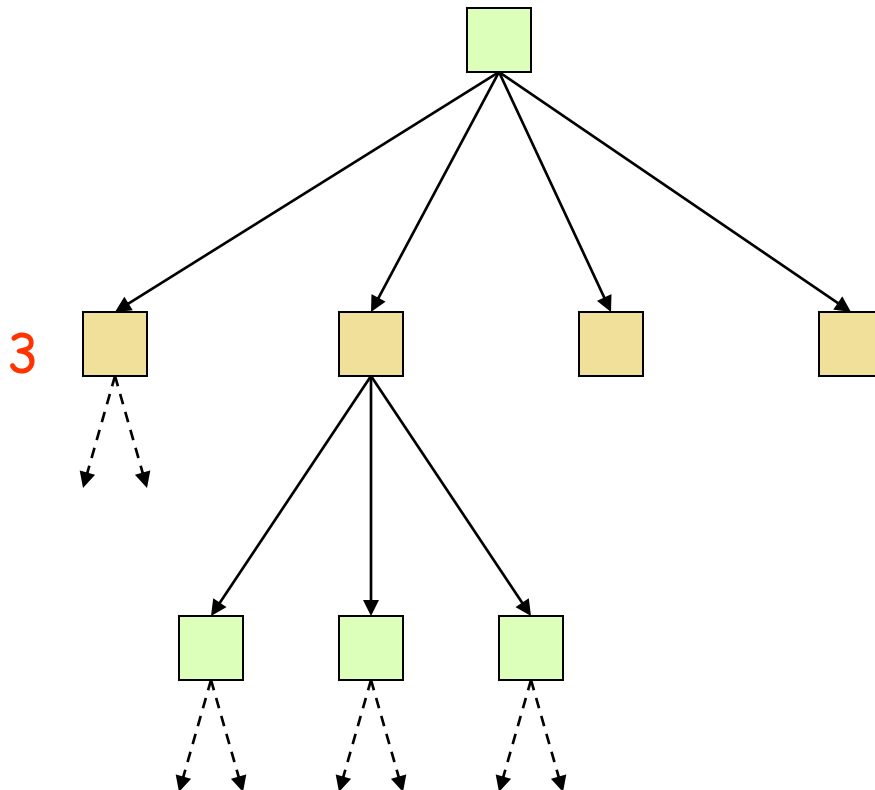
Can we do better?

Yes ! Much better !



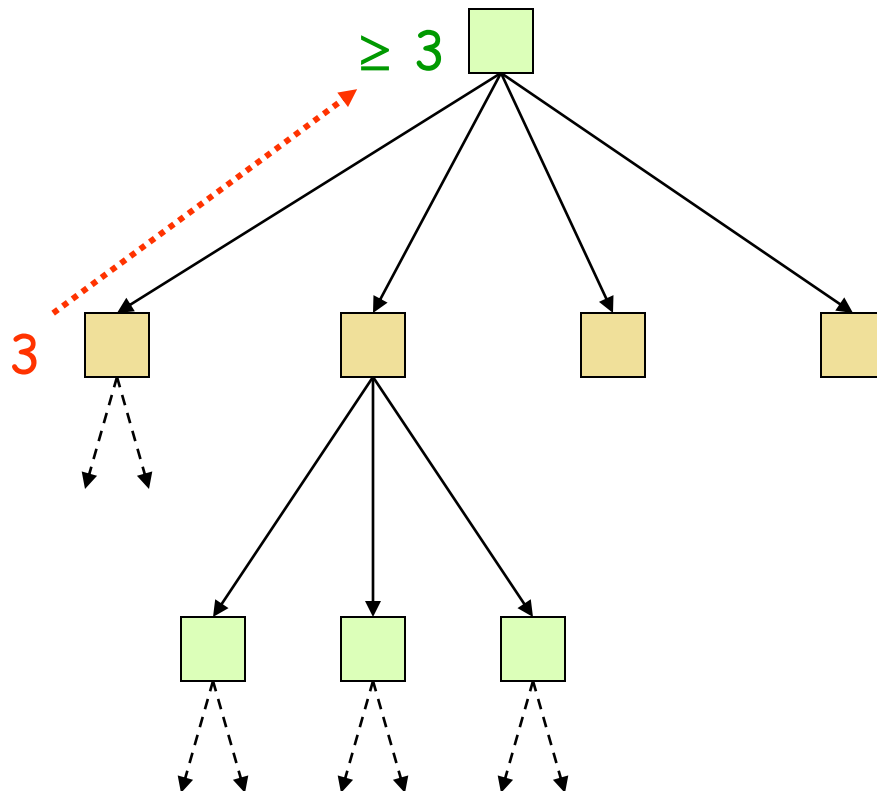
Can we do better?

Yes ! Much better !



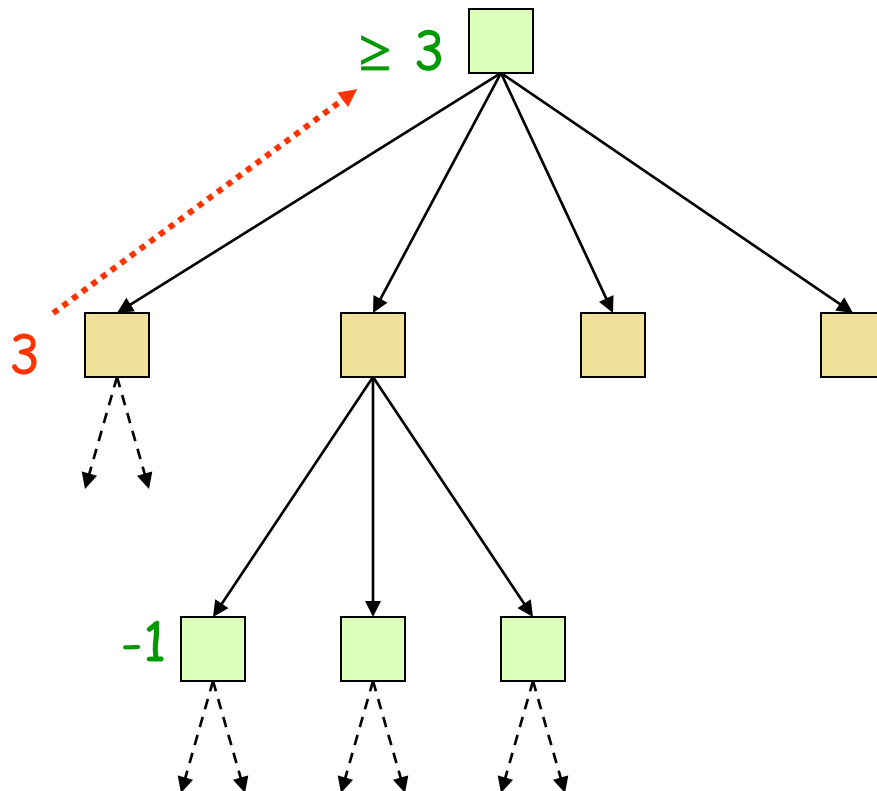
Can we do better?

Yes ! Much better !



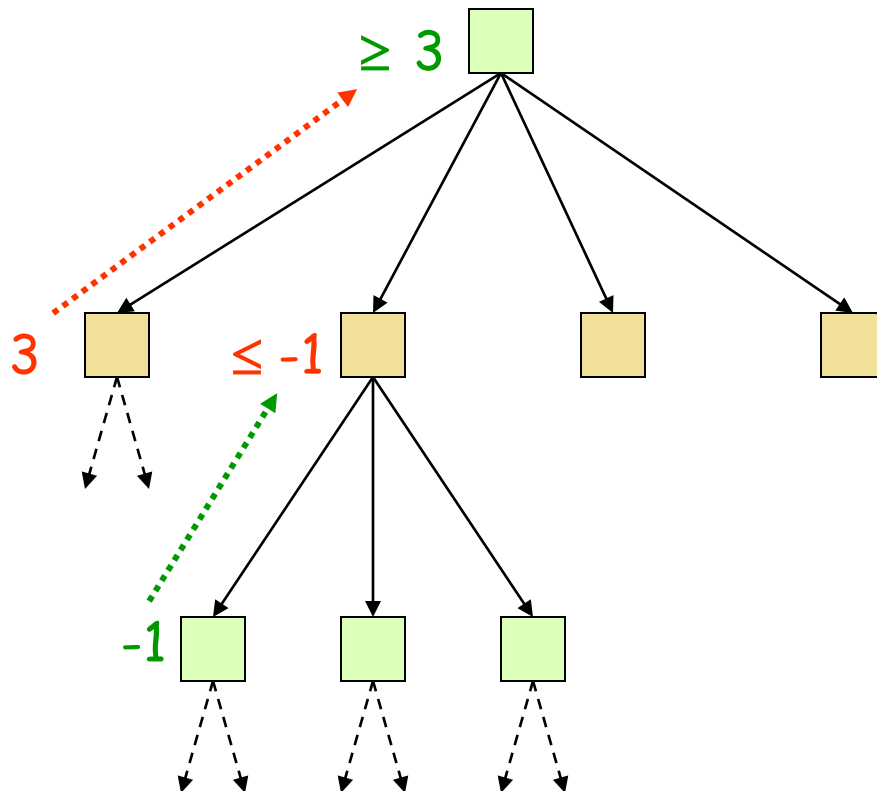
Can we do better?

Yes ! Much better !



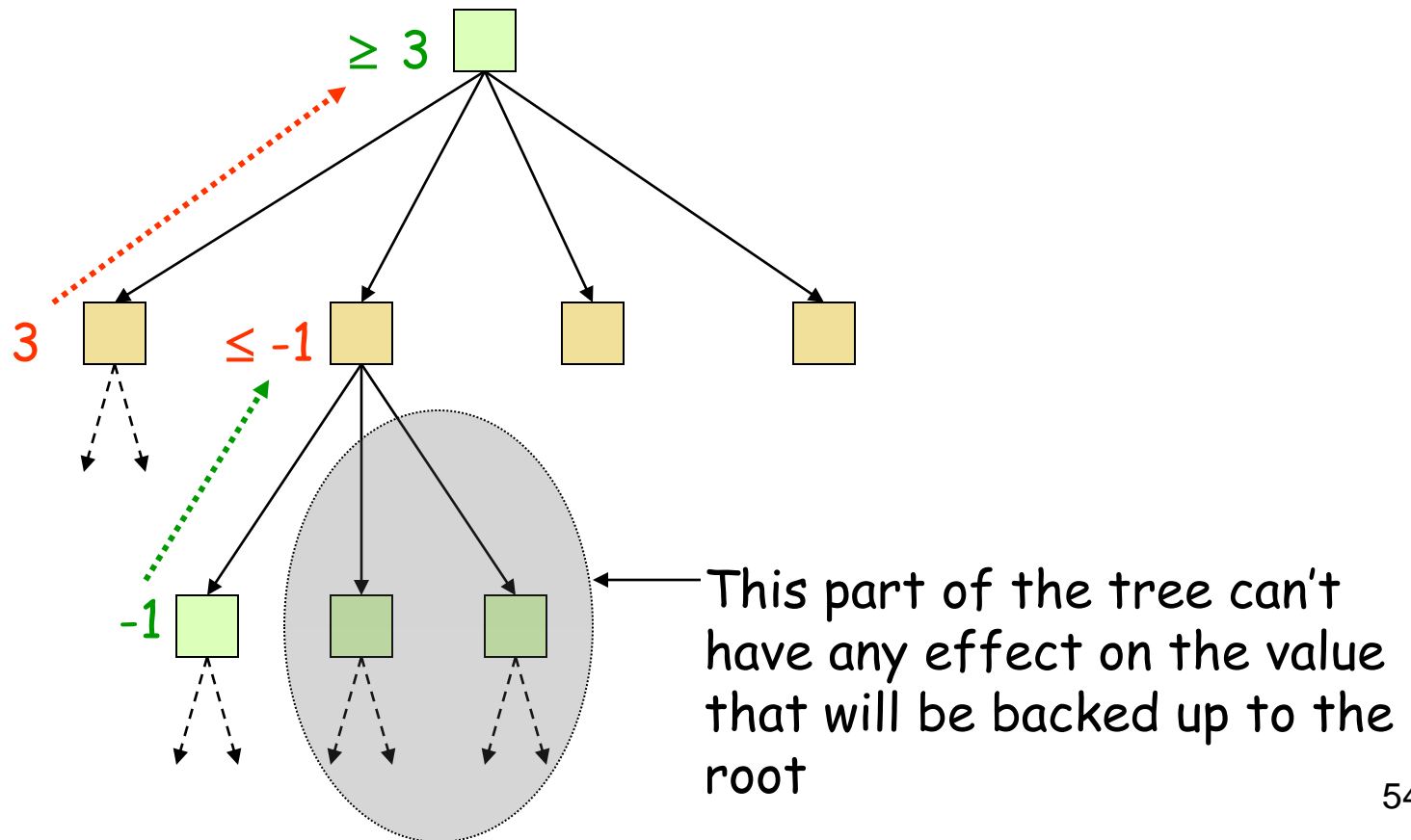
Can we do better?

Yes ! Much better !



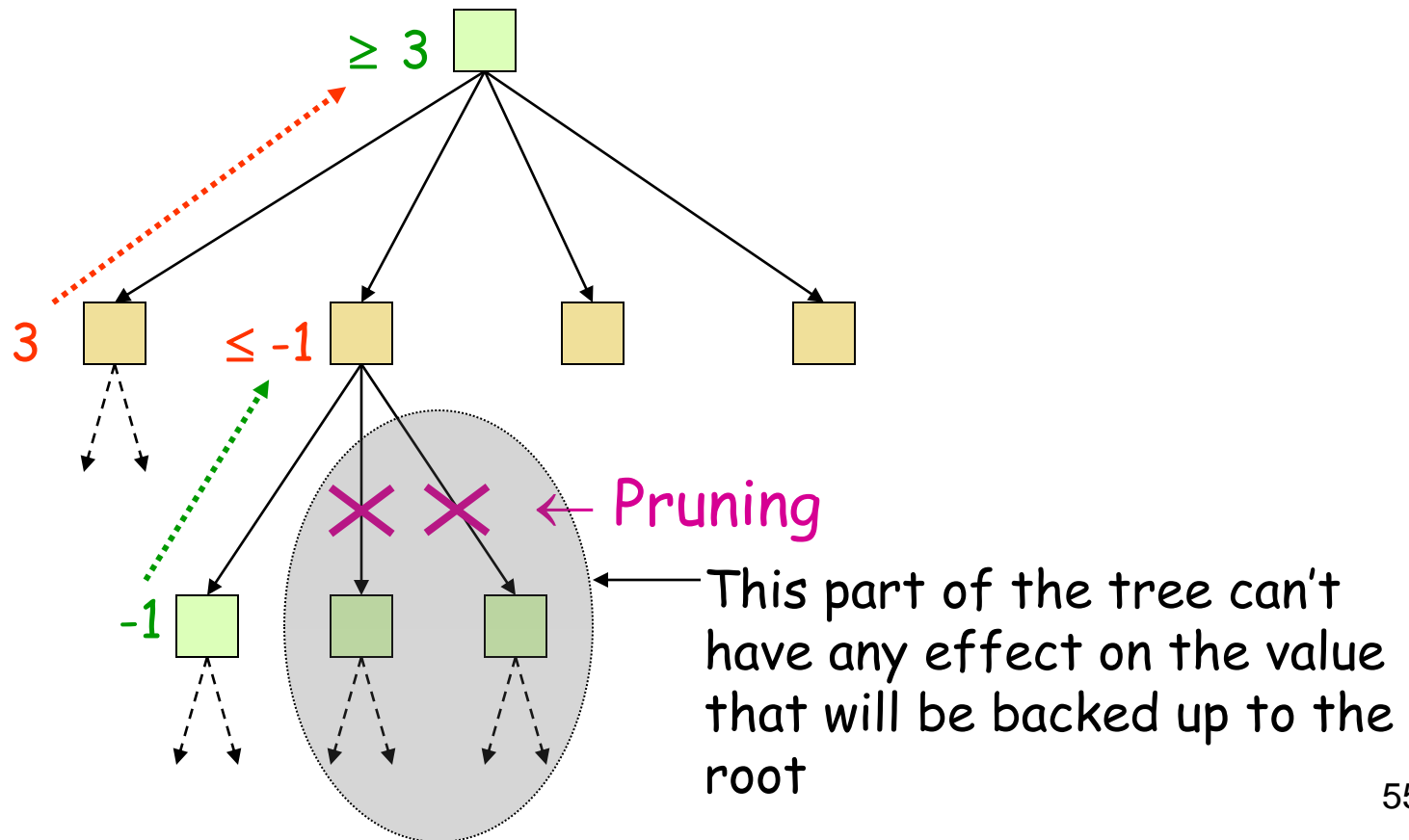
Can we do better?

Yes ! Much better !

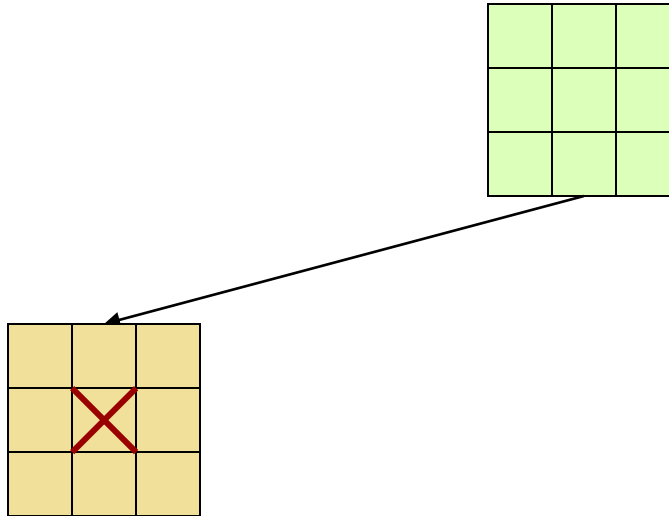


Can we do better?

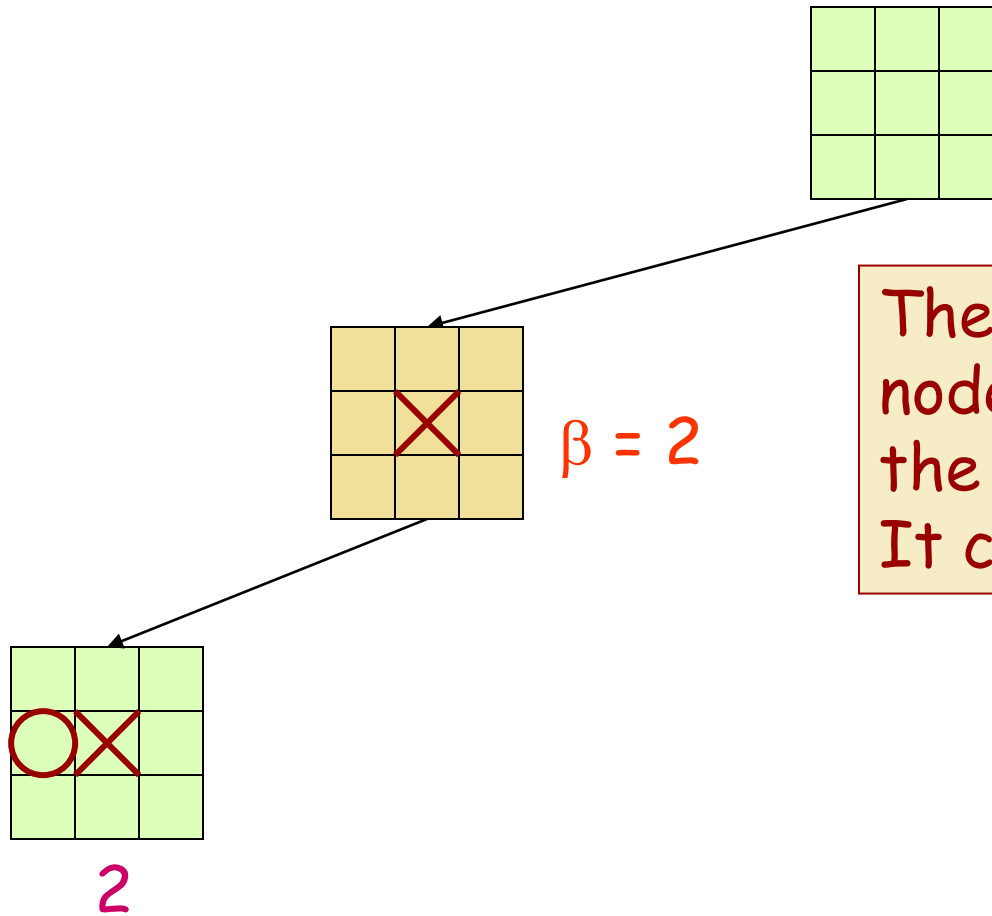
Yes ! Much better !



Example

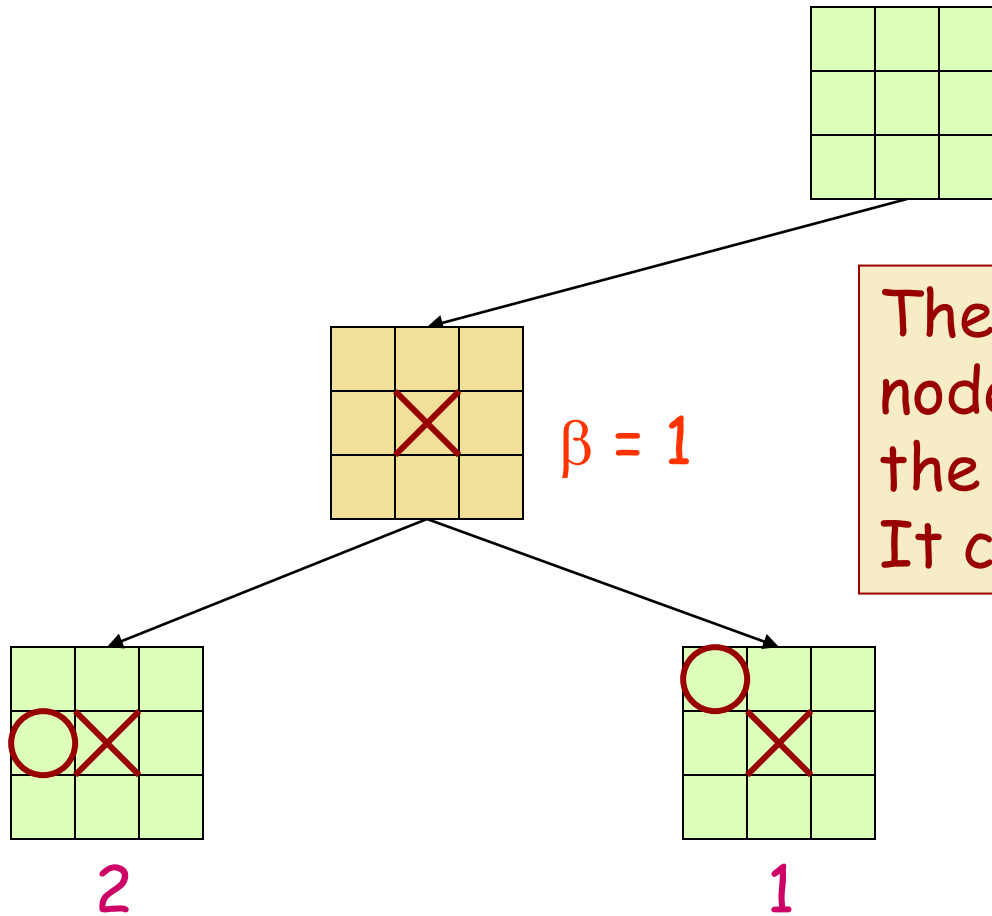


Example



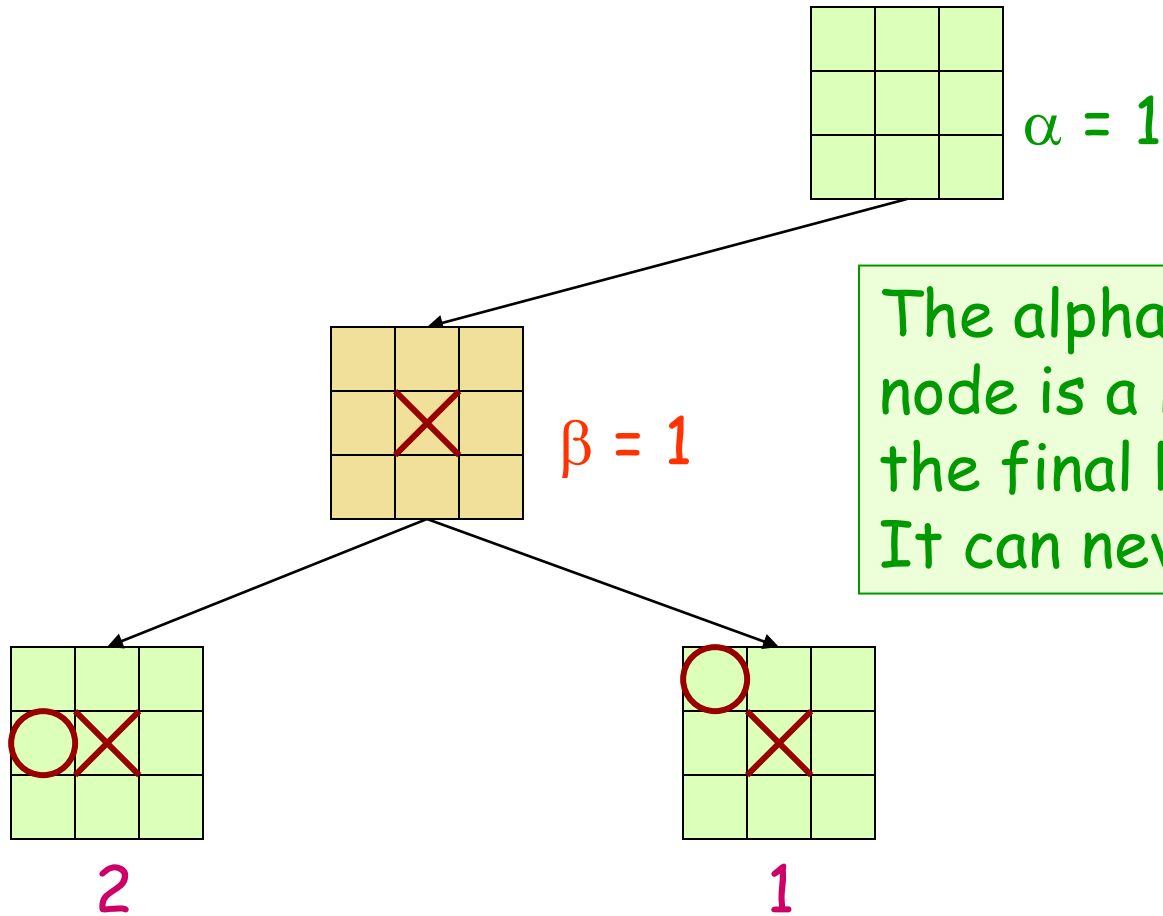
The beta value of a MIN node is an upper bound on the final backed-up value. It can never increase

Example

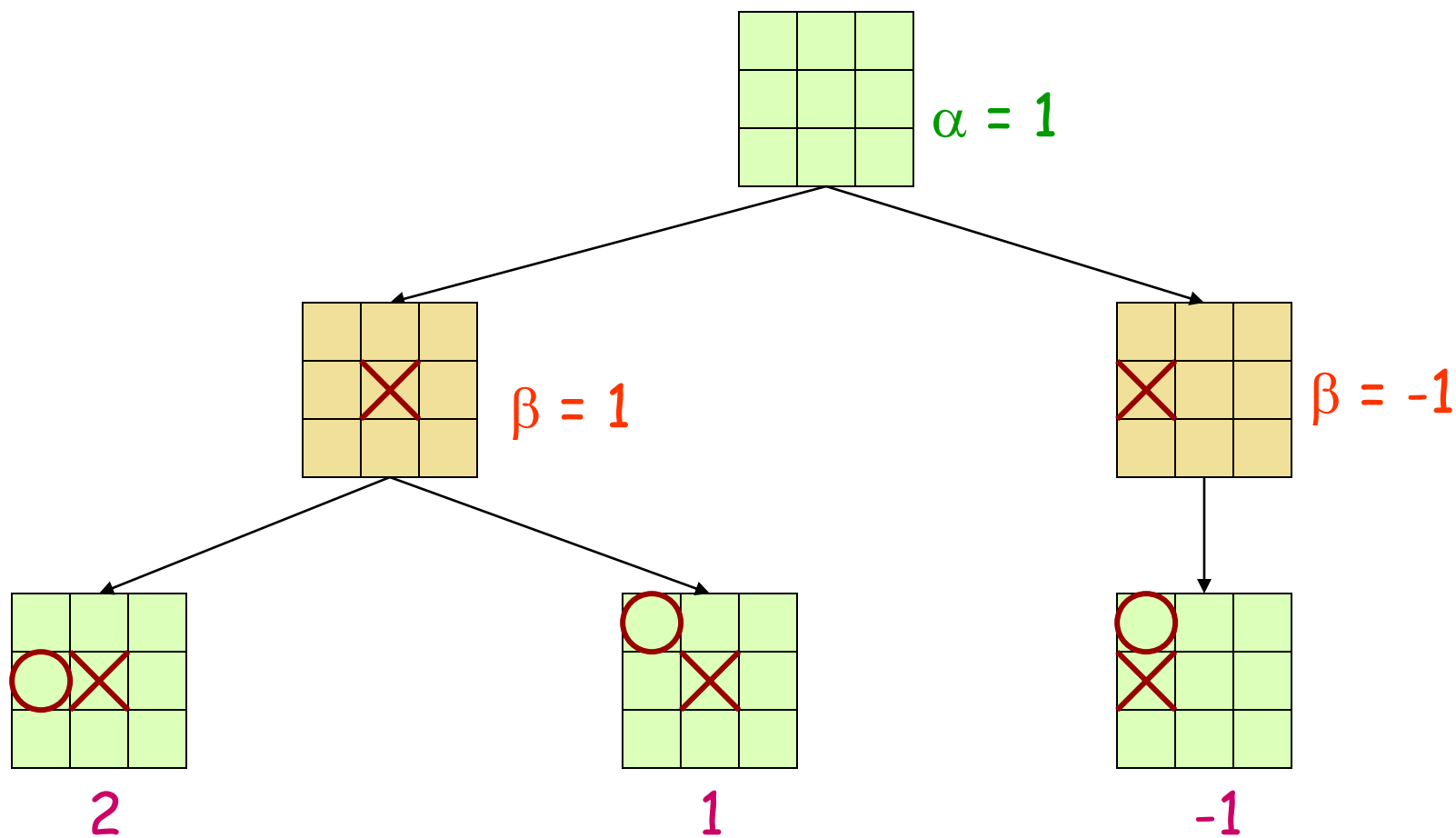


The beta value of a MIN node is an upper bound on the final backed-up value. It can never increase

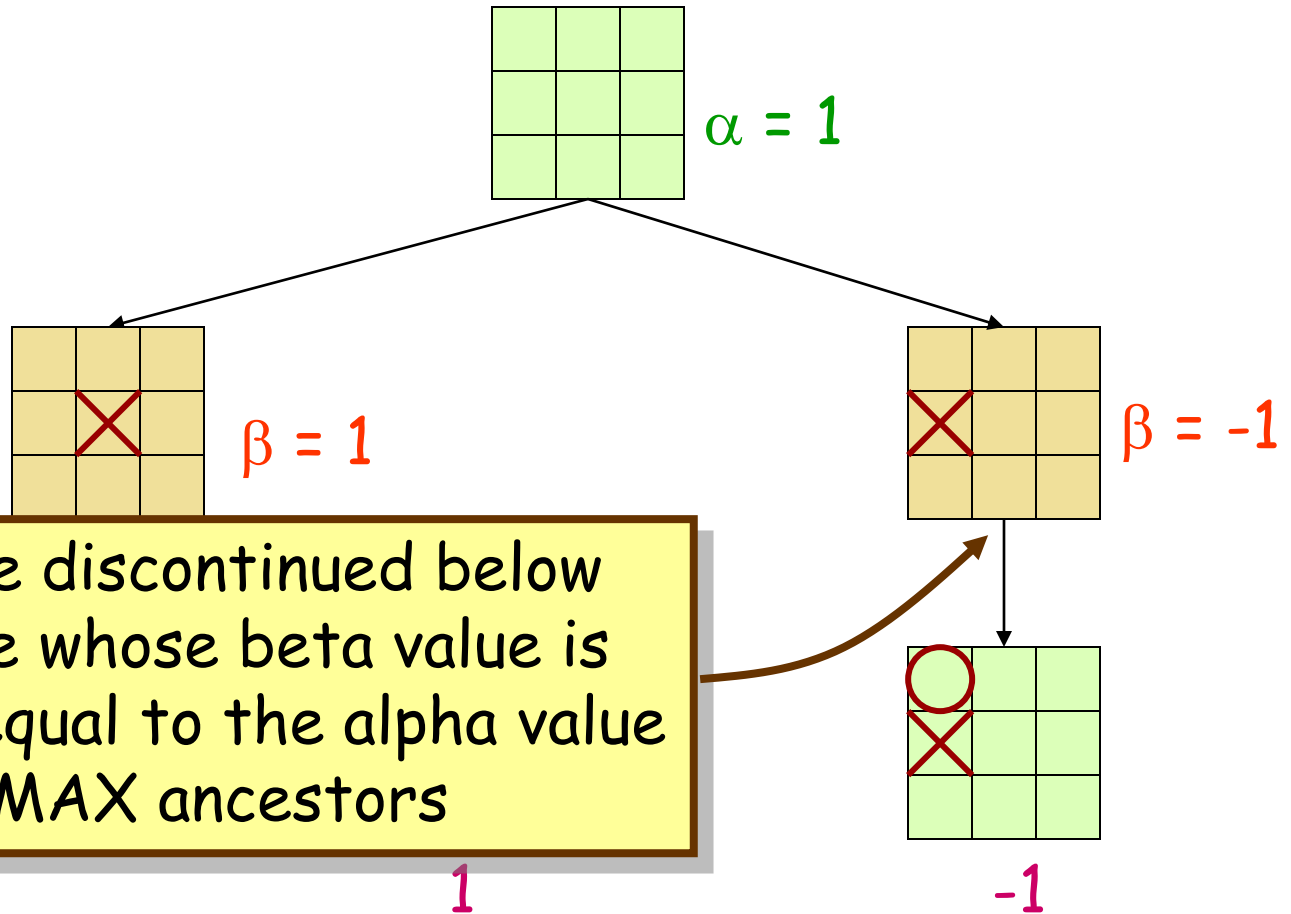
Example



Example



Example



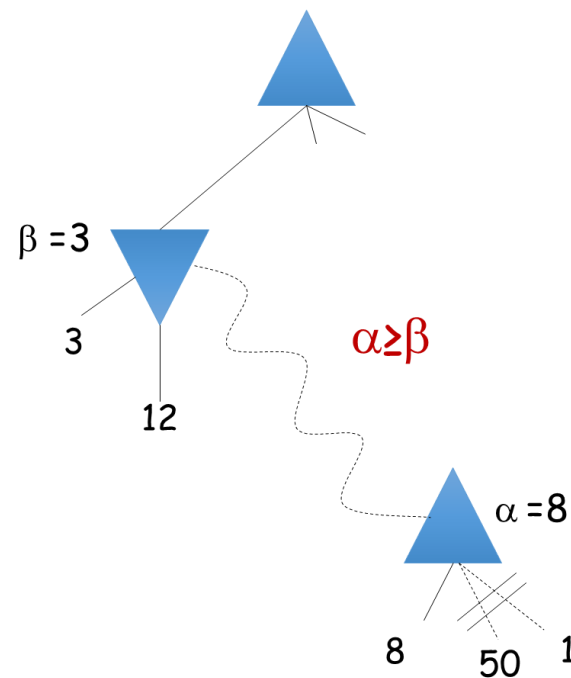
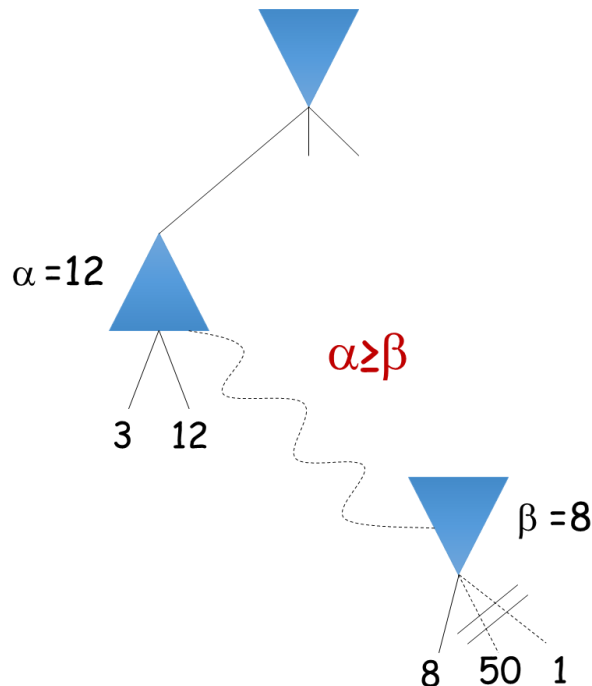
Search can be discontinued below any MIN node whose beta value is less than or equal to the alpha value of one of its MAX ancestors

Alpha-Beta Pruning

- Explore the game tree to depth h in depth-first manner
- Back up α and β values whenever possible
- Prune branches that can't lead to changing the final decision

Alpha-Beta Algorithm

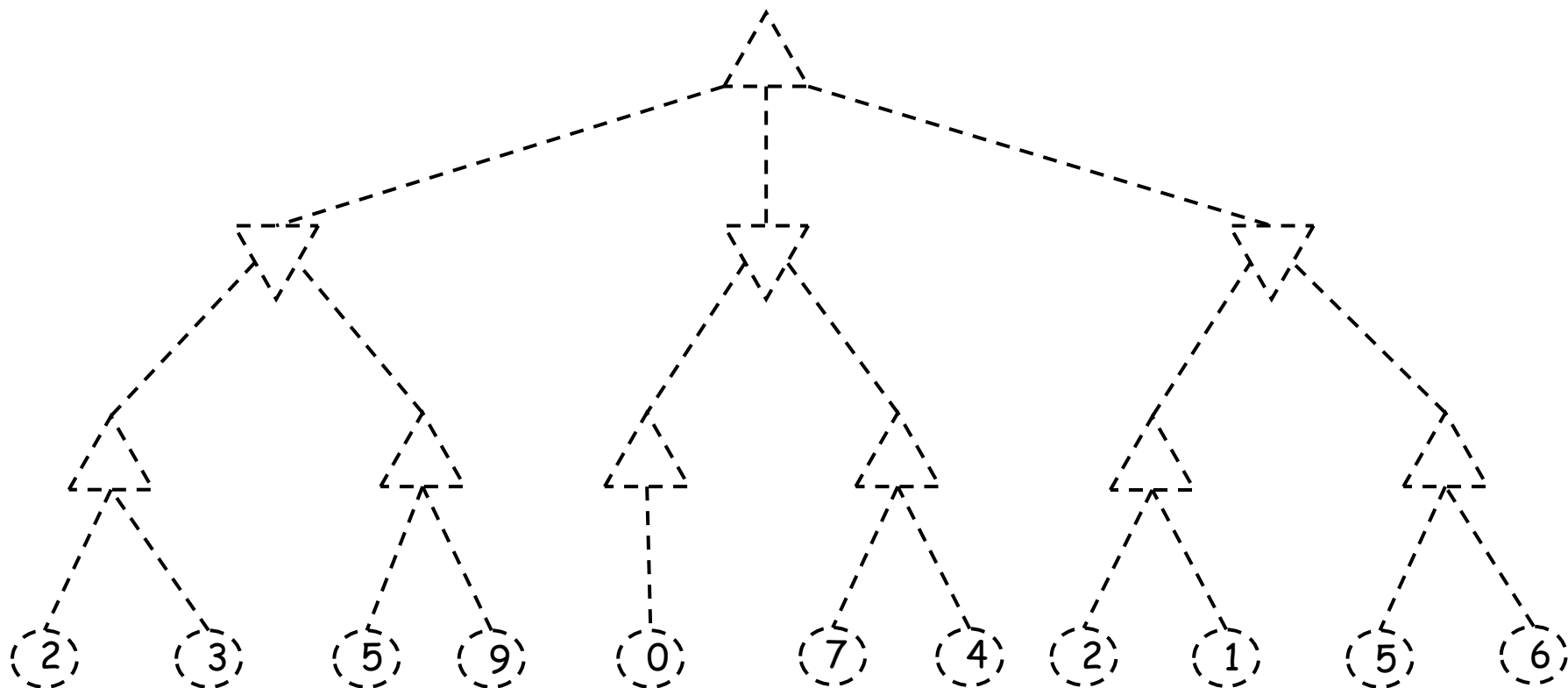
- **alpha (α)**
 - Value of the best (highest) choice found so far at any choice point along the path for **MAX**
- **beta (β)**
 - Value of the best (lowest) choice found so far at any choice point along the path for **MIN**



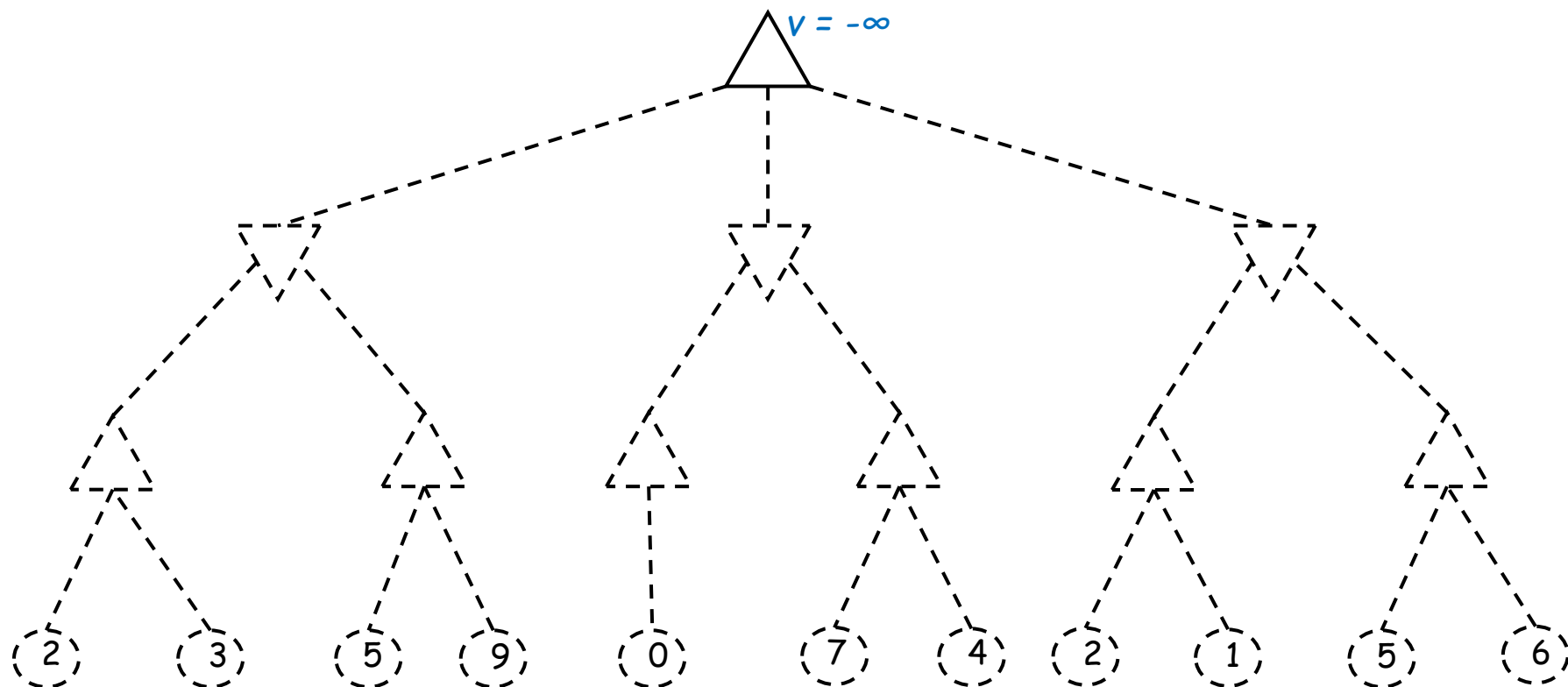
Alpha-Beta Algorithm

- Update the **alpha/beta** value of the parent of a node **N** when the search below N has been completed or discontinued
- Discontinue the search below a MAX node N if its **alpha value** is \geq the **beta** value of a MIN ancestor of N
- Discontinue the search below a MIN node N if its **beta value** is \leq the **alpha** value of a MAX ancestor of N

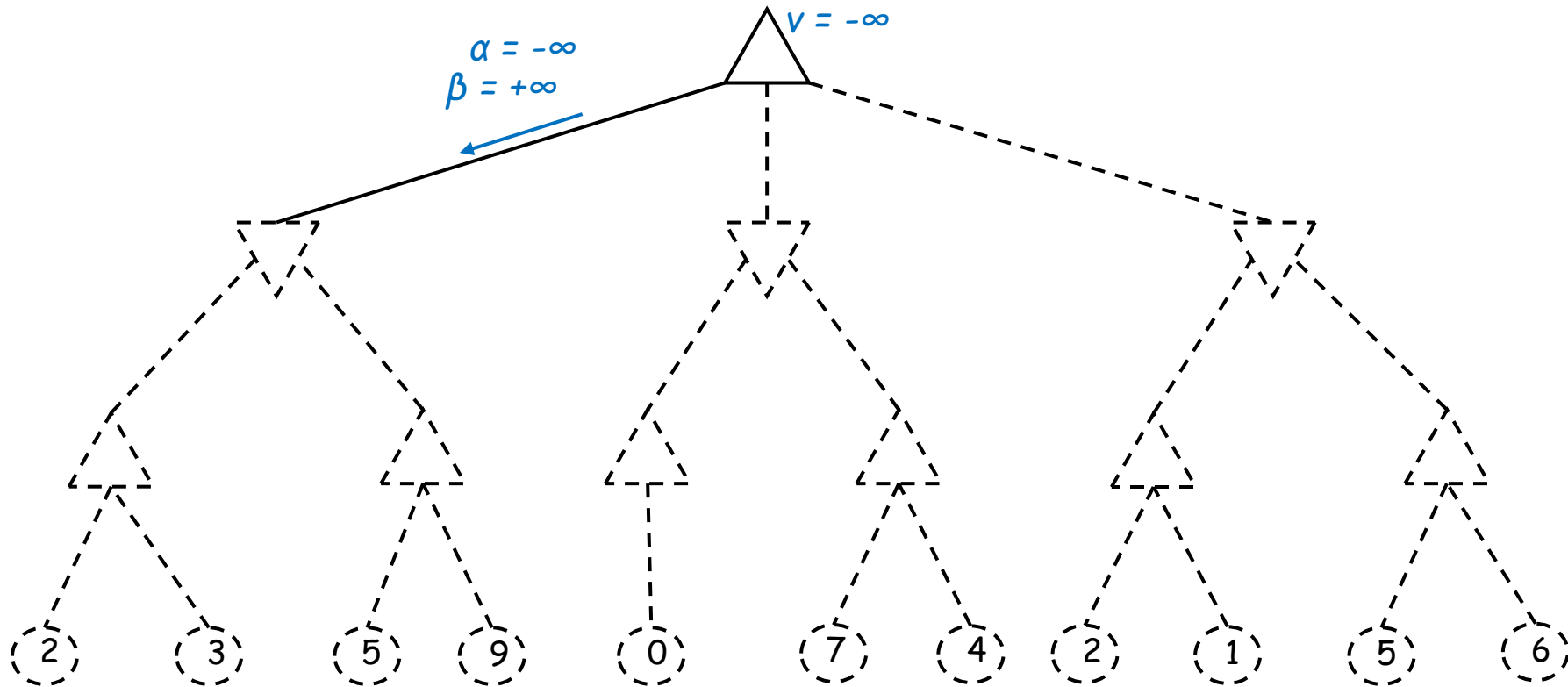
Example



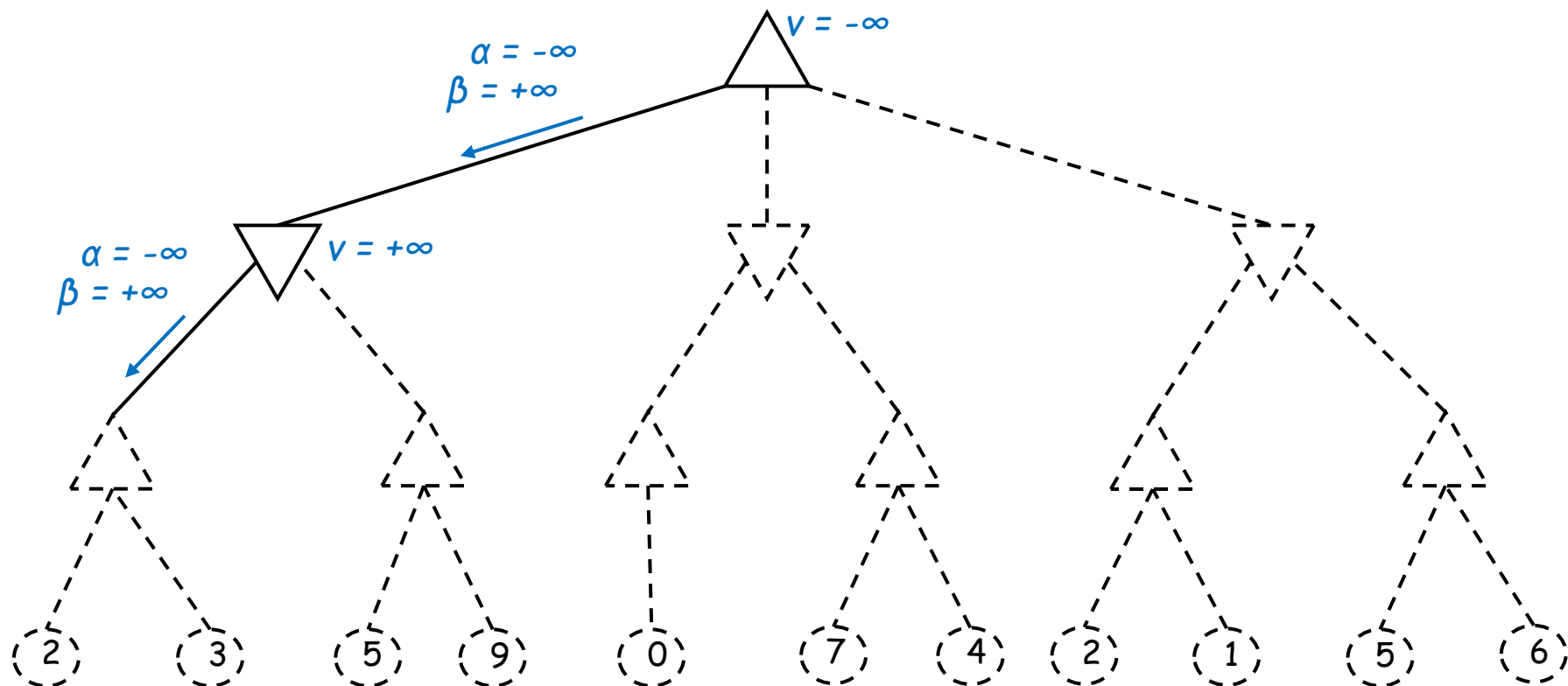
Example



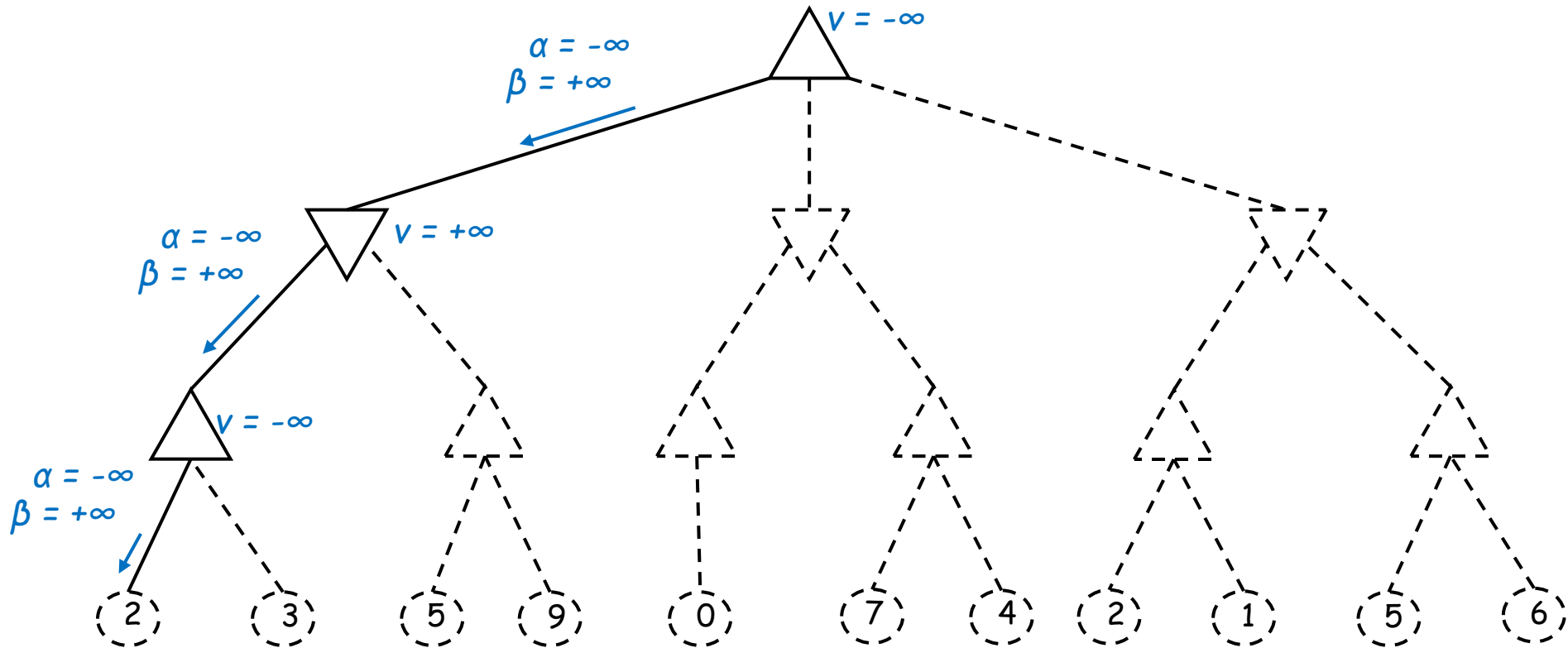
Example



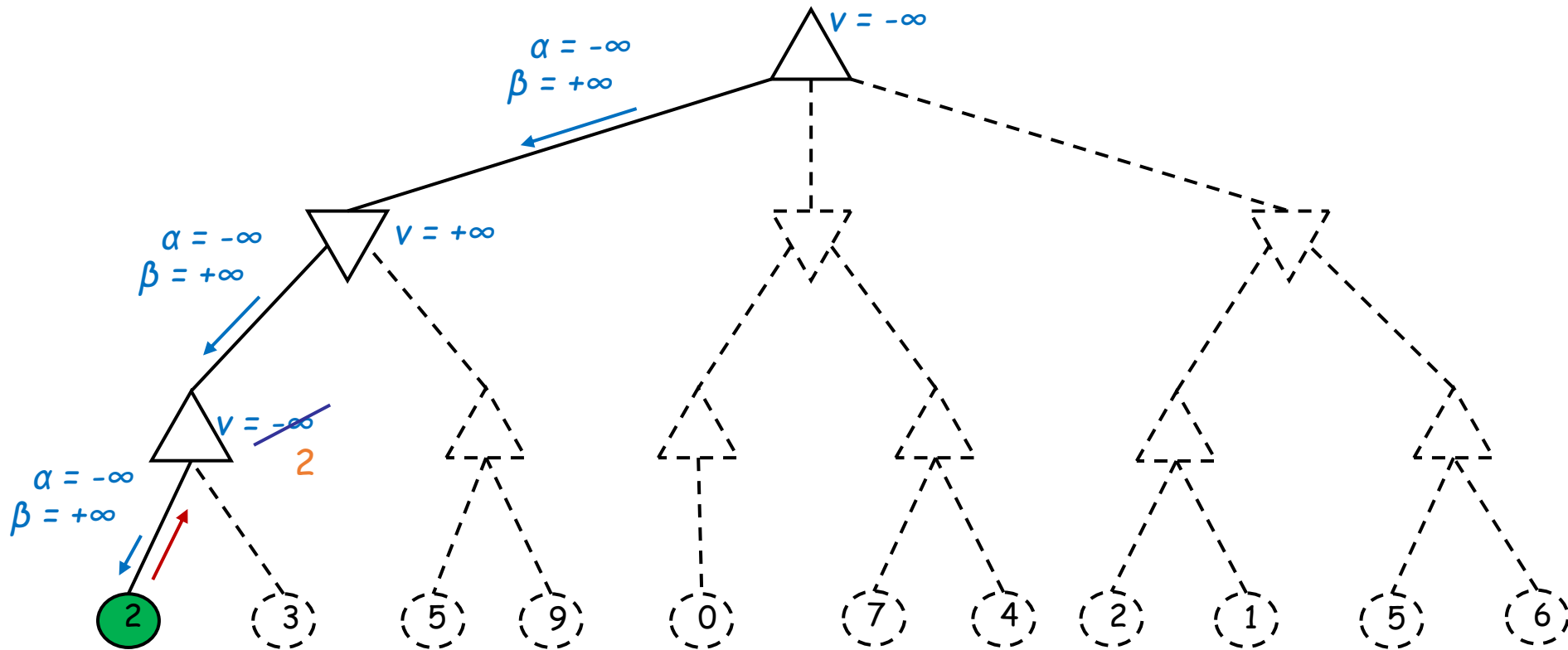
Example



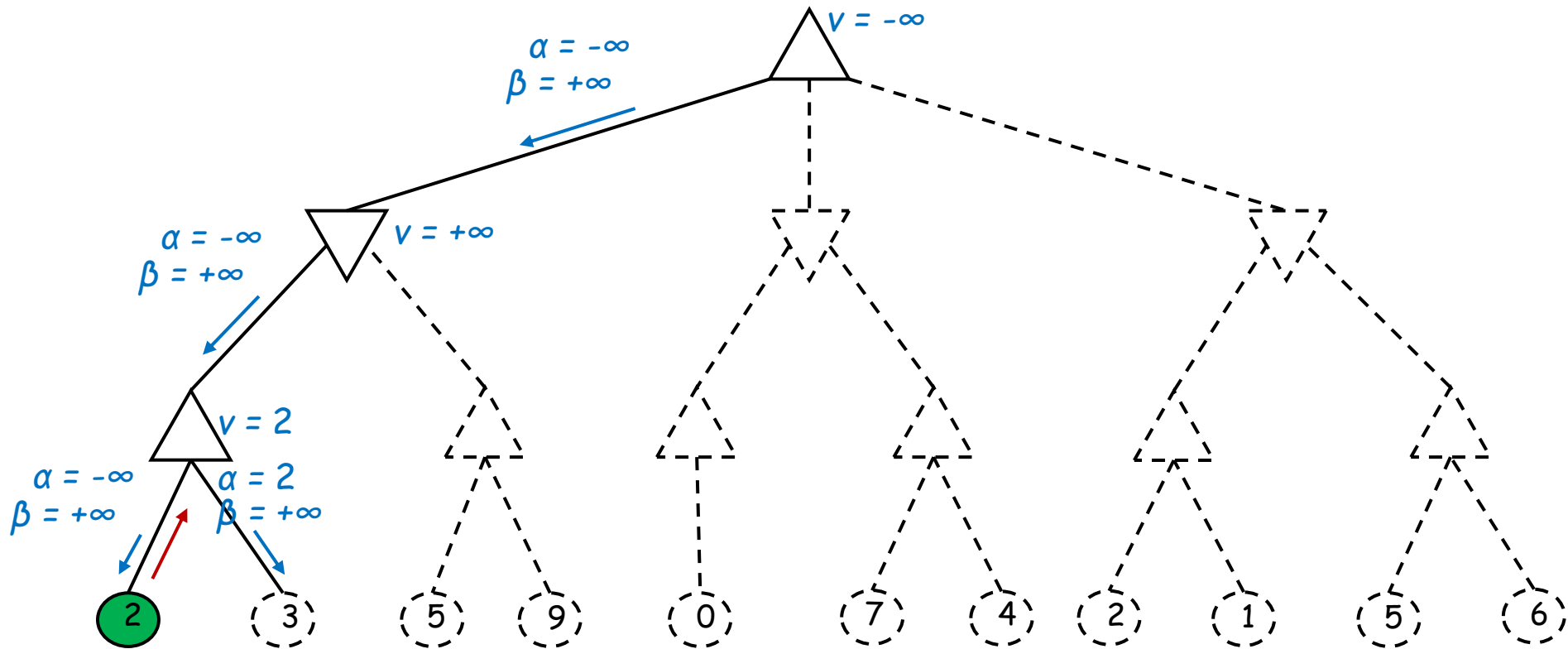
Example



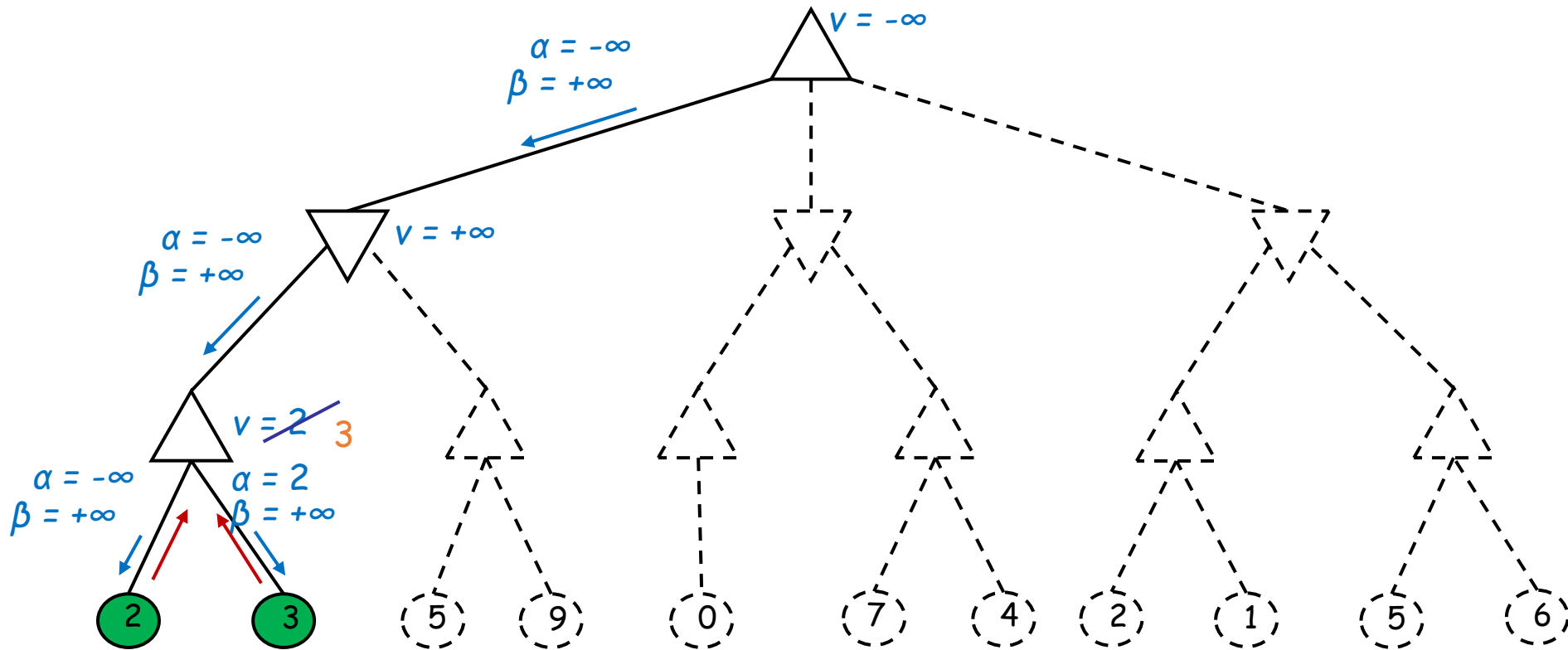
Example



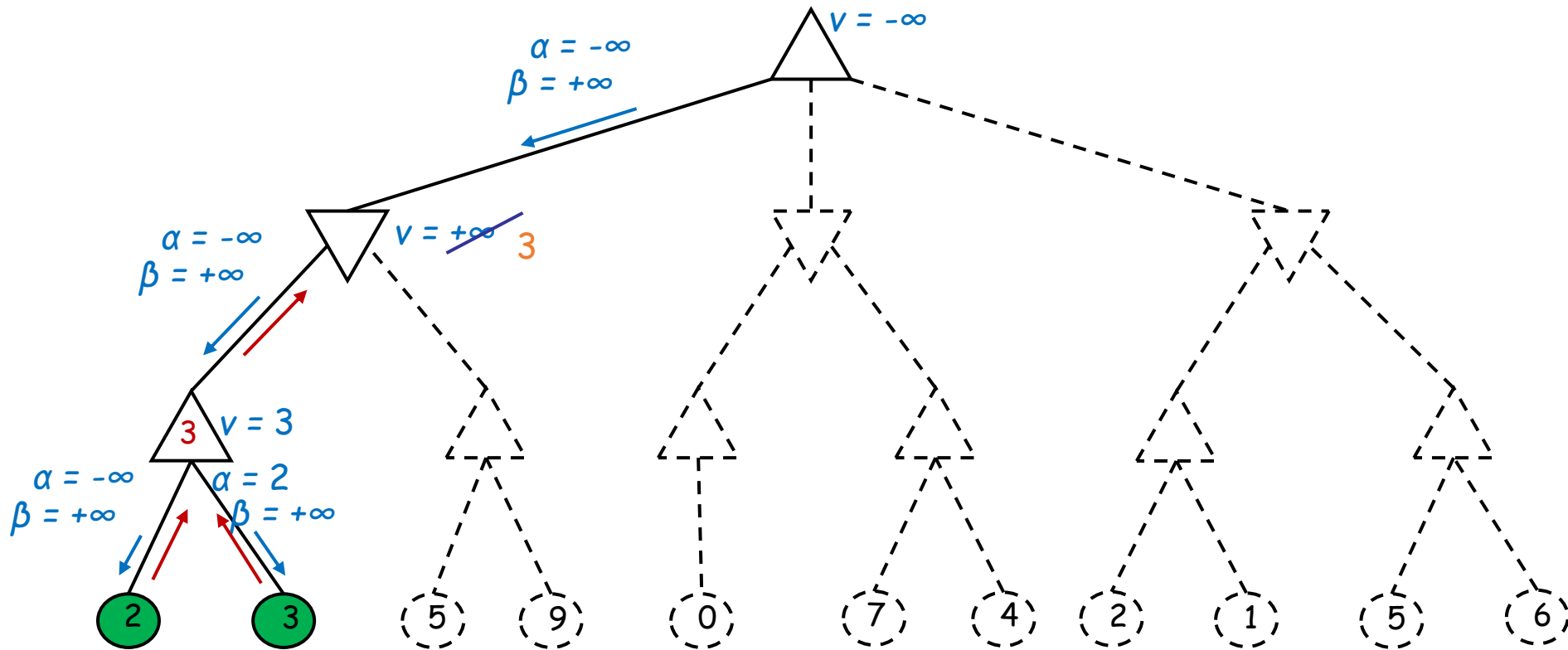
Example



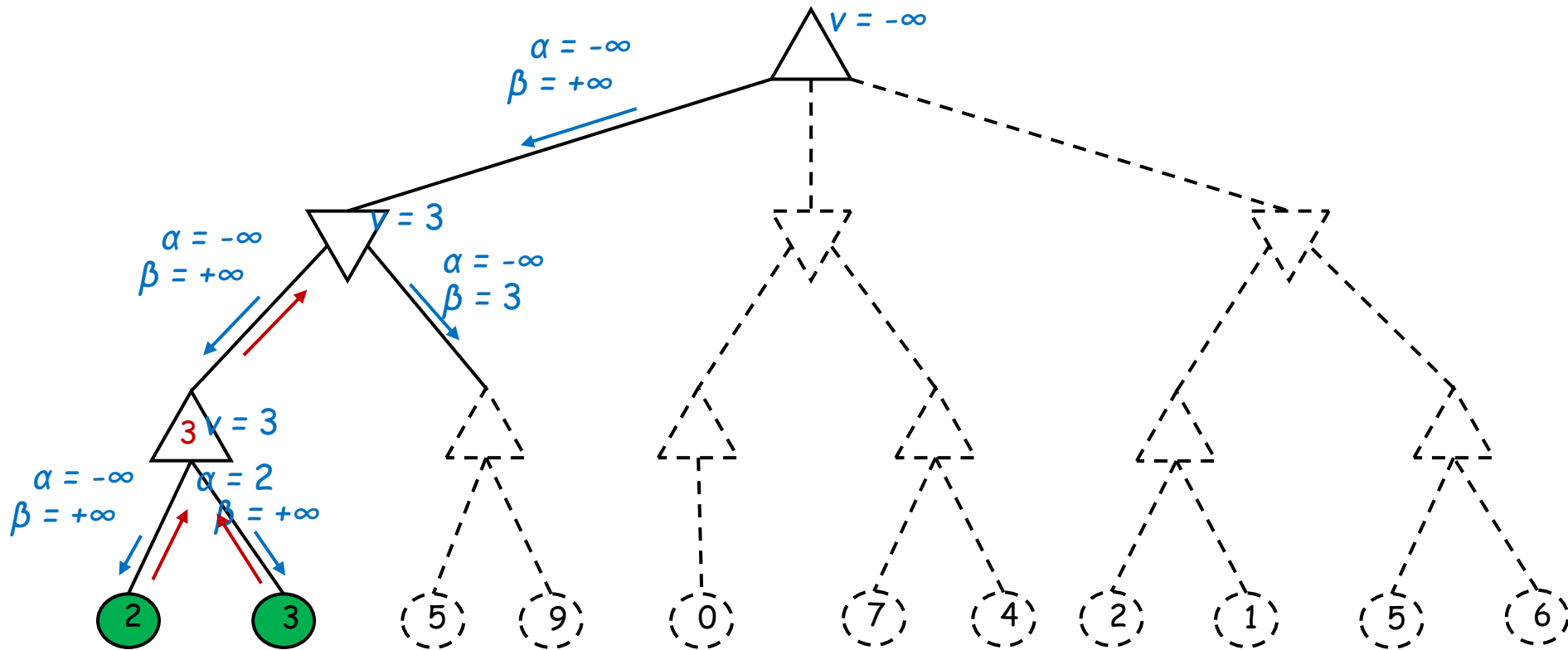
Example



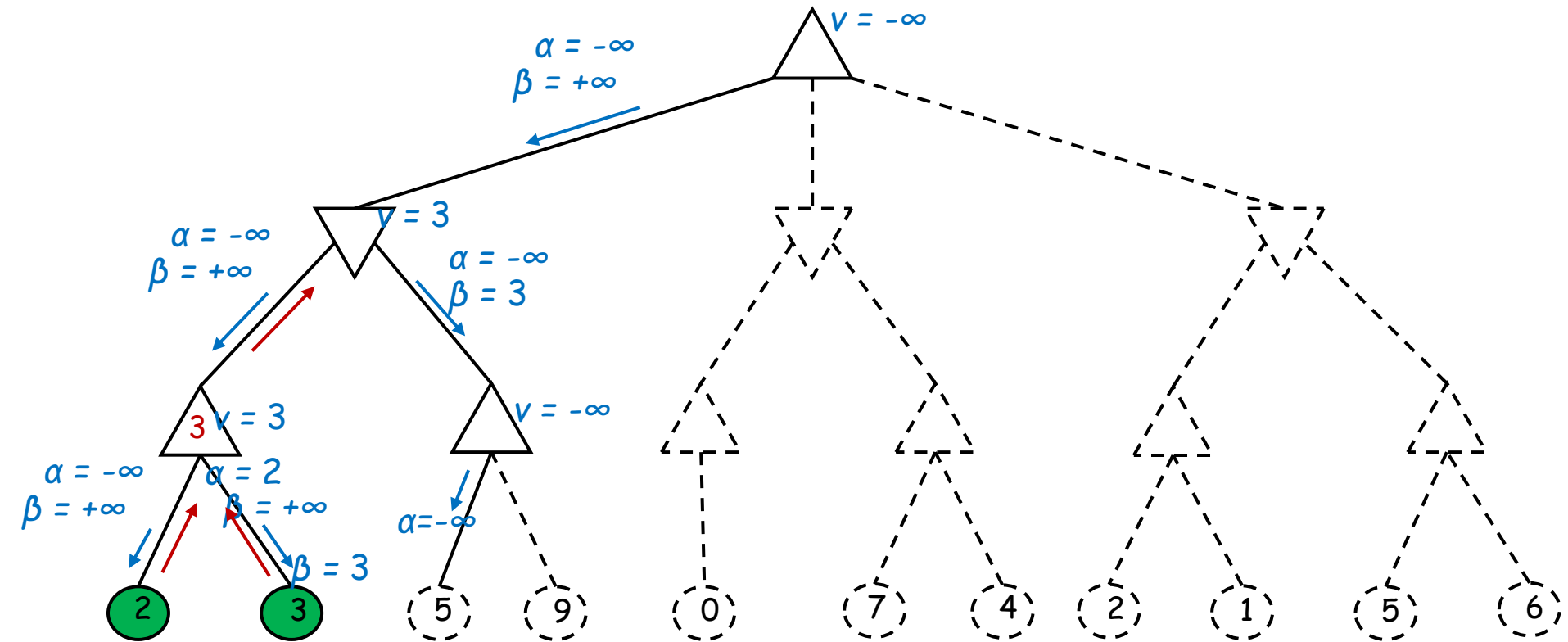
Example



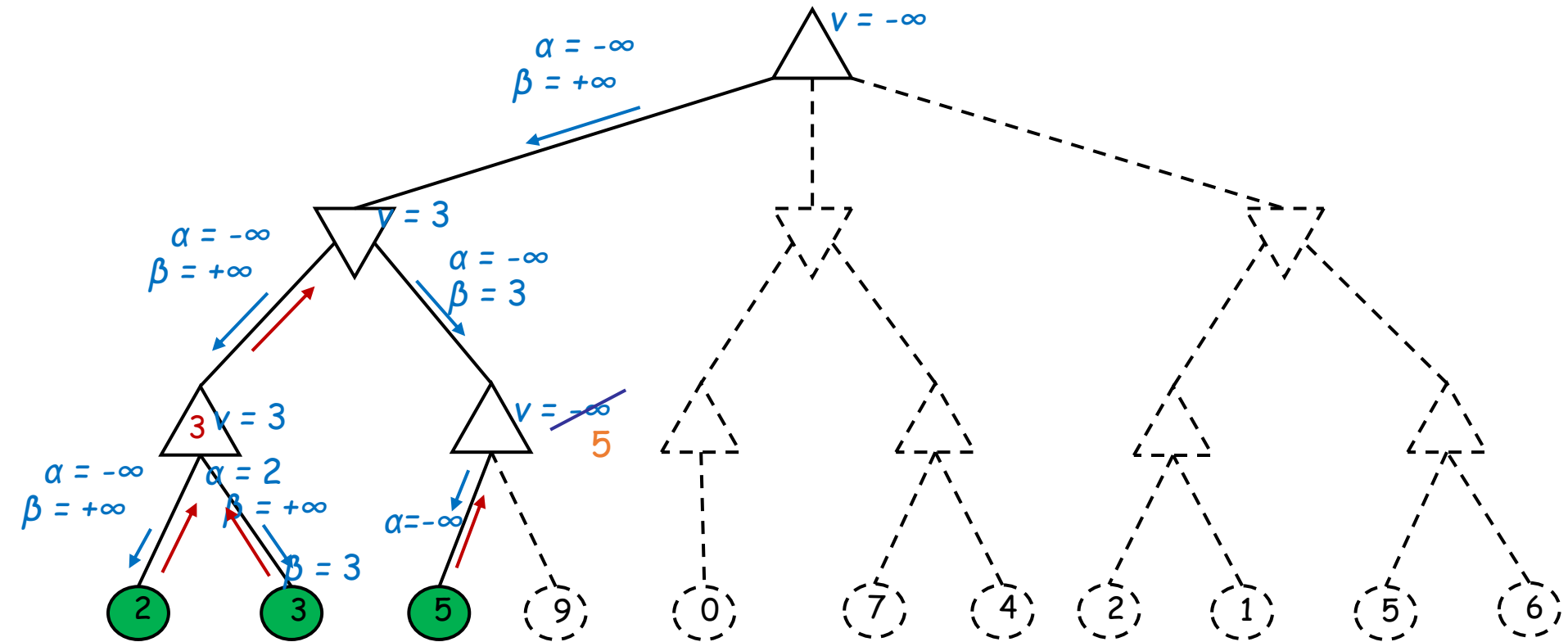
Example



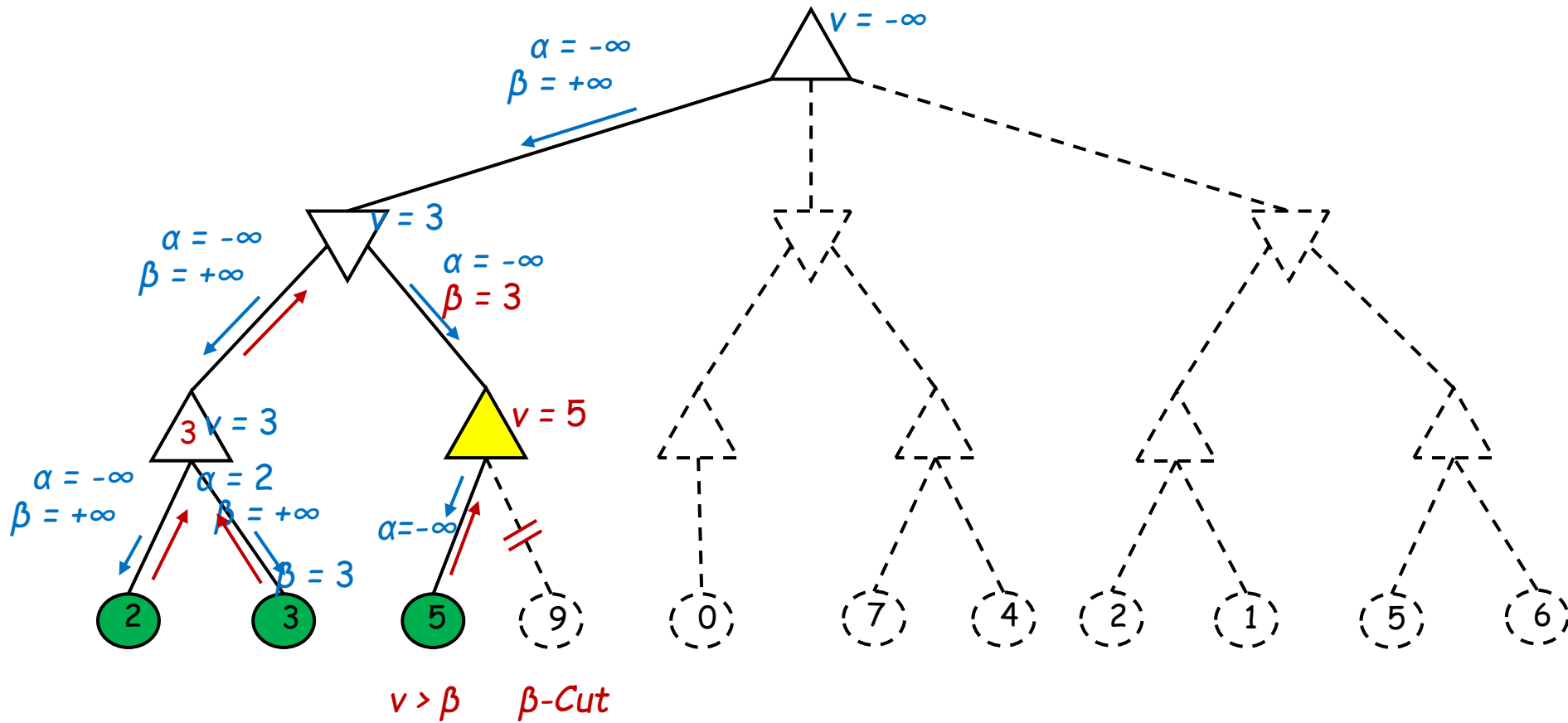
Example



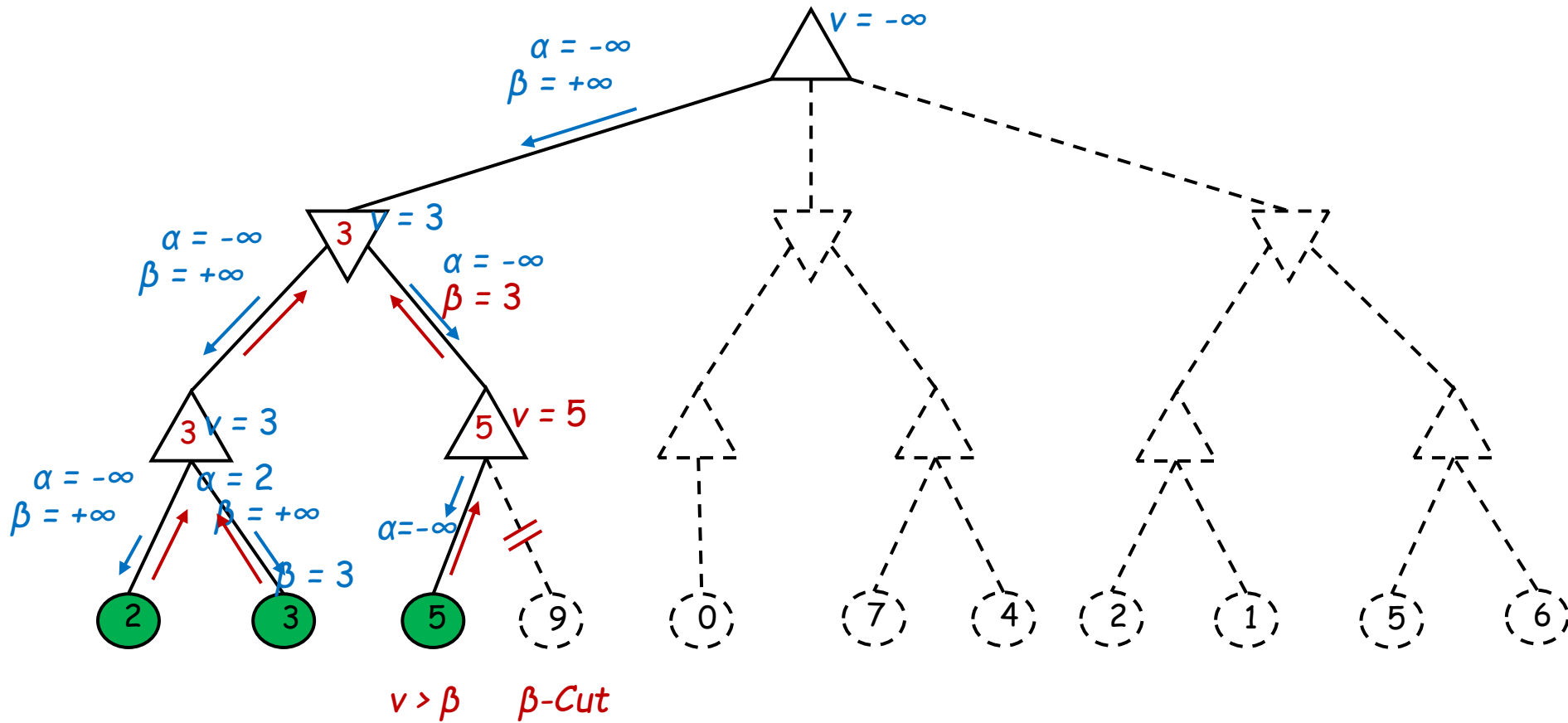
Example



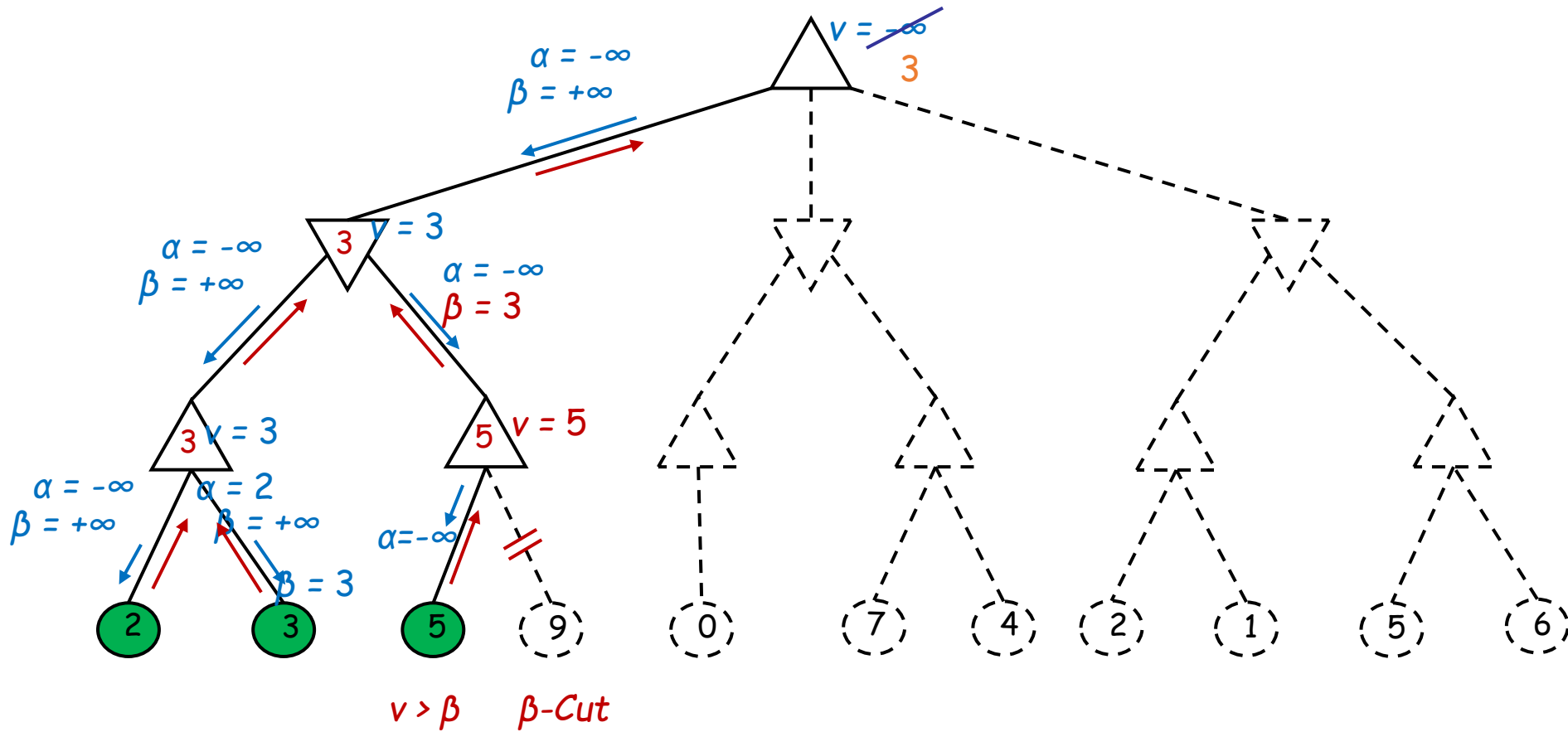
Example



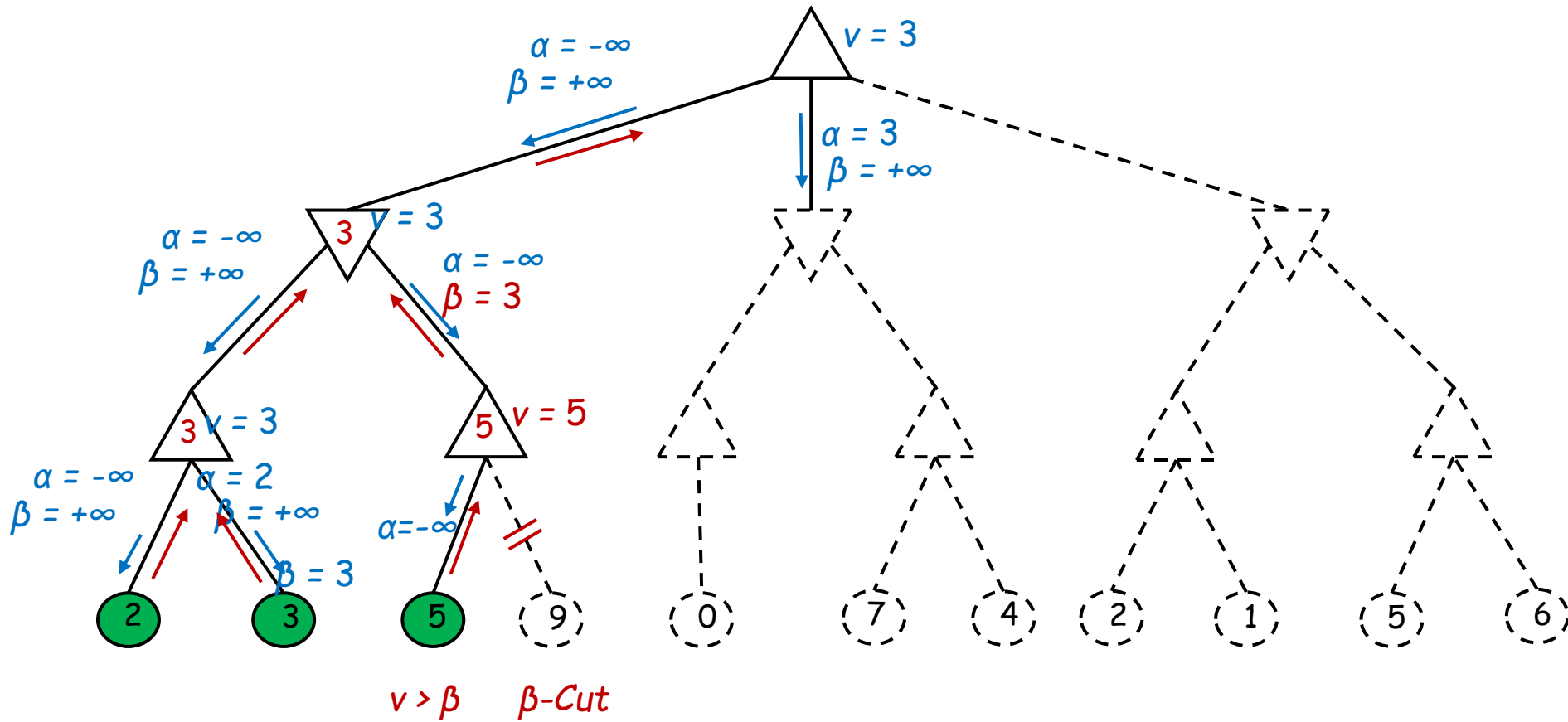
Example



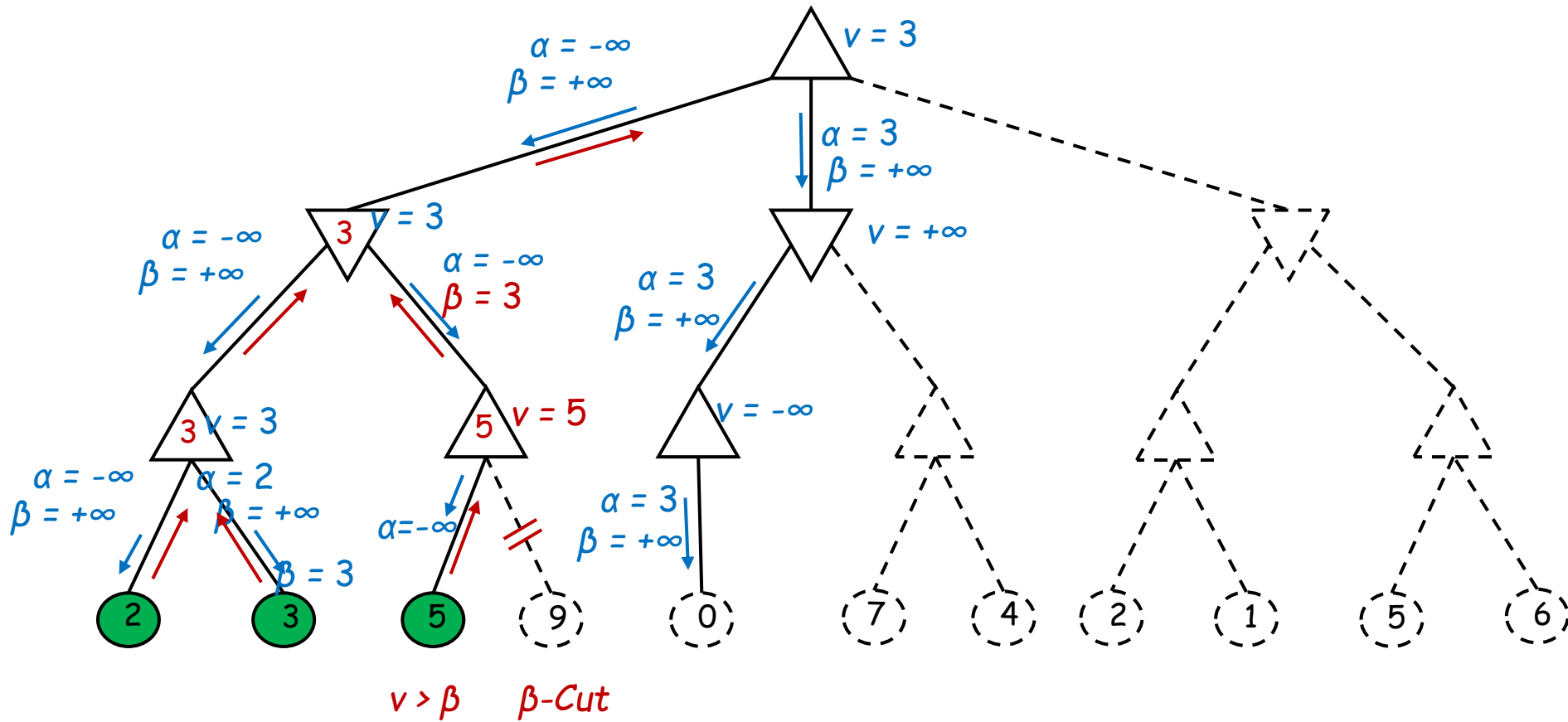
Example



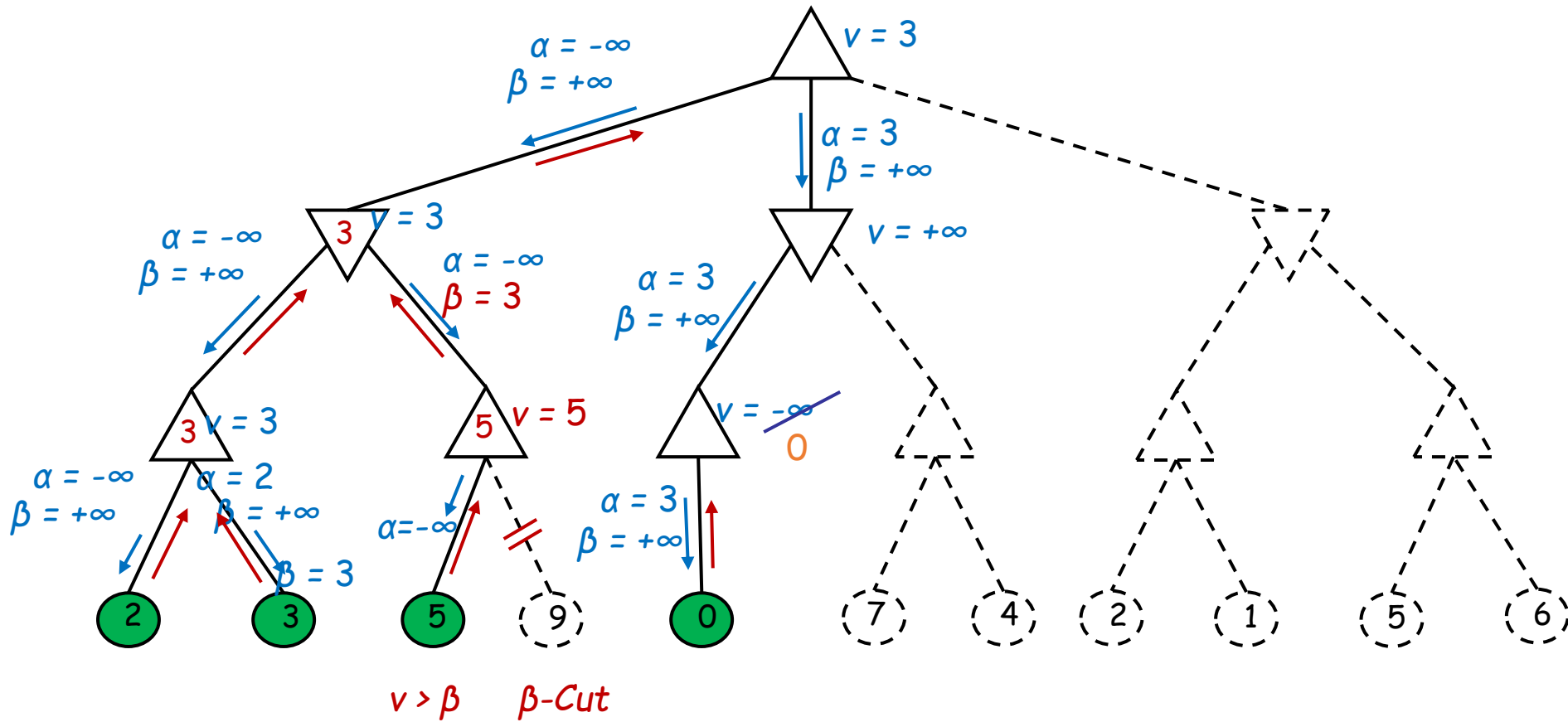
Example



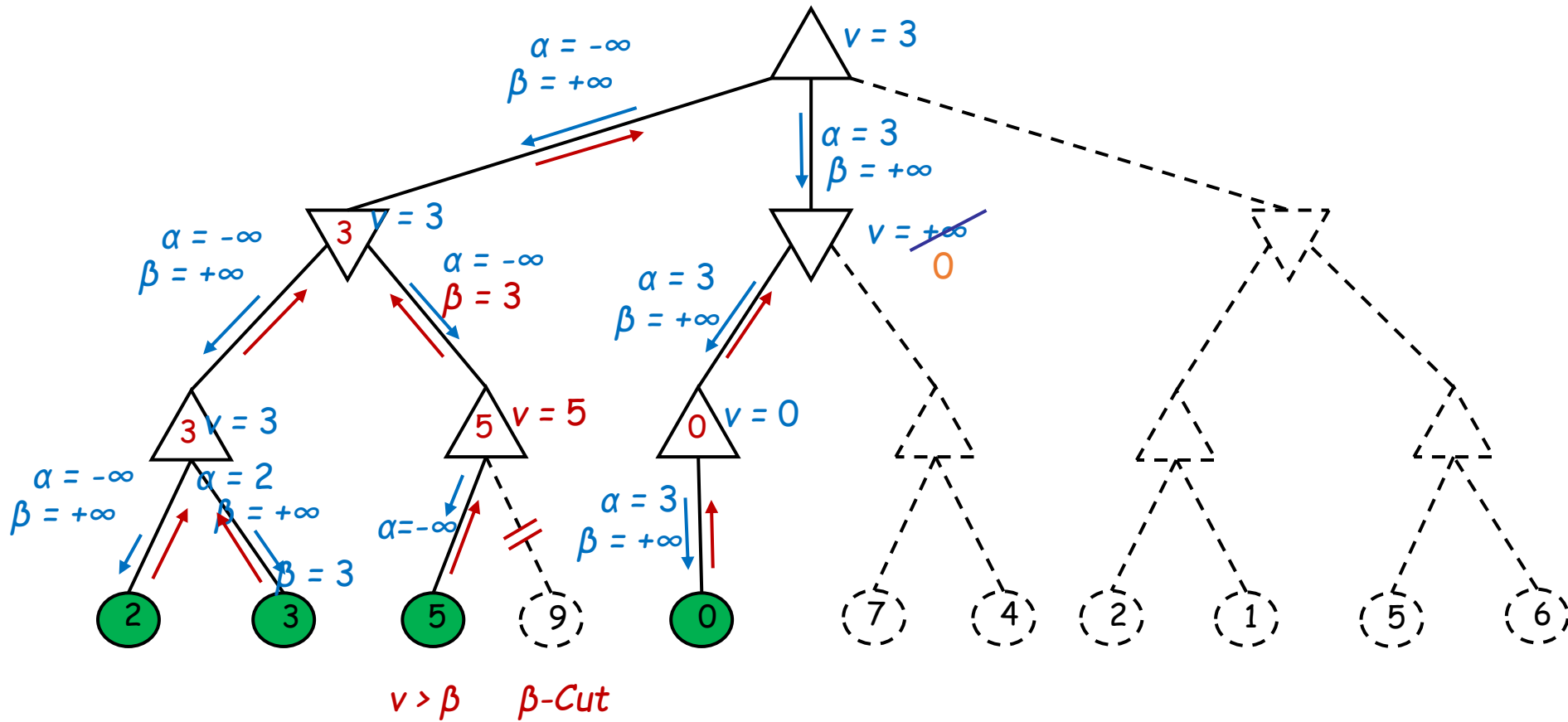
Example



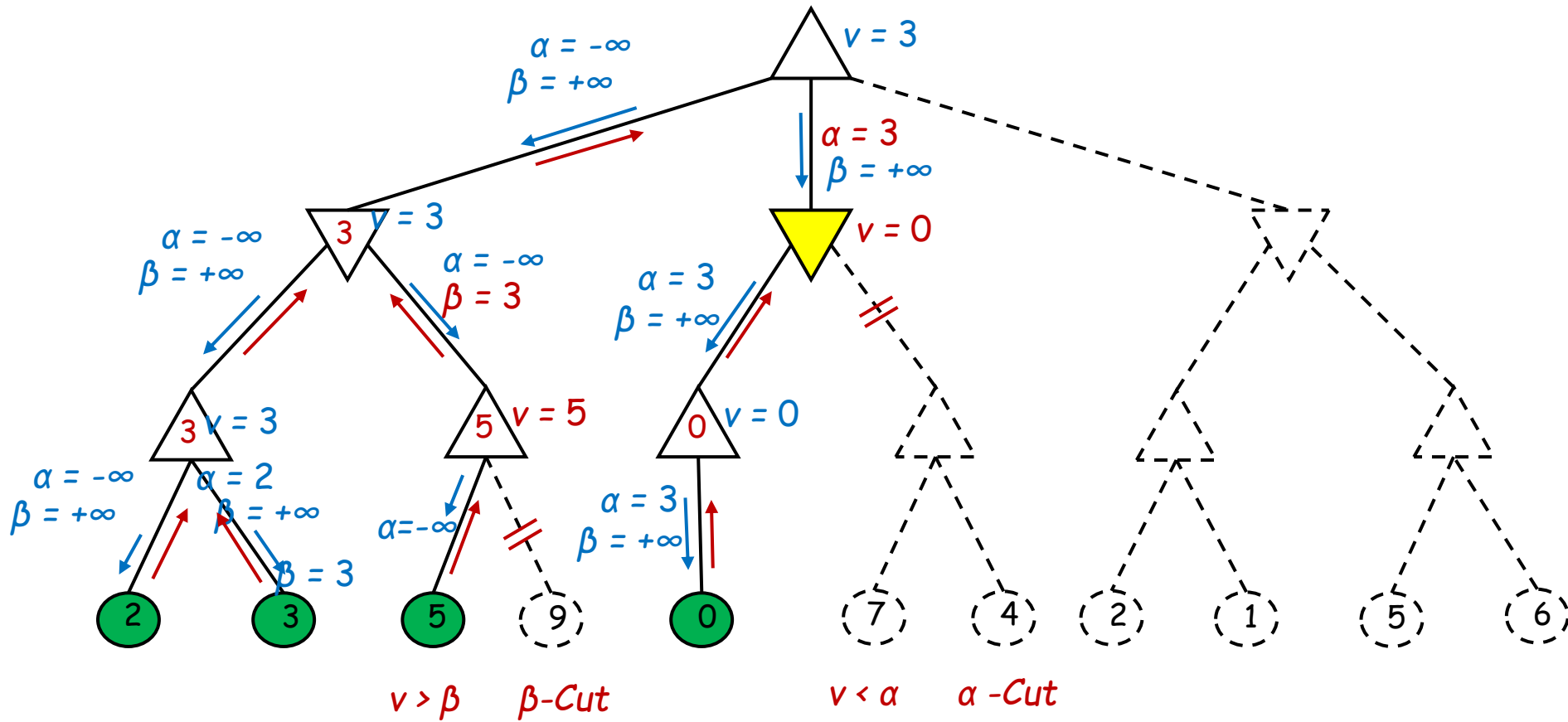
Example



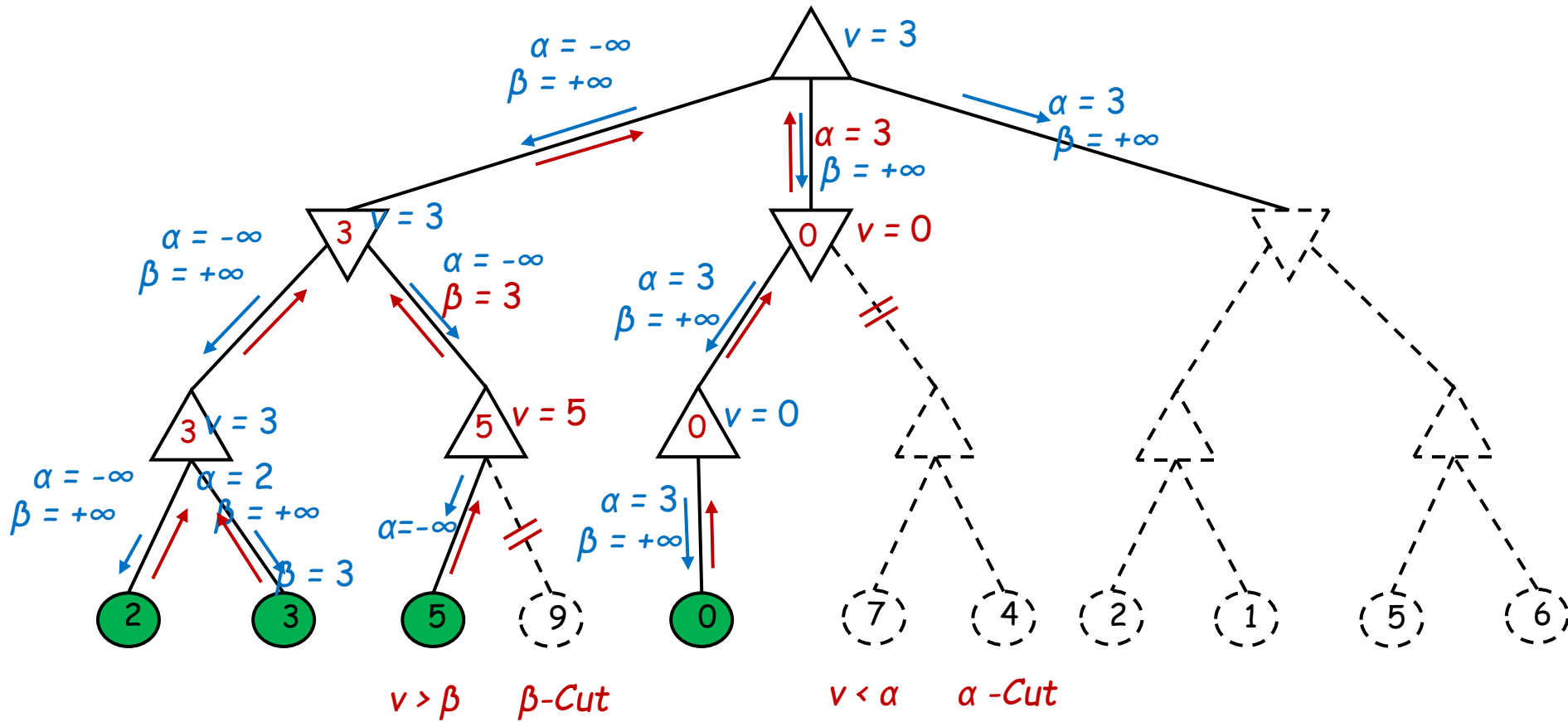
Example



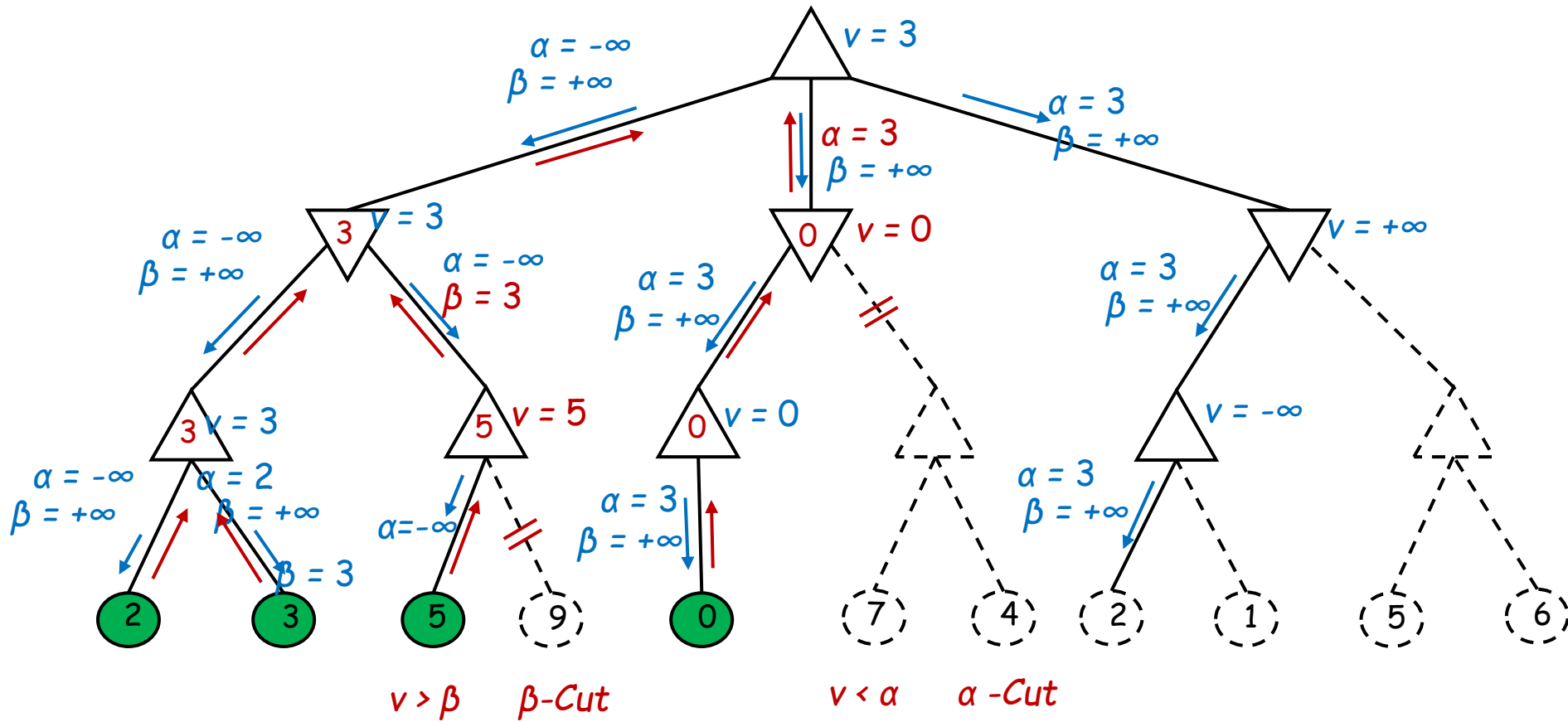
Example



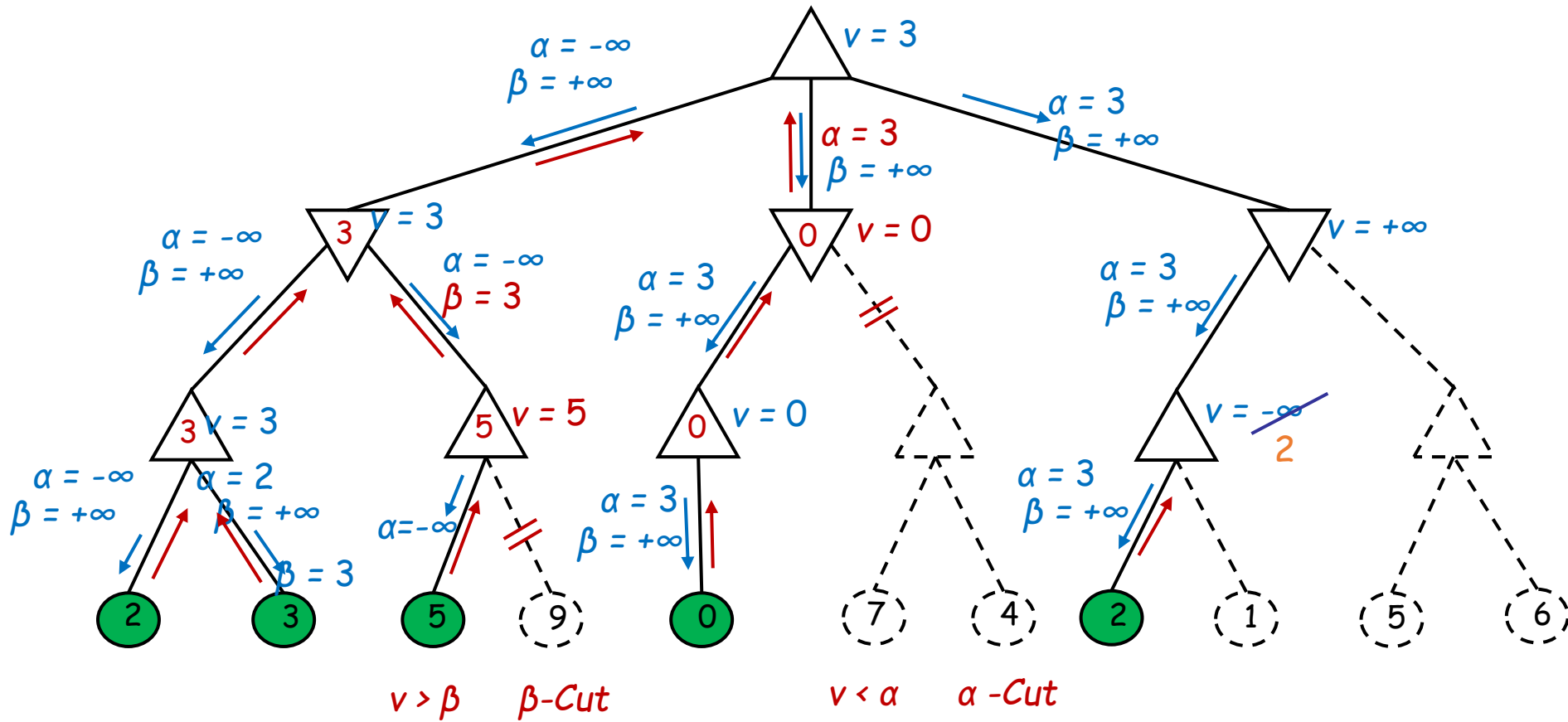
Example



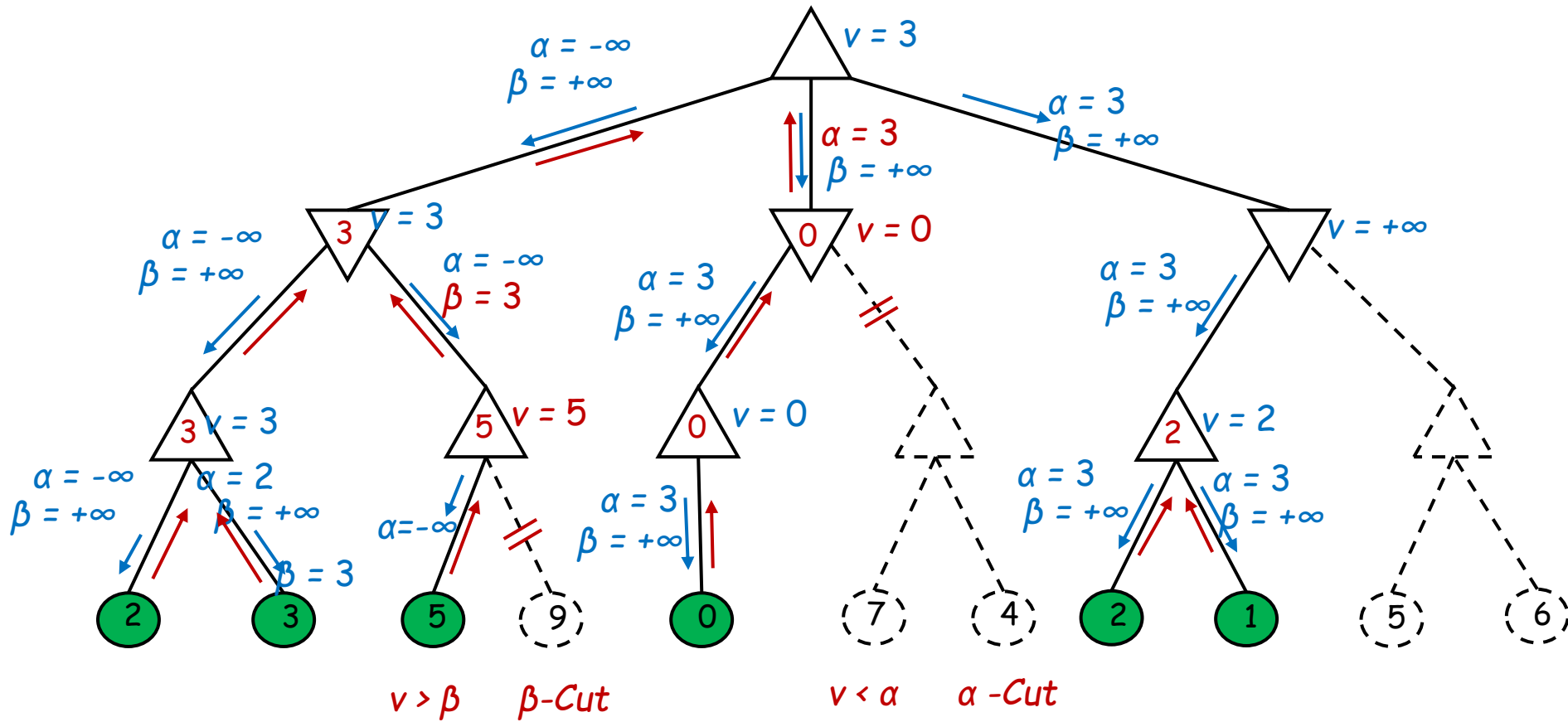
Example



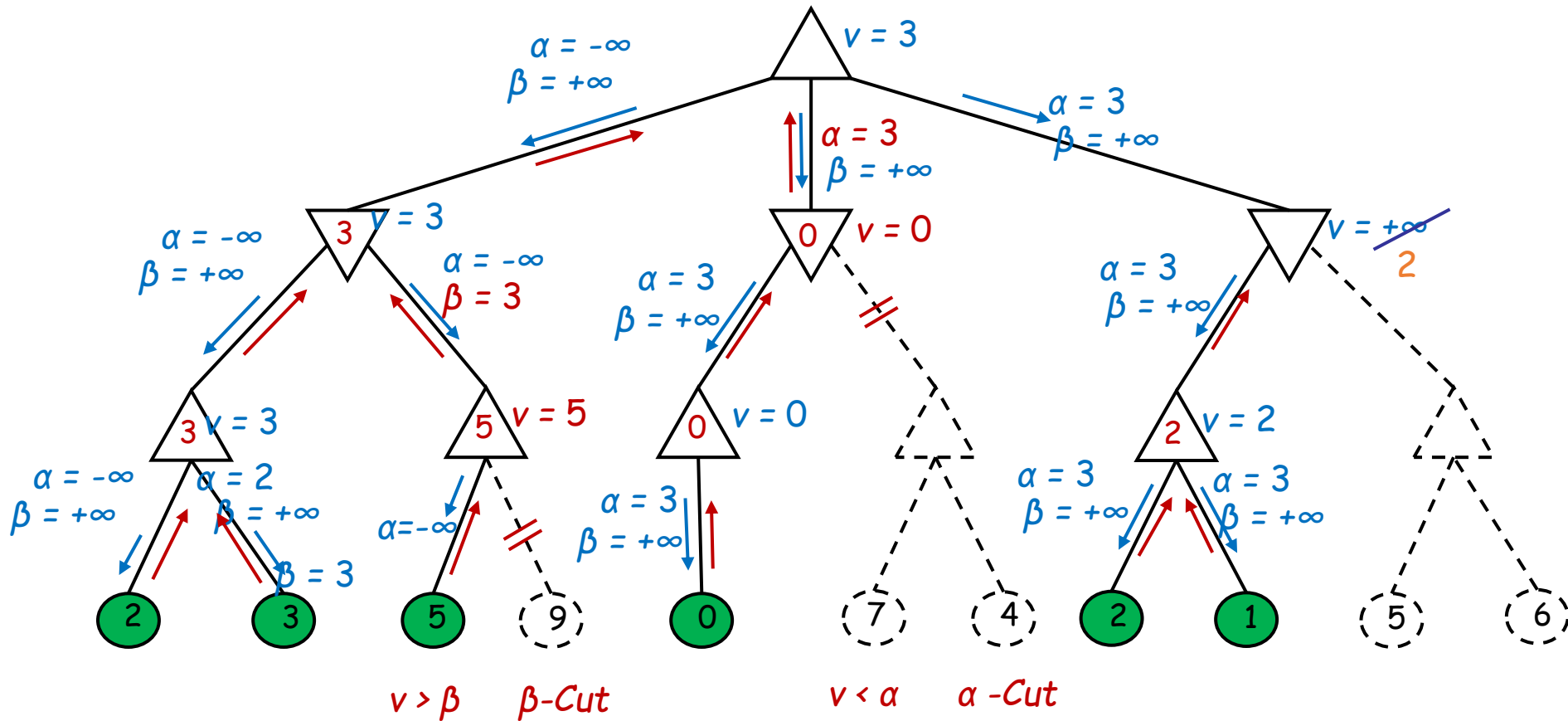
Example



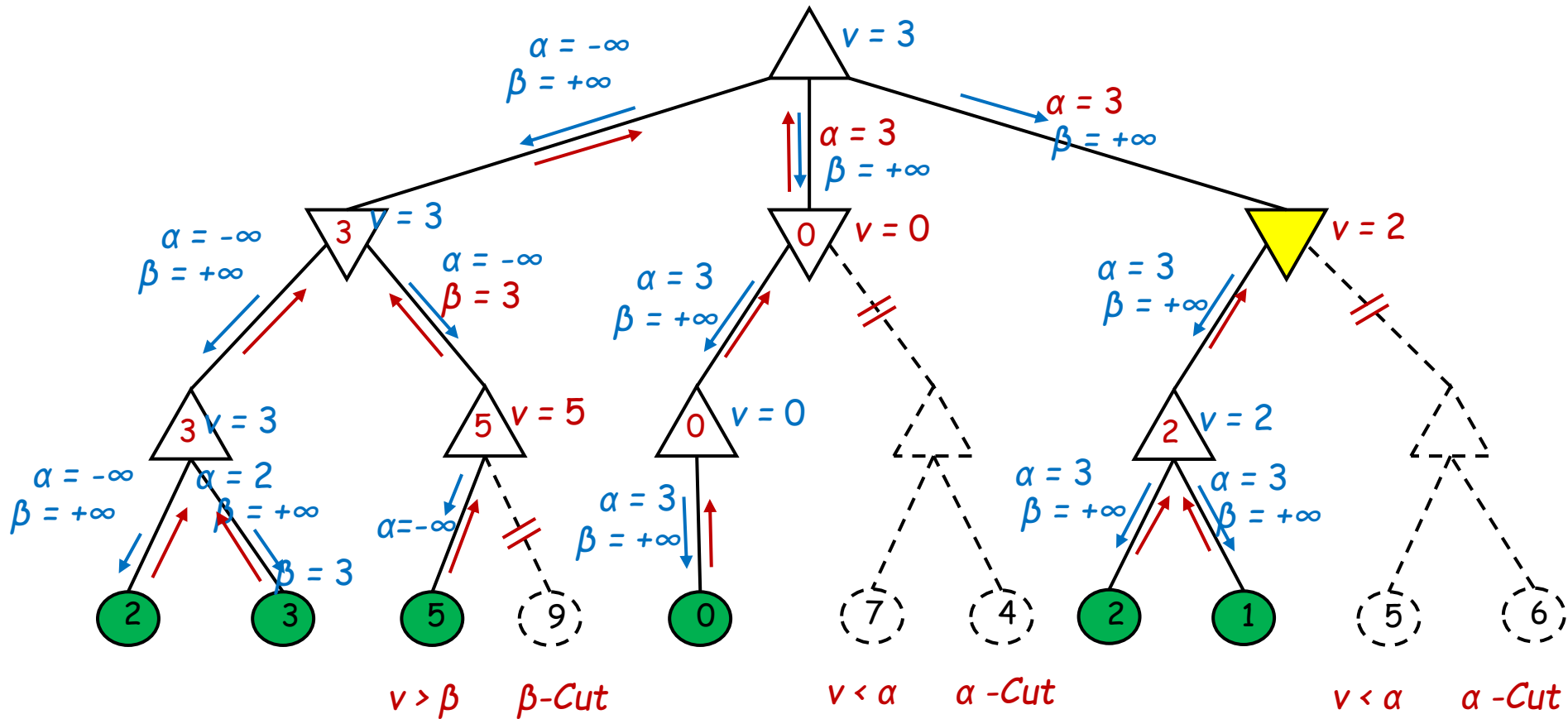
Example



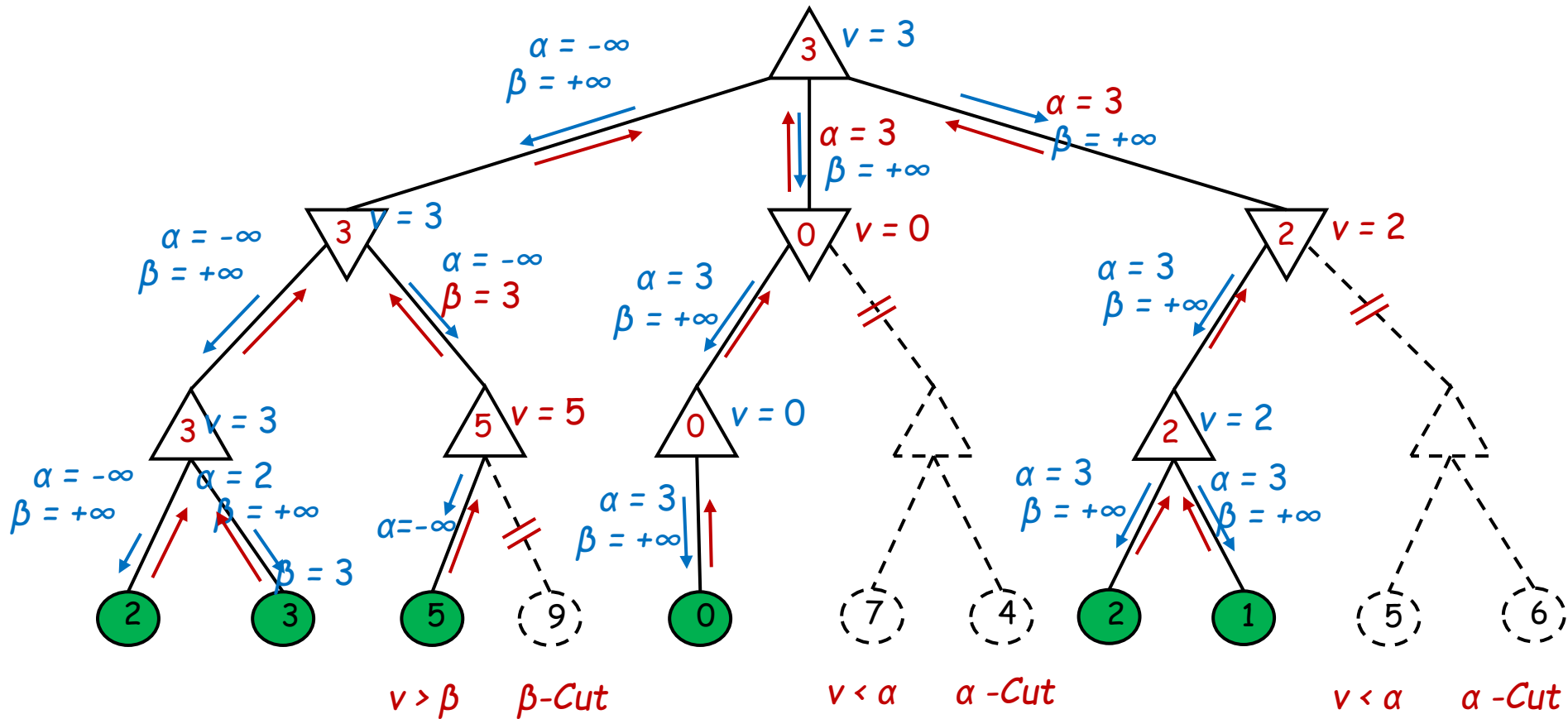
Example



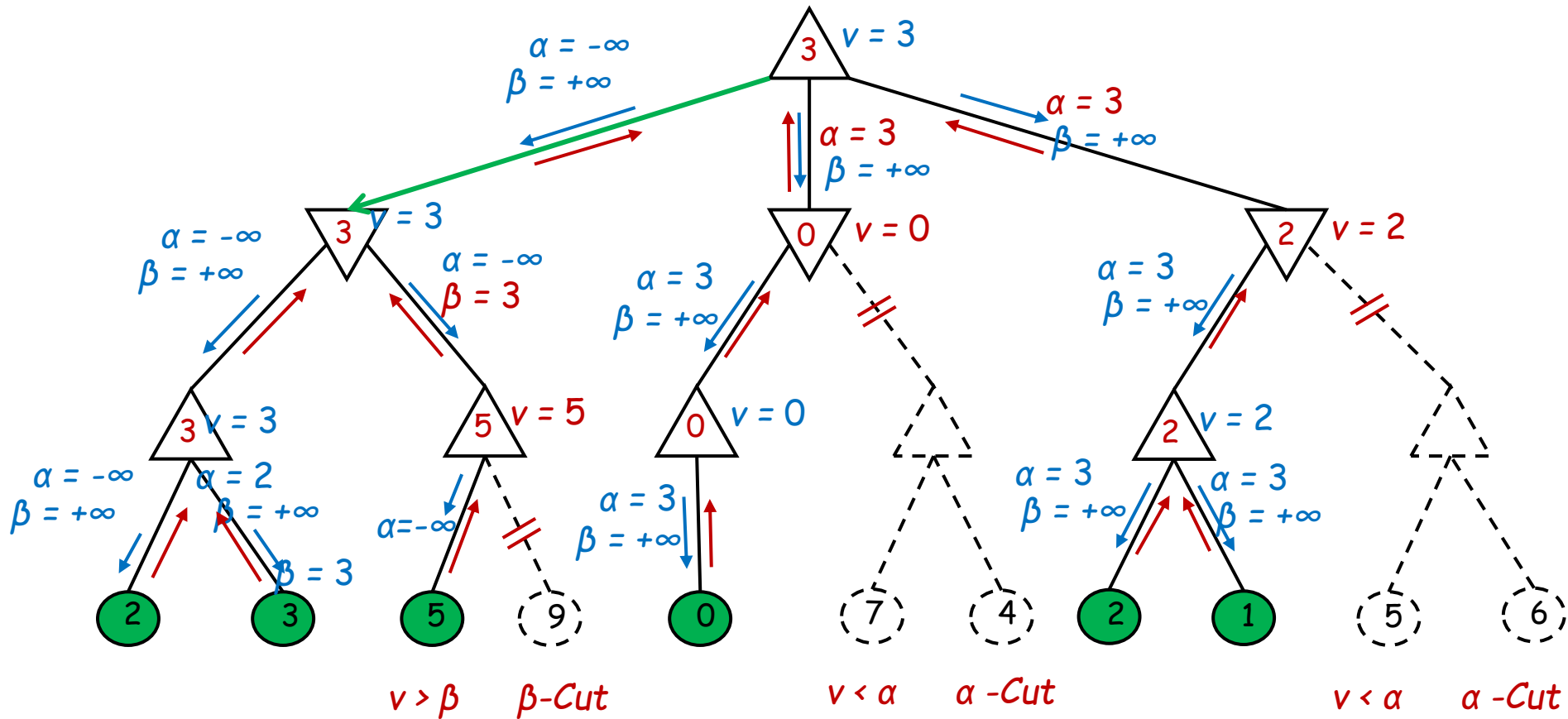
Example



Example

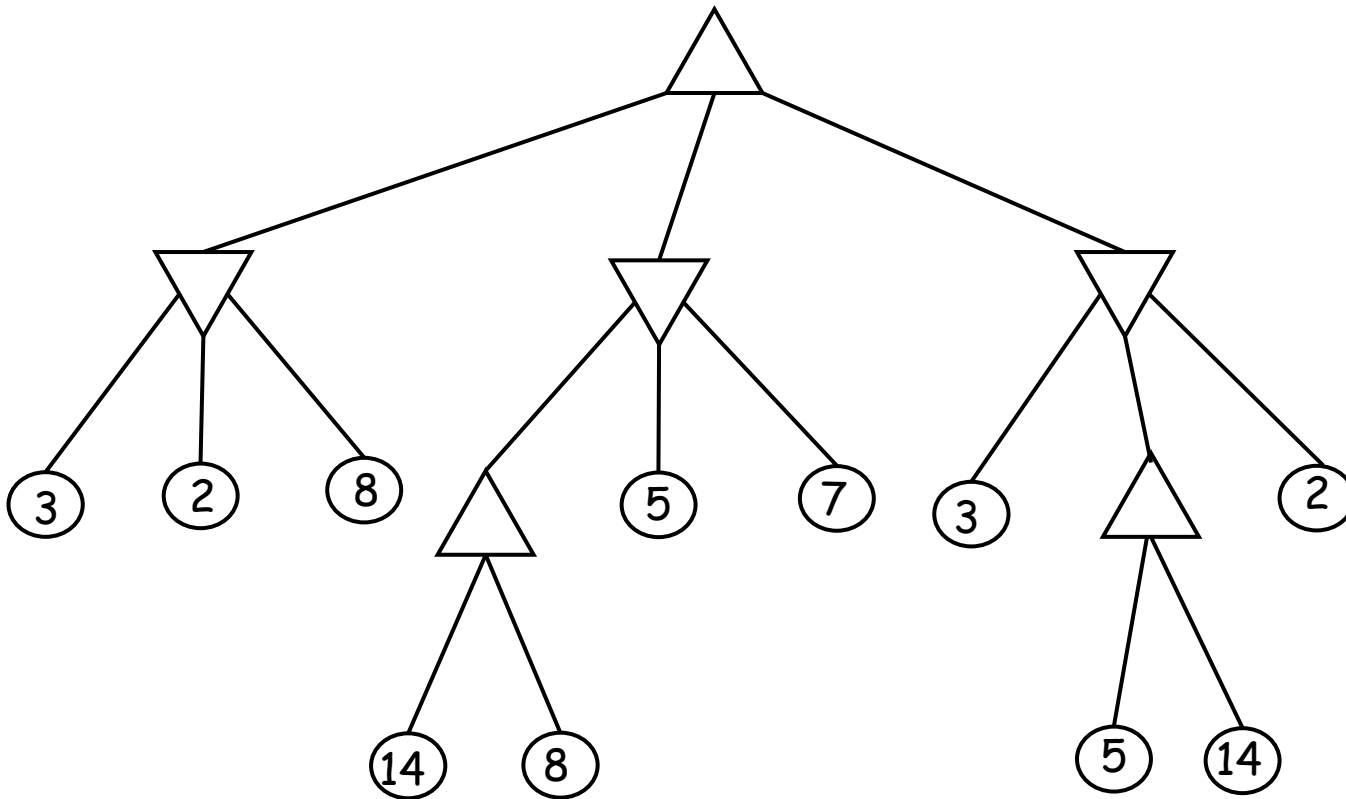


Example



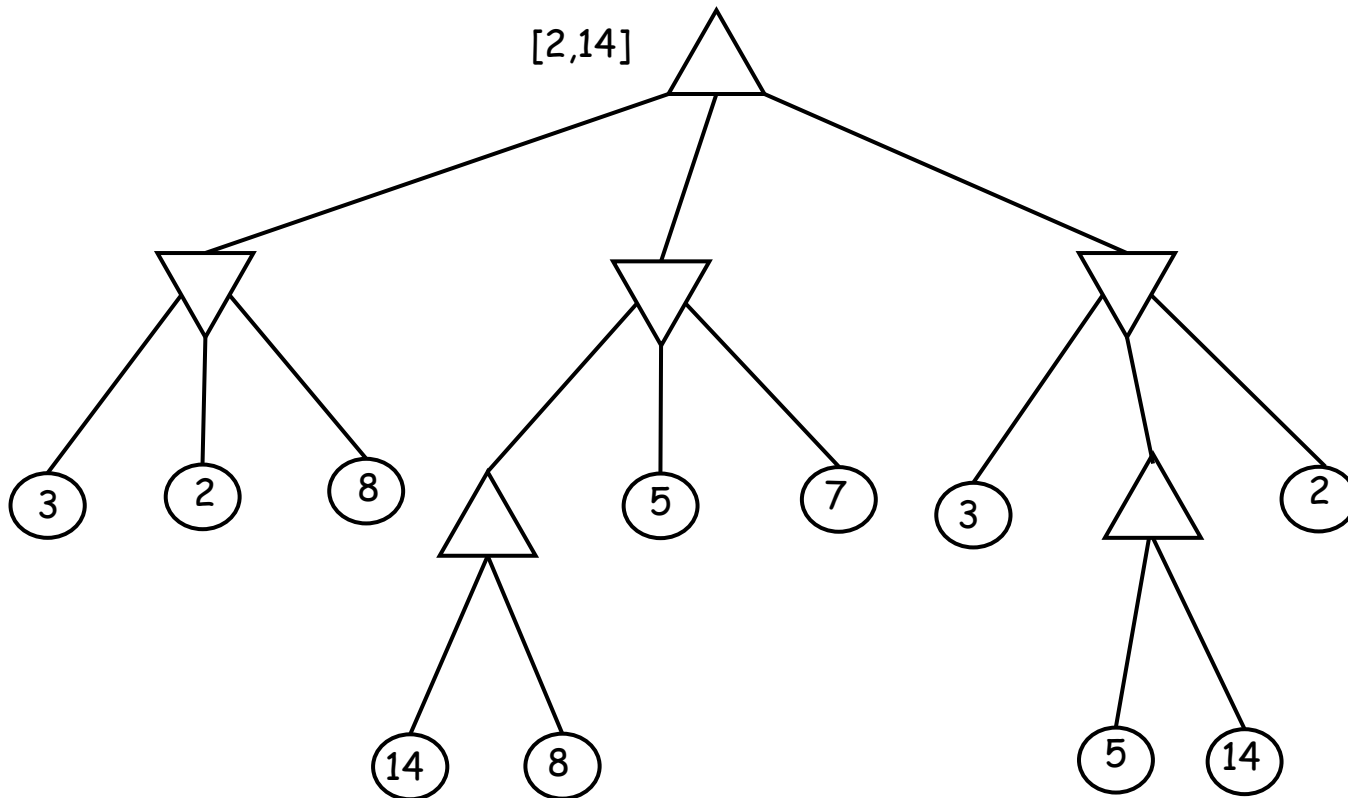
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



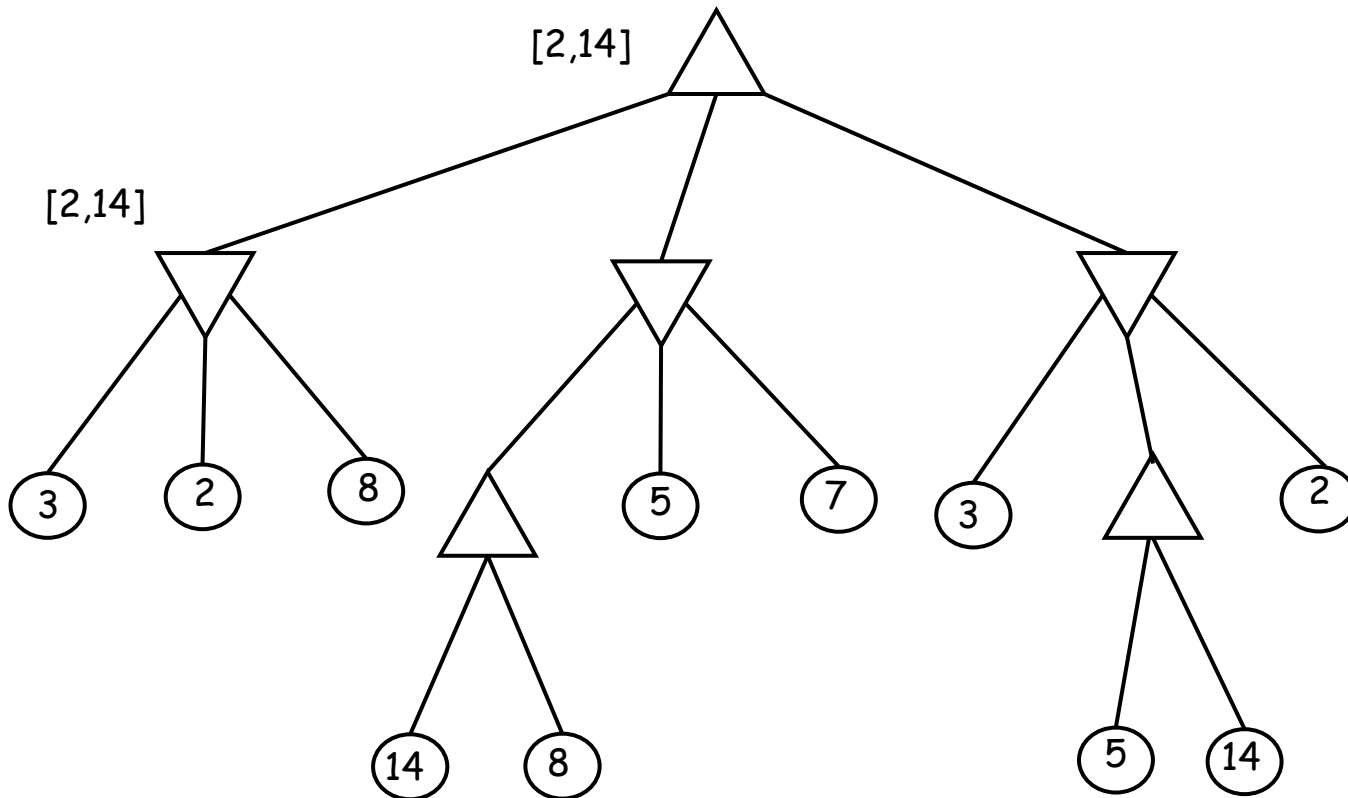
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



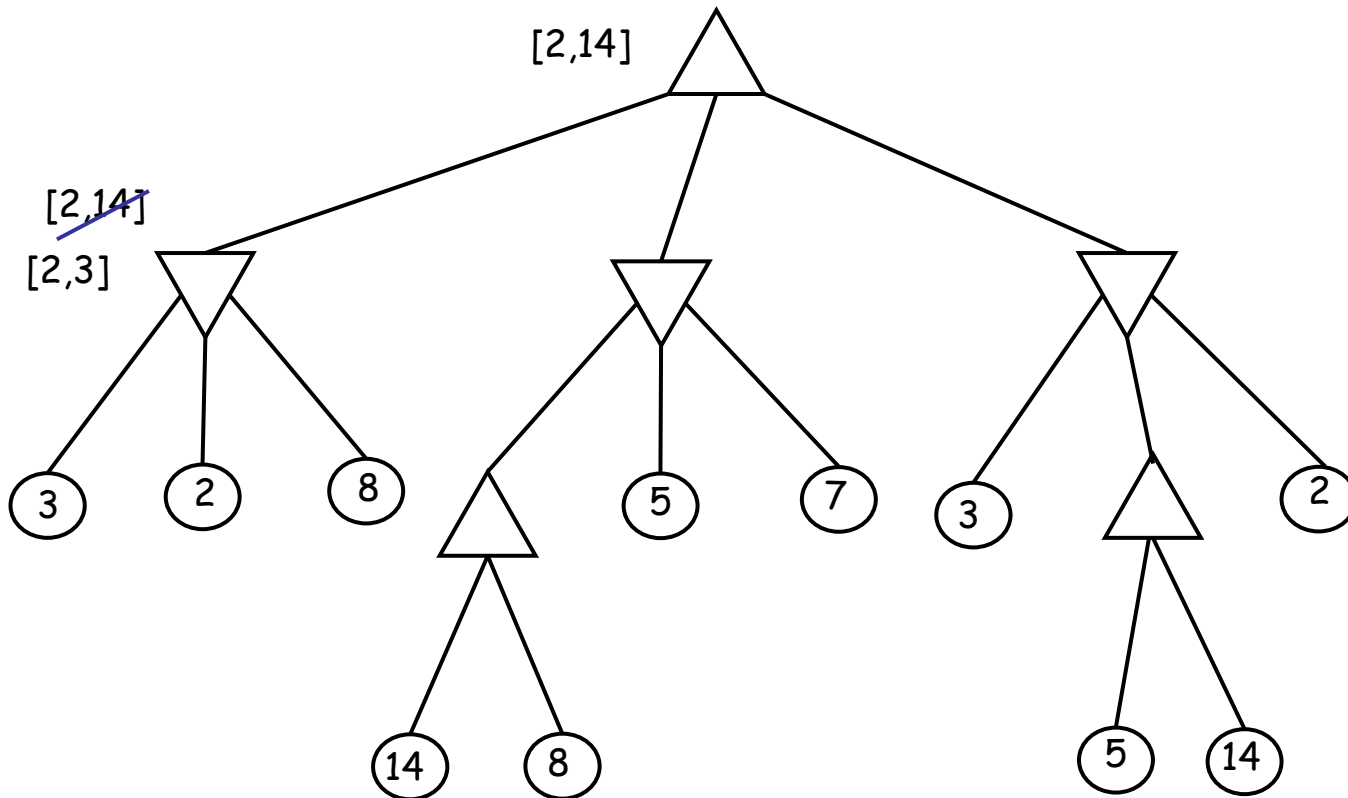
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



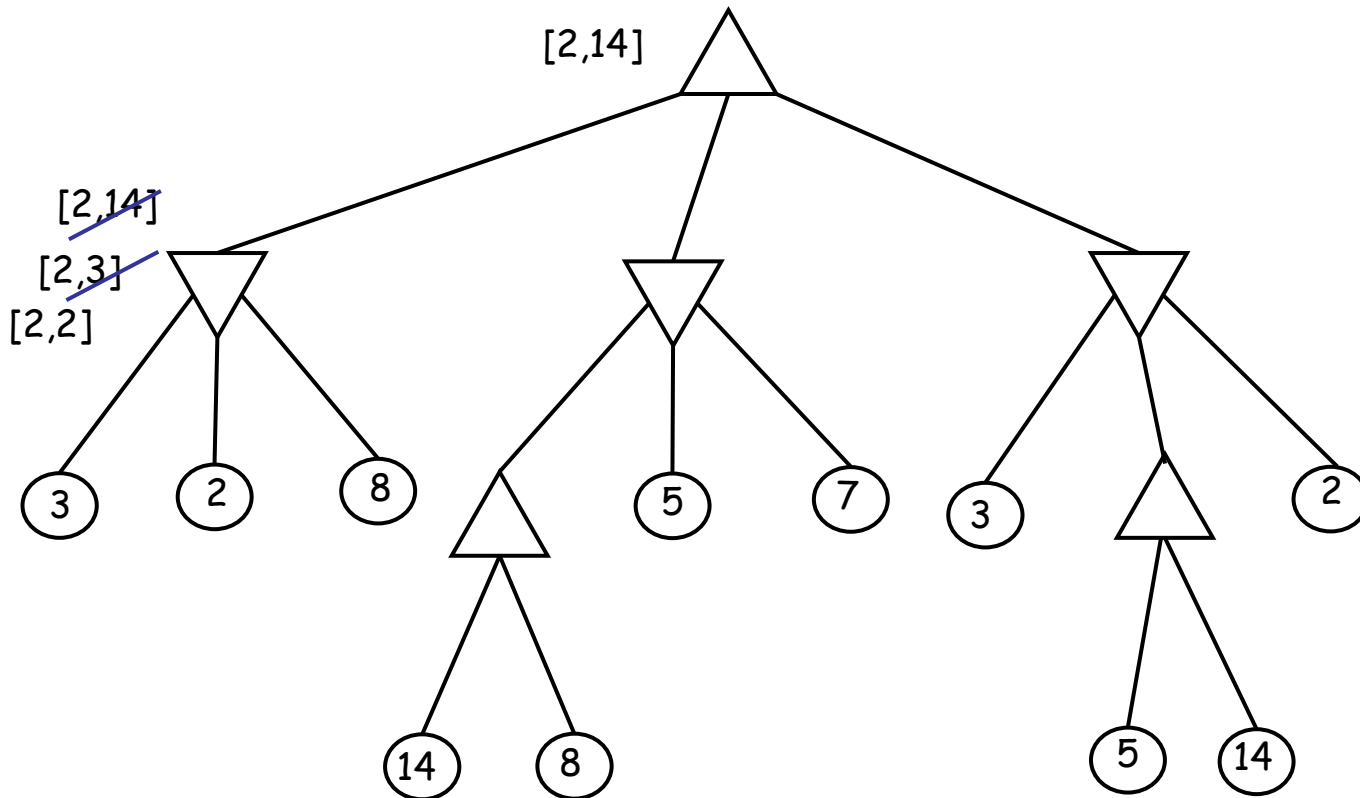
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



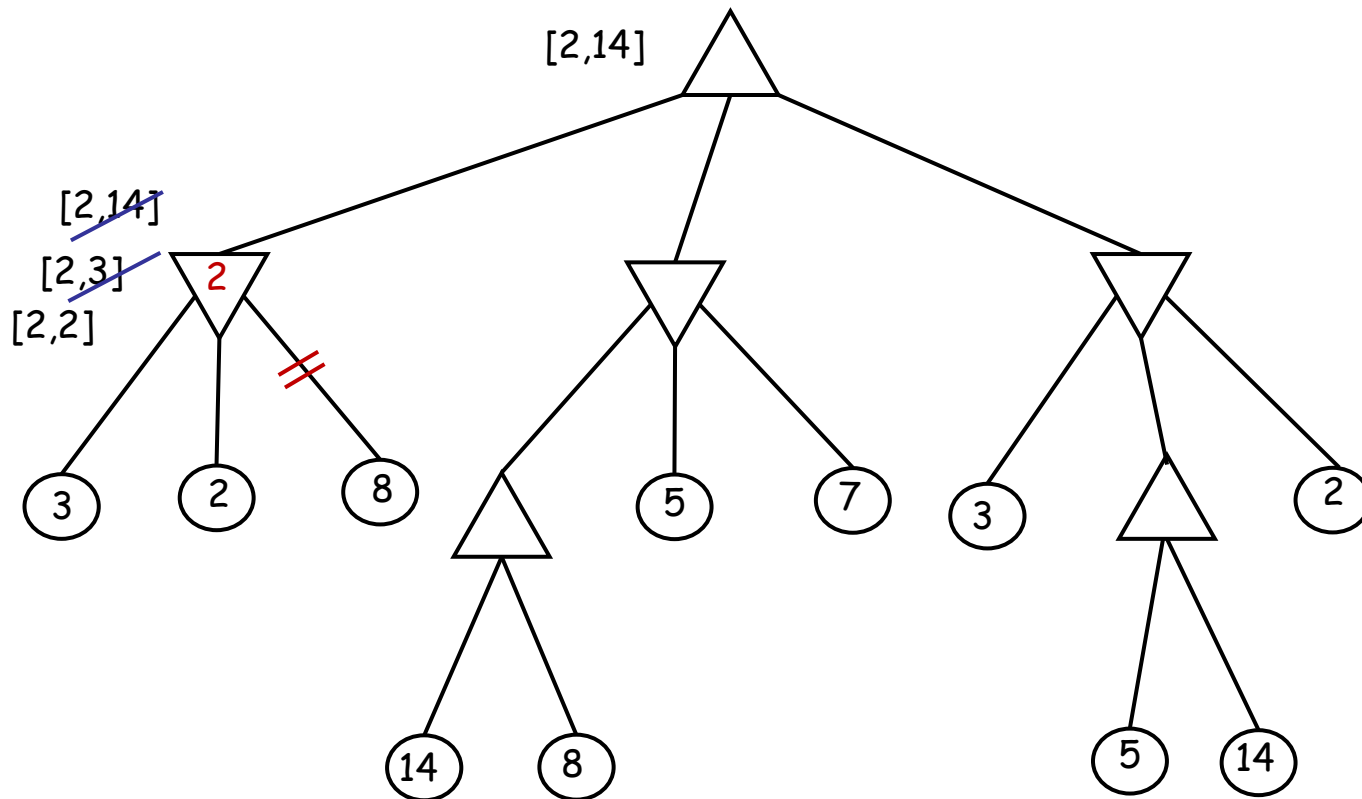
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



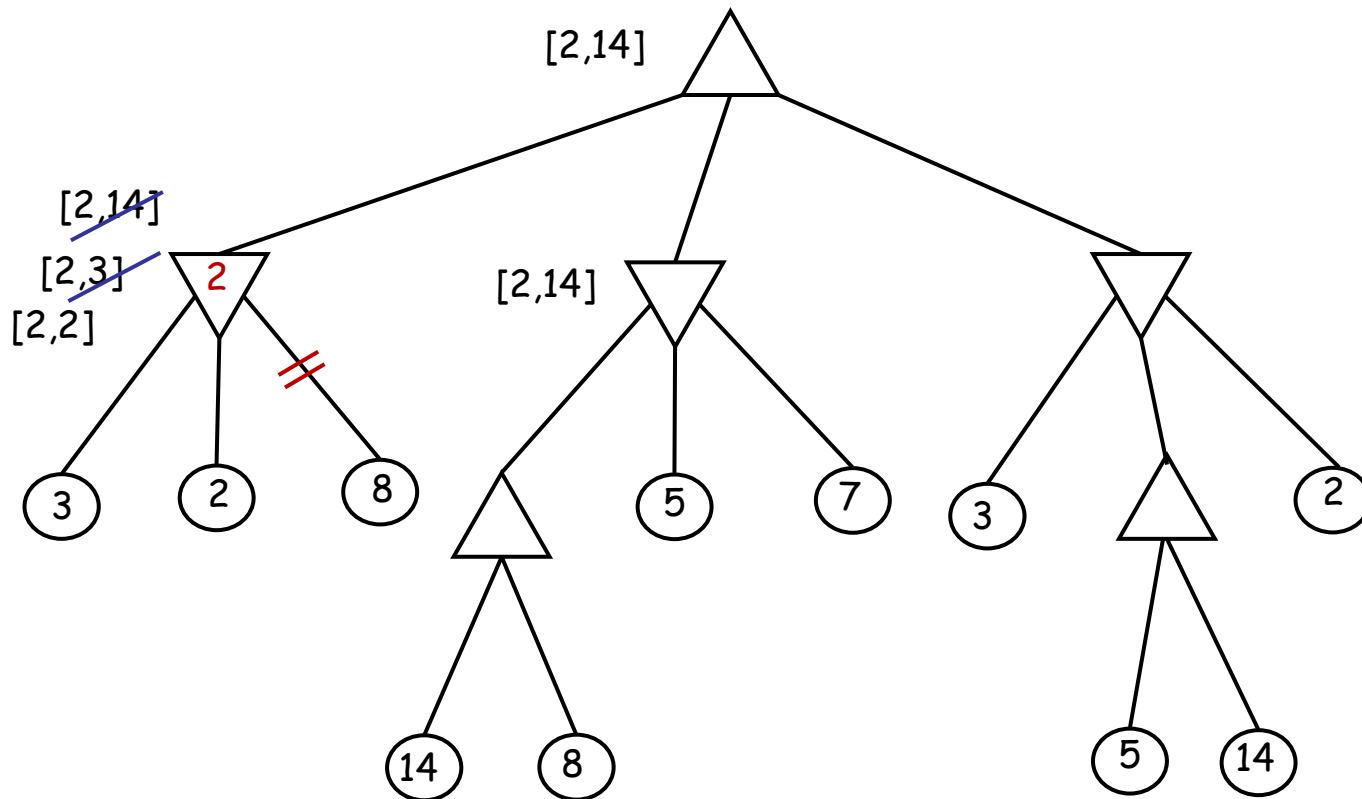
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



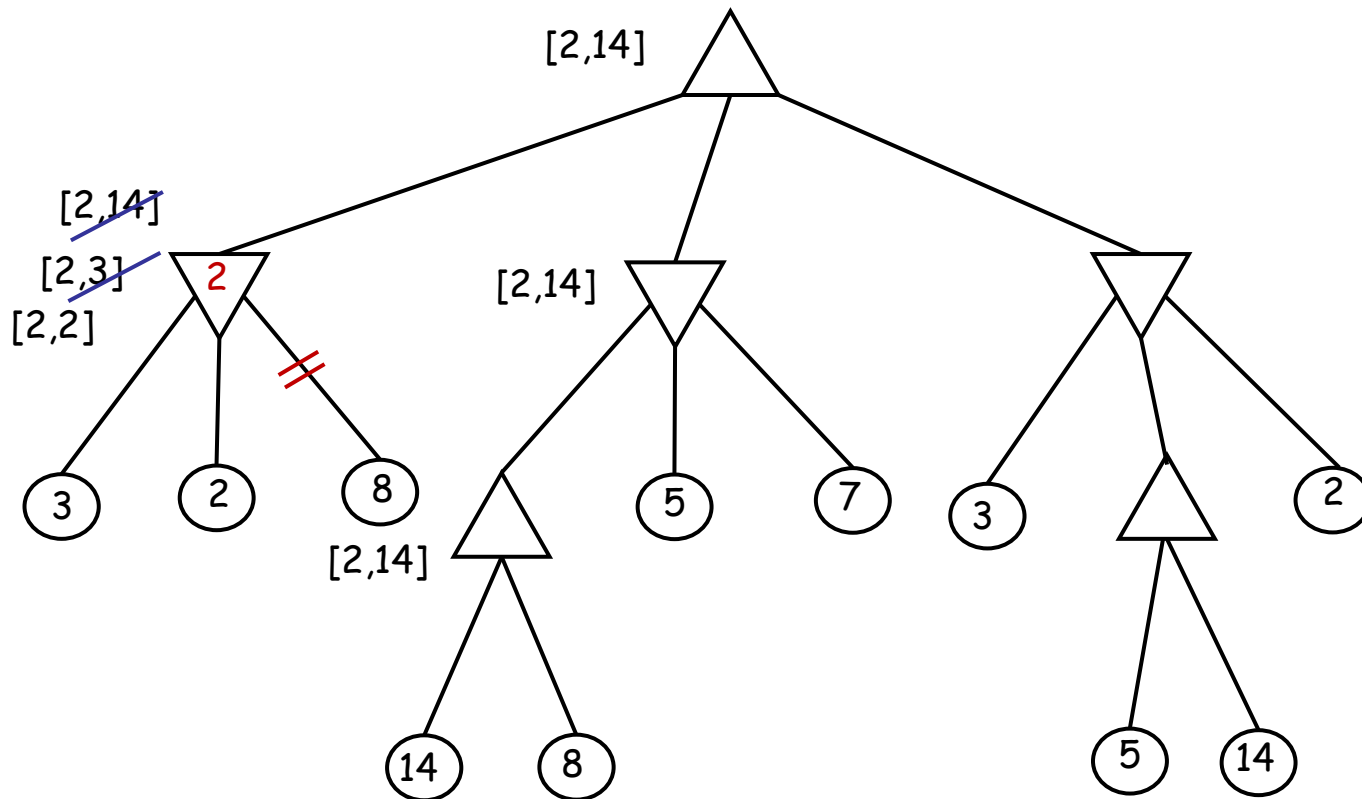
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



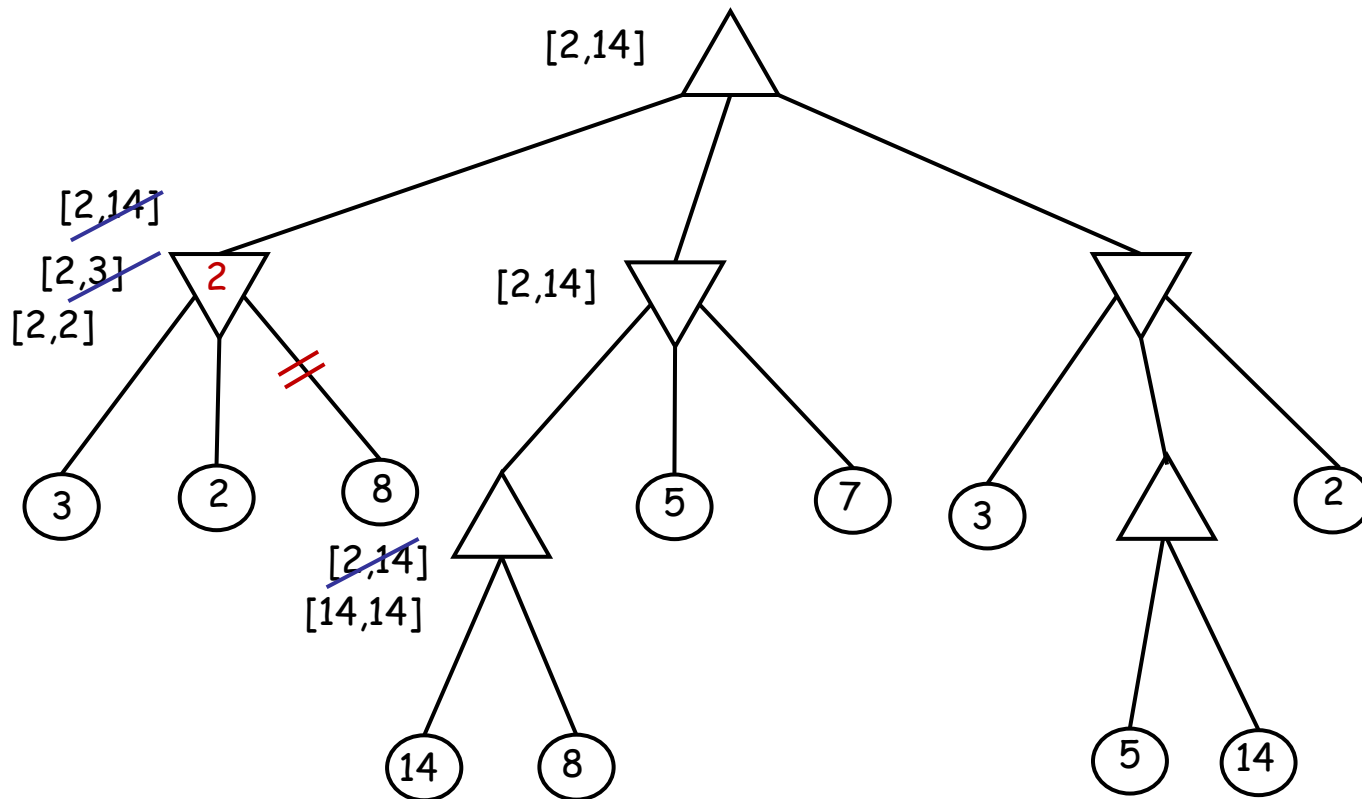
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



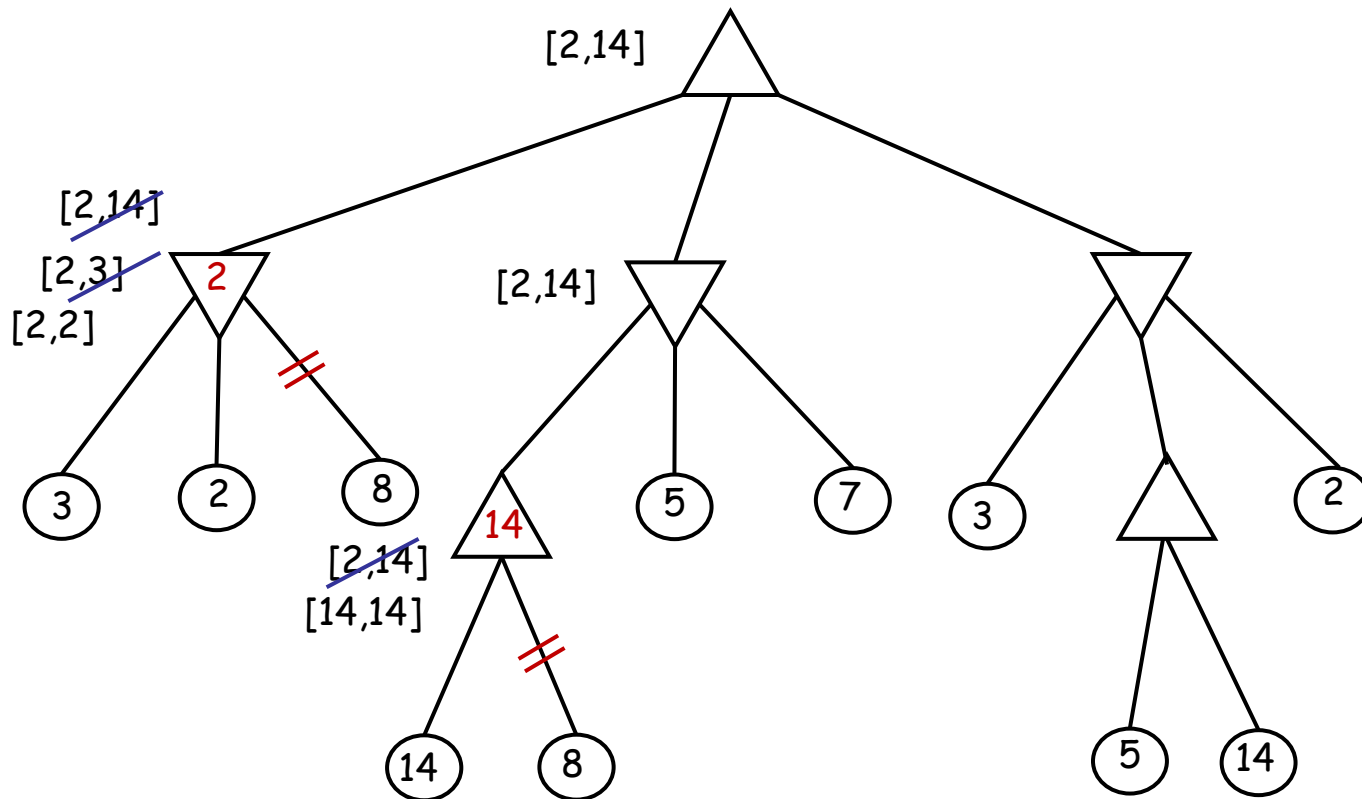
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



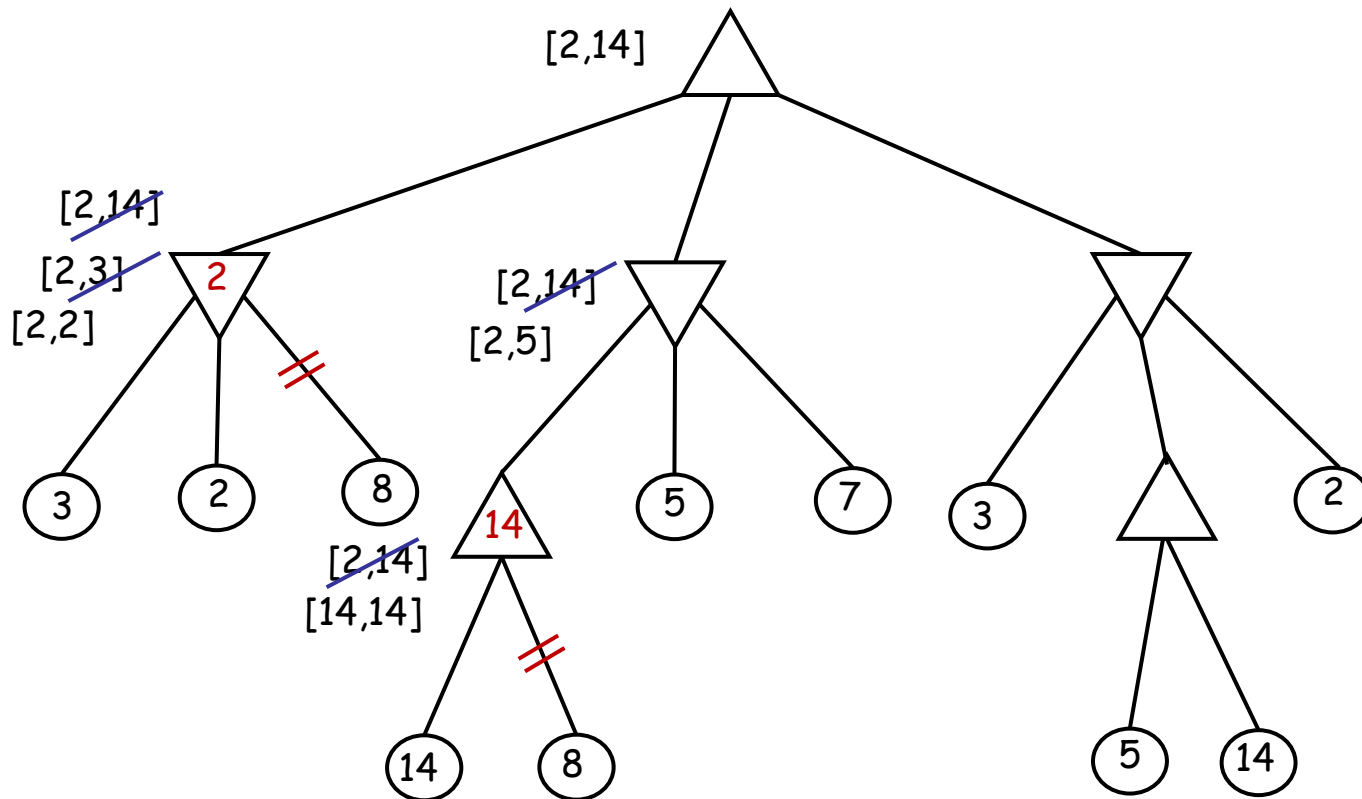
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



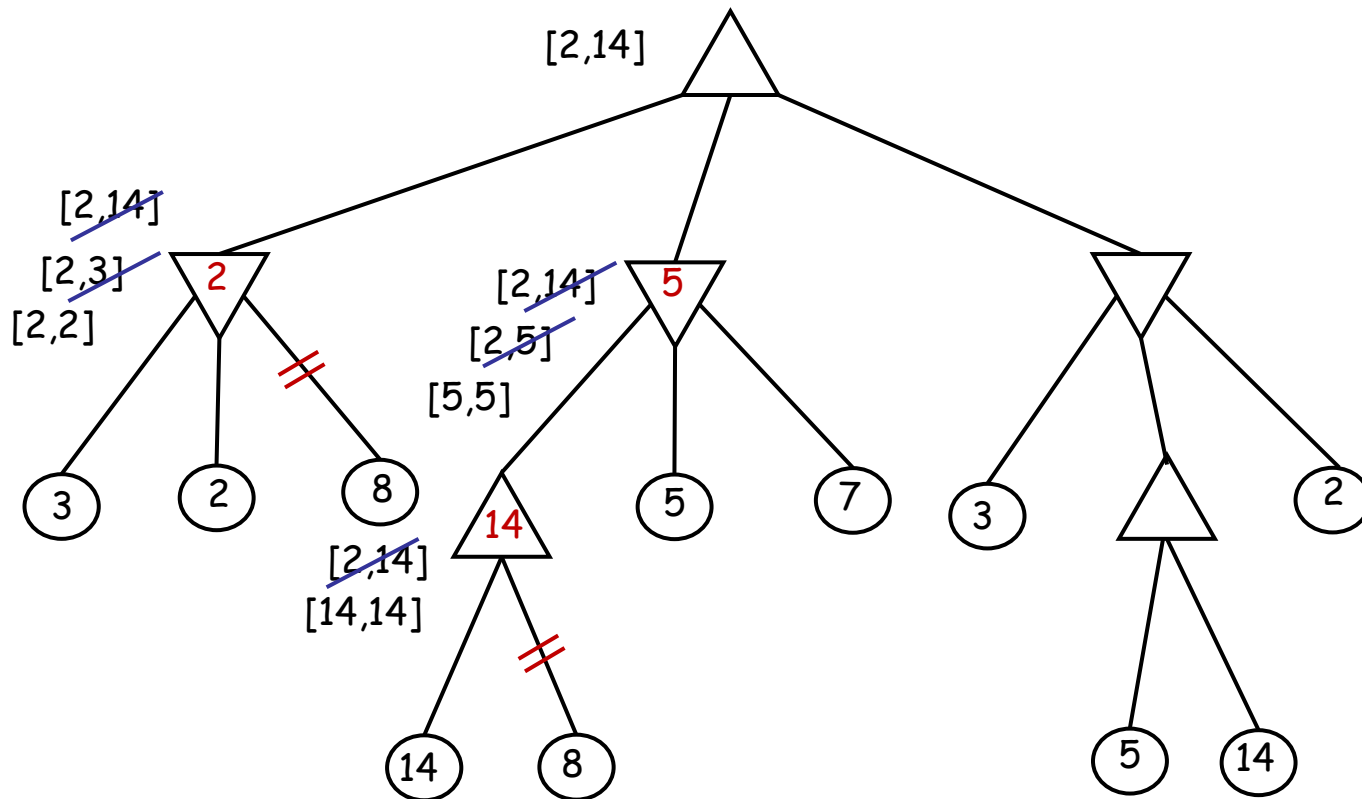
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



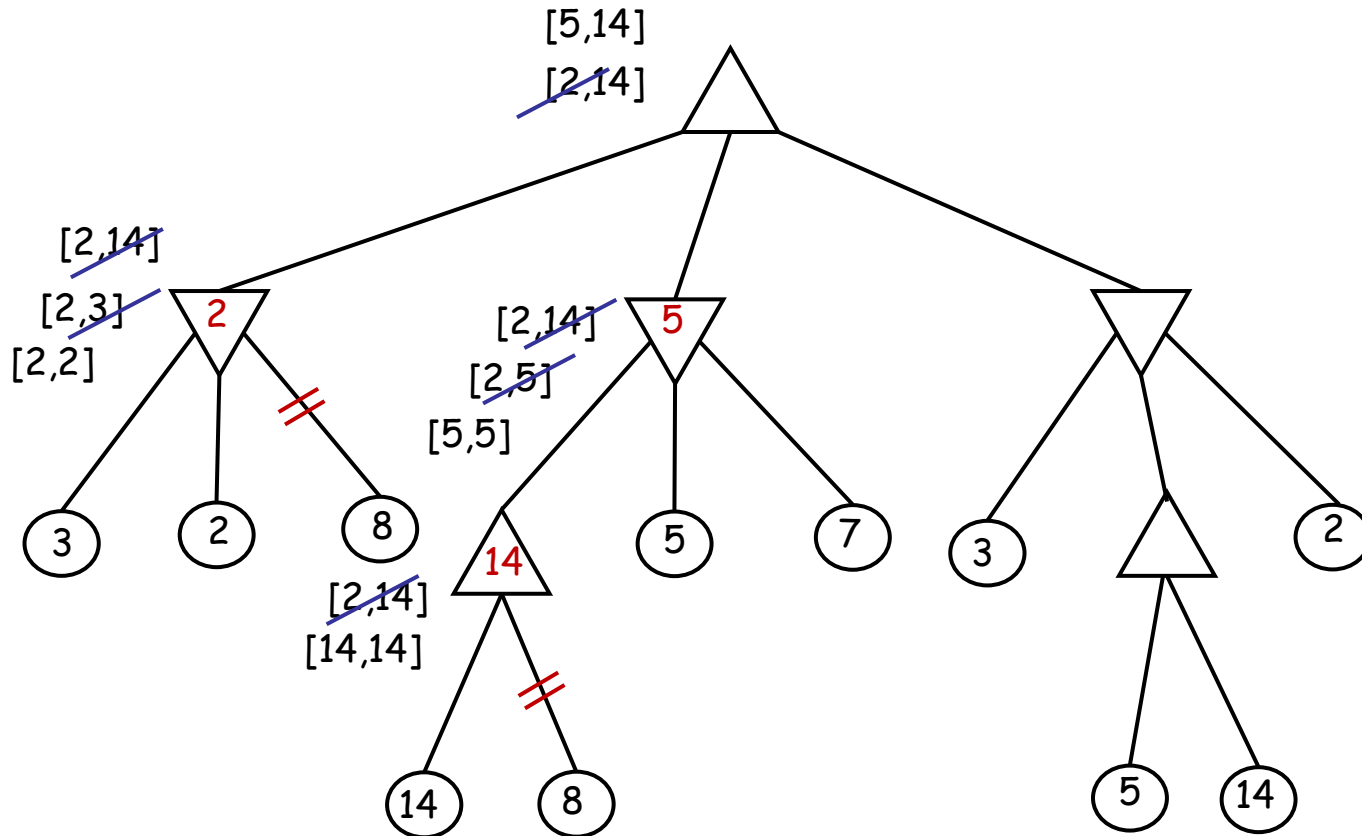
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



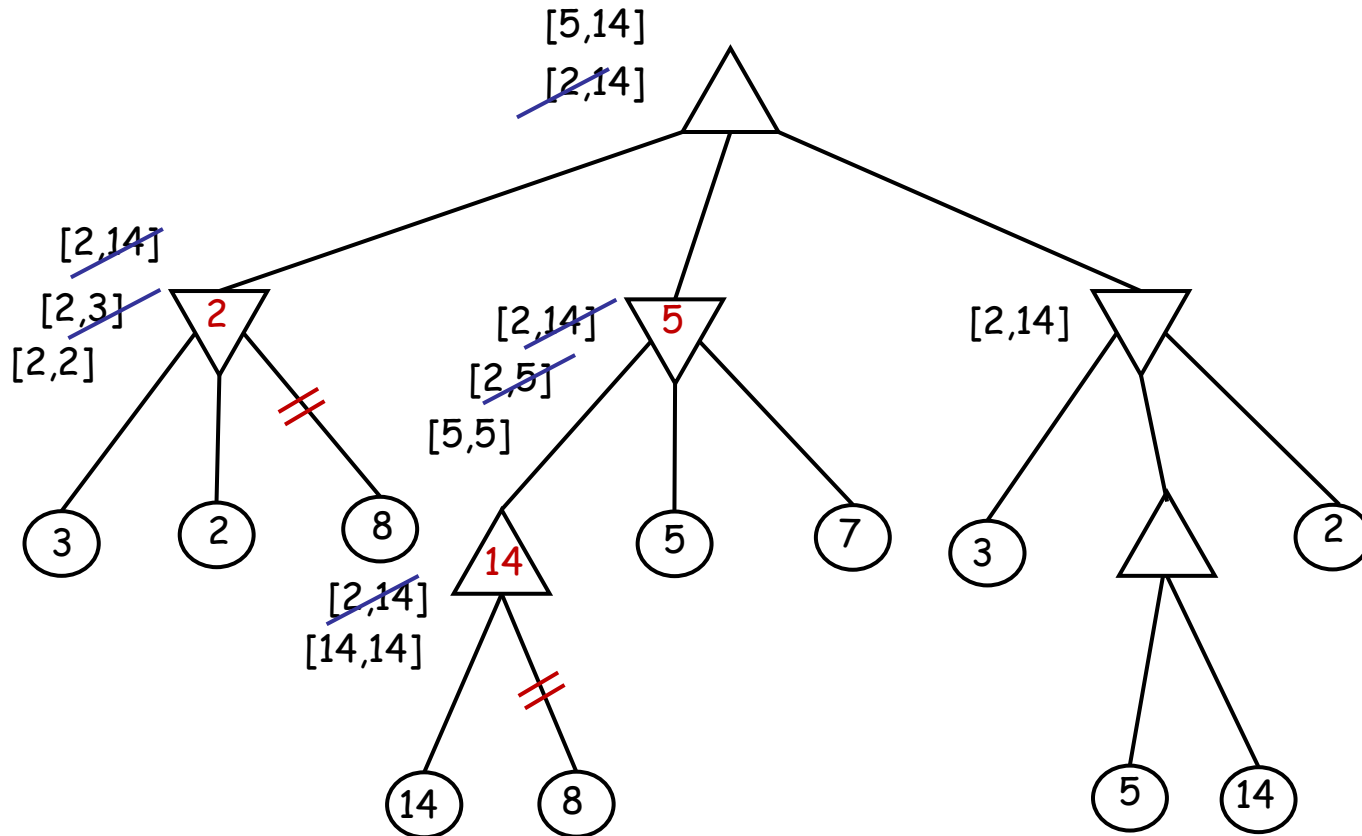
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



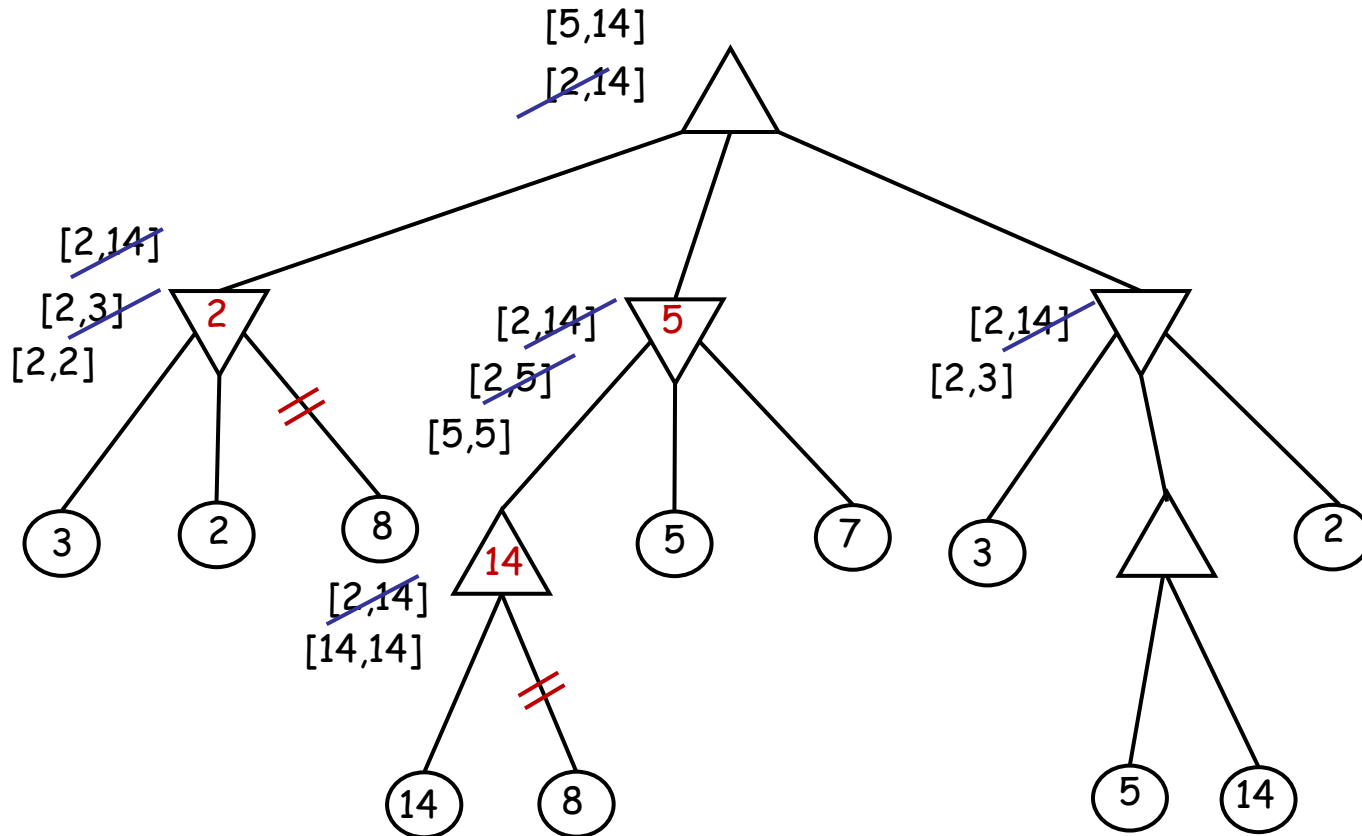
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



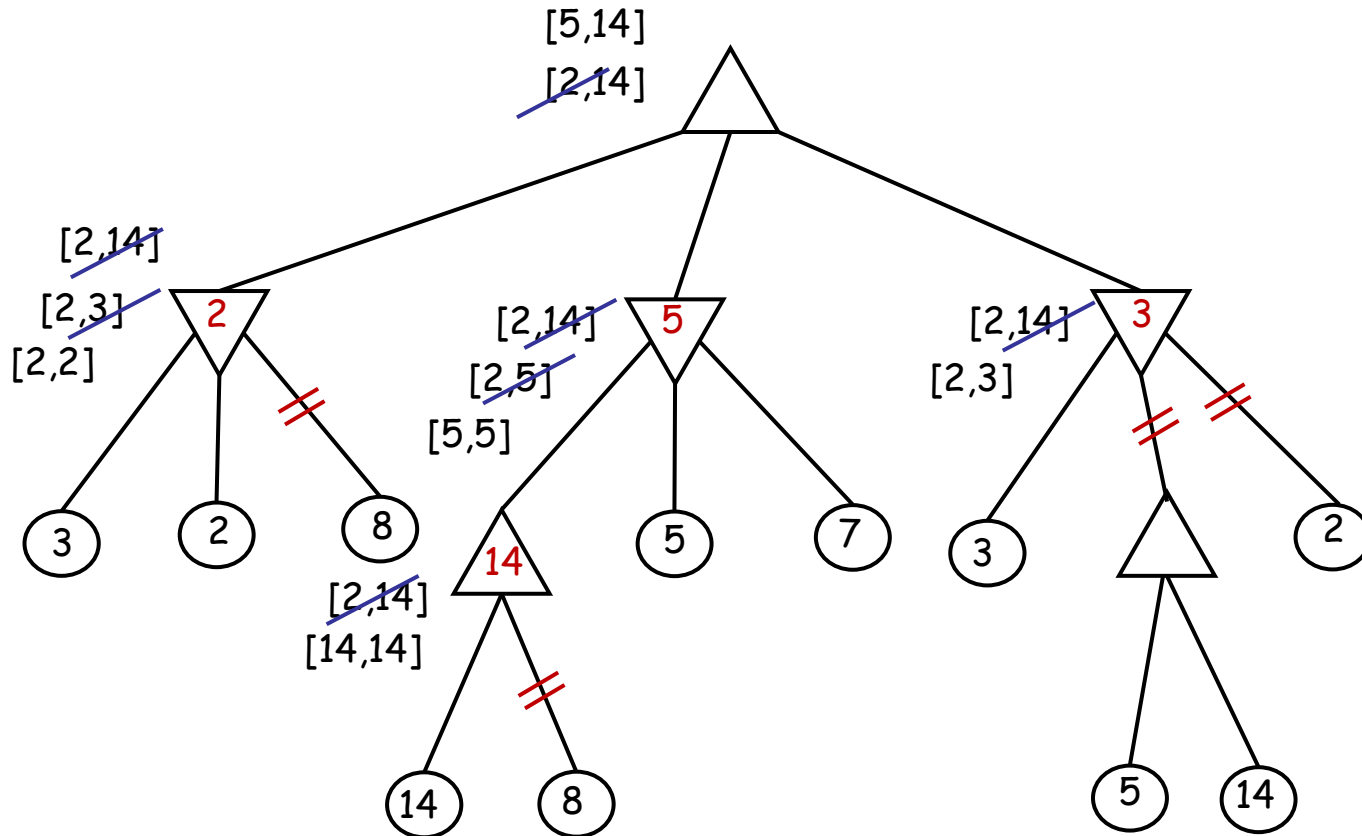
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



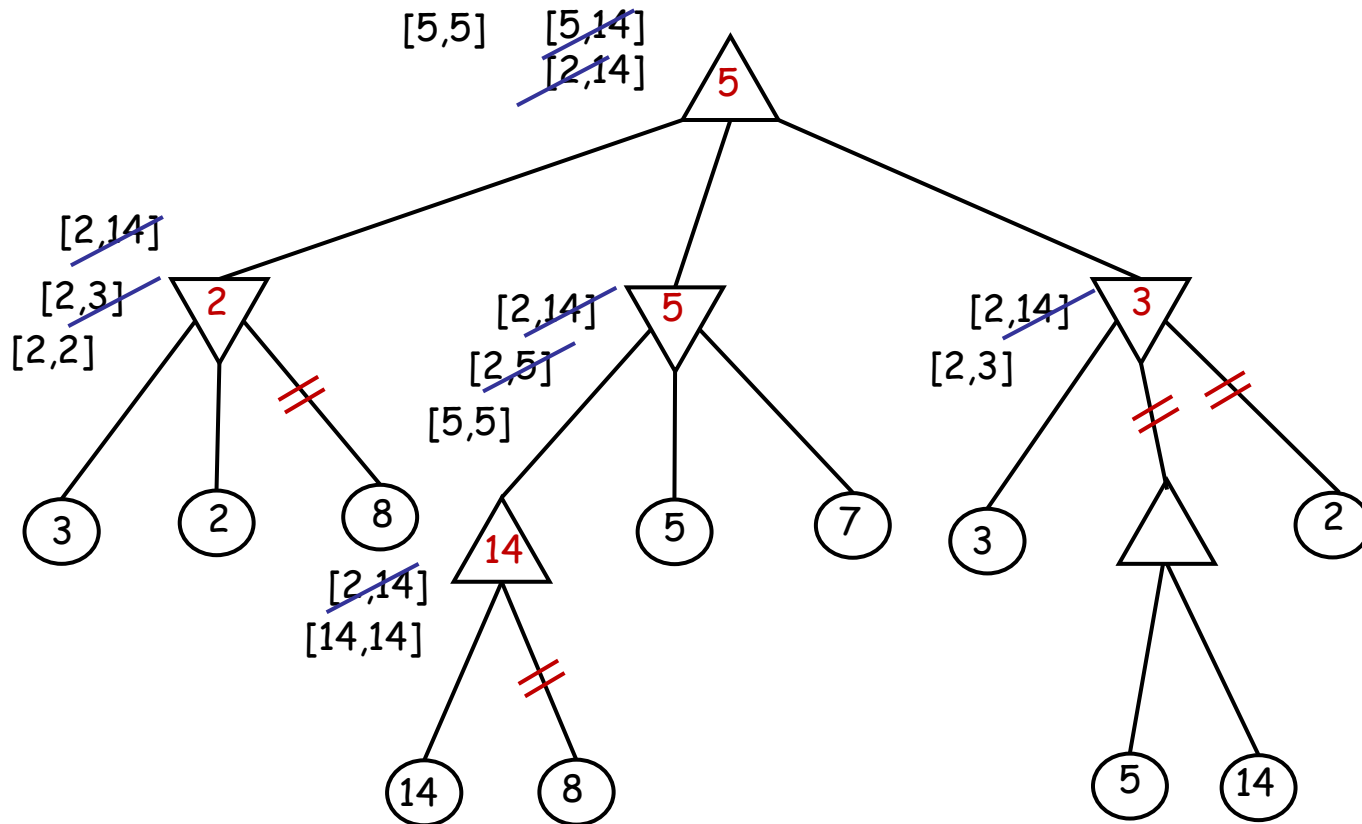
Example

- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example

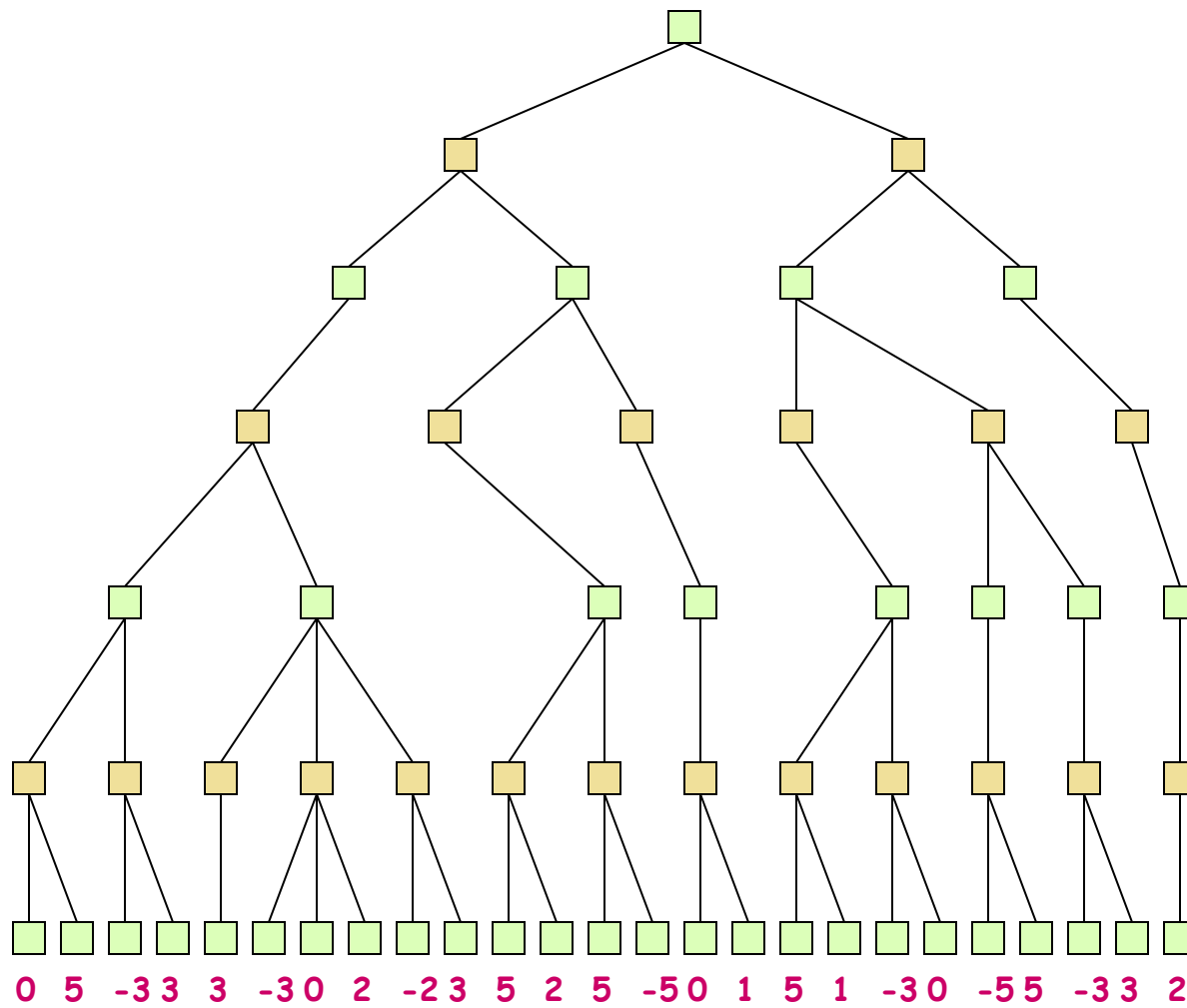


Example

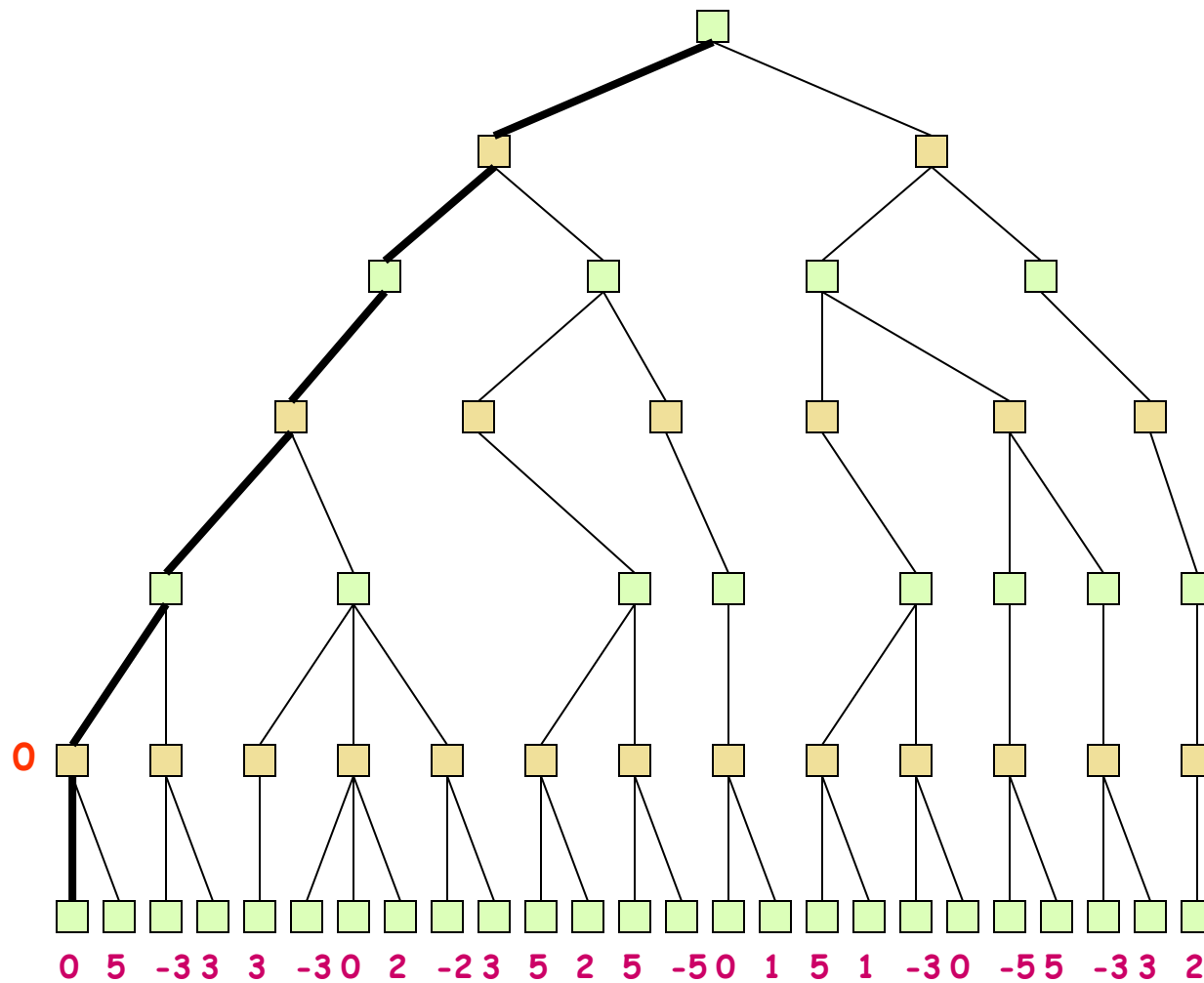
- Effect of knowing the lower bound and upper bound.
 - $[2, 14]$ for this example



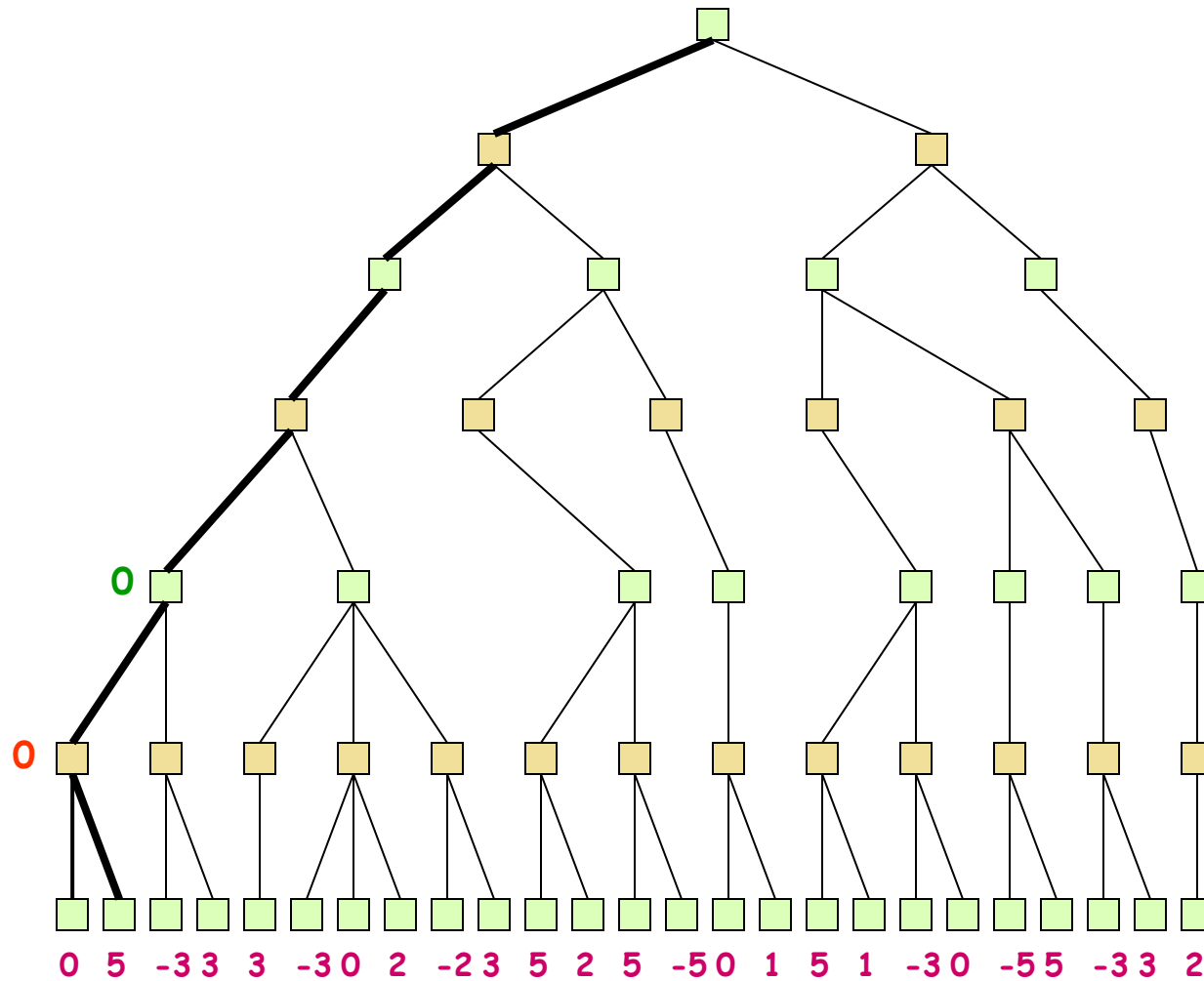
Example



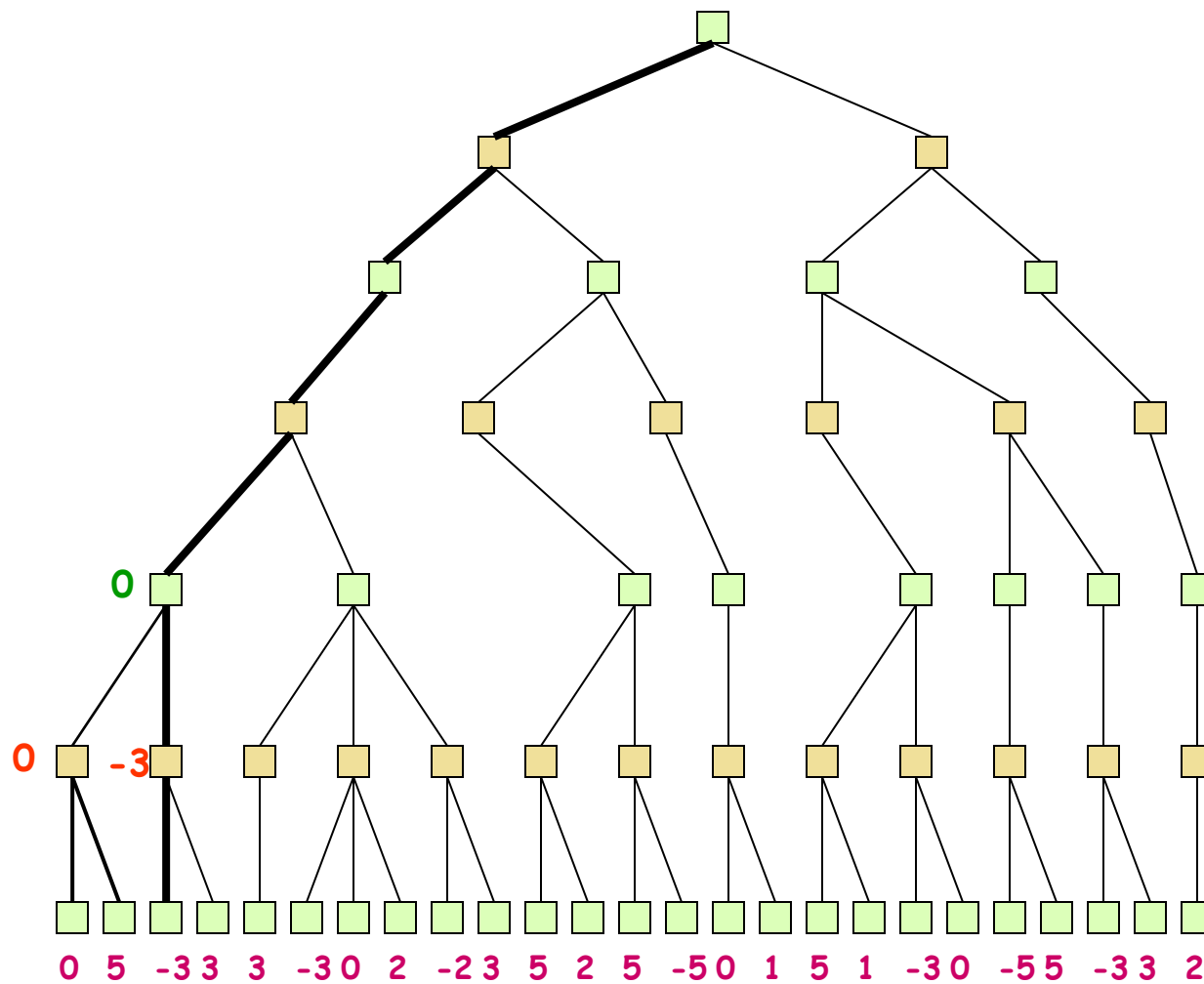
Example



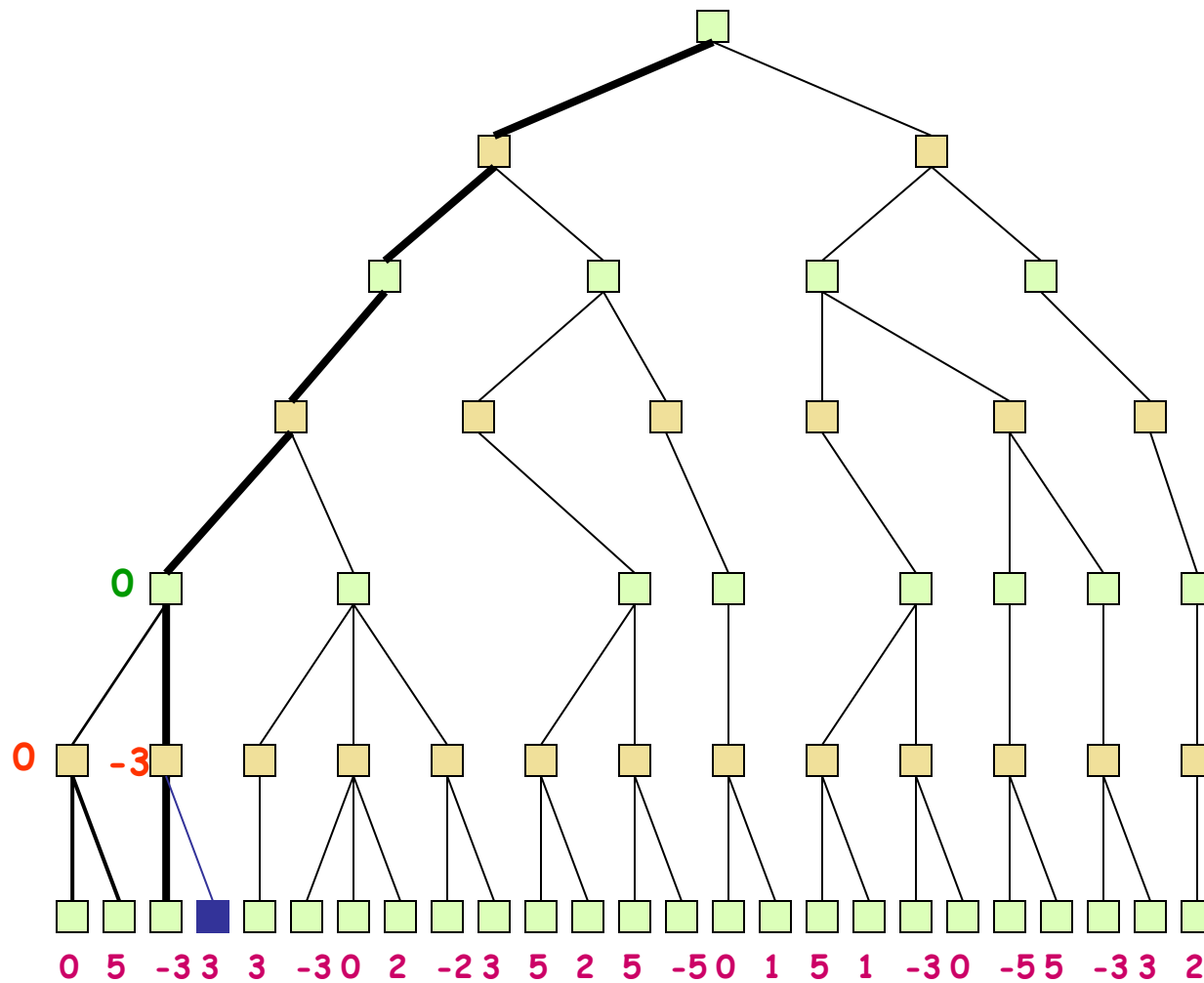
Example



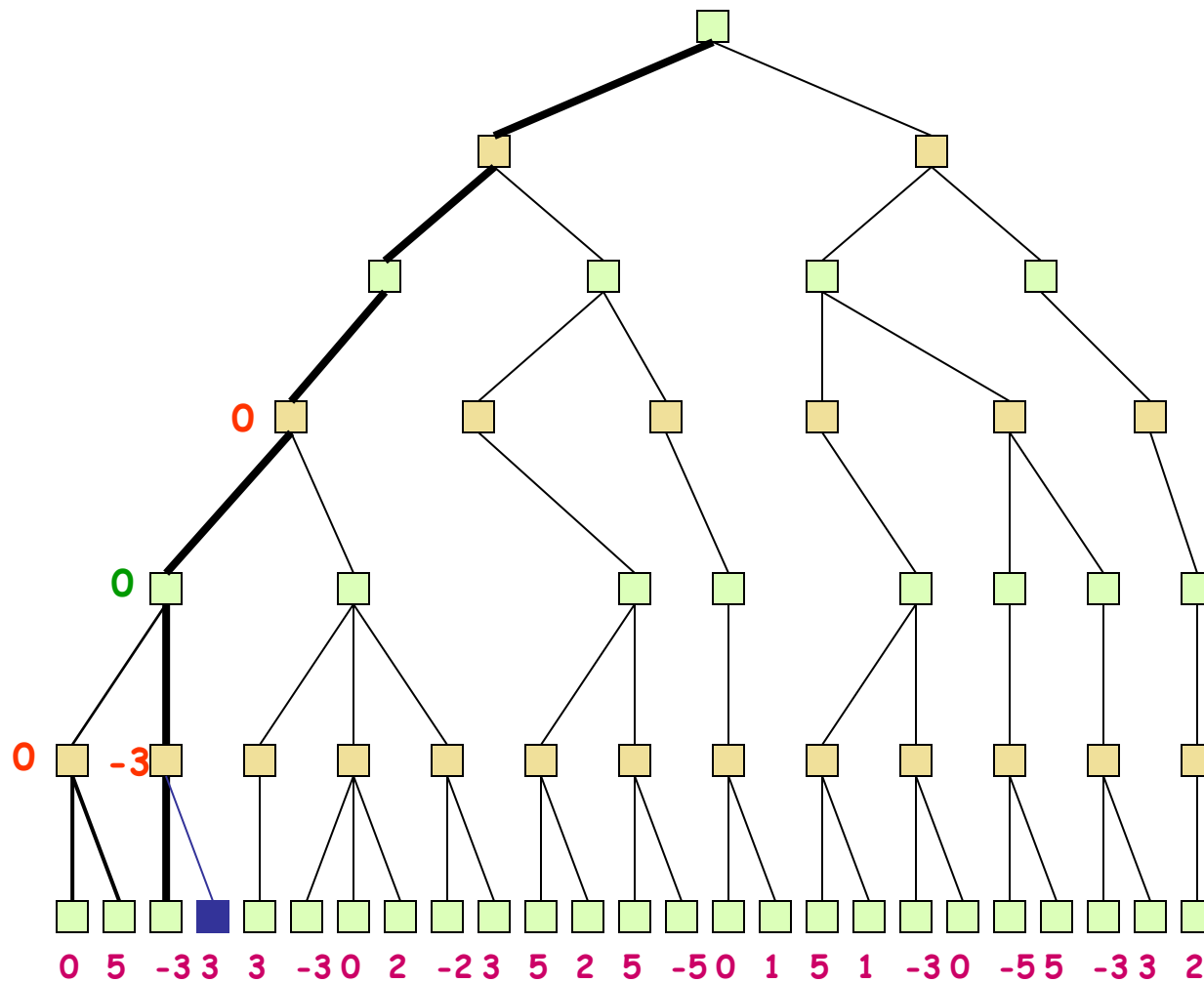
Example



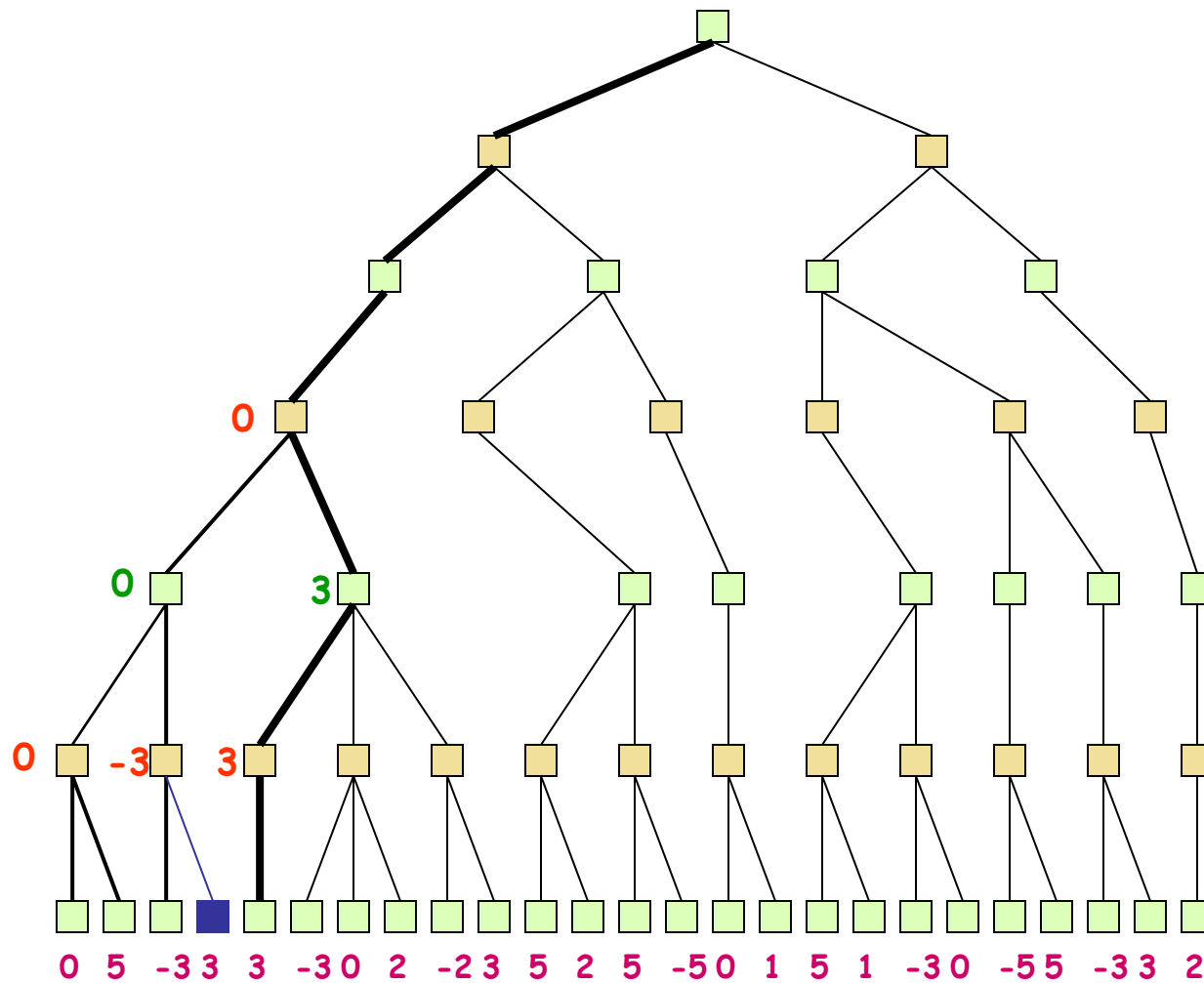
Example



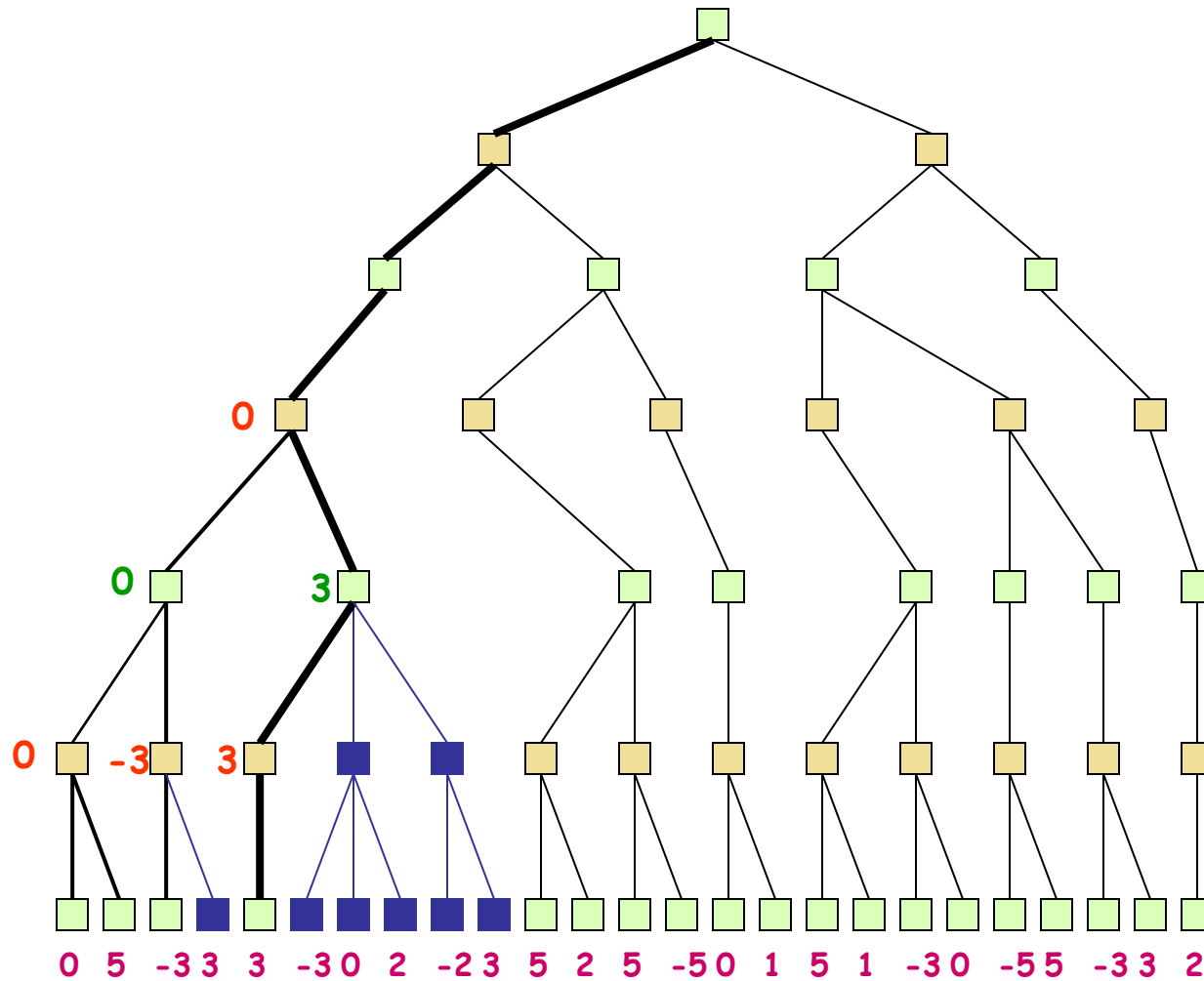
Example



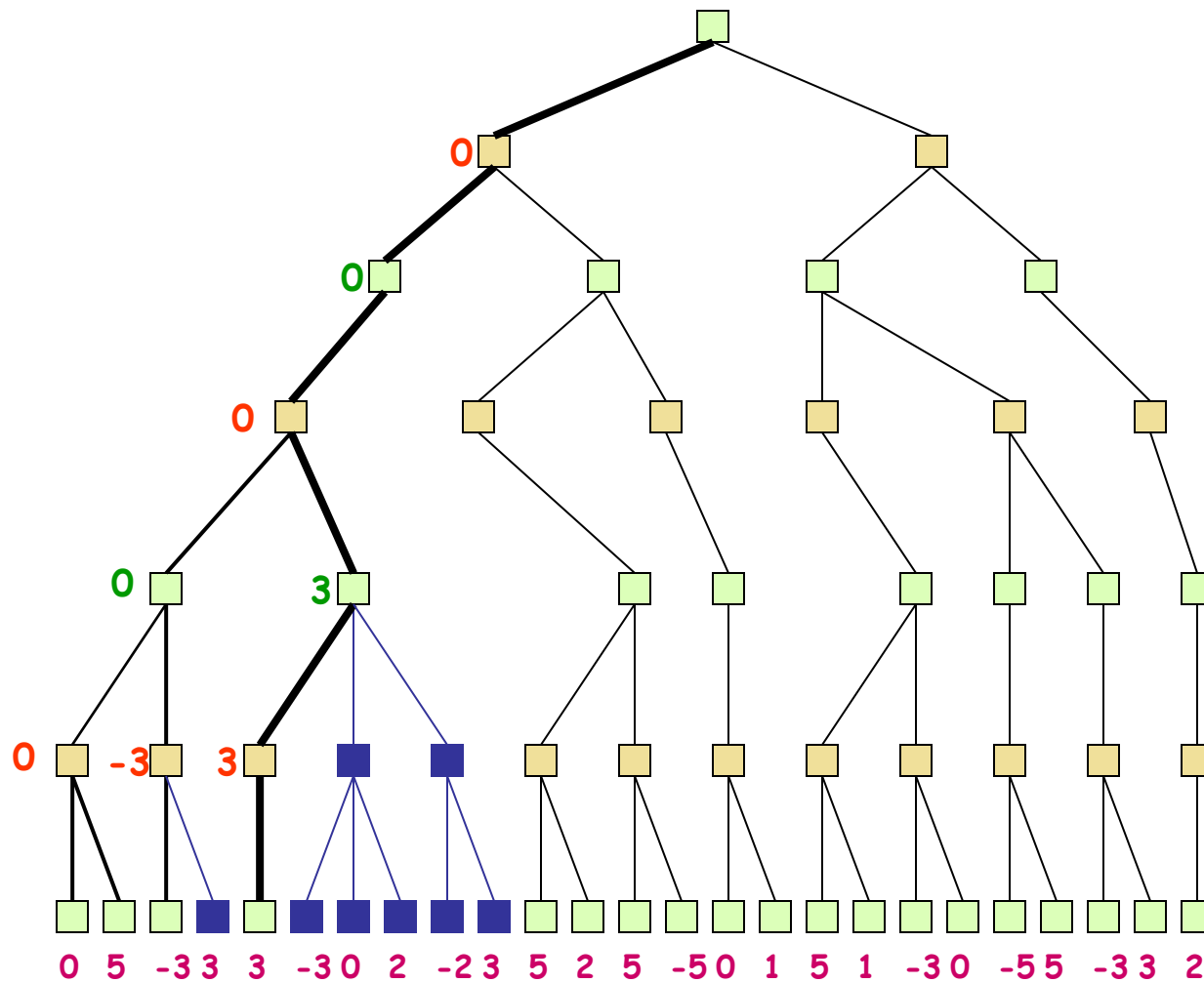
Example



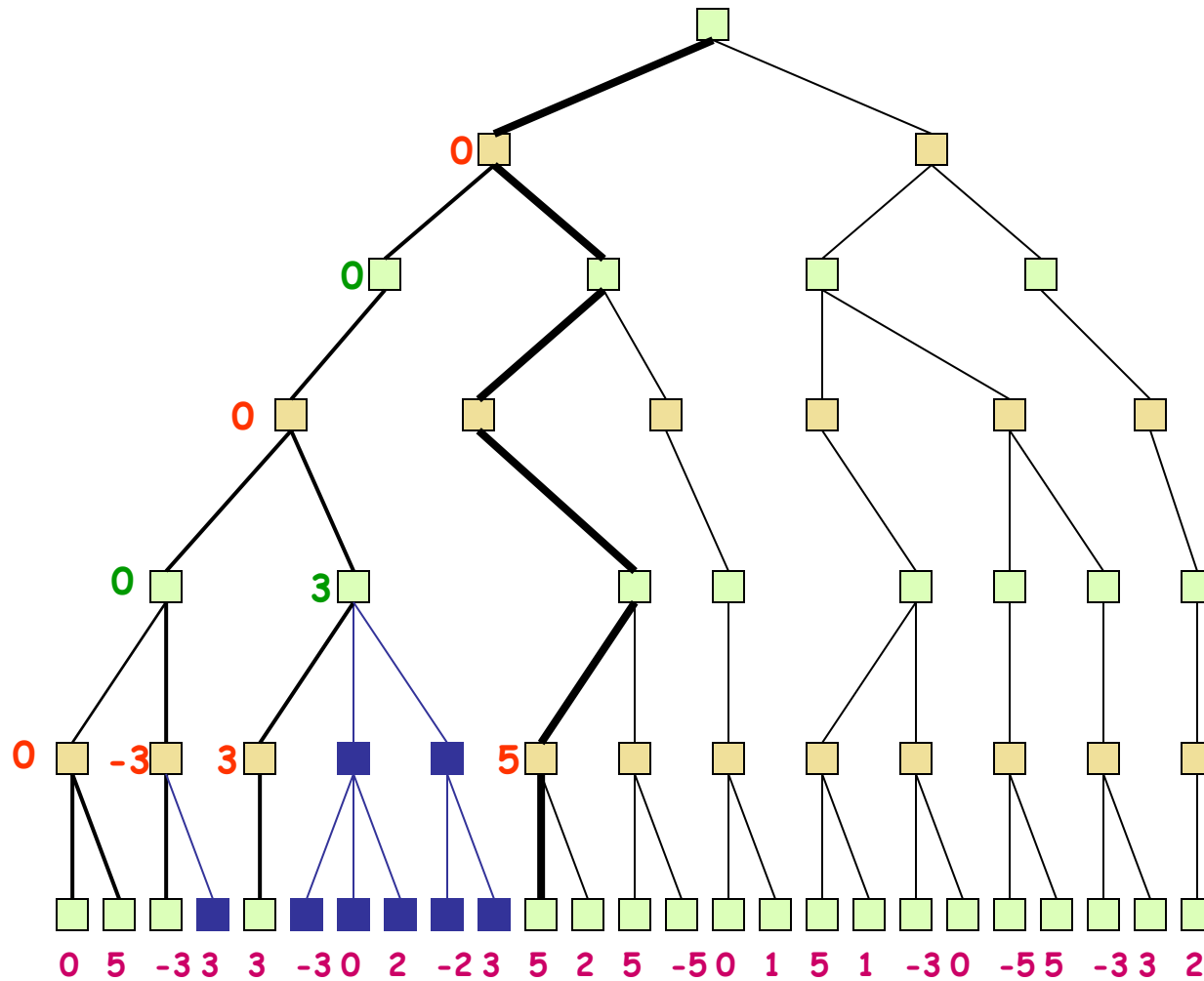
Example



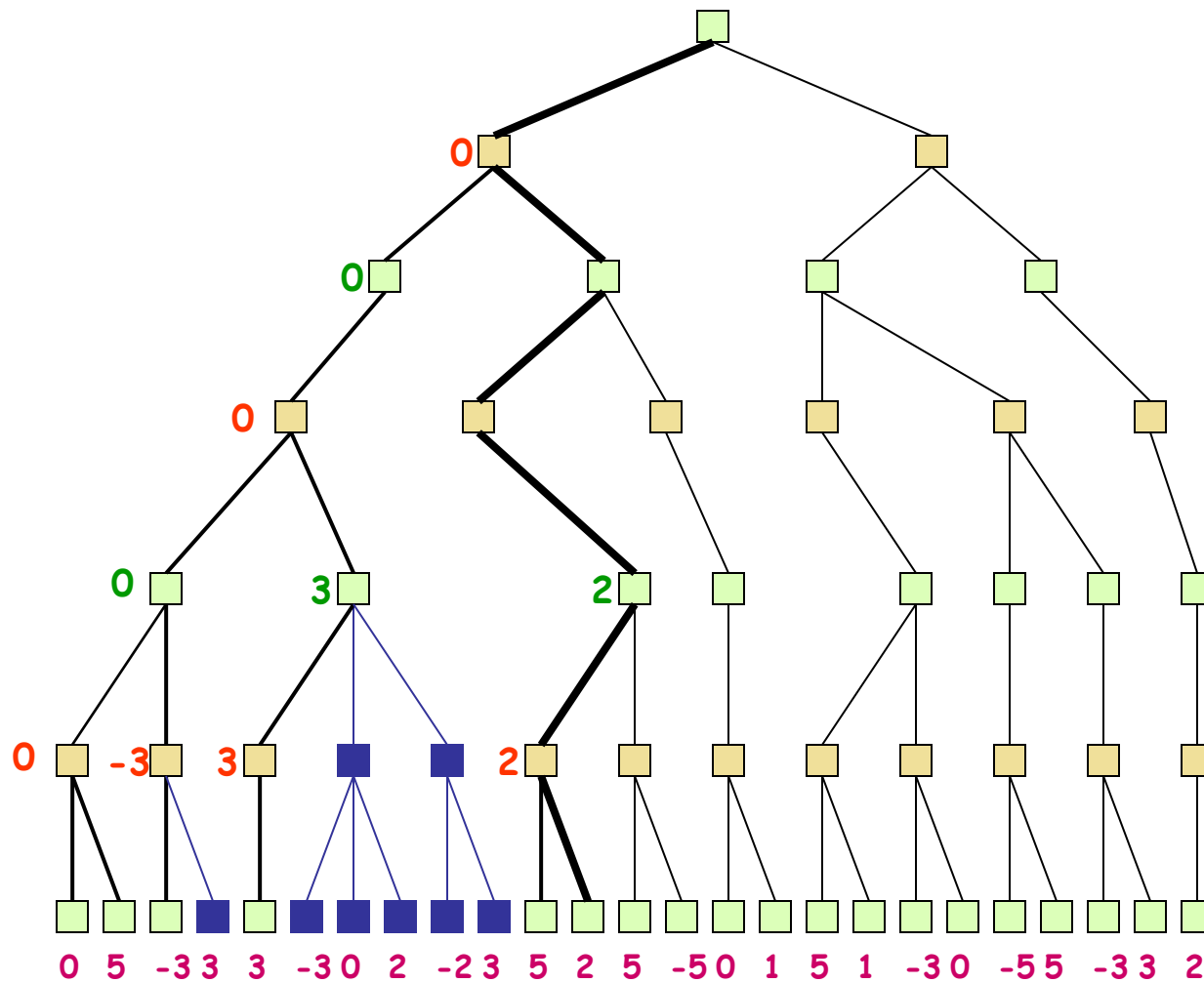
Example



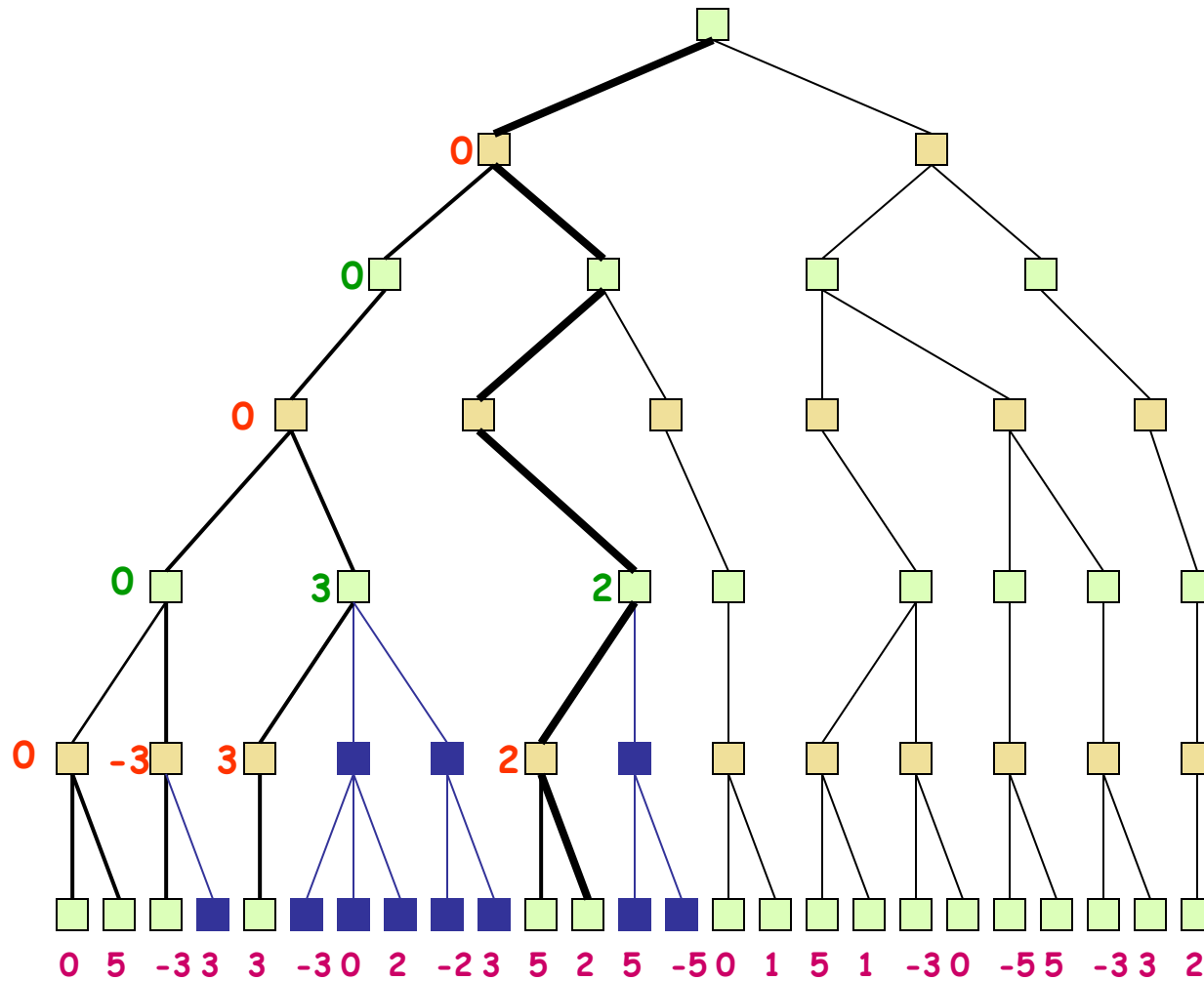
Example



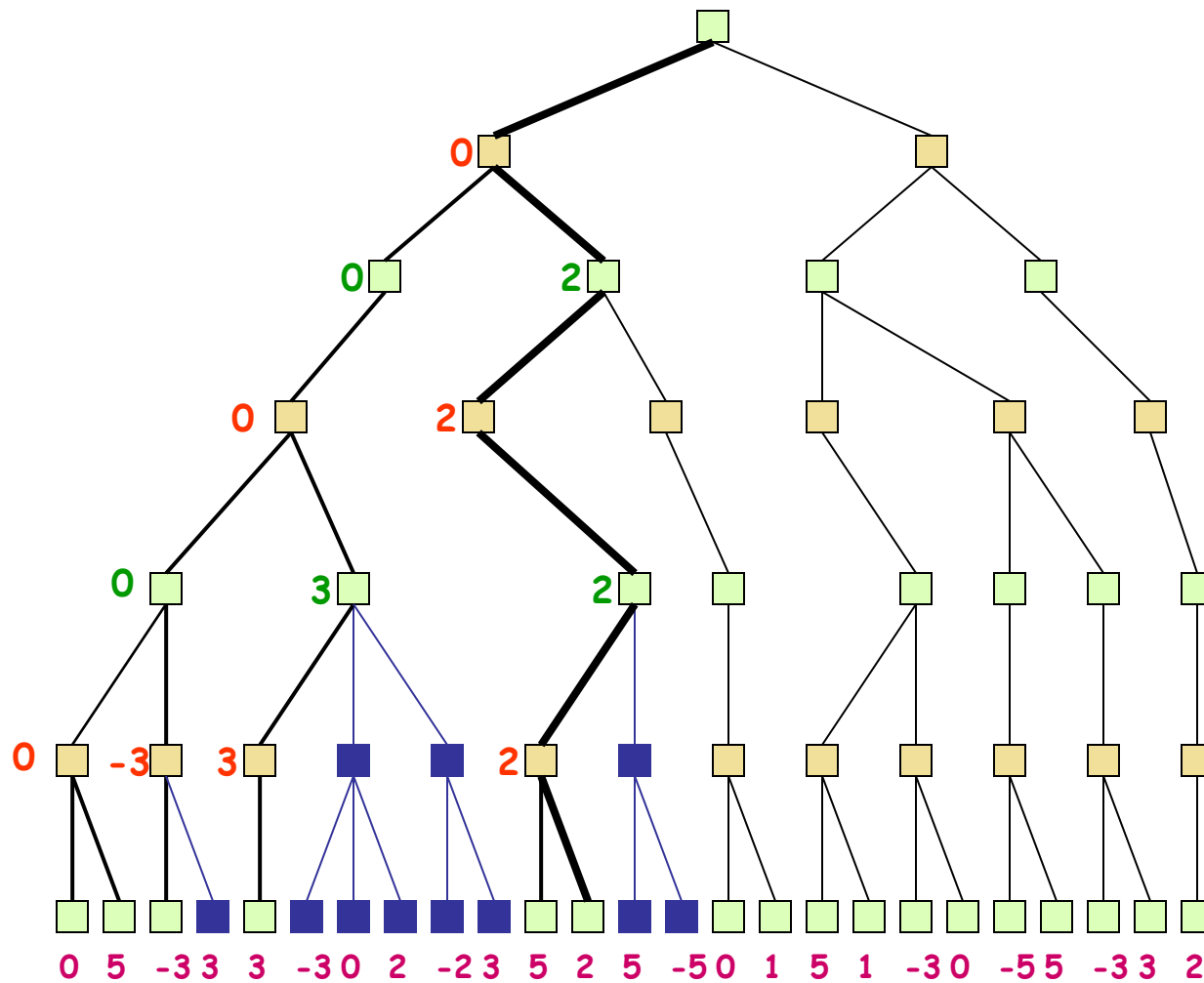
Example



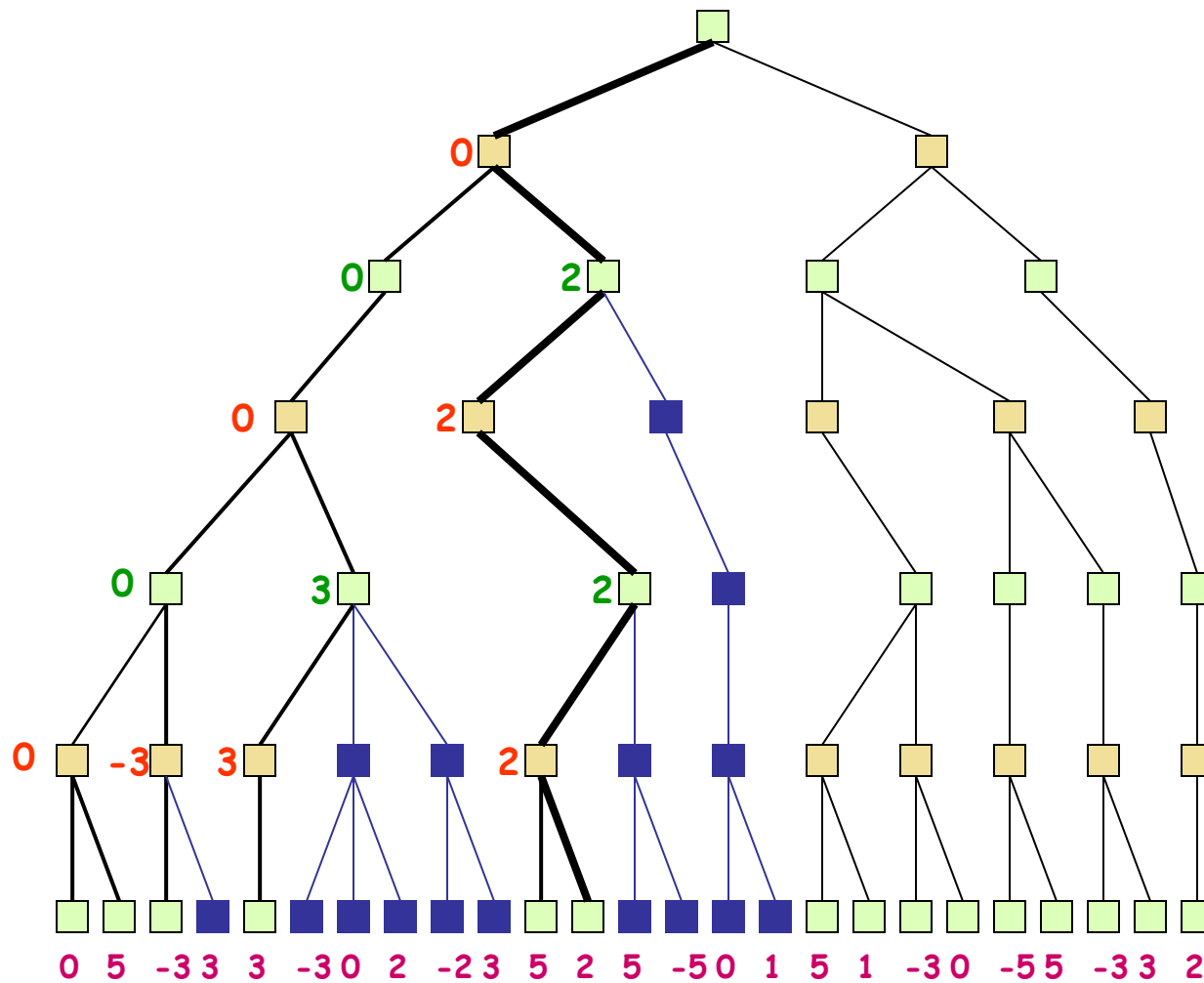
Example



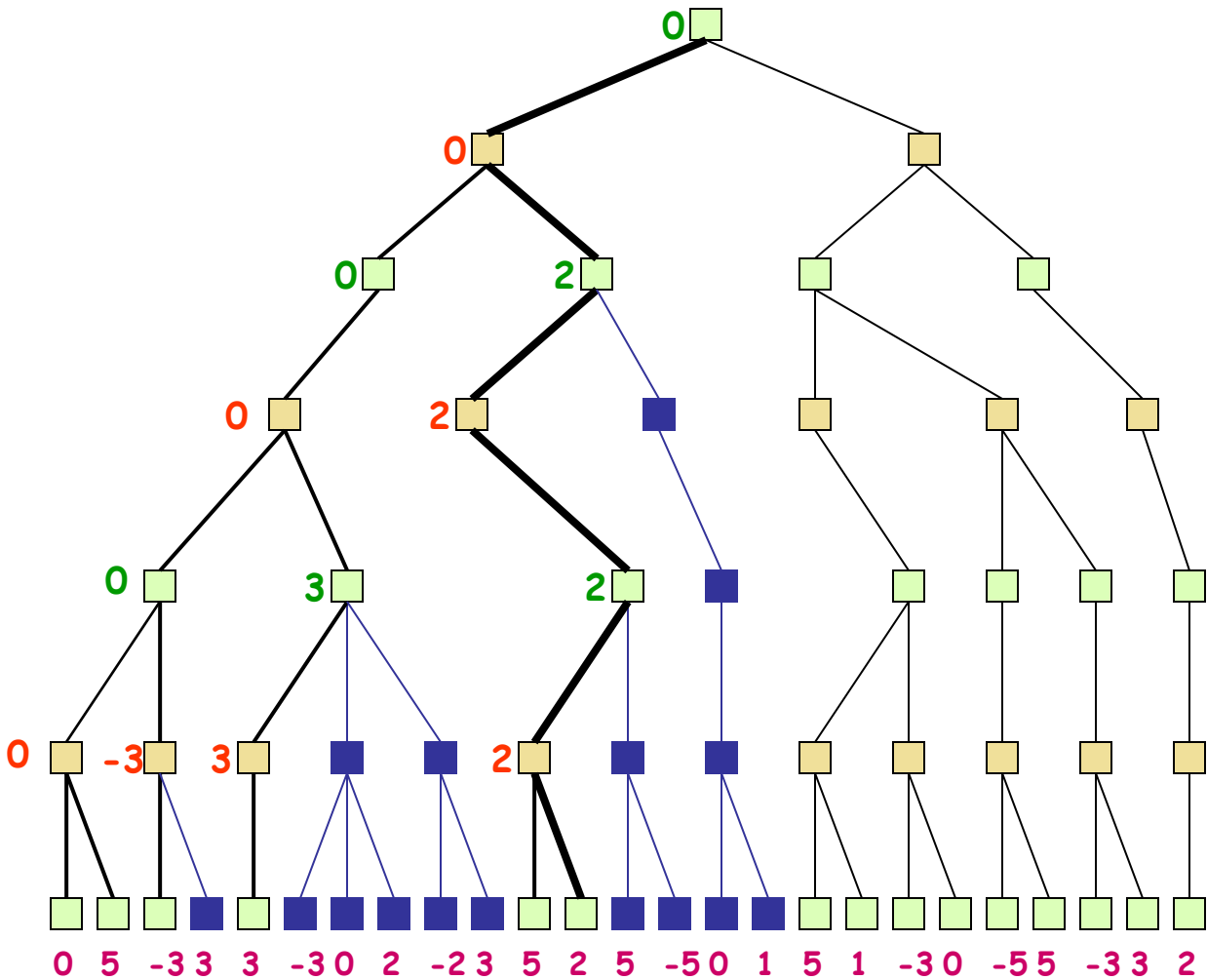
Example



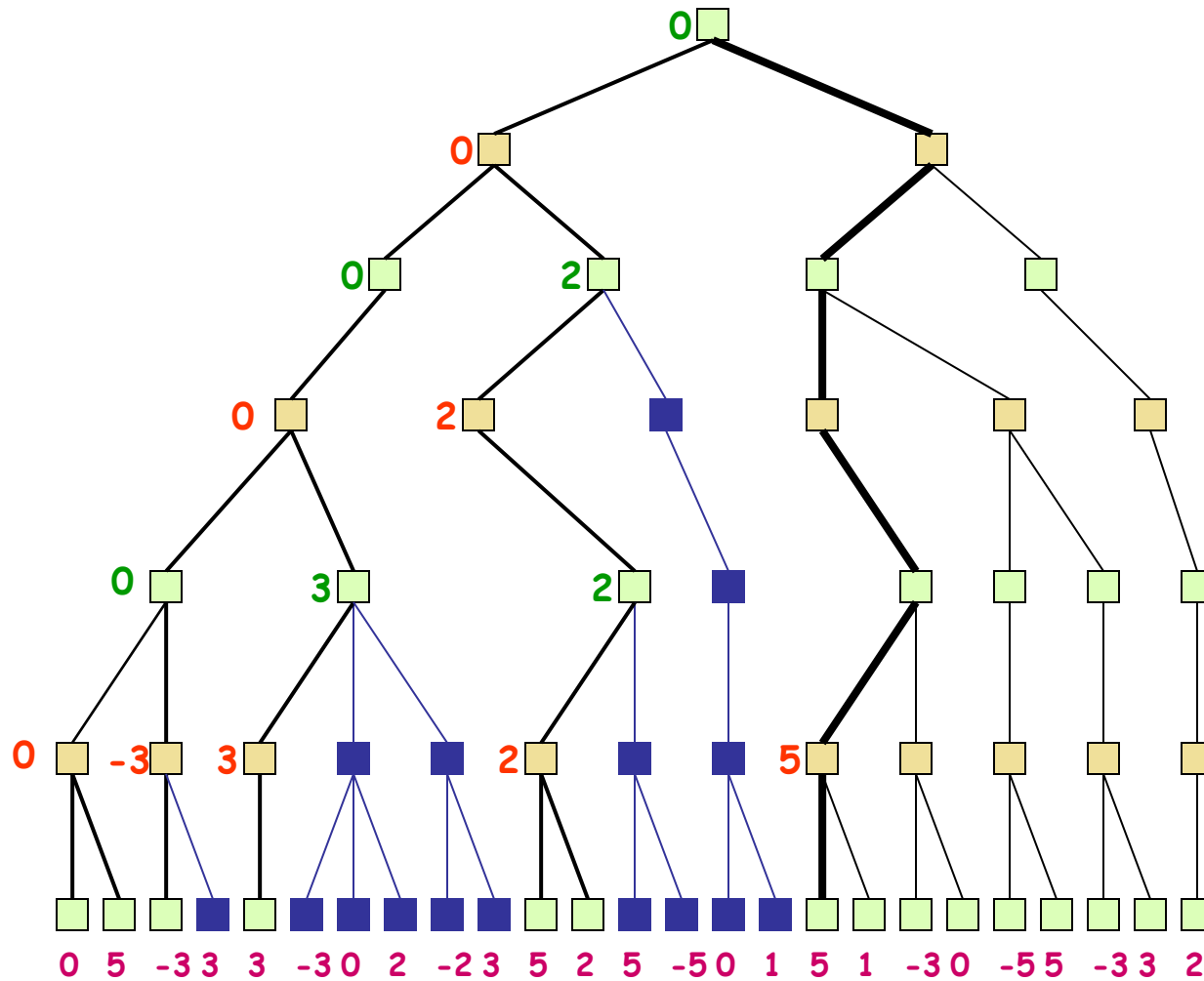
Example



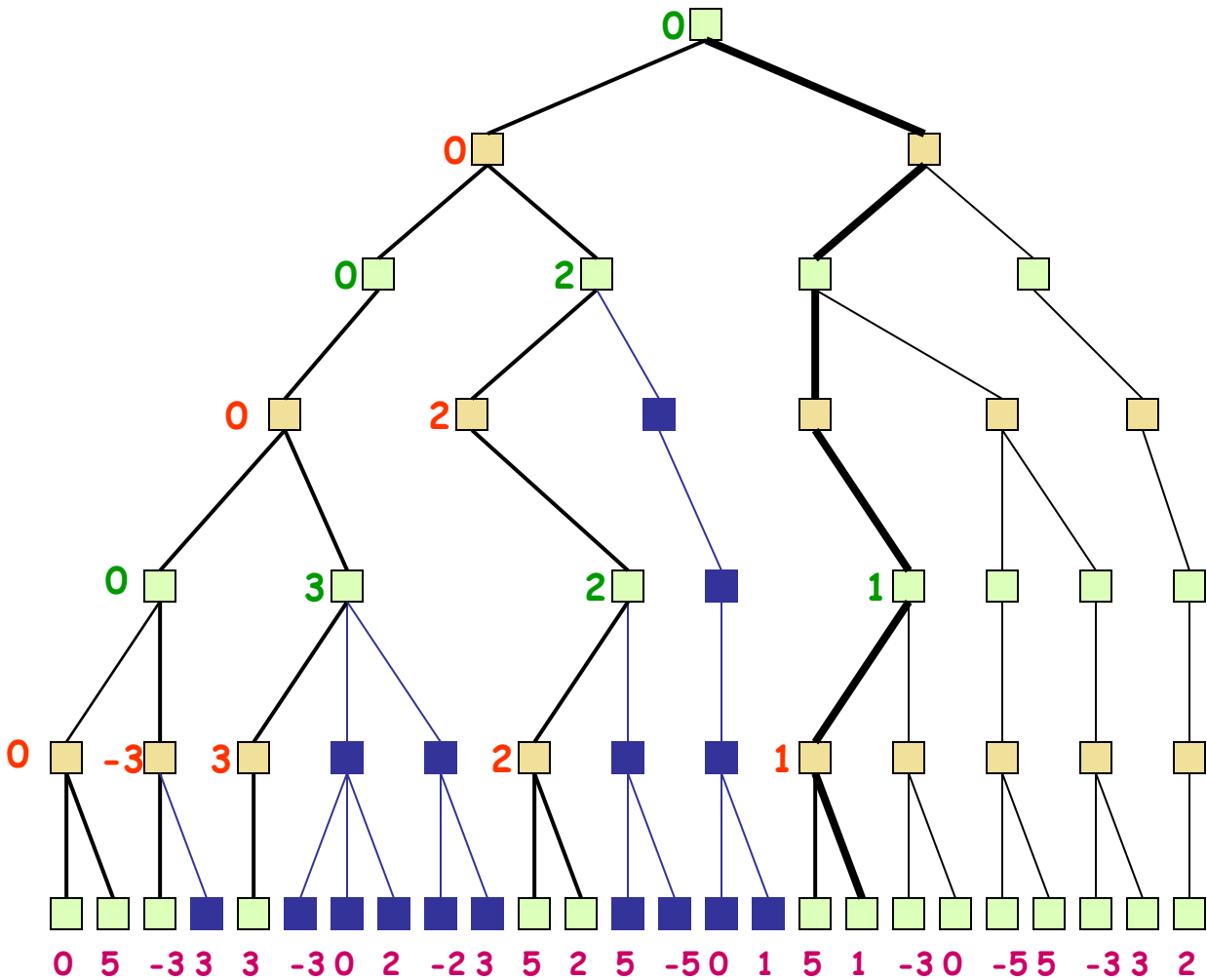
Example



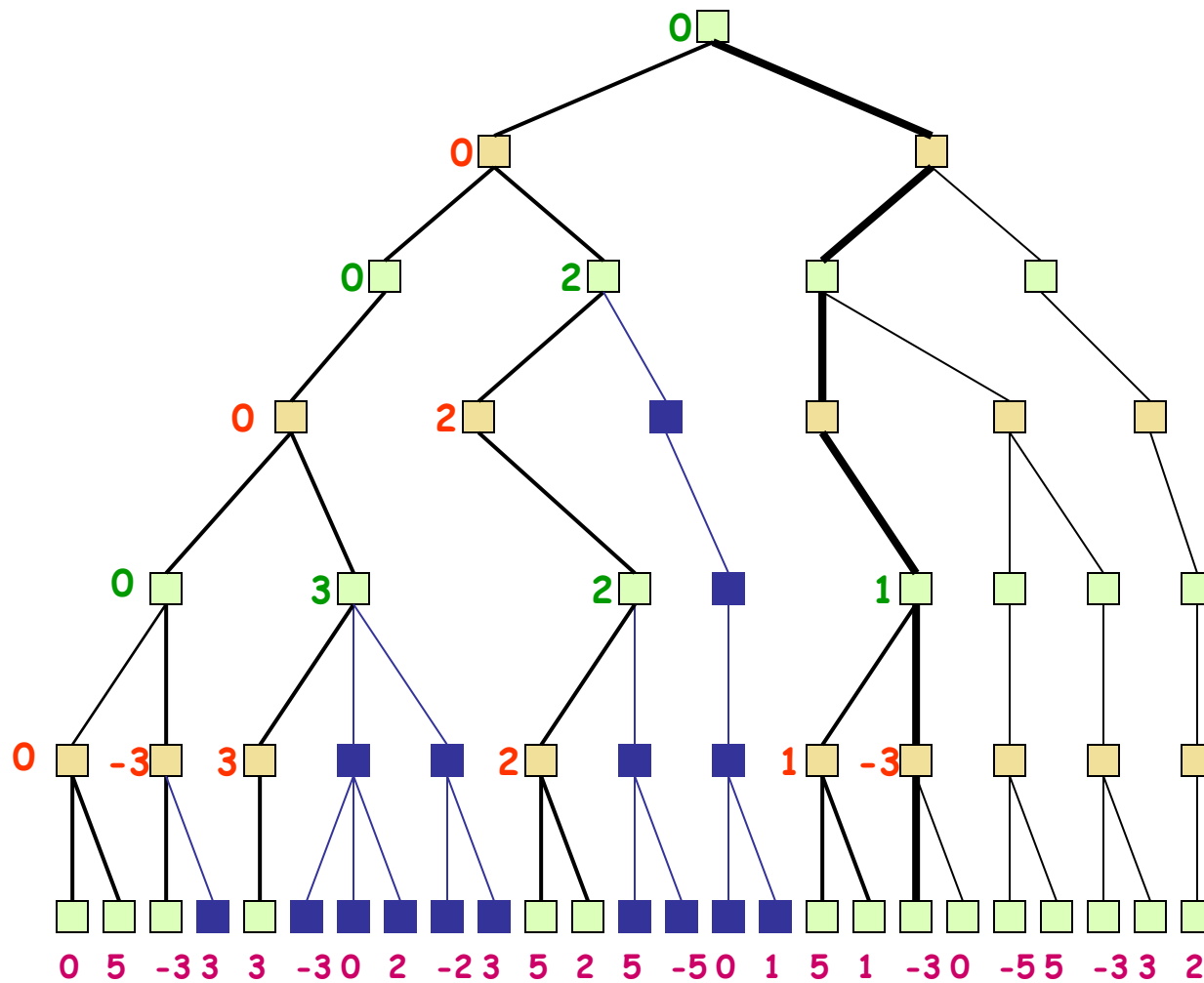
Example



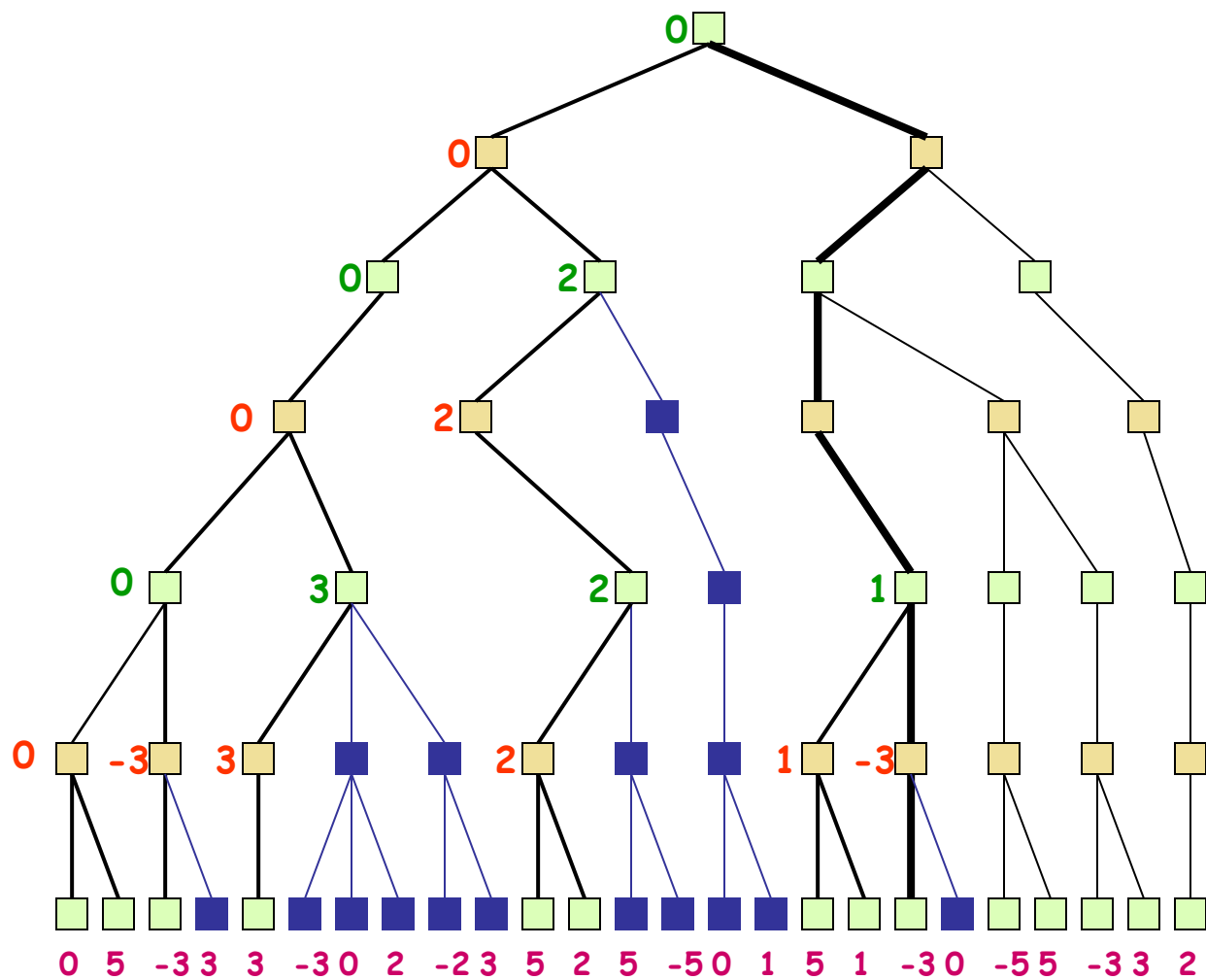
Example



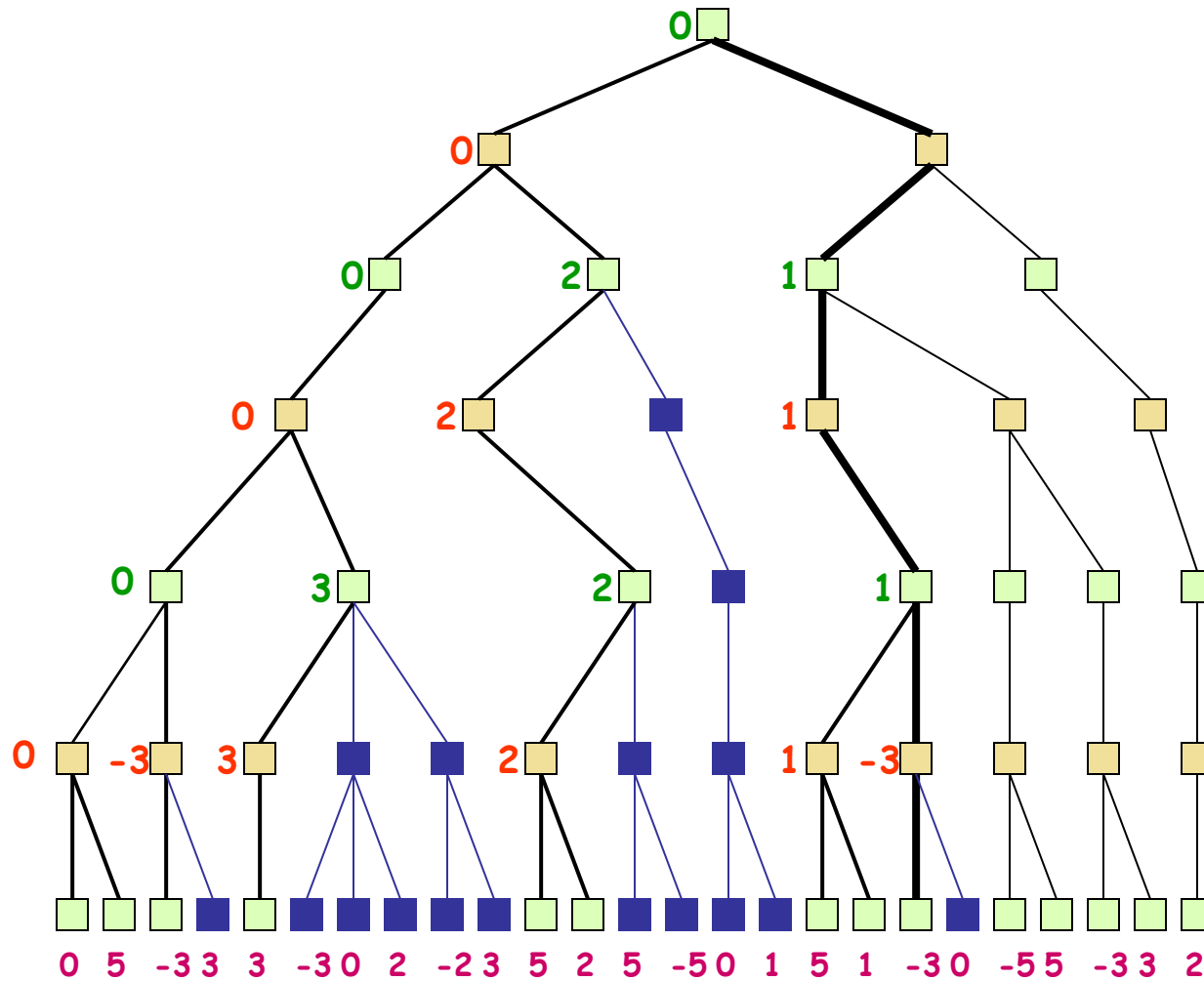
Example



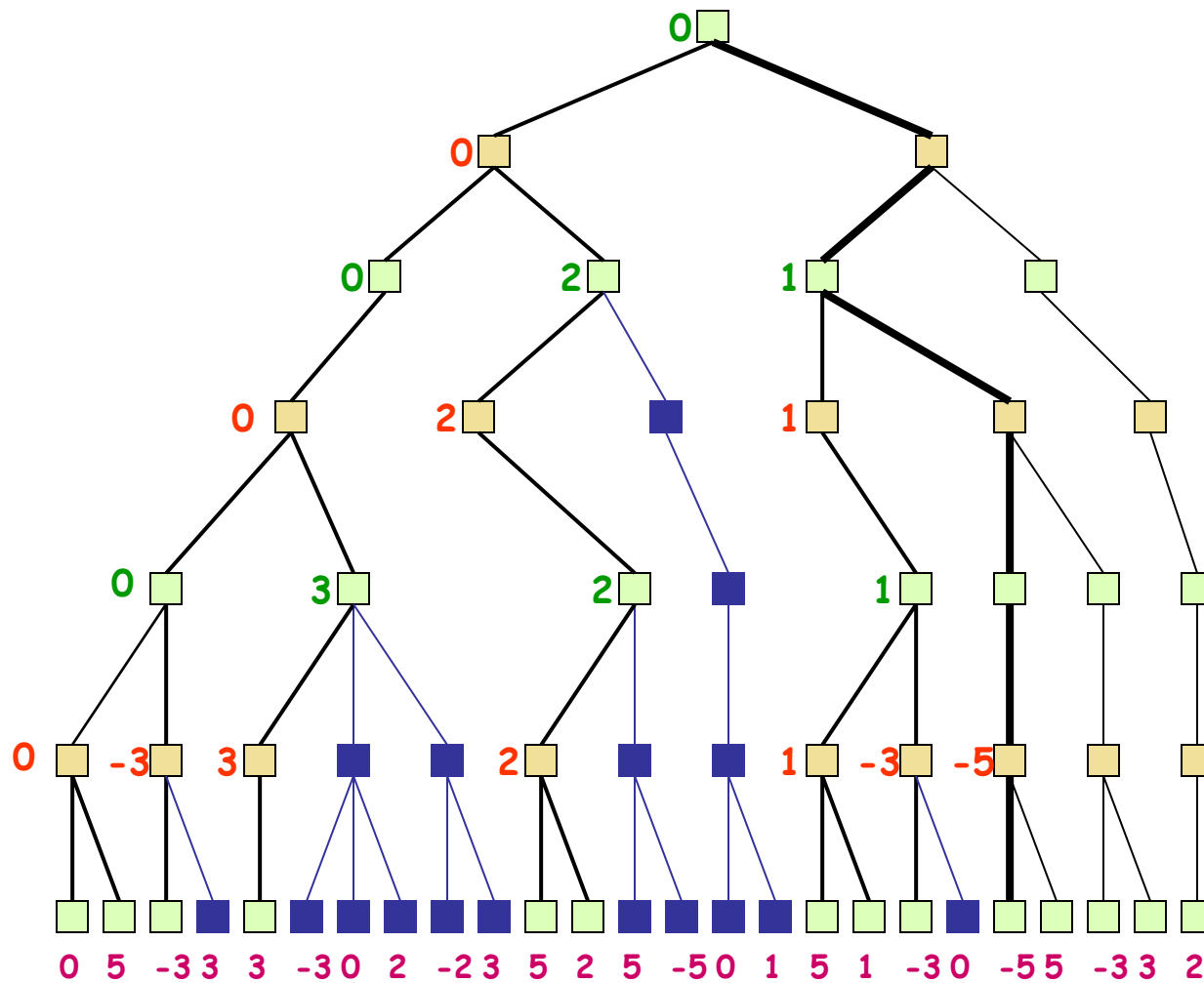
Example



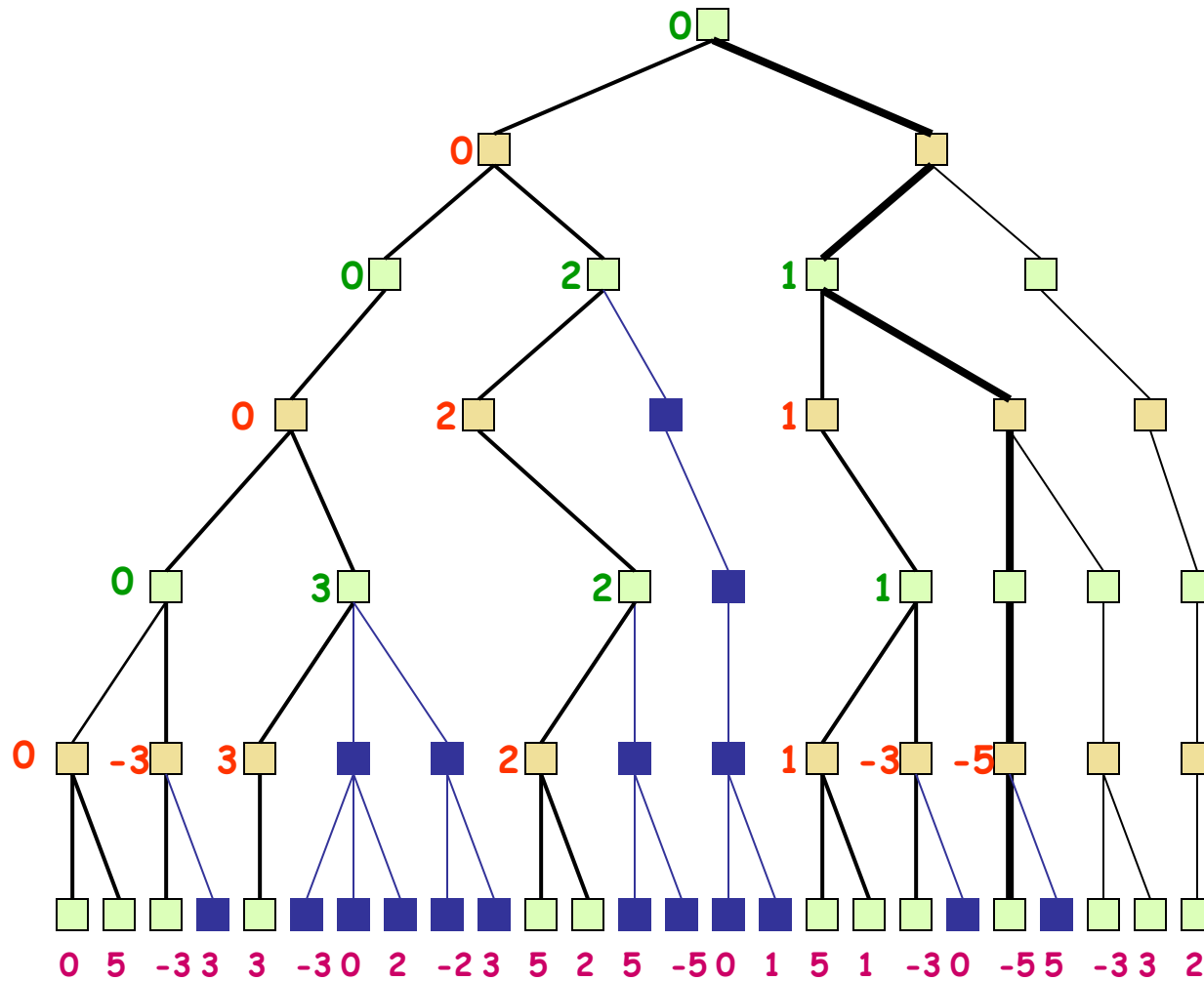
Example



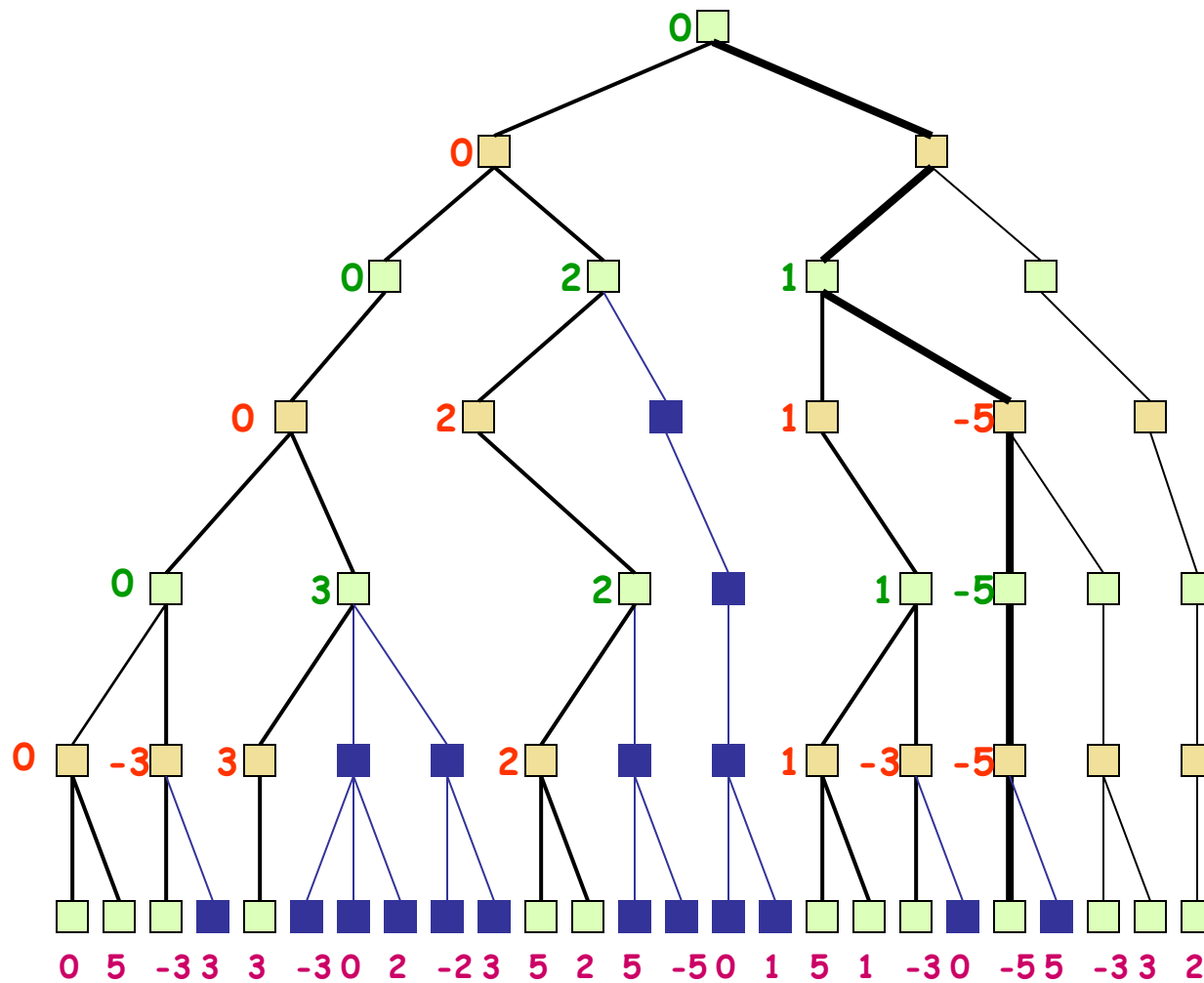
Example



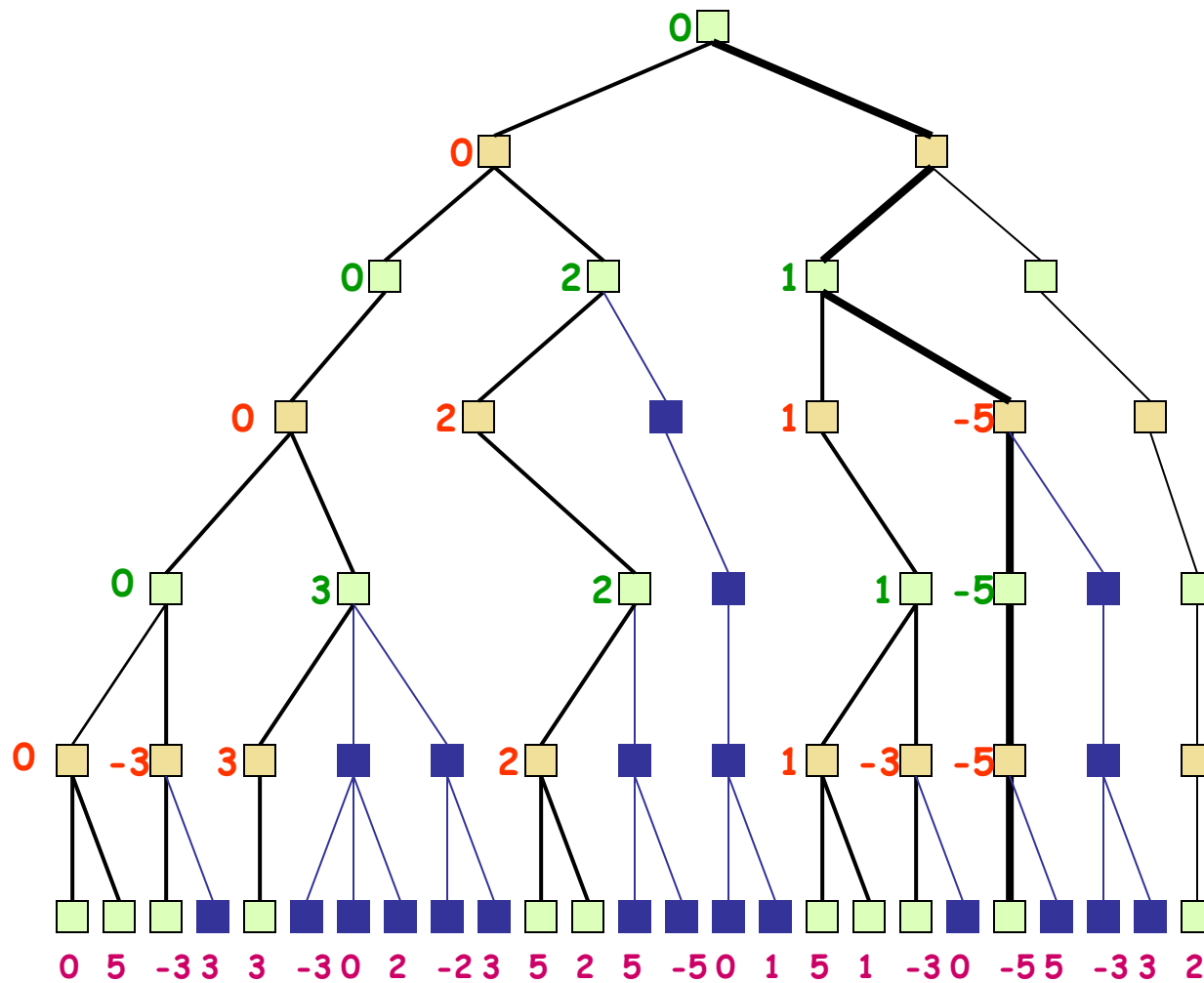
Example



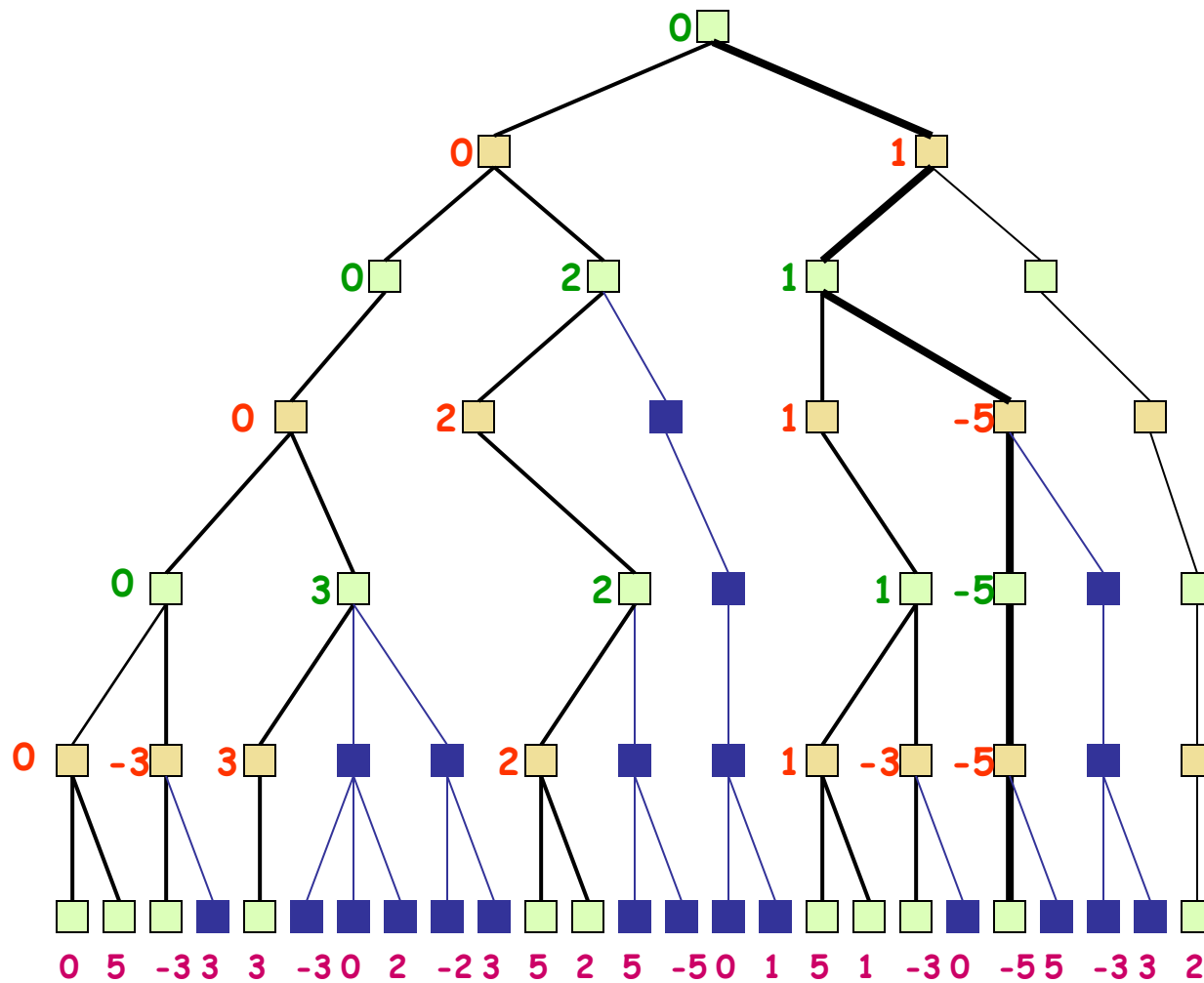
Example



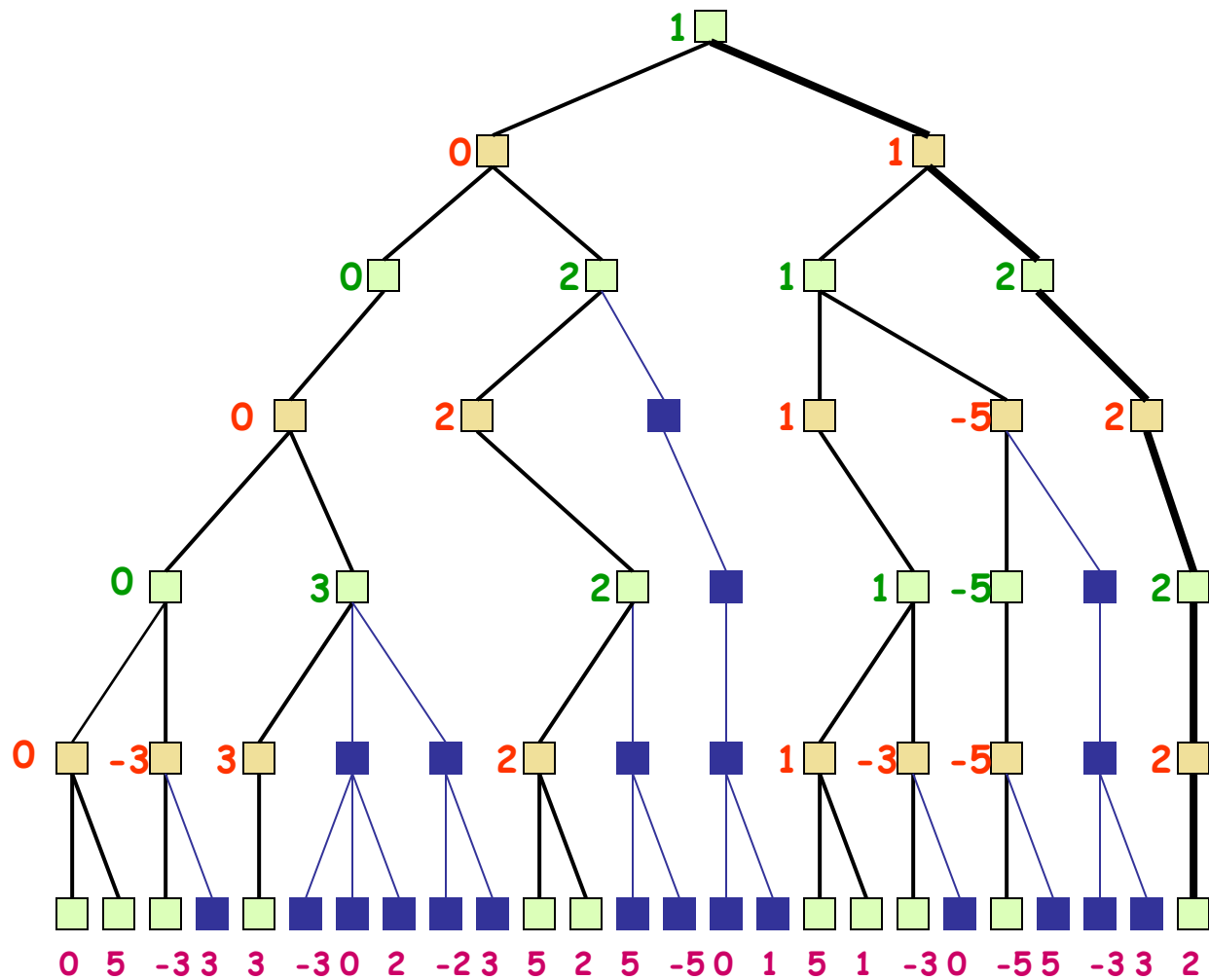
Example



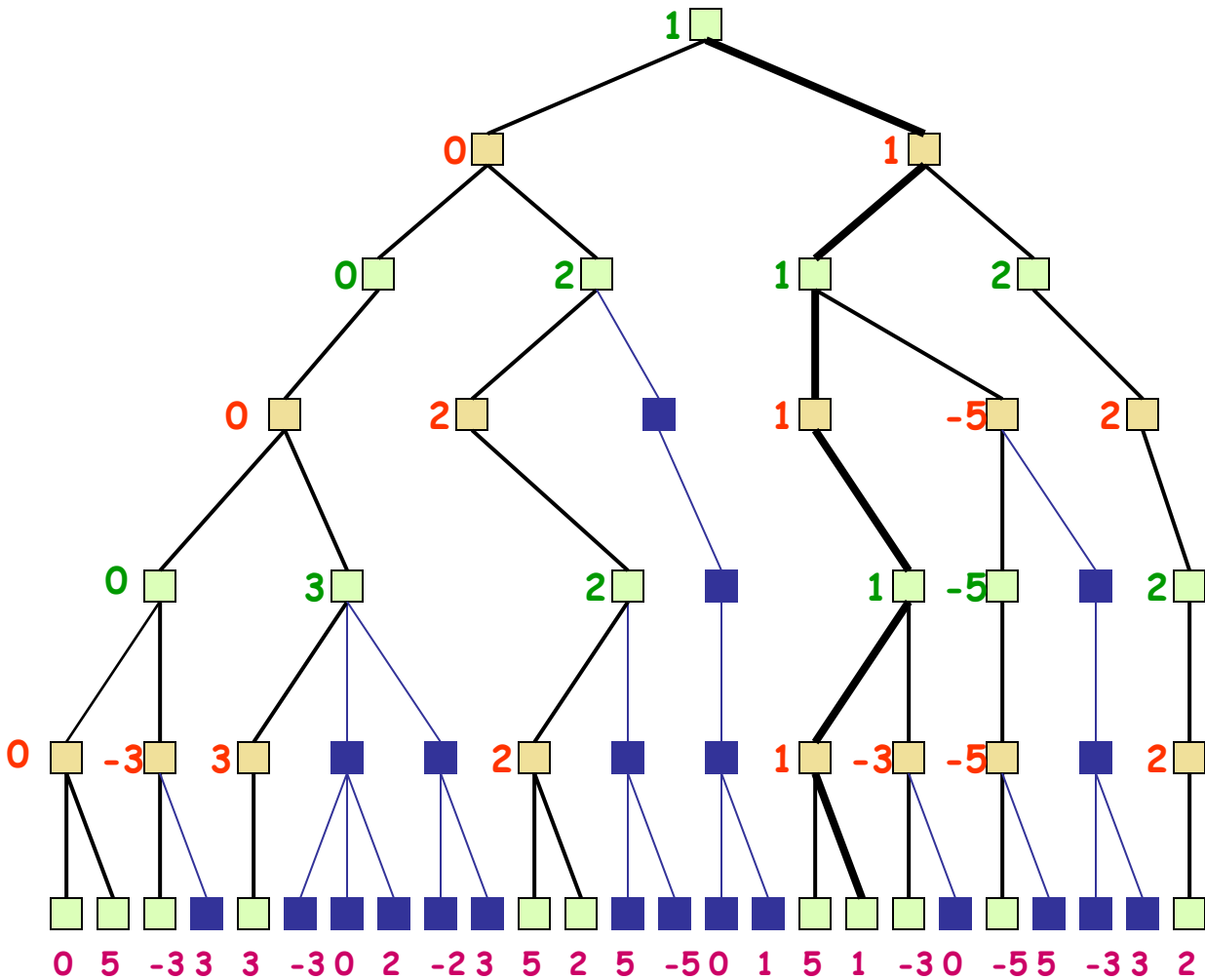
Example



Example

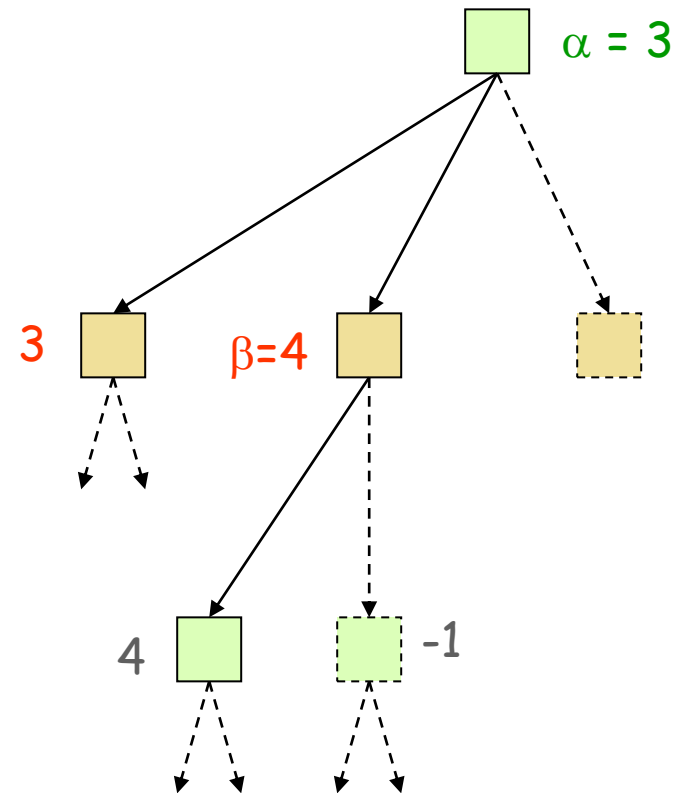
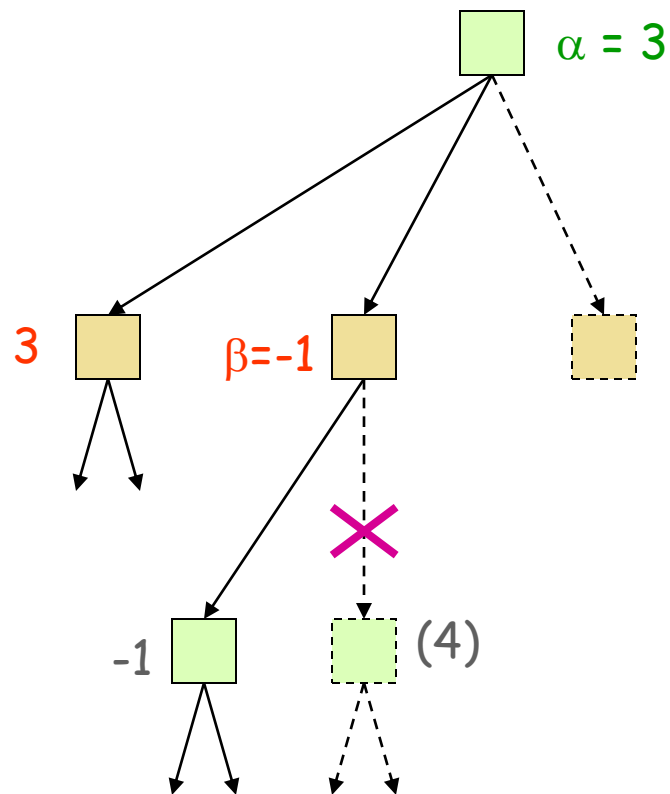


Example



How much do we gain?

Consider these two cases:



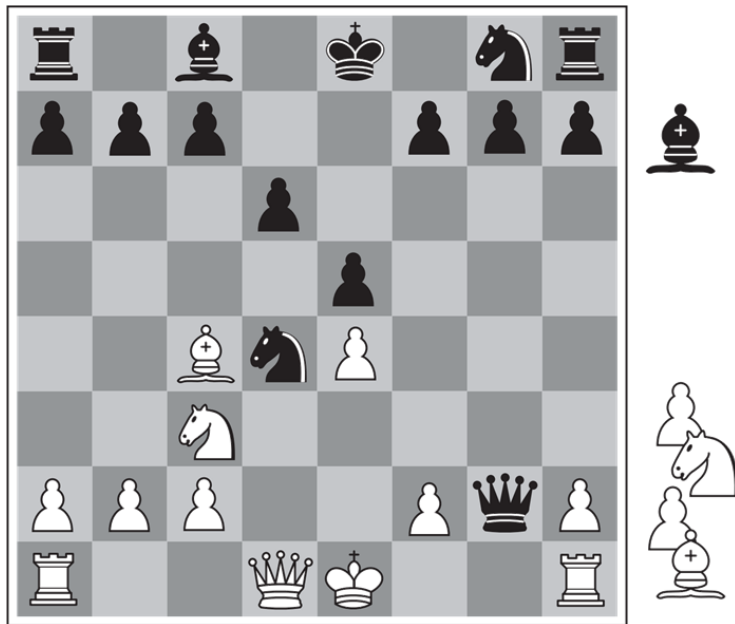
How much do we gain?

- Assume a game tree of uniform branching factor b
- Minimax examines $O(b^h)$ nodes, so does alpha-beta in the worst-case
- The gain for alpha-beta is maximum when:
 - The MIN children of a MAX node are ordered in decreasing backed up values
 - The MAX children of a MIN node are ordered in increasing backed up values
- Then alpha-beta examines $O(b^{h/2})$ nodes [Knuth and Moore, 1975]
- But this requires an oracle (if we knew how to order nodes perfectly, we would not need to search the game tree)
- If nodes are ordered at random, then the average number of nodes examined by alpha-beta is $\sim O(b^{3h/4})$

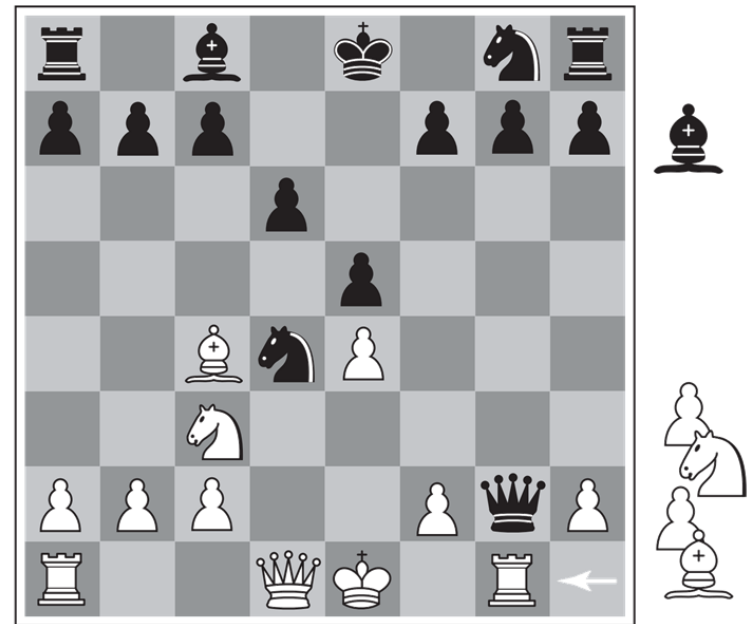
Cutting off search: simple depth limit

if CUTOFF-TEST($state$, $depth$) then return EVAL($state$)

- Problem1: **non-quiet** positions
 - Few more plies make big difference in evaluation value



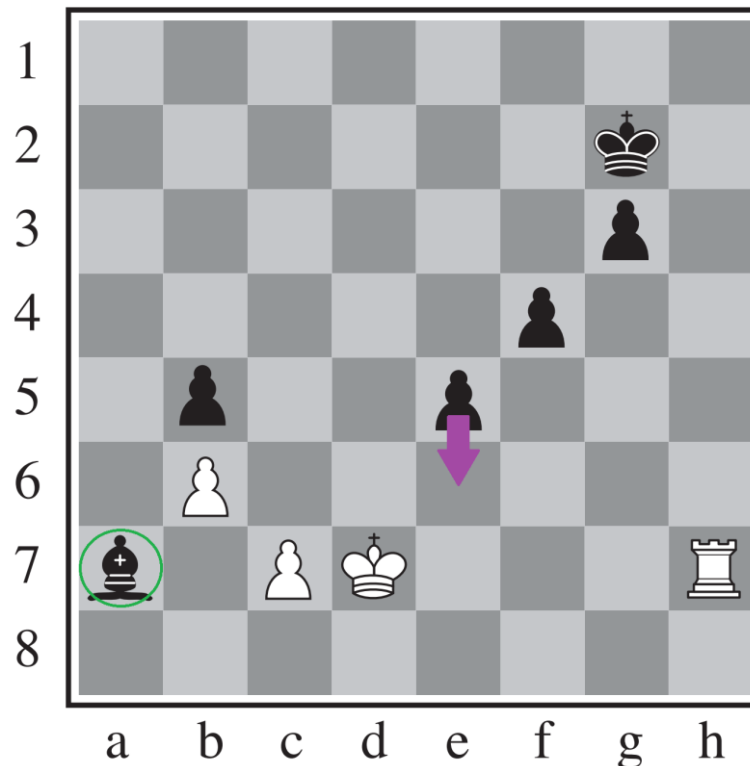
(a) White to move



(b) White to move

Cutting off search: simple depth limit

- Problem 2: **horizon effect**
 - Delaying tactics against opponent's move that **causes serious unavoidable damage** (because of pushing the damage beyond the horizon that the player can see)



More sophisticated cutting off

- Cutoff only on quiescent positions
 - **Quiescent search**: expanding non-quiescent positions until reaching quiescent ones
- Horizon effect
 - **Singular extension**: a move that is clearly better than all other moves in a given position.
 - Once reaching the depth limit, check to see if the singular extension is a legal move.
 - It makes the tree deeper but it does not add many nodes to the tree due to few possible singular extensions.

Forward pruning

- Some moves at a given node are pruned immediately without further consideration
 - Most humans playing chess consider only a few moves from each position
- beam search
 - Consider only a "beam" of the n best moves (according to the evaluation function) rather than considering all possible moves.

Unfortunately, this approach is rather dangerous because there is no guarantee that the best move will not be pruned away.

Speed up the search process

- Table lookup
- Example: Chess
 - Books describing good play in the opening and endgame in chess have been available for about a century
 - For the openings, the computer is mostly relying on the expertise of humans
 - Computers can also gather statistics from a database of previously played games
 - Usually after ten moves we end up in a rarely seen position, and the program must switch from table lookup to search.
 - Near the end of the game there are again fewer possible positions, and thus more chance to do lookup.
 - Computer analysis of endgames goes far beyond anything achieved by humans.

Stochastic games

MAX

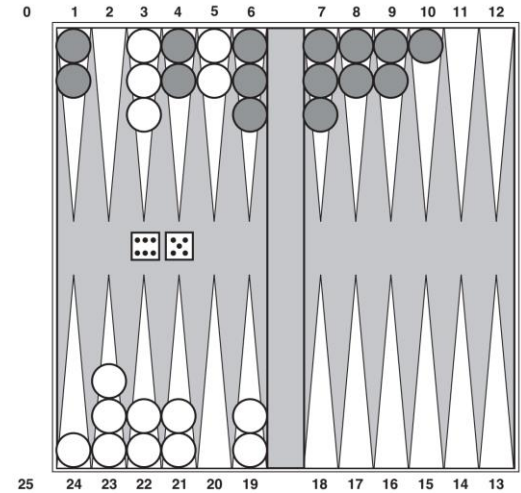
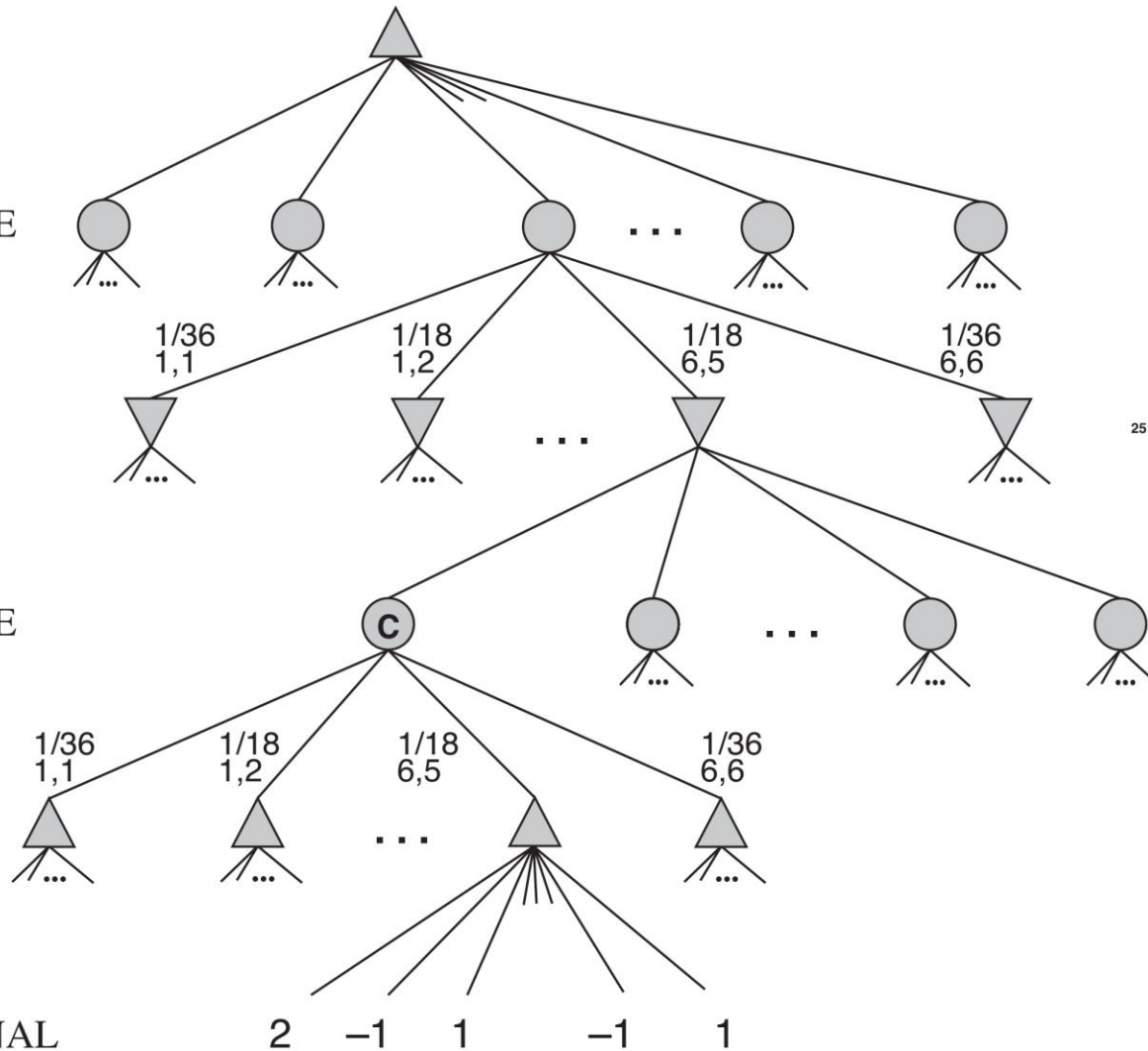
CHANCE

MIN

CHANCE

MAX

TERMINAL



Stochastic games

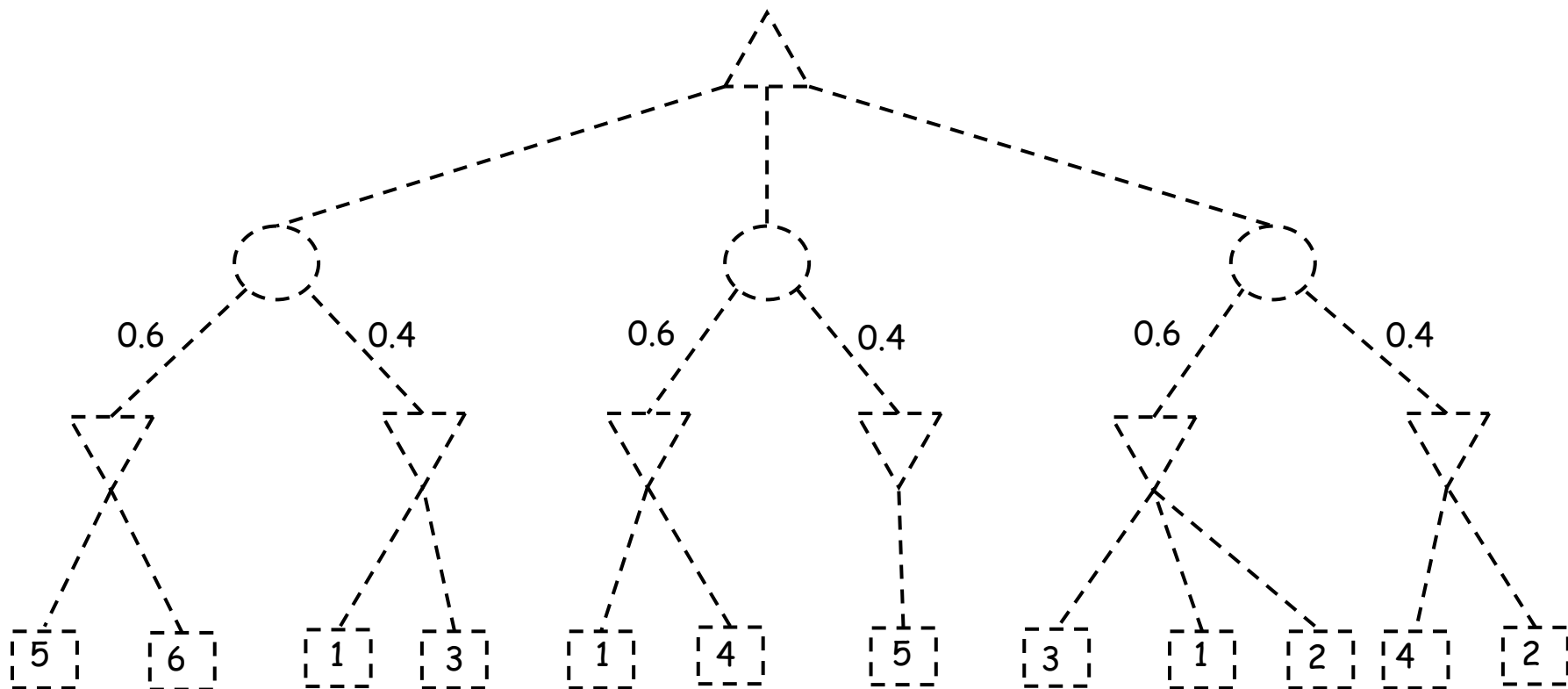
- Obviously, we still want to pick the move that leads to the best position.
 - Positions do **not have** definite minimax values.
 - Instead, we can only calculate the **expected value** of a position
 - the average over all possible outcomes of the chance nodes

EXPECTIMINIMAX(s) =

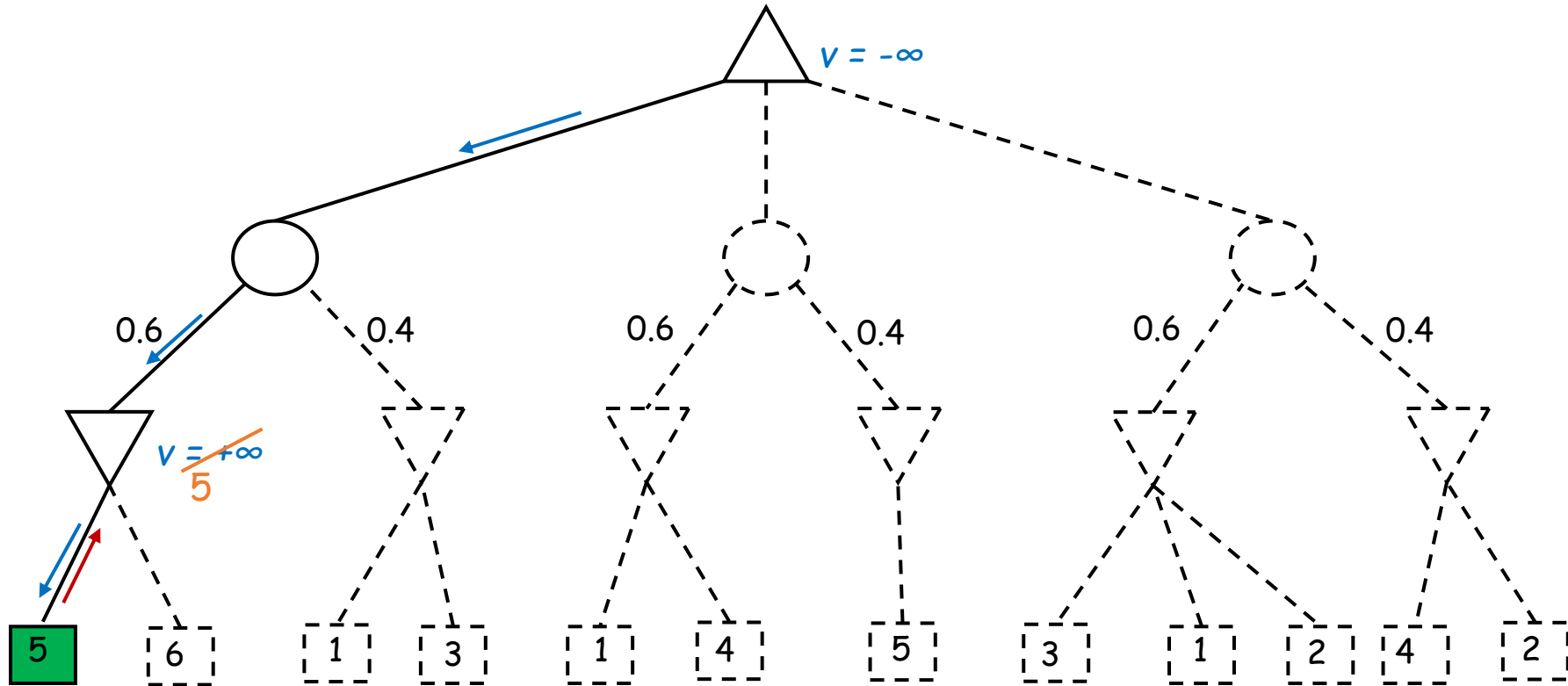
$$\begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_a \text{EXPECTIMINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MIN} \\ \sum_r P(r) \text{EXPECTIMINIMAX}(\text{RESULT}(s, r)) & \text{if } \text{PLAYER}(s) = \text{CHANCE} \end{cases}$$

where r represents a possible dice roll (or other chance event) and $\text{RESULT}(s, r)$ is the same state as s , with the additional fact that the result of the dice roll is r .

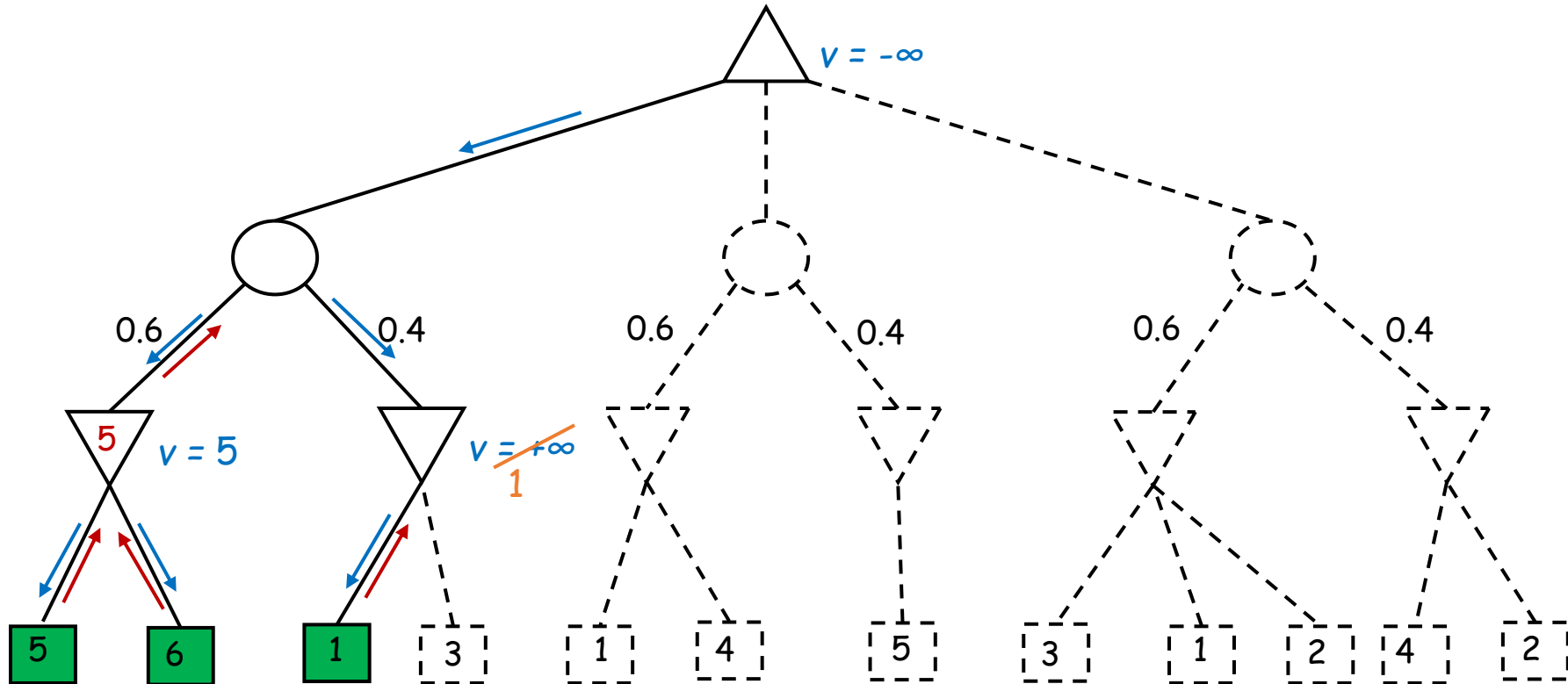
Stochastic games: Example



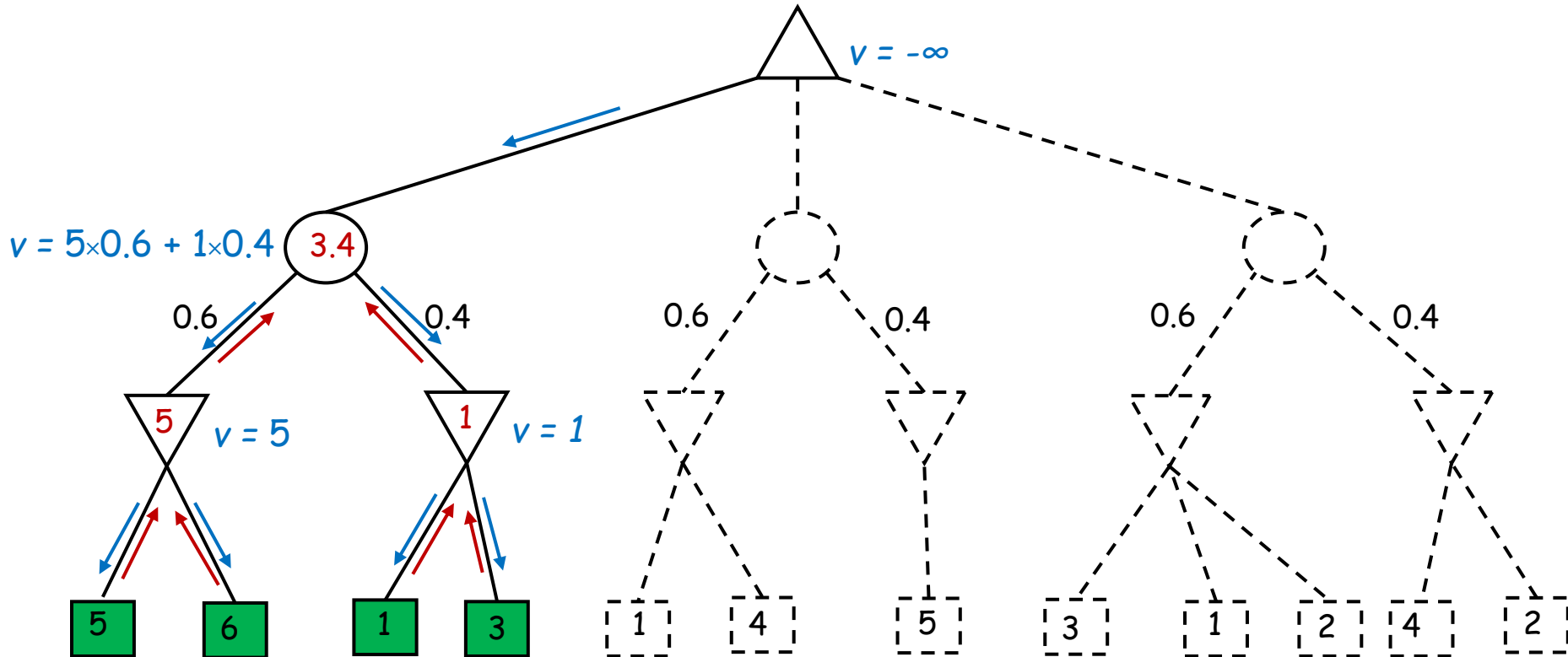
Stochastic games: Example



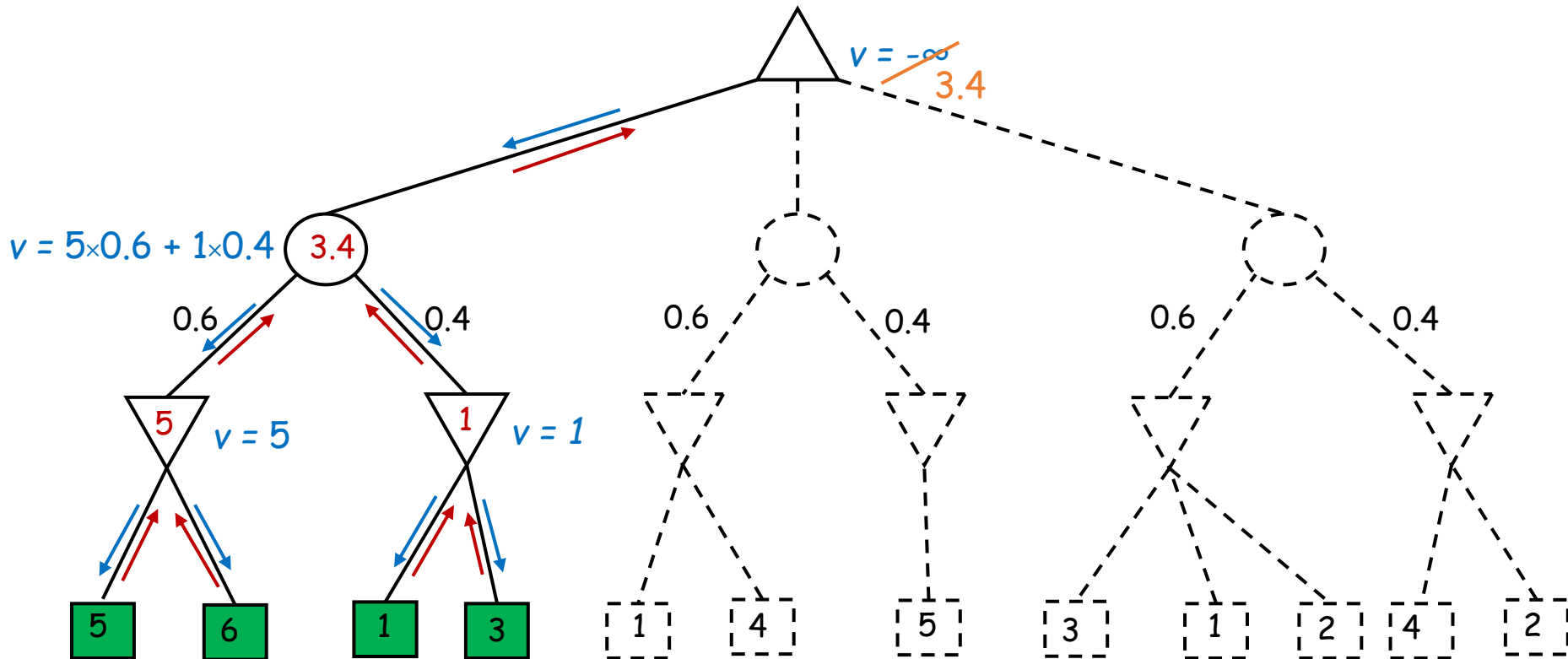
Stochastic games: Example



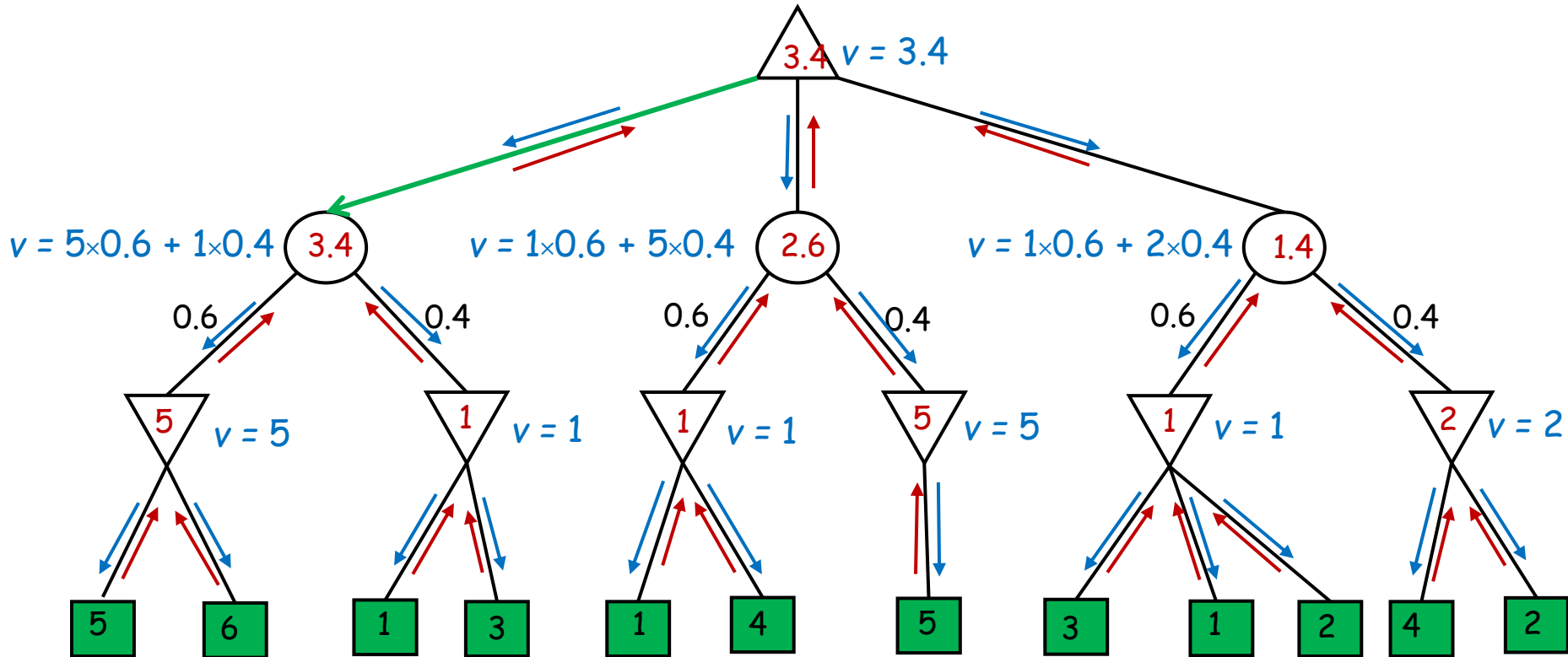
Stochastic games: Example



Stochastic games: Example

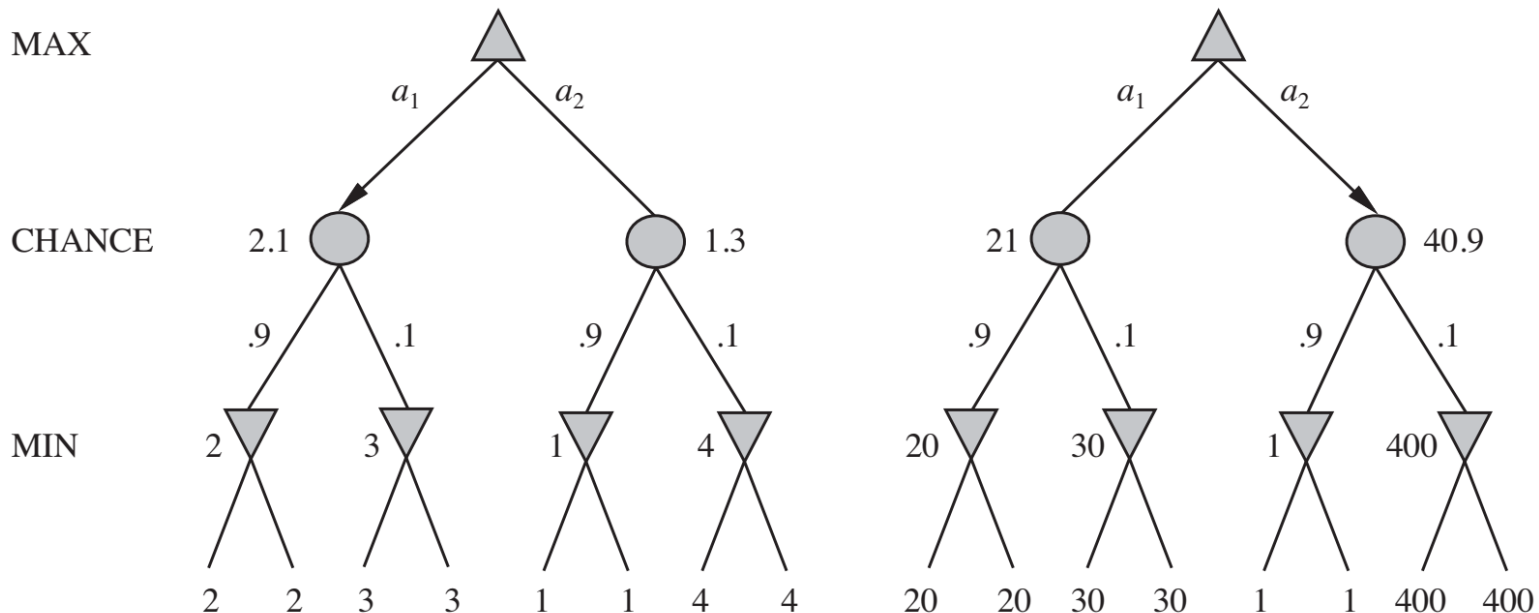


Stochastic games: Example



Evaluation functions for games of chance

- The presence of chance nodes means that one has to be more careful about what the evaluation values mean.
- An **order preserving transformation** on leaf values is not a sufficient condition.
- Evaluation function **must be a positive linear transformation** of the expected utility of a position ($ax+b$ ($a>0$))

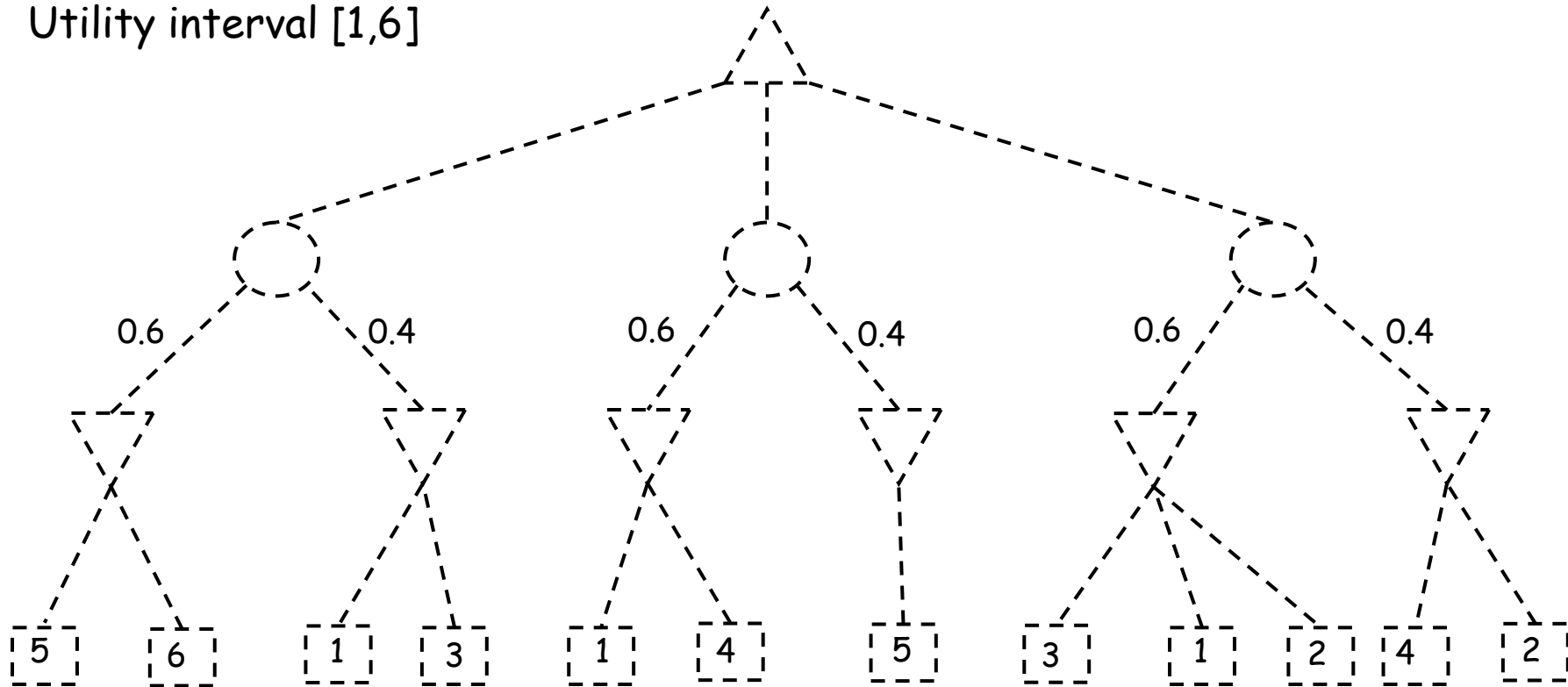


Complexity in stochastic games

- $O(b^m n^m)$
 - Backgammon: $b \approx 20$ (can be up to 4000 for double dice rolls), $n = 21$ (no. of different possible dice rolls)
 - b is the branching factor and m is the maximum depth of the game tree. n is the number of distinct rolls.

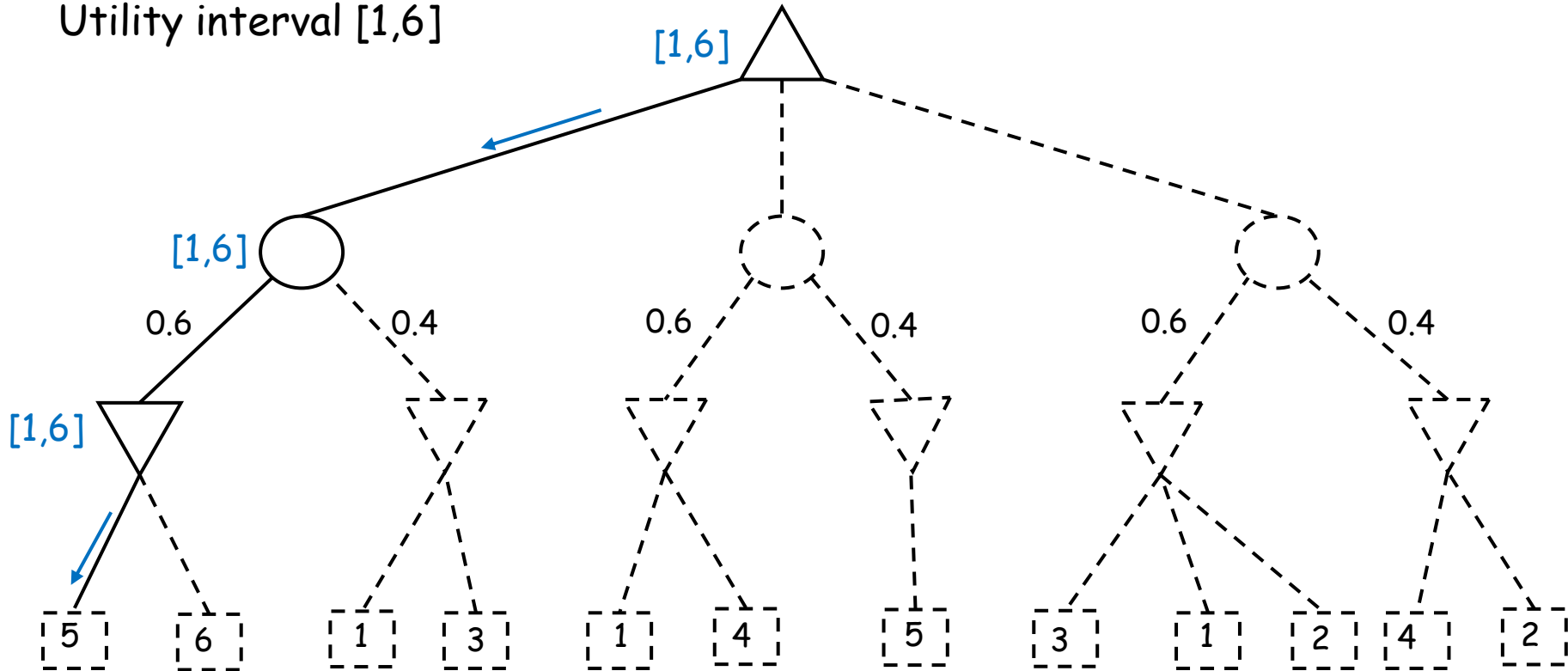
Alpha-Beta Pruning in stochastic games

Utility interval $[1,6]$

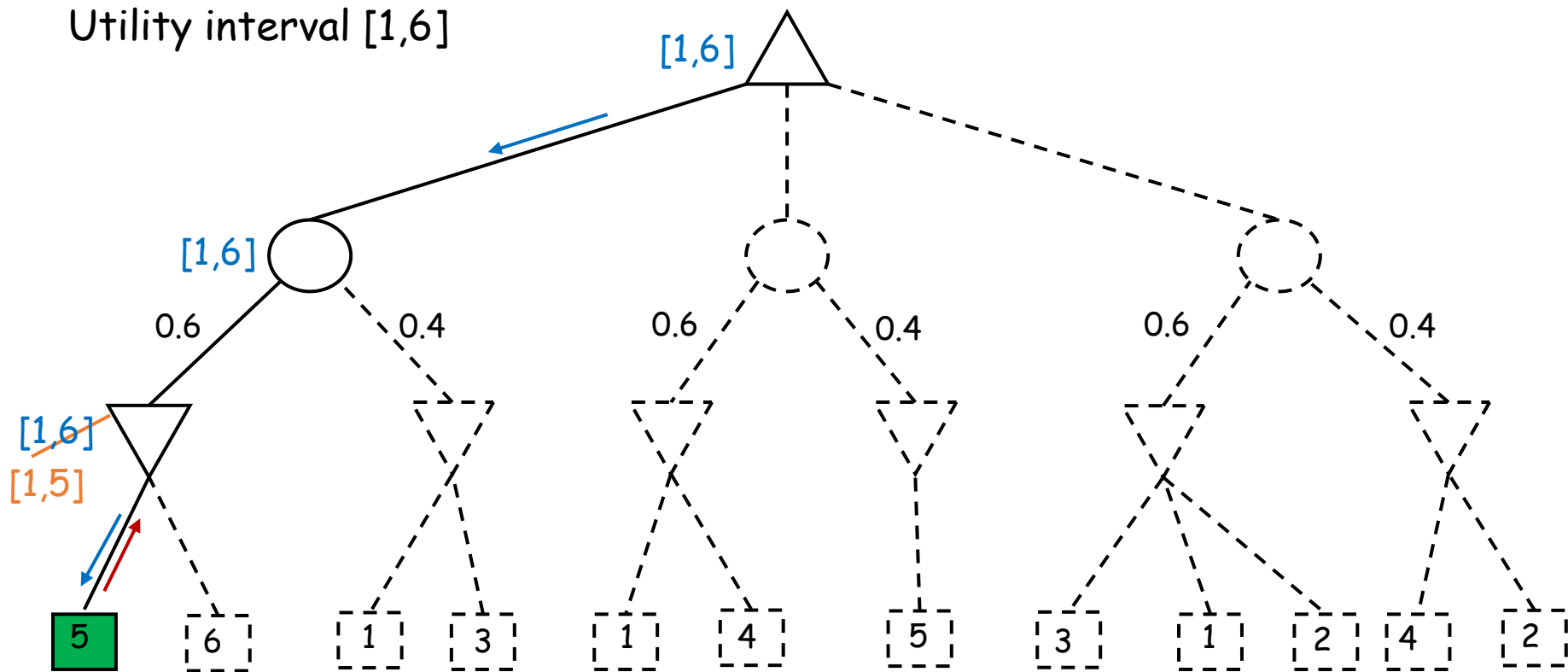


Alpha-Beta Pruning in stochastic games

Utility interval [1,6]

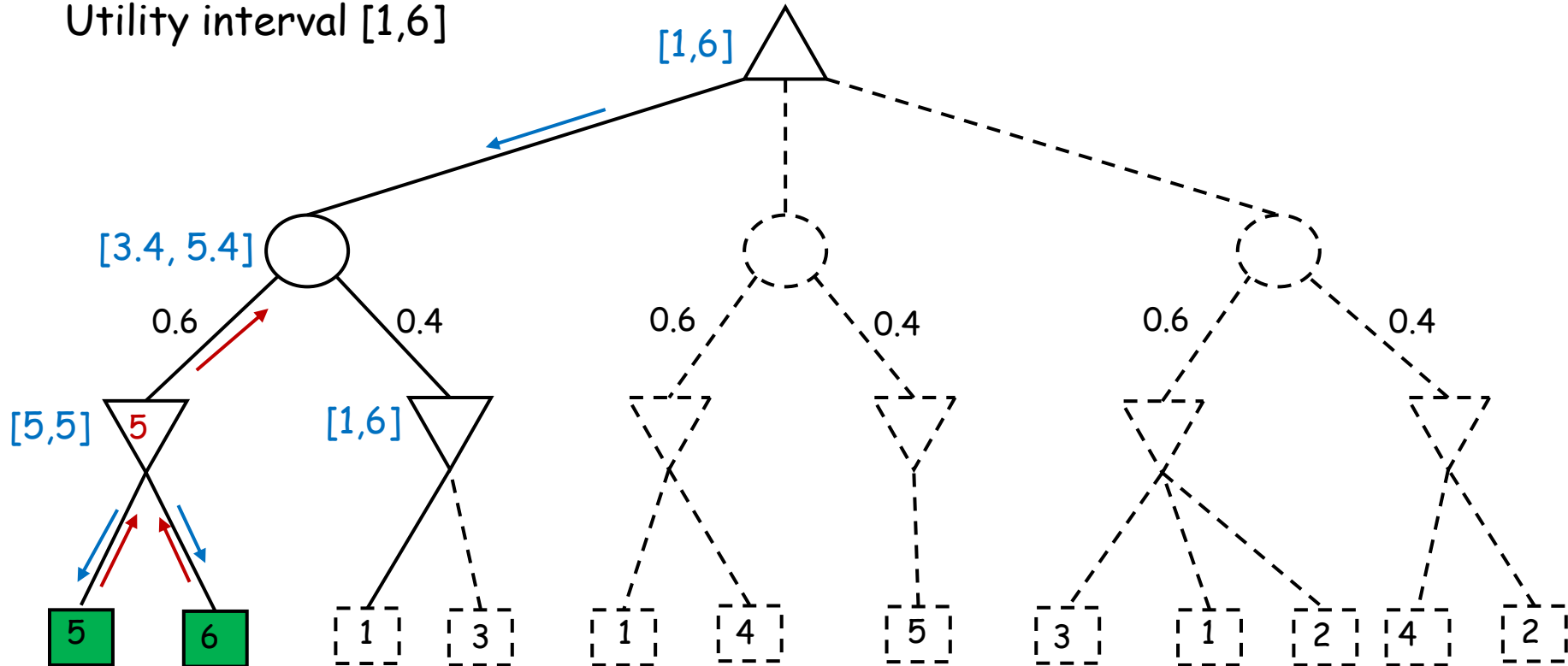


Alpha-Beta Pruning in stochastic games



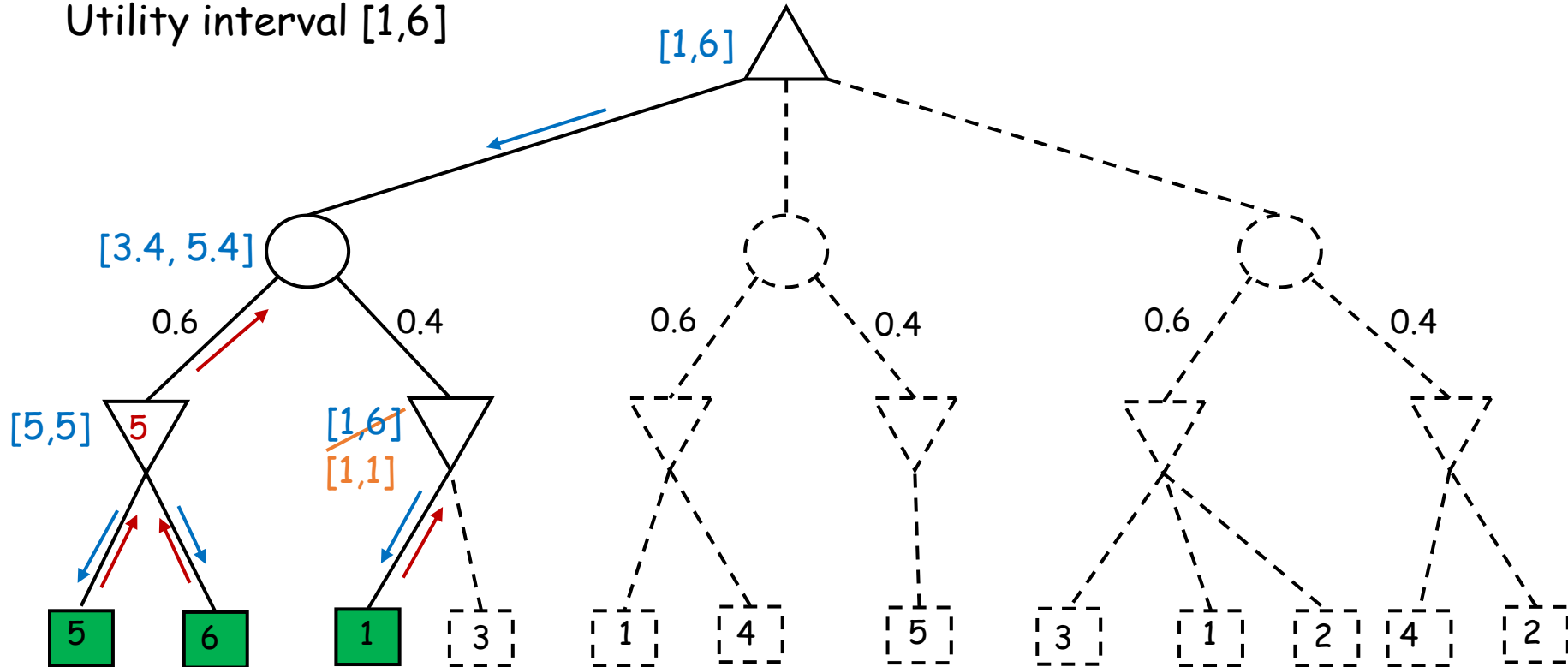
Alpha-Beta Pruning in stochastic games

Utility interval $[1,6]$



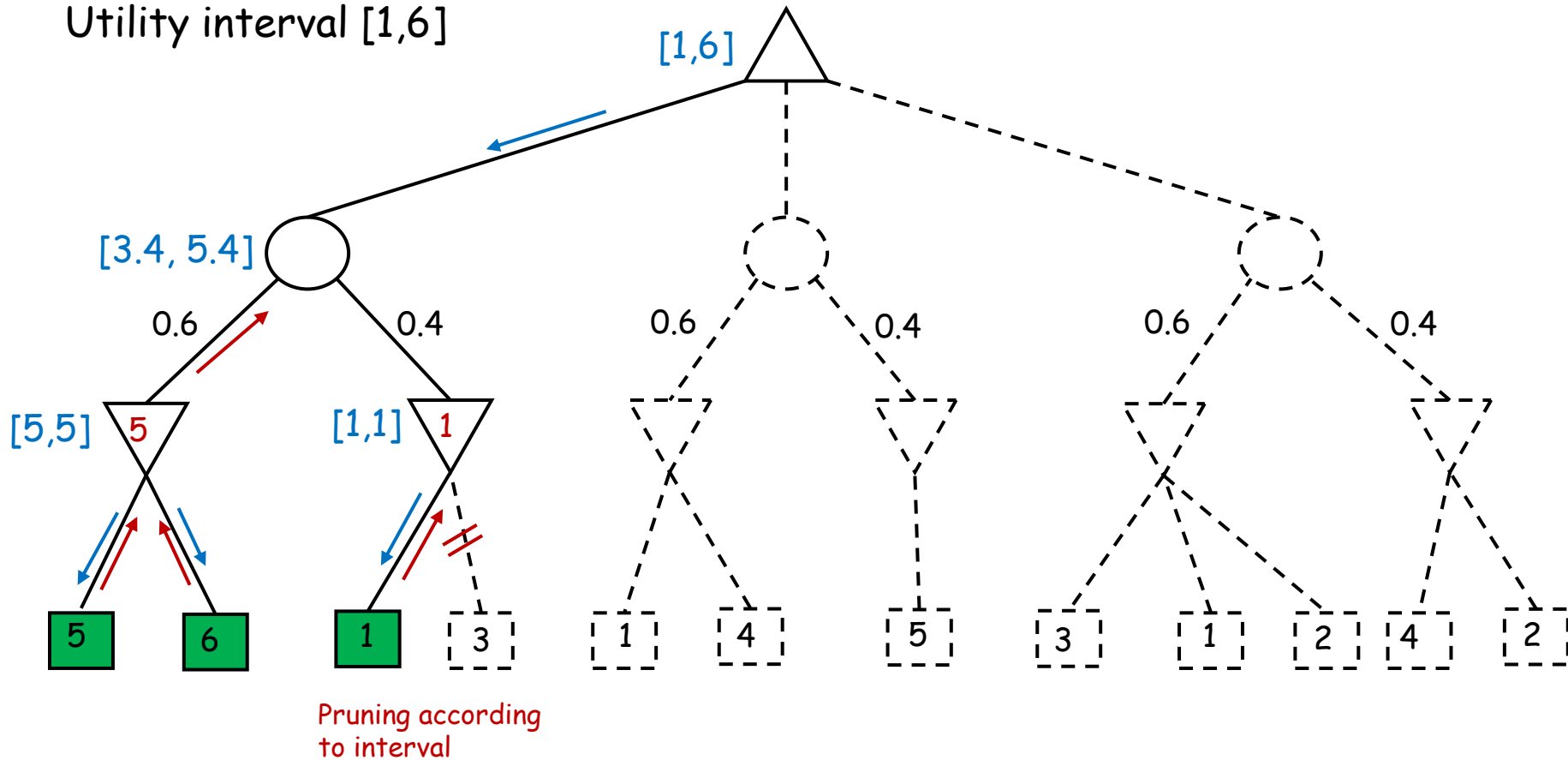
Alpha-Beta Pruning in stochastic games

Utility interval $[1,6]$



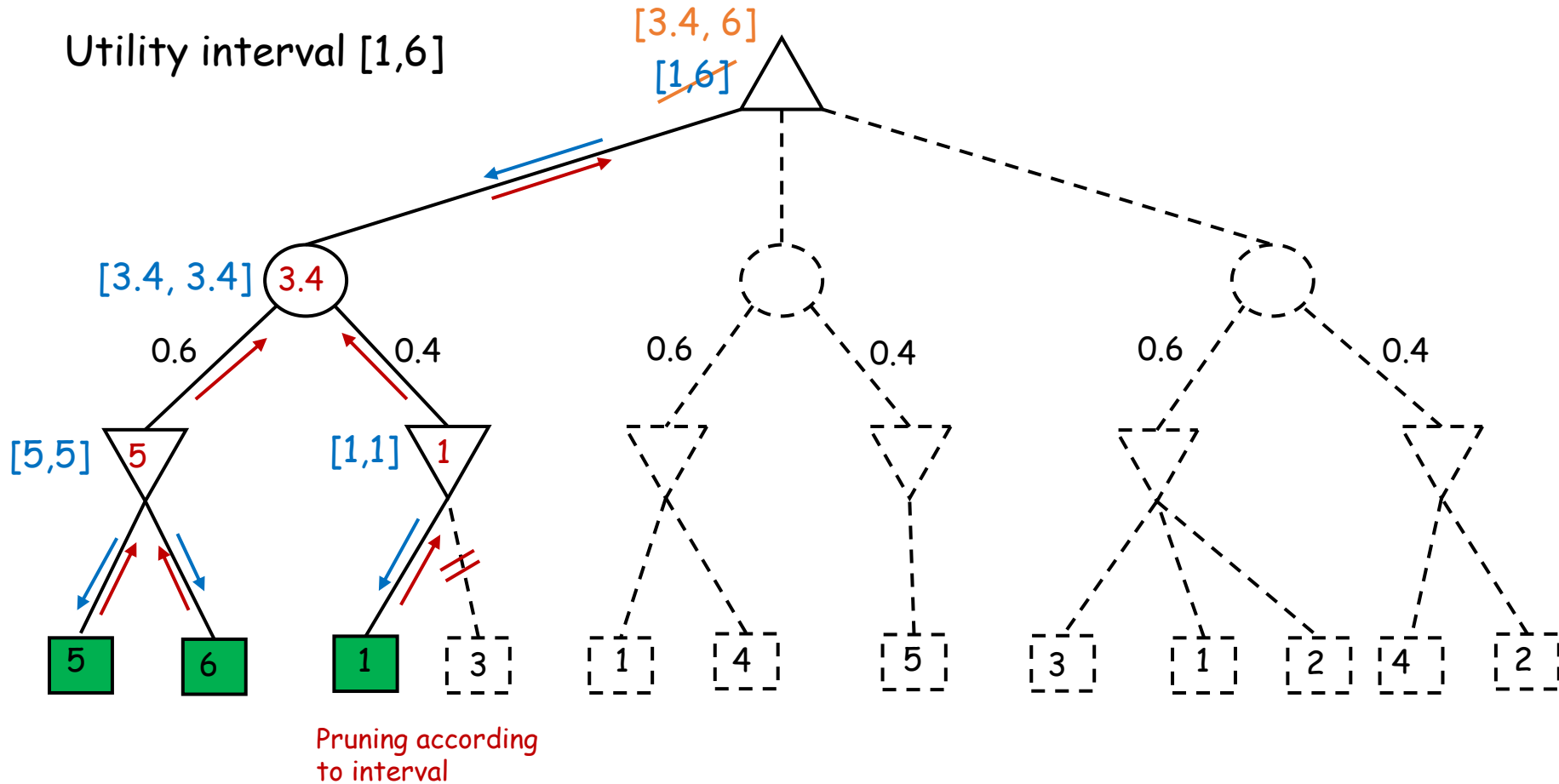
Alpha-Beta Pruning in stochastic games

Utility interval $[1,6]$



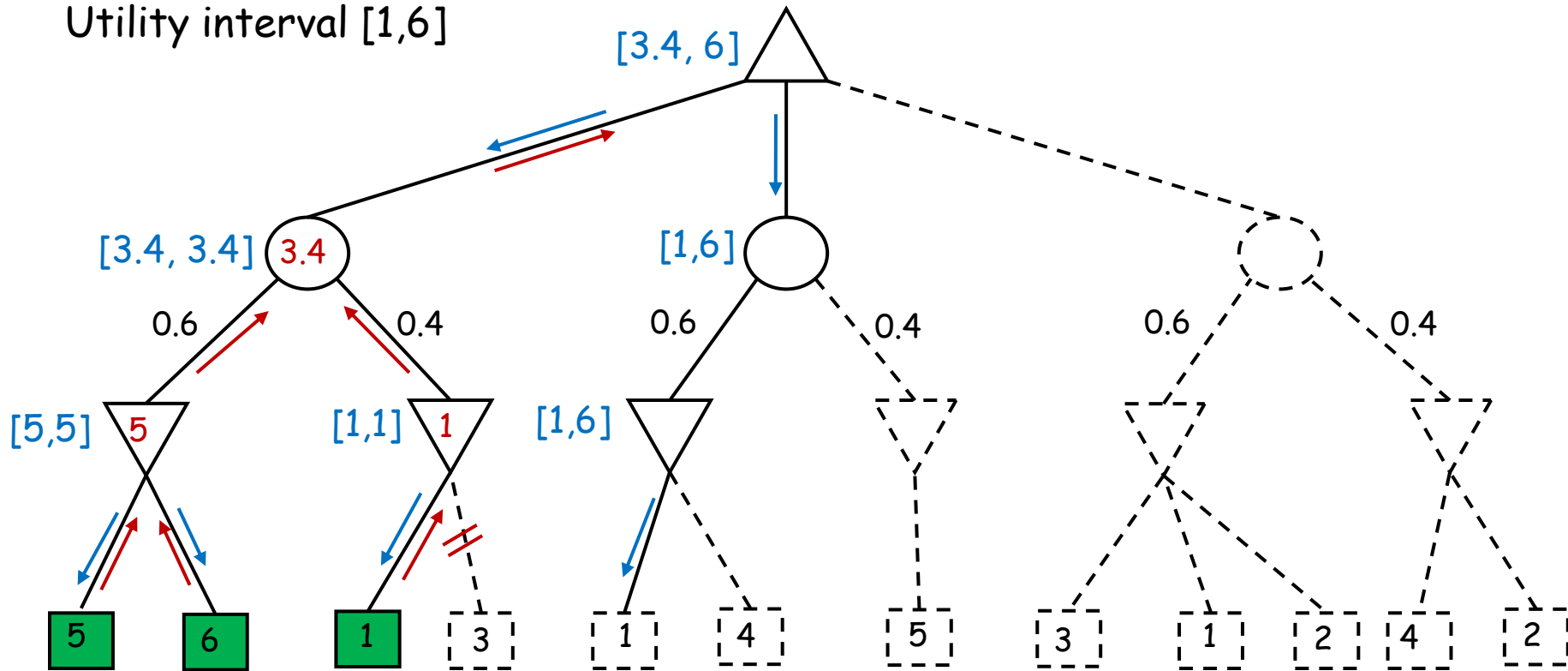
Alpha-Beta Pruning in stochastic games

Utility interval [1,6]



Alpha-Beta Pruning in stochastic games

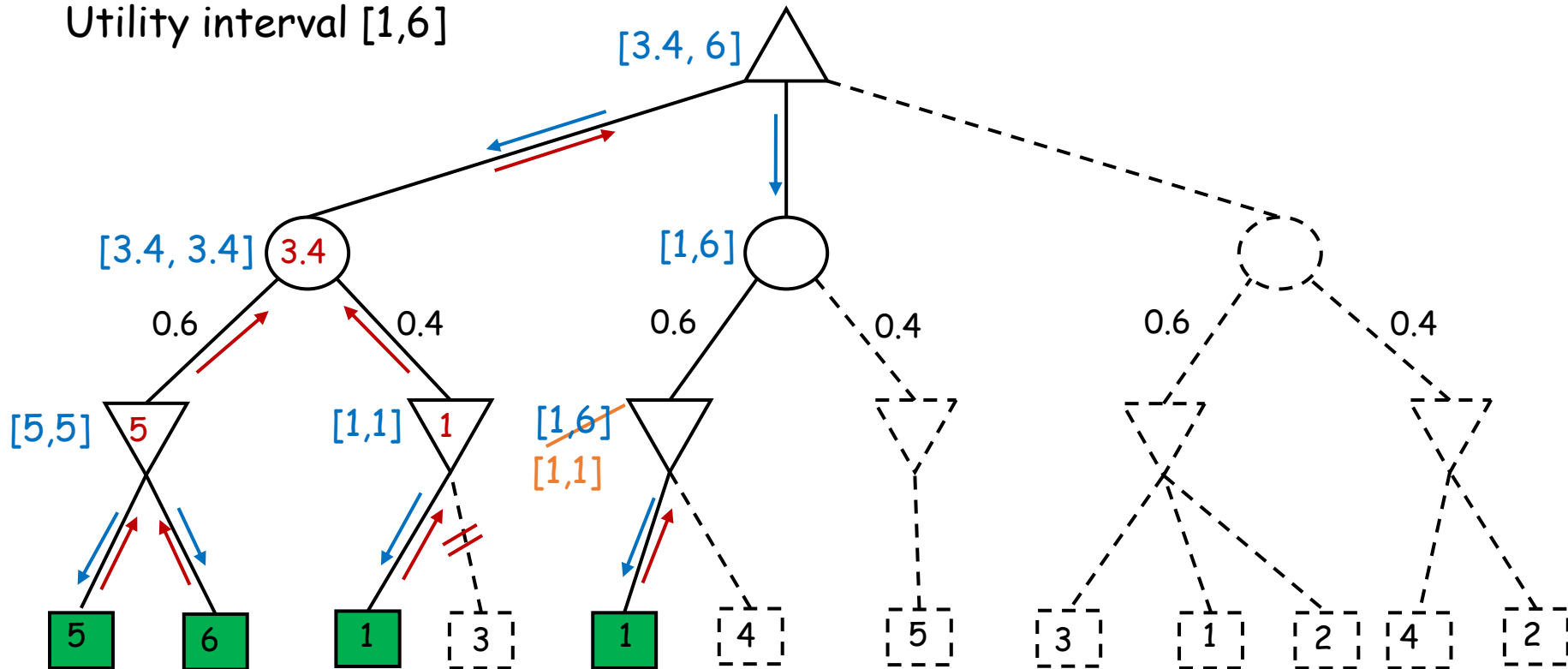
Utility interval $[1,6]$



Pruning according
to interval

Alpha-Beta Pruning in stochastic games

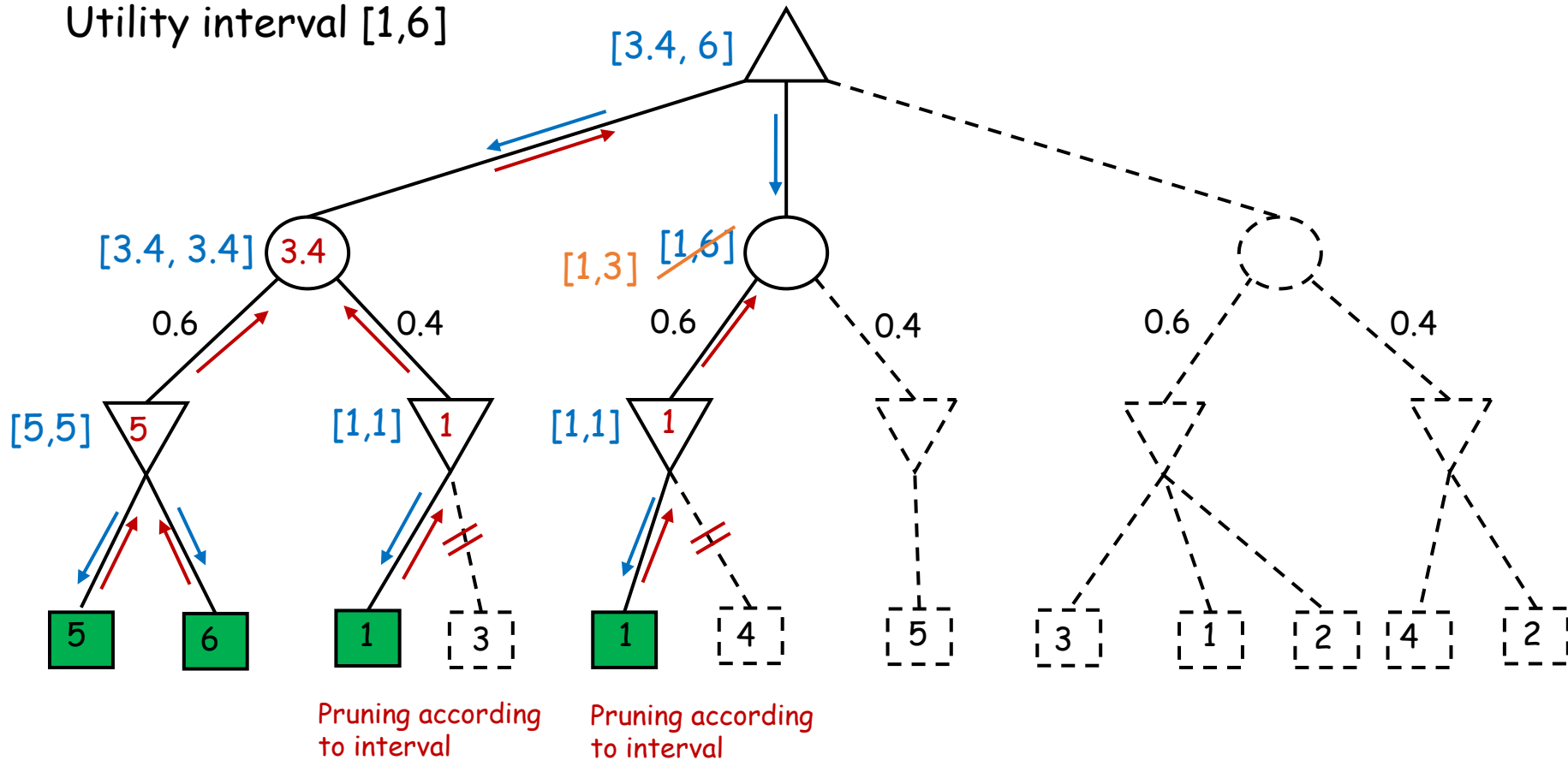
Utility interval $[1,6]$



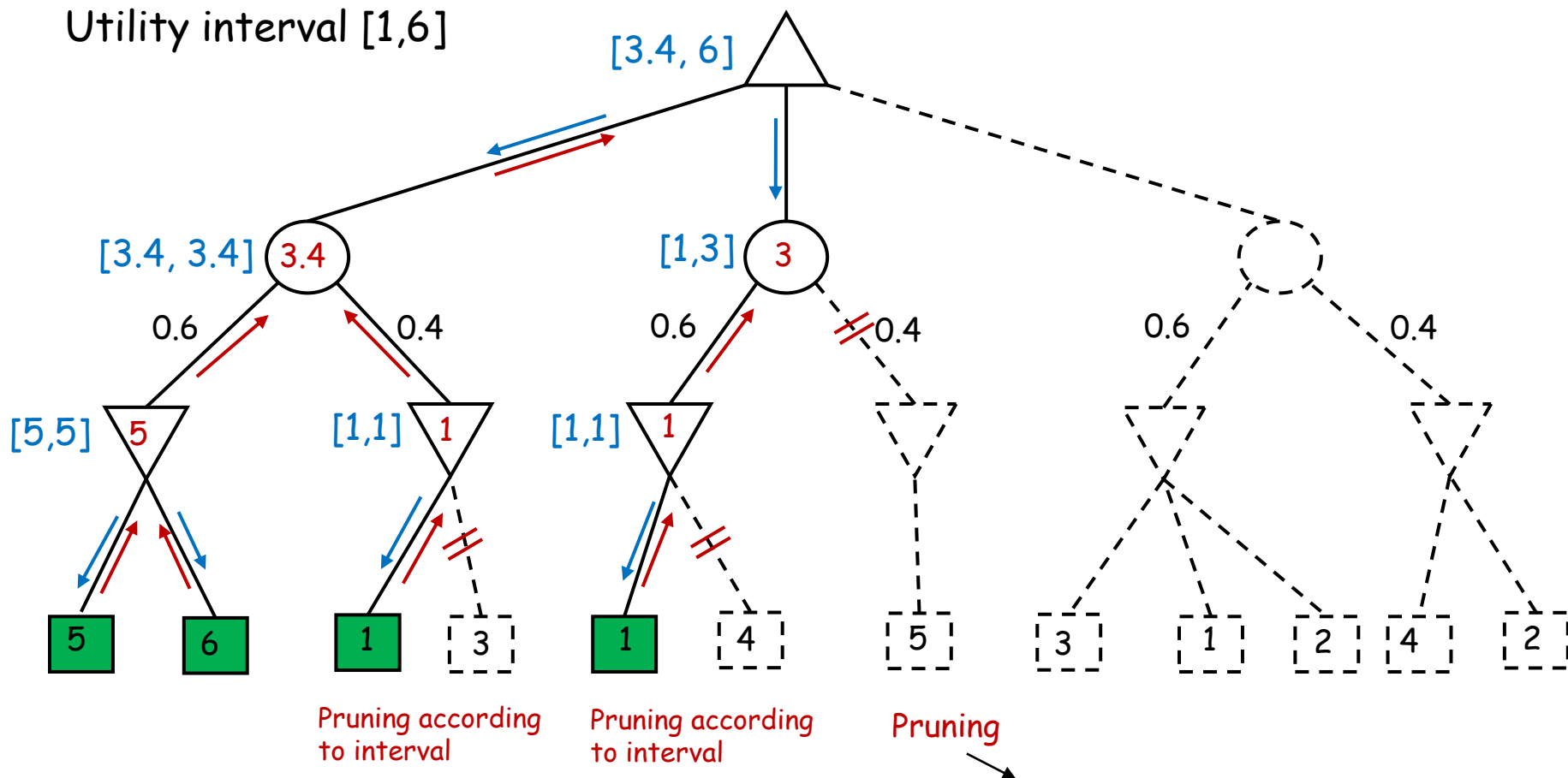
Pruning according
to interval

Alpha-Beta Pruning in stochastic games

Utility interval [1,6]

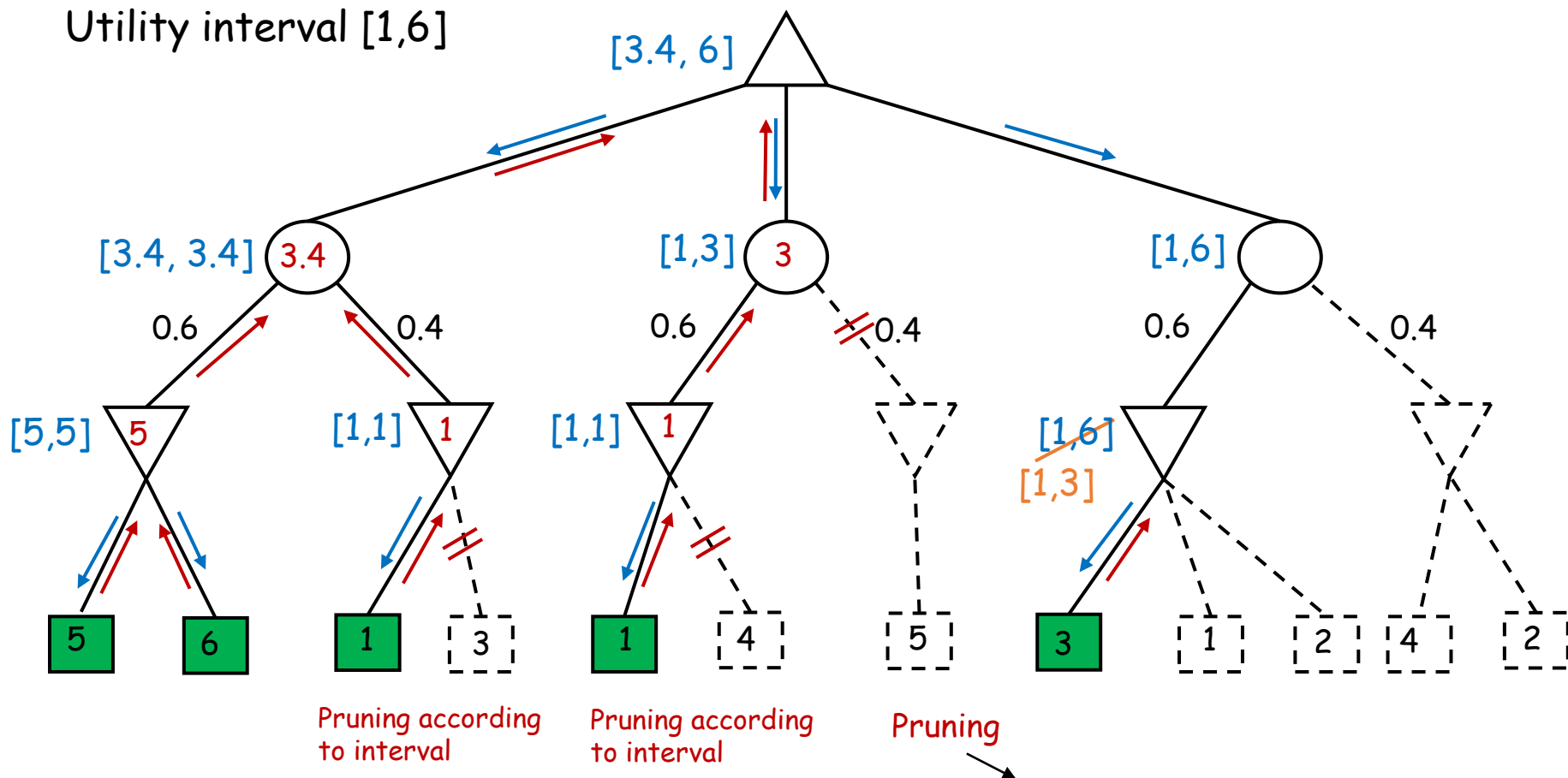


Alpha-Beta Pruning in stochastic games



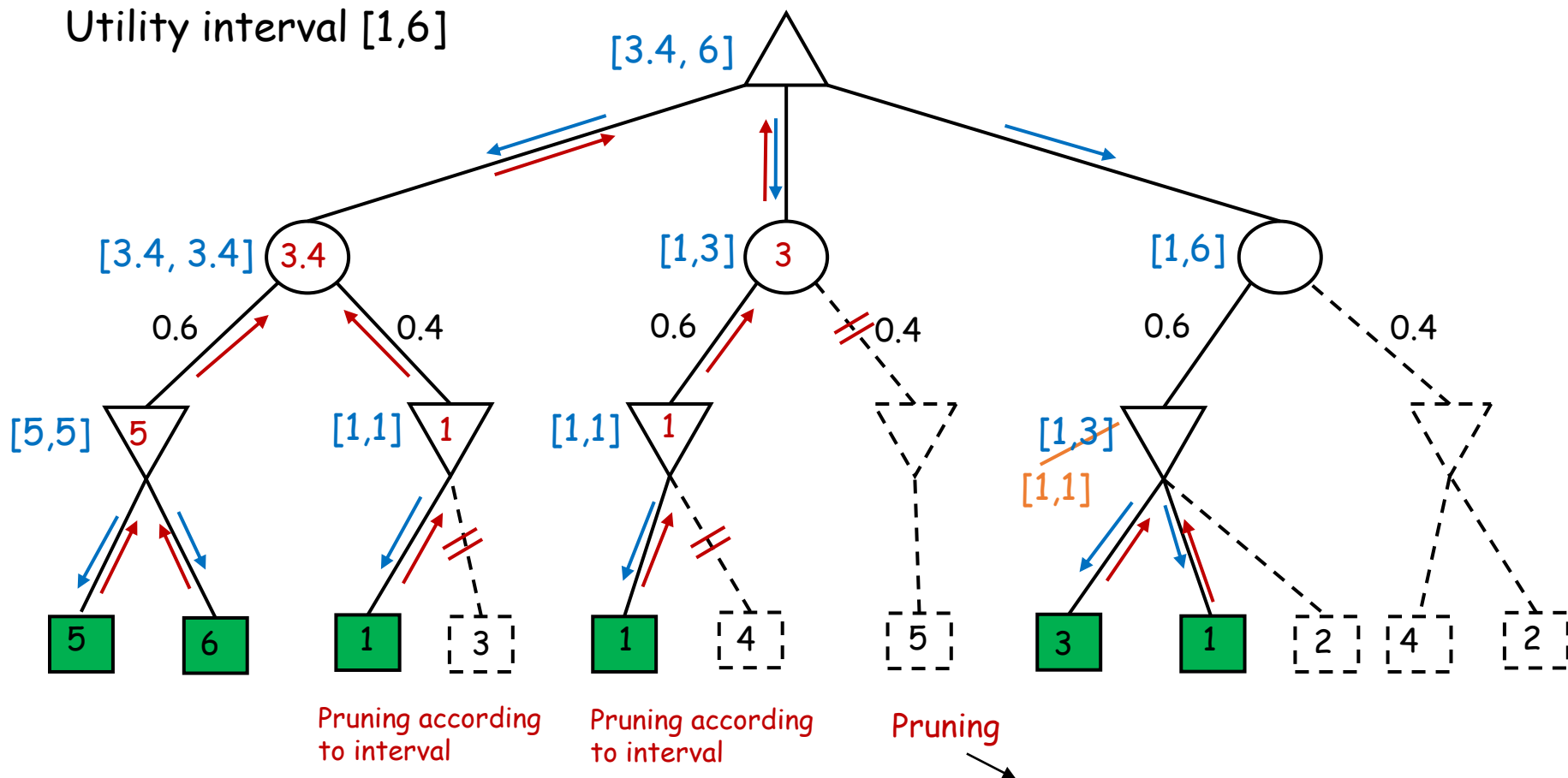
With maximum utility (6), the expected value of chance node is lower than the minimum utility of MAX

Alpha-Beta Pruning in stochastic games



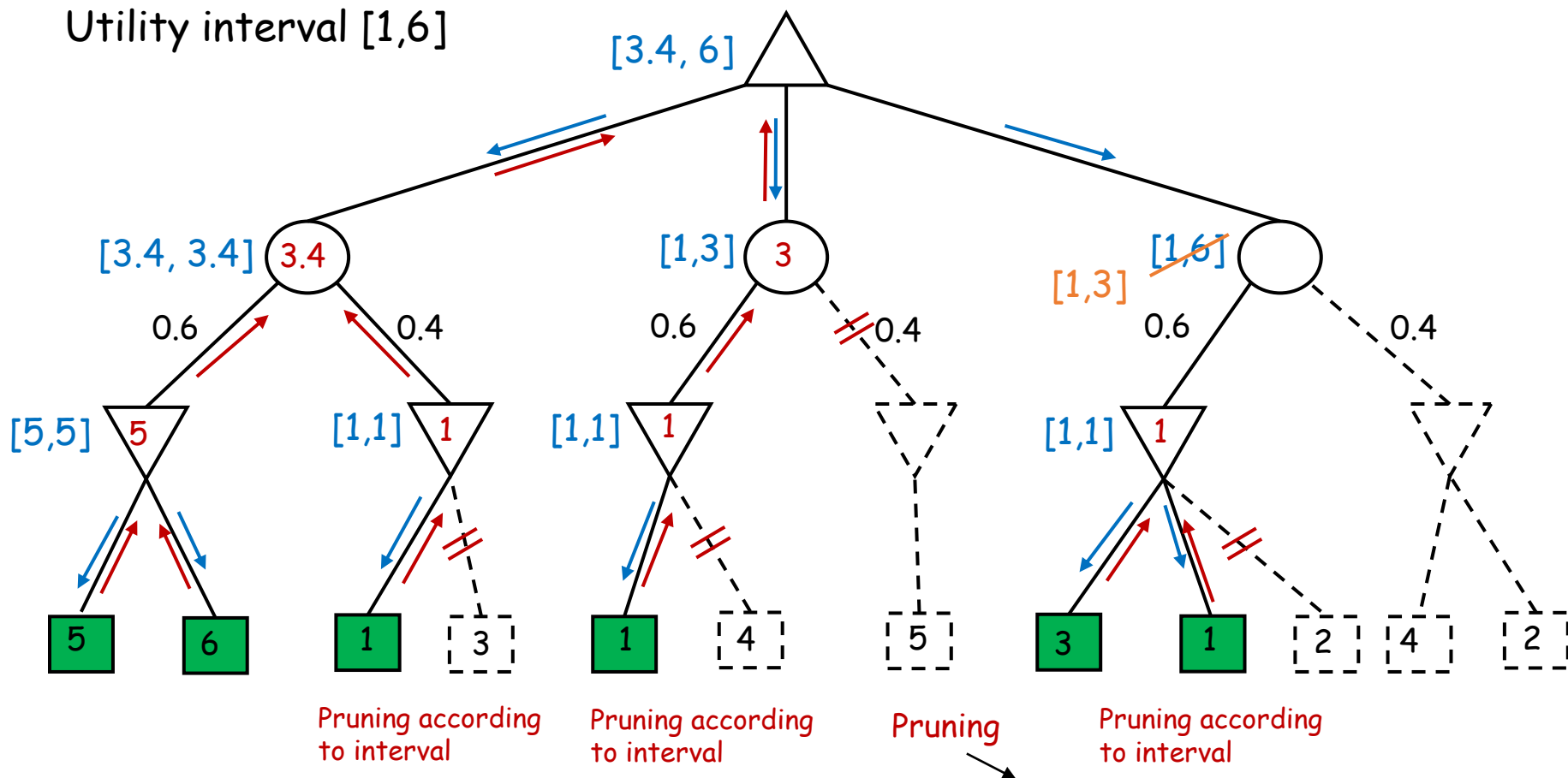
With maximum utility (6), the expected value of chance node is lower than the minimum utility of MAX

Alpha-Beta Pruning in stochastic games



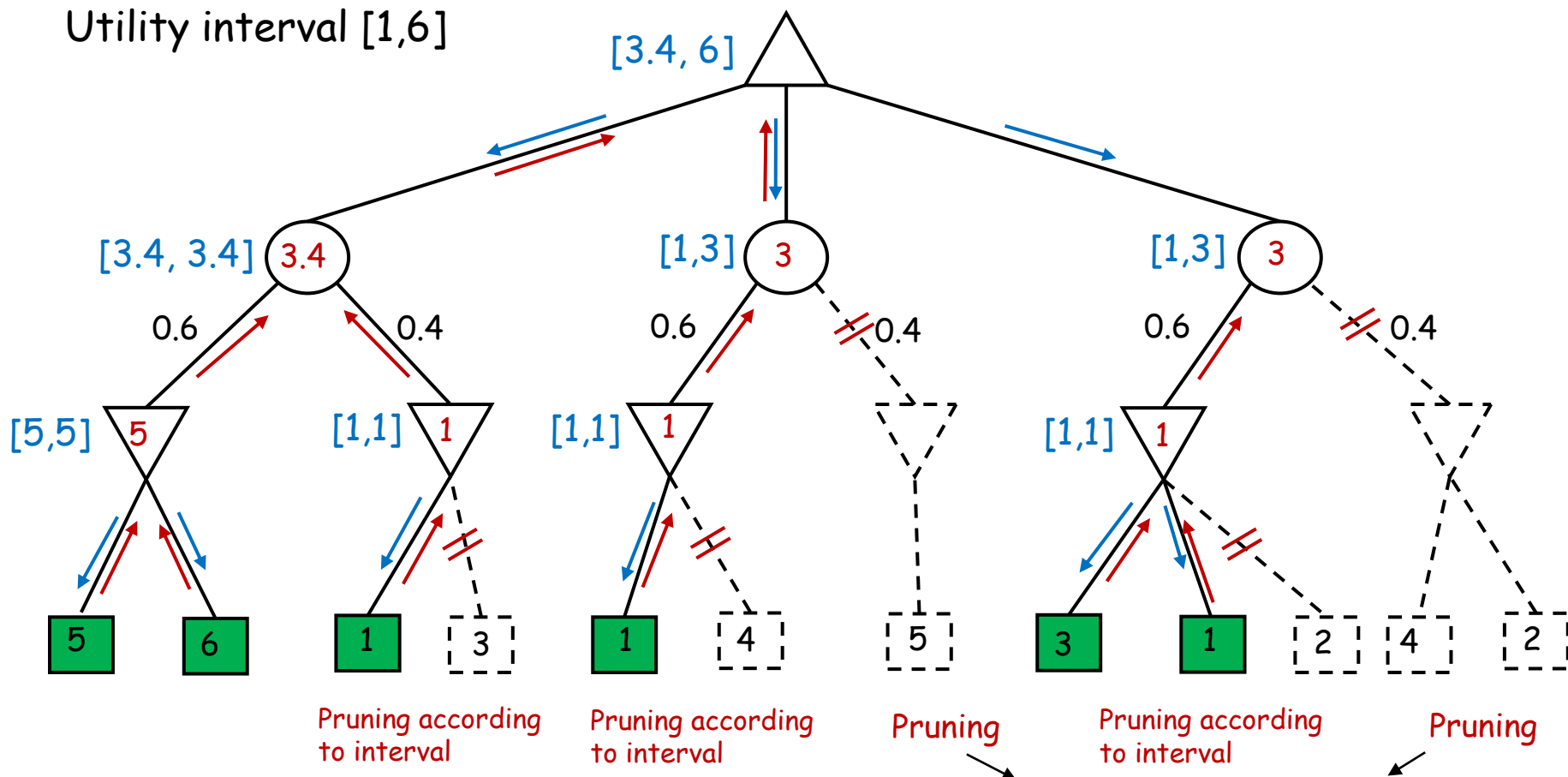
With maximum utility (6), the expected value of chance node is lower than the minimum utility of MAX

Alpha-Beta Pruning in stochastic games



With maximum utility (6), the expected value of chance node is lower than the minimum utility of MAX

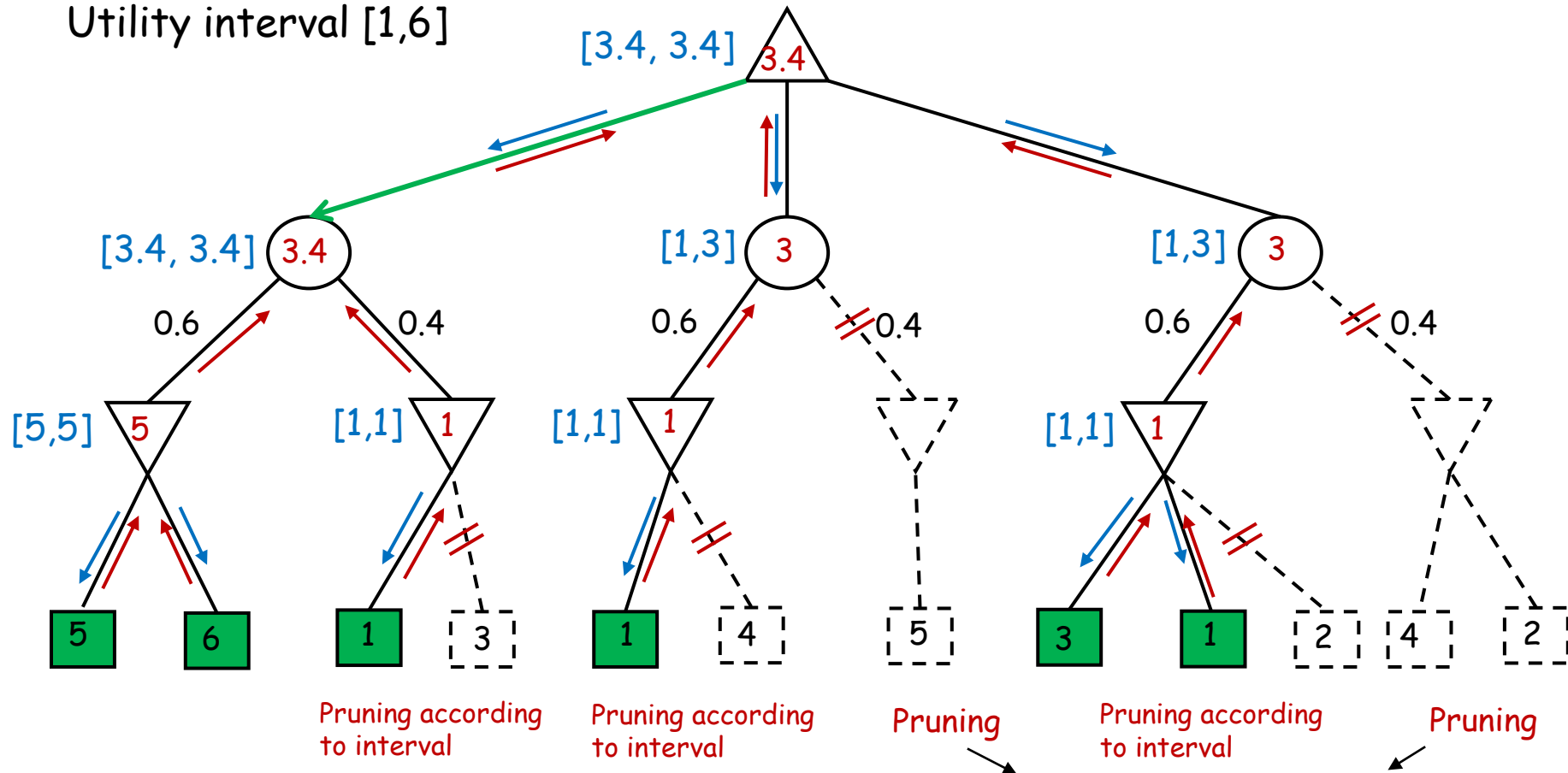
Alpha-Beta Pruning in stochastic games



With maximum utility (6), the expected value of chance node is lower than the minimum utility of MAX

Alpha-Beta Pruning in stochastic games

Utility interval $[1,6]$



With maximum utility (6), the expected value of chance node is lower than the minimum utility of MAX

State-of-the-Art

Checkers: Tinsley vs. Chinook



Name: Marion Tinsley
Profession: Teach mathematics
Hobby: Checkers
Record: Over 42 years
loses only 3 games
of checkers
World champion for over 40
years

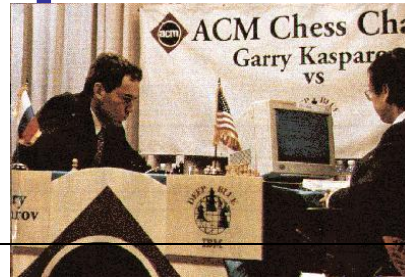
Mr. Tinsley suffered his 4th and 5th losses against Chinook

Chinook



First computer to become official world champion of Checkers!

Chess: Kasparov vs. Deep Blue



Kasparov

Deep Blue

5'10"

Height

6' 5"

176 lbs

Weight

2,400 lbs

34 years

Age

4 years

50 billion neurons

Computers

32 RISC processors
+ 256 VLSI chess engines

2 pos/sec

Speed

200,000,000 pos/sec

Extensive

Knowledge

Primitive

Electrical/chemical

Power Source

Electrical

Enormous

Ego

None

1997: Deep Blue wins by 3 wins, 1 loss, and 2 draws

Chess: Kasparov vs. Deep Junior



Deep Junior

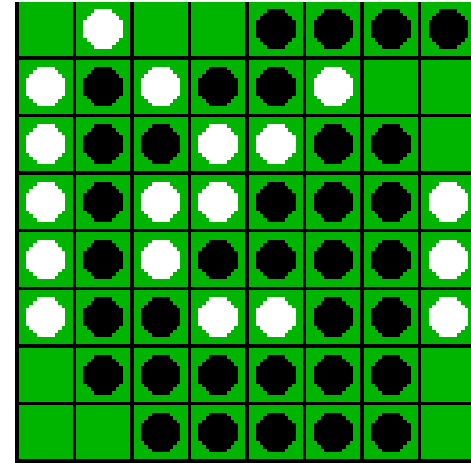
8 CPU, 8 GB RAM, Win 2000
2,000,000 pos/sec
Available at \$100

August 2, 2003: Match ends in a 3/3 tie!

Othello: Murakami vs. Logistello



Takeshi Murakami
World Othello Champion



1997: The Logistello software crushed Murakami
by 6 games to 0

Go: Goemate vs. ??



Name: Chen Zhixing

Profession: Retired

Computer skills:

self-taught programmer

Author of Goemate (arguably the
best Go program available today)



Gave Goemate a 9 stone
handicap and still easily
beat the program,
thereby winning \$15,000

Go: Goemate vs. ??



Name: Chen Zhixing
Profession: Retired
Computer skills:

Go has too high a branching factor
for existing search techniques

Current and future software must
rely on huge databases and pattern-
recognition techniques

thereby winning \$15,000



Secrets

- Many game programs are based on alpha-beta + iterative deepening + extended/singular search + transposition tables + huge databases + ...
- For instance, Chinook searched all checkers configurations with 8 pieces or less and created an endgame database of 444 billion board configurations
- The methods are general, but their implementation is dramatically improved by many specifically tuned-up enhancements (e.g., the evaluation functions) like an F1 racing car

Perspective on Games: Con and Pro

Chess is the *Drosophila* of artificial intelligence. However, computer chess has developed much as genetics might have if the geneticists had concentrated their efforts starting in 1910 on breeding racing *Drosophila*. We would have some science, but mainly we would have very fast fruit flies.

John McCarthy

Saying Deep Blue doesn't really think about chess is like saying an airplane doesn't really fly because it doesn't flap its wings.

Drew McDermott