CS3243 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

ADVERSARIAL SEARCH

nicholas.teh@u.nus.edu

MIDTERM TOPICS

- Modelling a Search Problem / Uninformed Search
- Informed Search
- Local Search
- Adversarial Search

- **Week 7 Saturday (6 Mar 0830 - 1000)**
- **5 Questions, 45 marks**

JUST A NOTE

- You don't have to know/trace the *pseudocode*. Just use the technique that we'll be going through today.
- You don't have to know the time/space complexity of alpha-beta pruning.
- Those are all not important, both in practice and in our course.

CS3243 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

CONTENT SUMMARY

nicholas.teh@u.nus.edu

KEY CONCEPTS

- **Adversarial Search**
  - Tracing tree with Alpha-Beta pruning algorithm
    - Detect nodes that are pruned/not evaluated
      - Knowing values/range of values for which the nodes will/ will not be pruned
      - Knowing what happens when some values change (could solve by brute force) **Key is to be fast!**
- The Minimax algorithm

CS3243 INTRODUCTION TO ARTIFICIAL INTELLIGENCE

ADVERSARIAL SEARCH

nicholas.teh@u.nus.edu

MASTERING ADVERSARIAL SEARCH GRAPHS

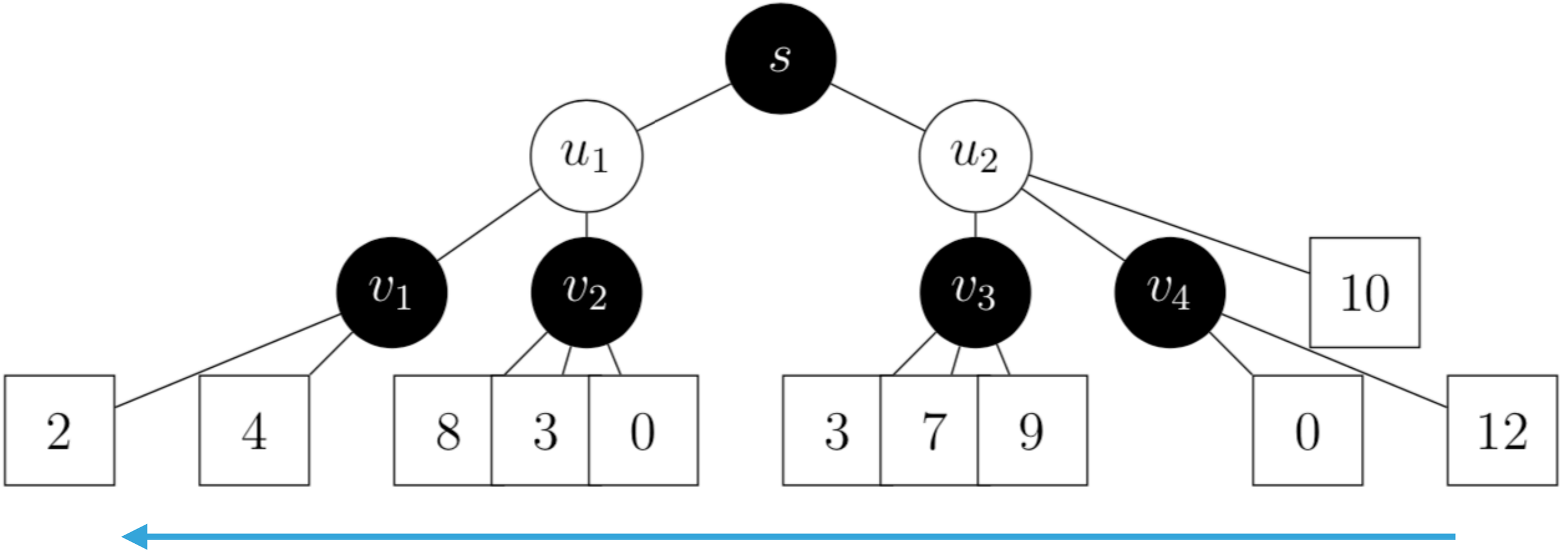
- α: worst case (lower bound) for MAX α ≥ #
  β: worst case (upper bound) for MIN β ≤ #
- Have α for every MAX node, start with -∞
  Have β for every MIN node, start with ∞
- When propagating upwards (or deep-compare),
  PRUNING: If [child] α ≥ β [parent], then prune child's remaining
  PRUNING: If [child] β ≤ α [parent], then prune child's remaining
  If prune, value don't copy upwards
- Compare all the way up for conflict if **deep tree** (or simply copy values down) - more on this later.

MASTERING ADVERSARIAL SEARCH GRAPHS

- Comparison vs Update
  - Compare finish all the arcs of a node before update parent (and only update **if no pruning occurred**)
- Note the direction of evaluation (left to right or right to left)
- Equality = Prune

TUTORIAL 4 QUESTION 3

- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.

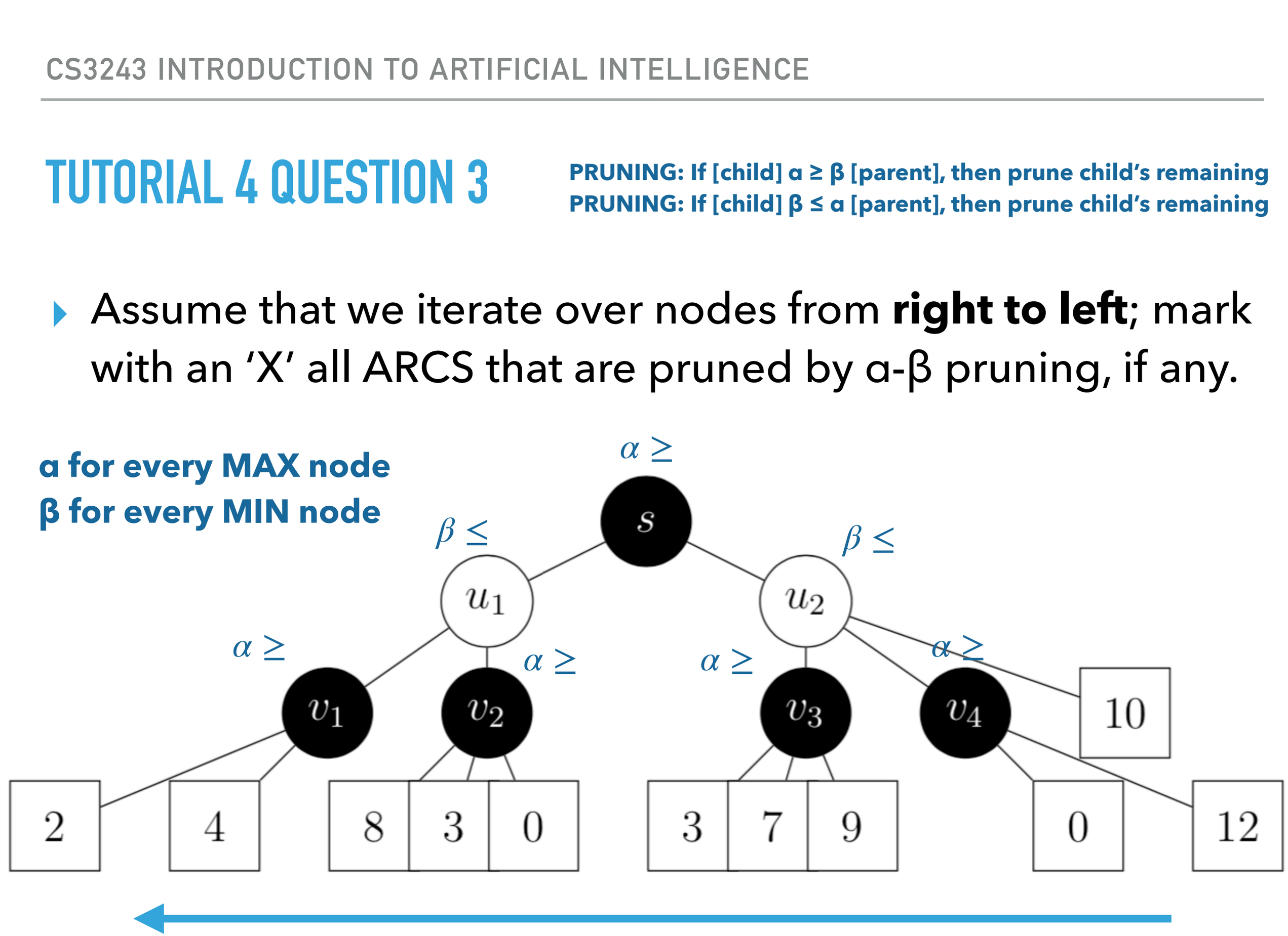


TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.

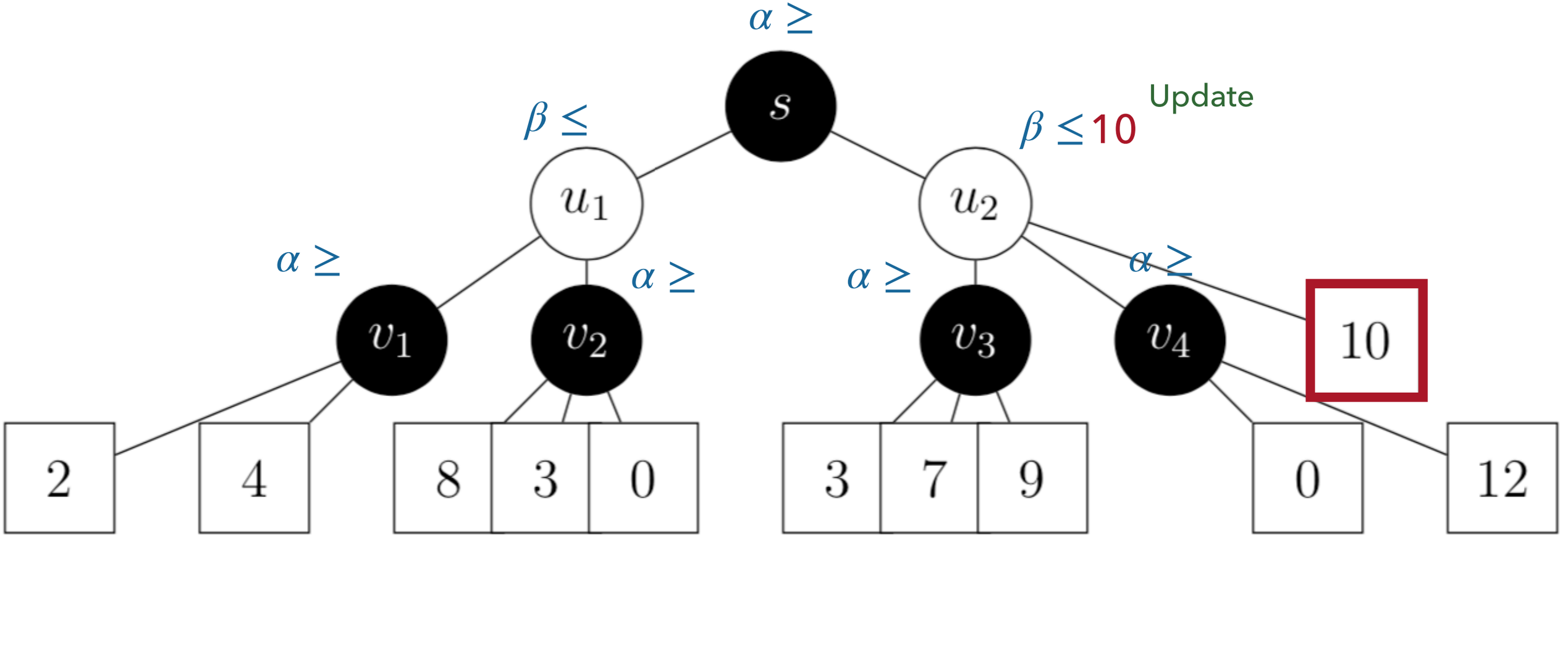
α for every MAX node
β for every MIN node



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

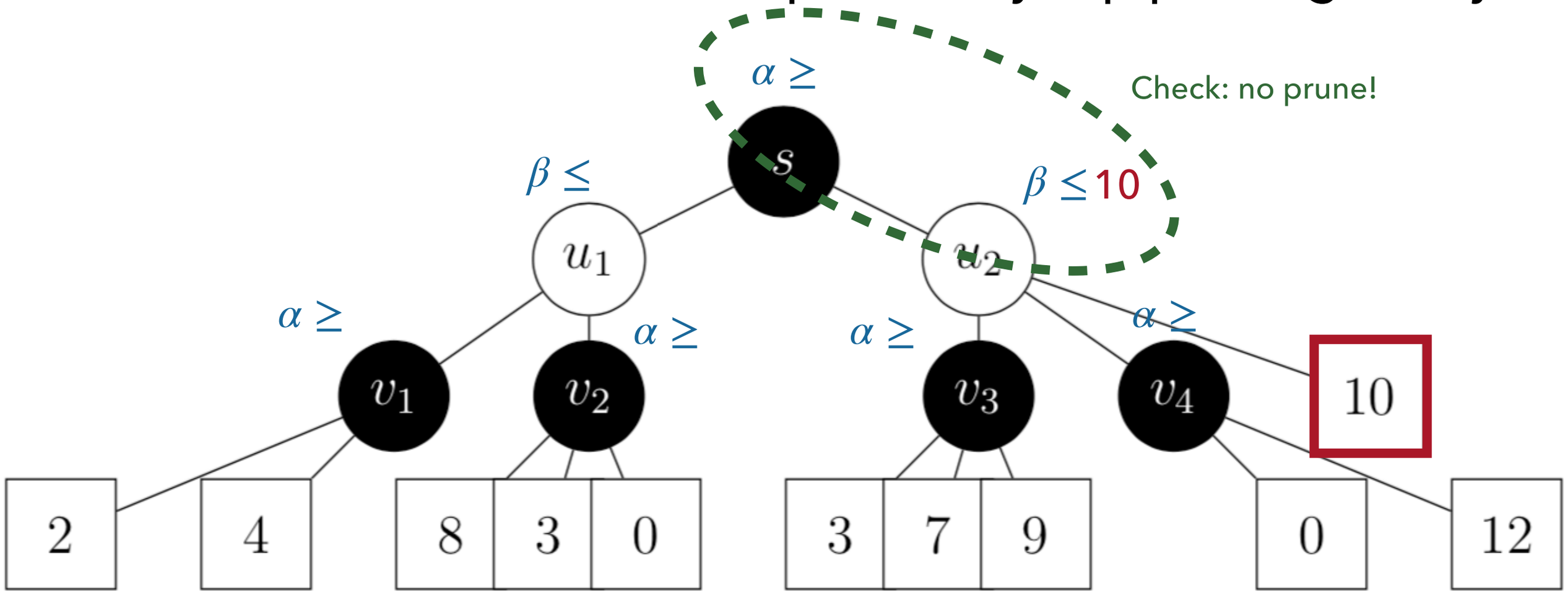
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

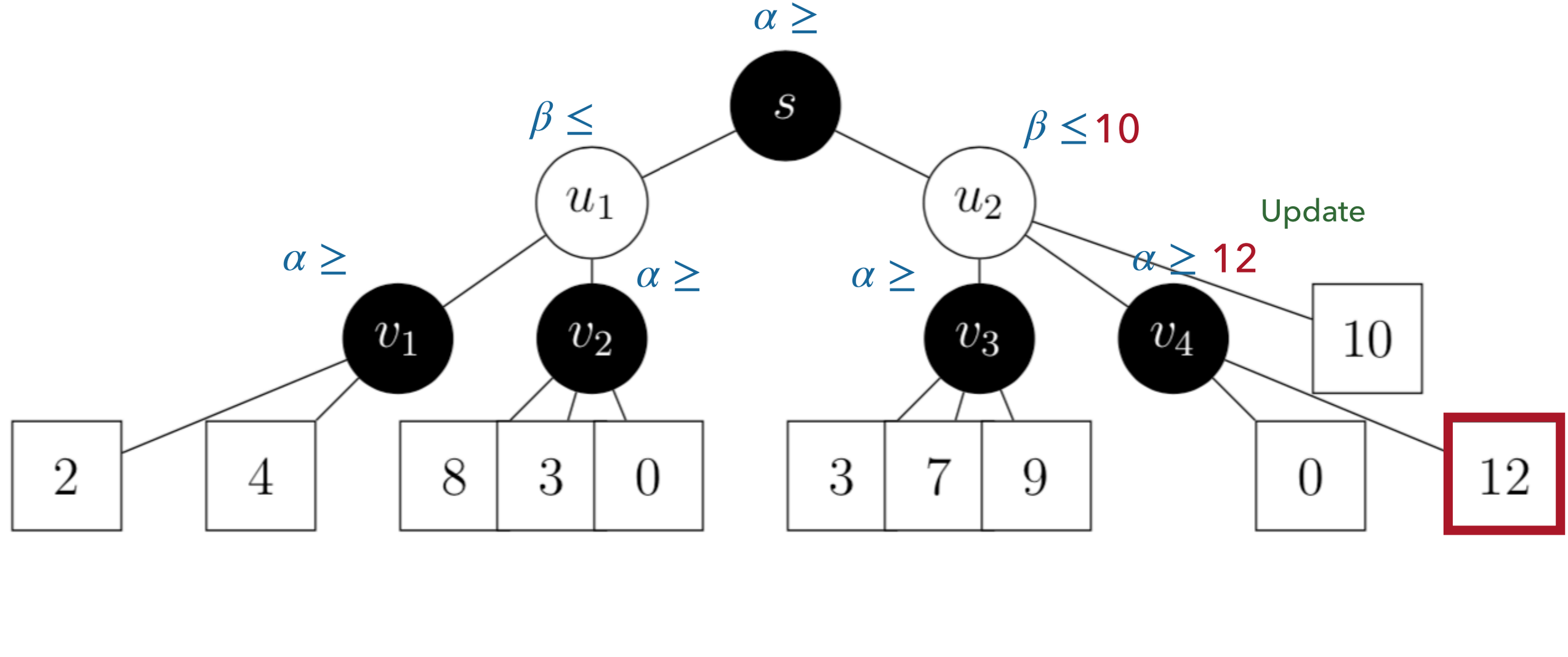
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

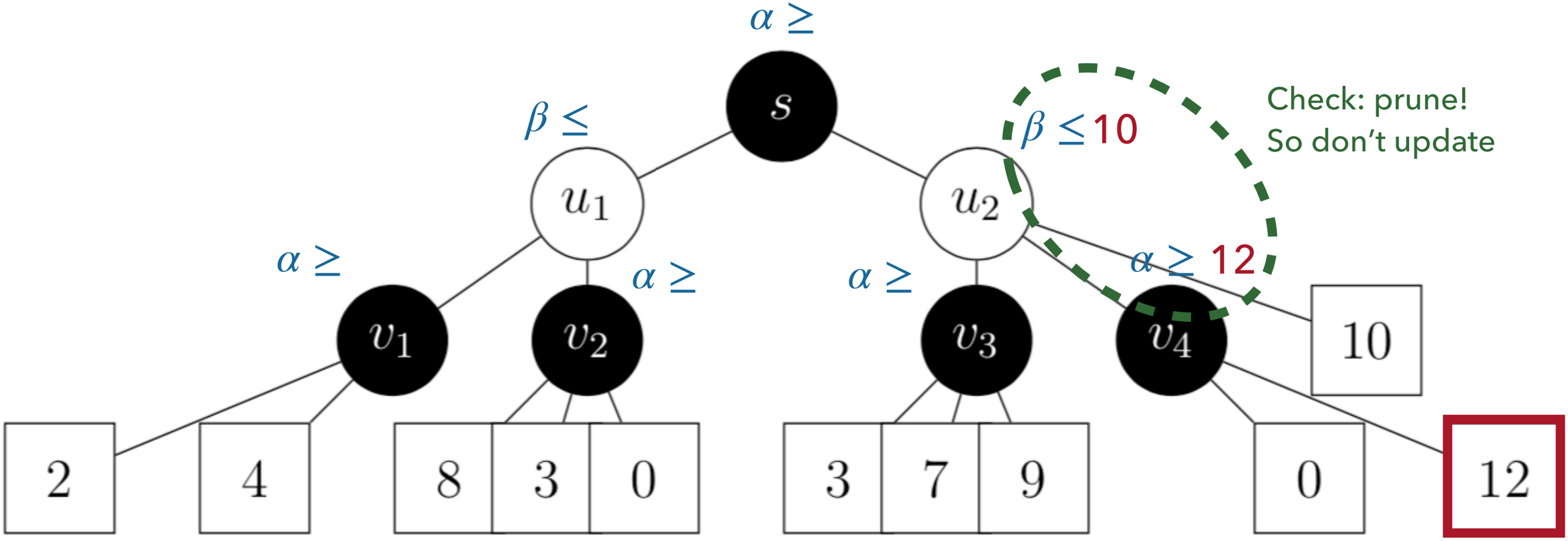
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

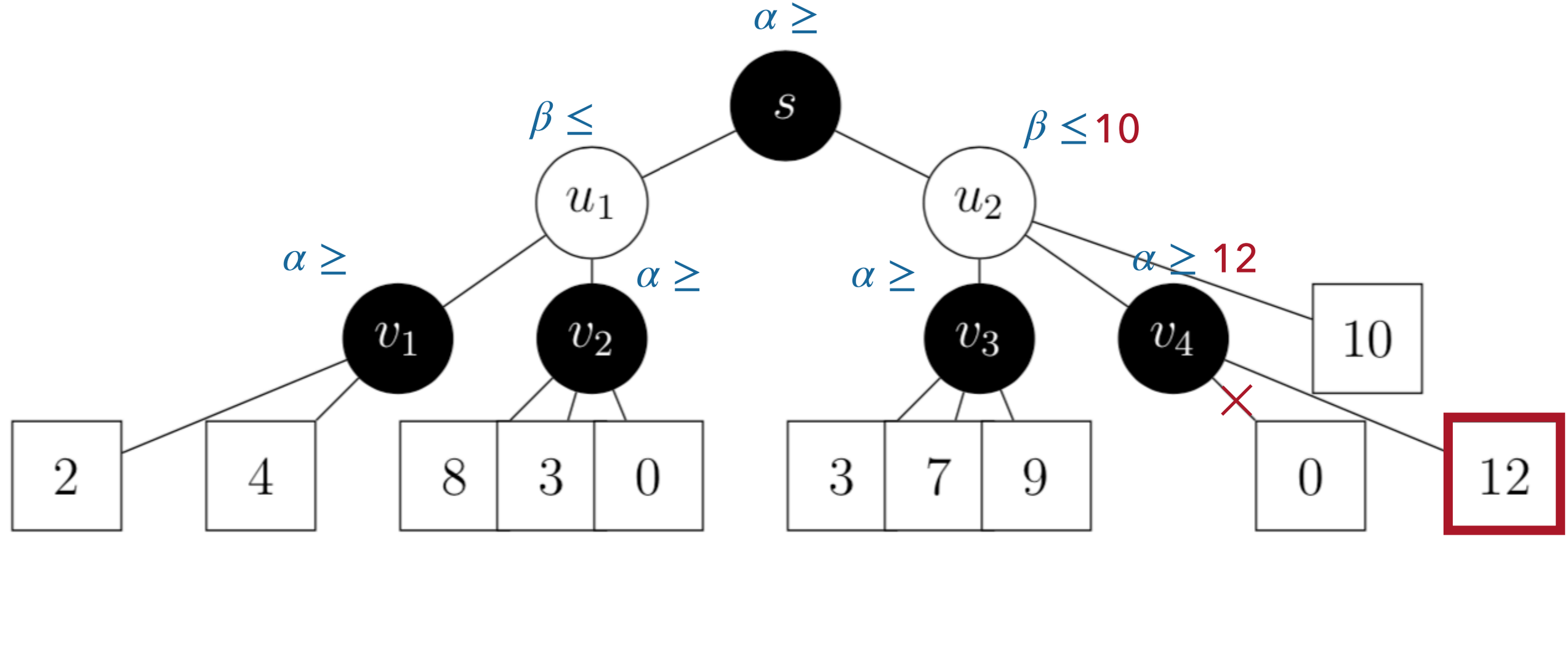
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

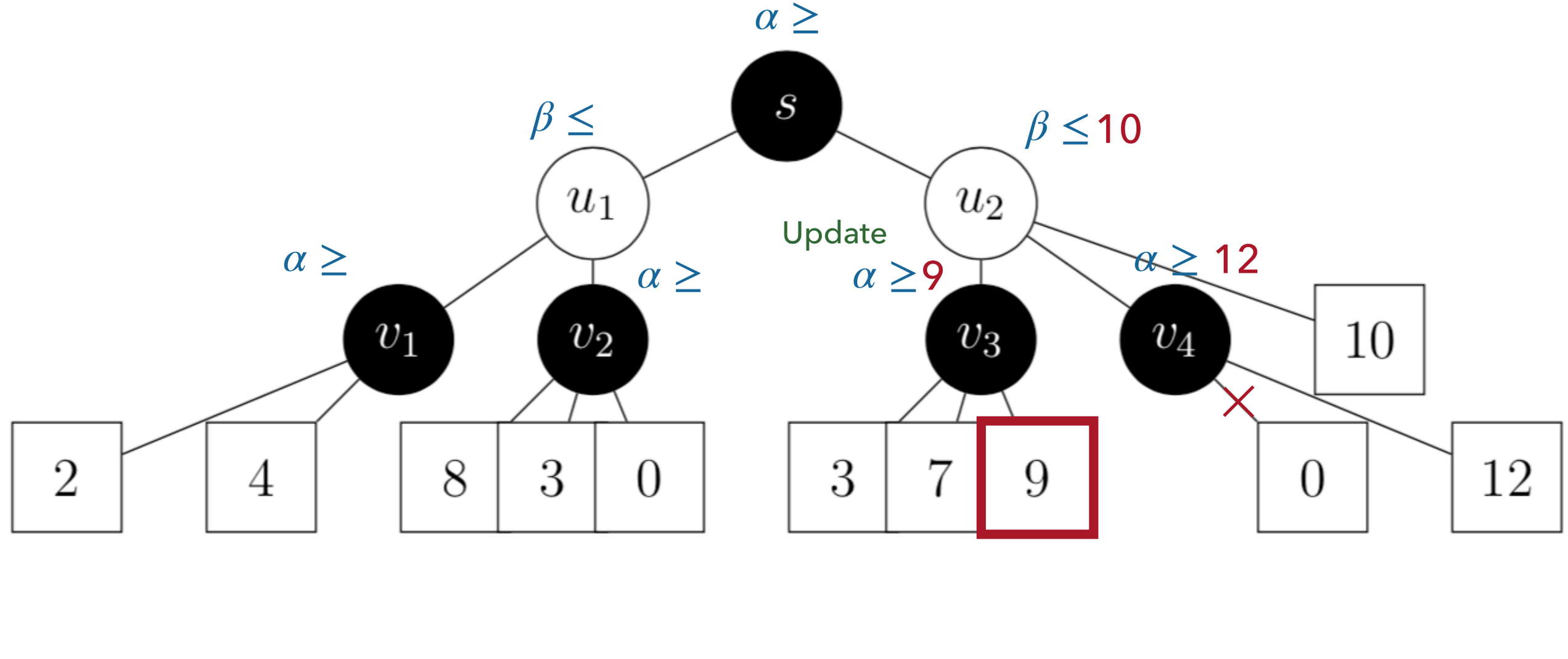
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

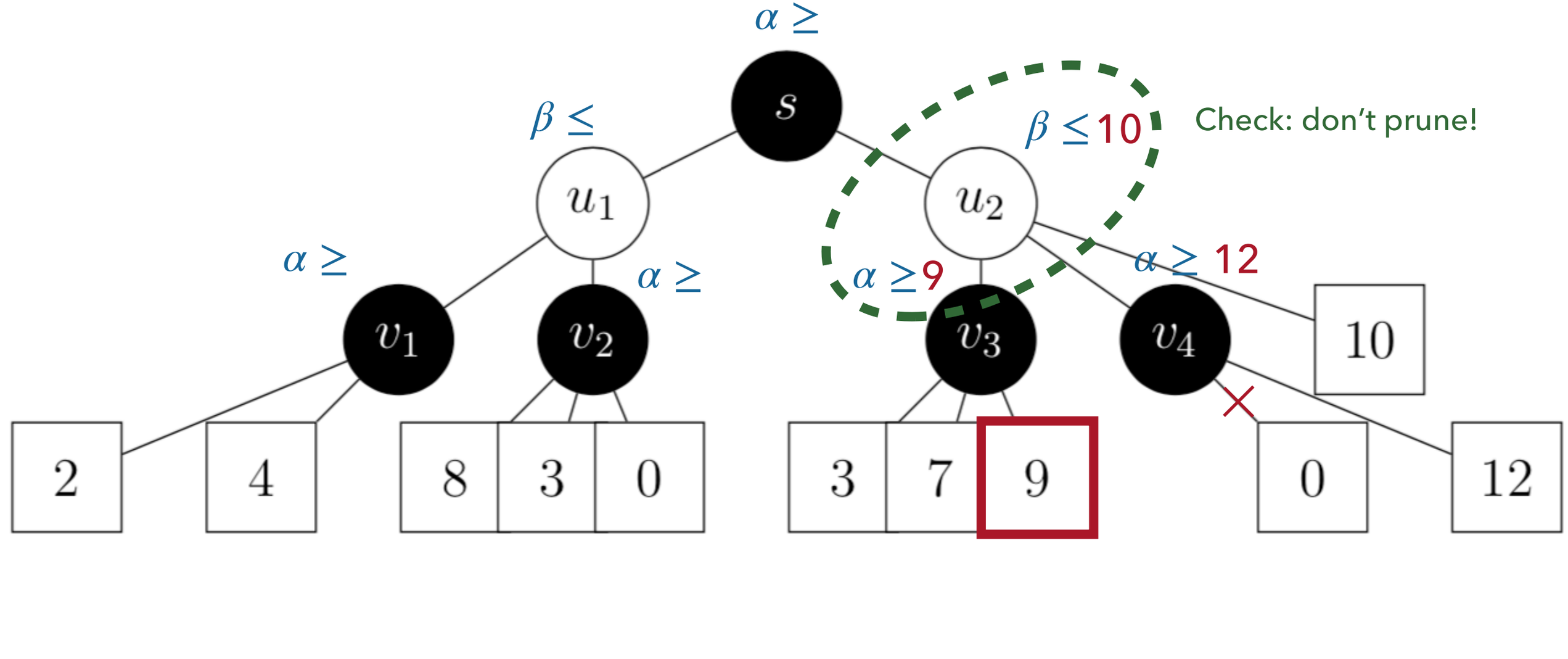
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

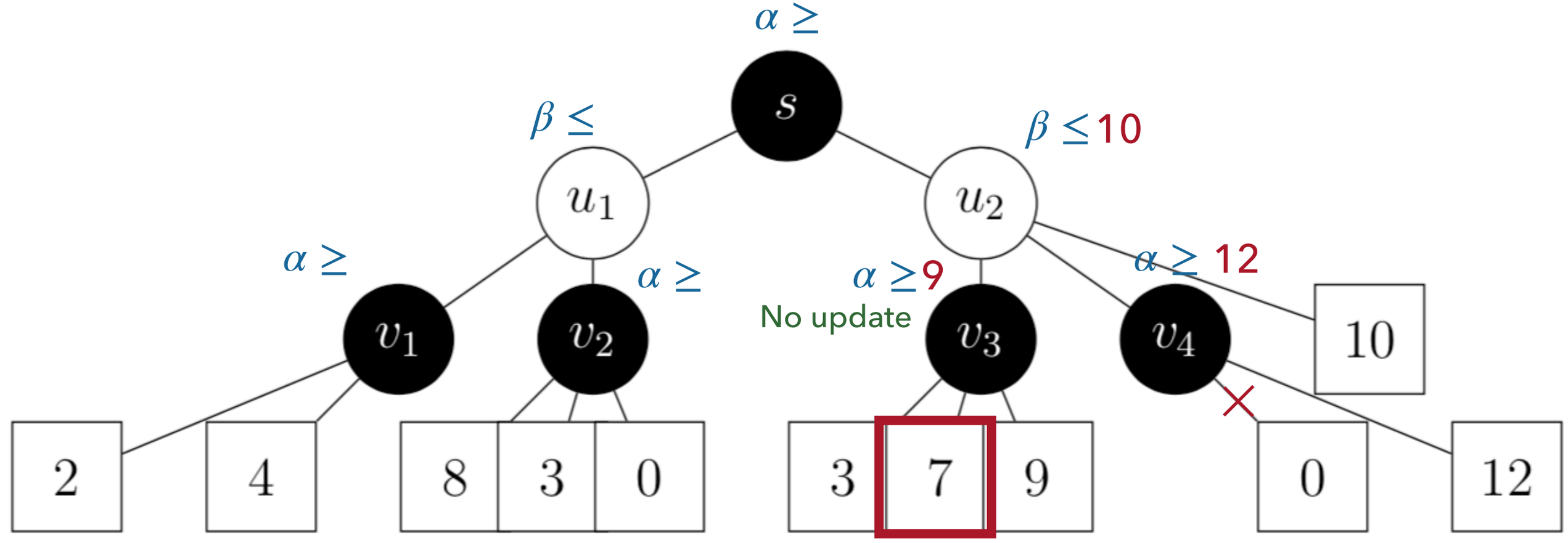
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.

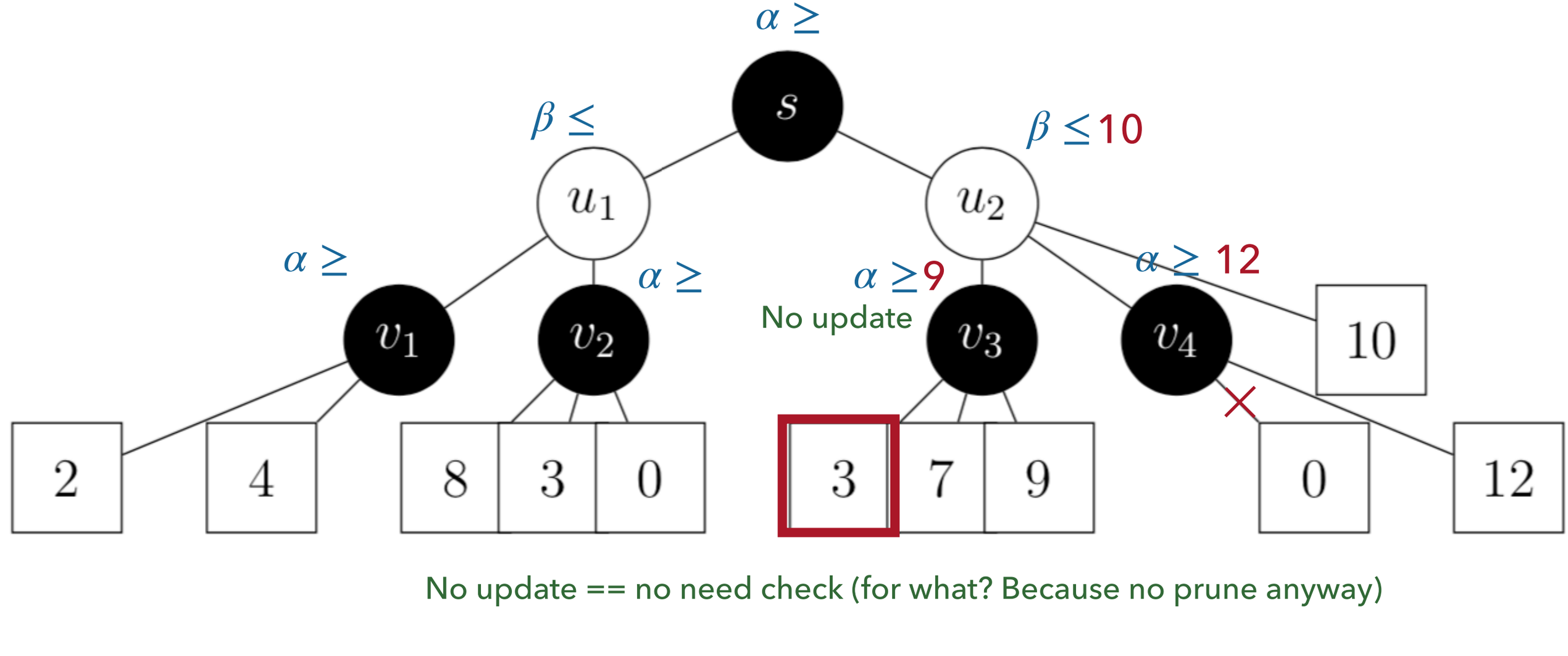


No update == no need check (for what? Because no prune anyway)

TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.

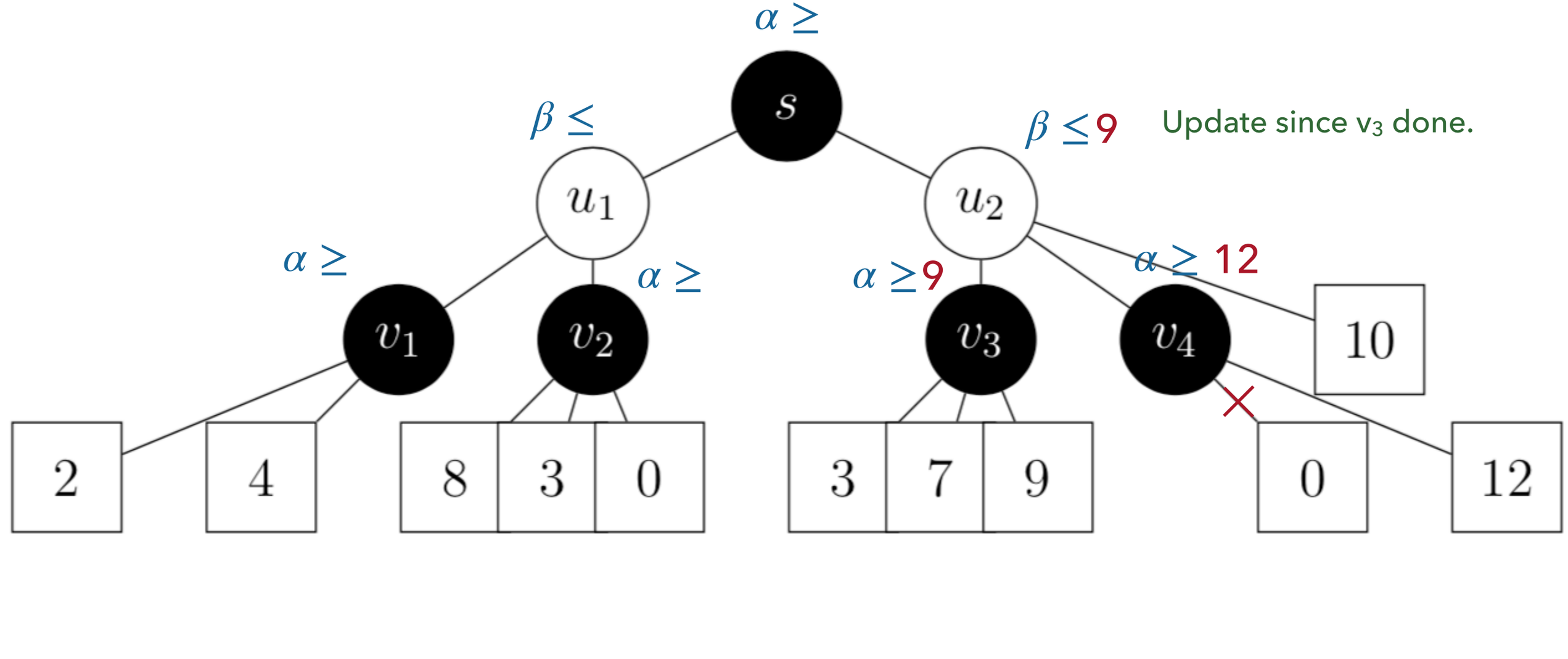


No update == no need check (for what? Because no prune anyway)

TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

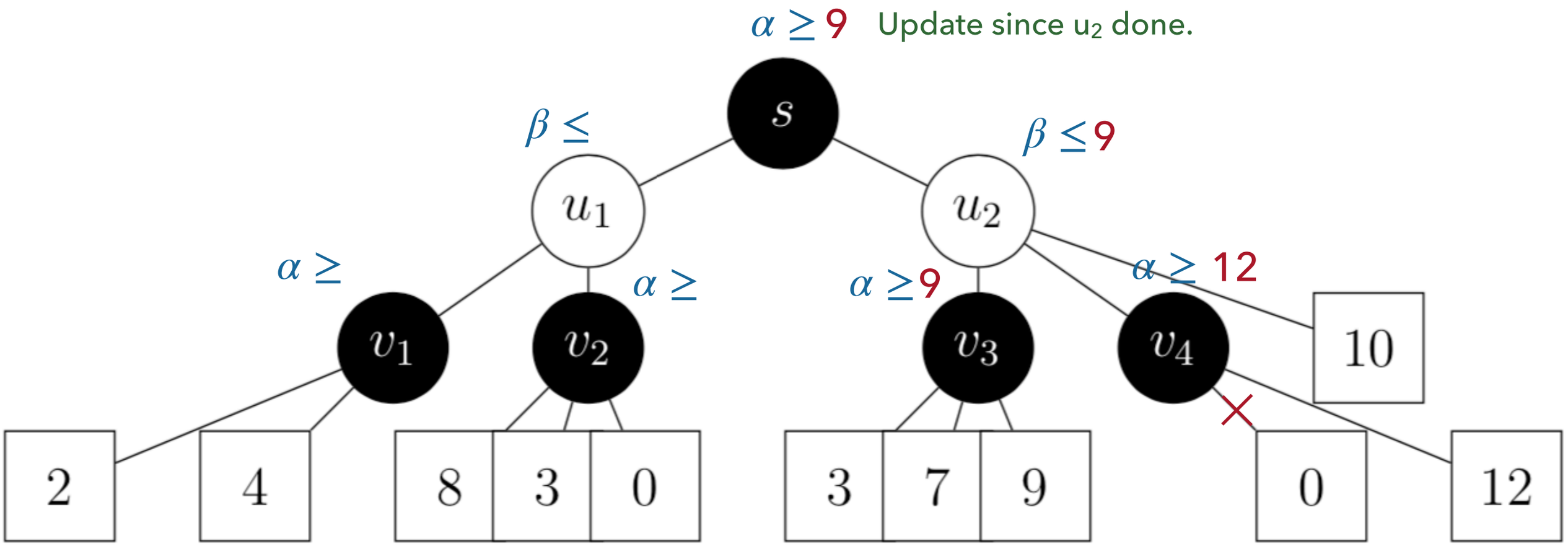
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

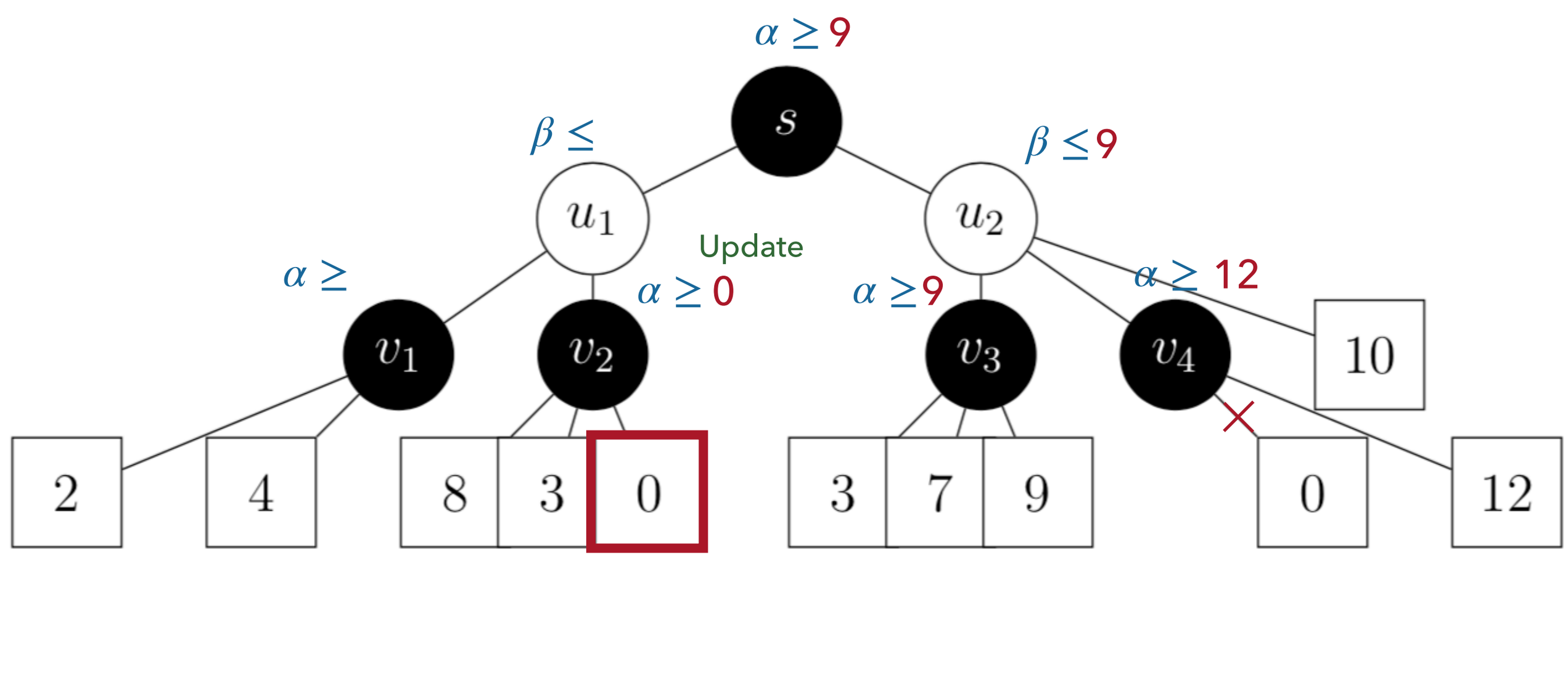
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

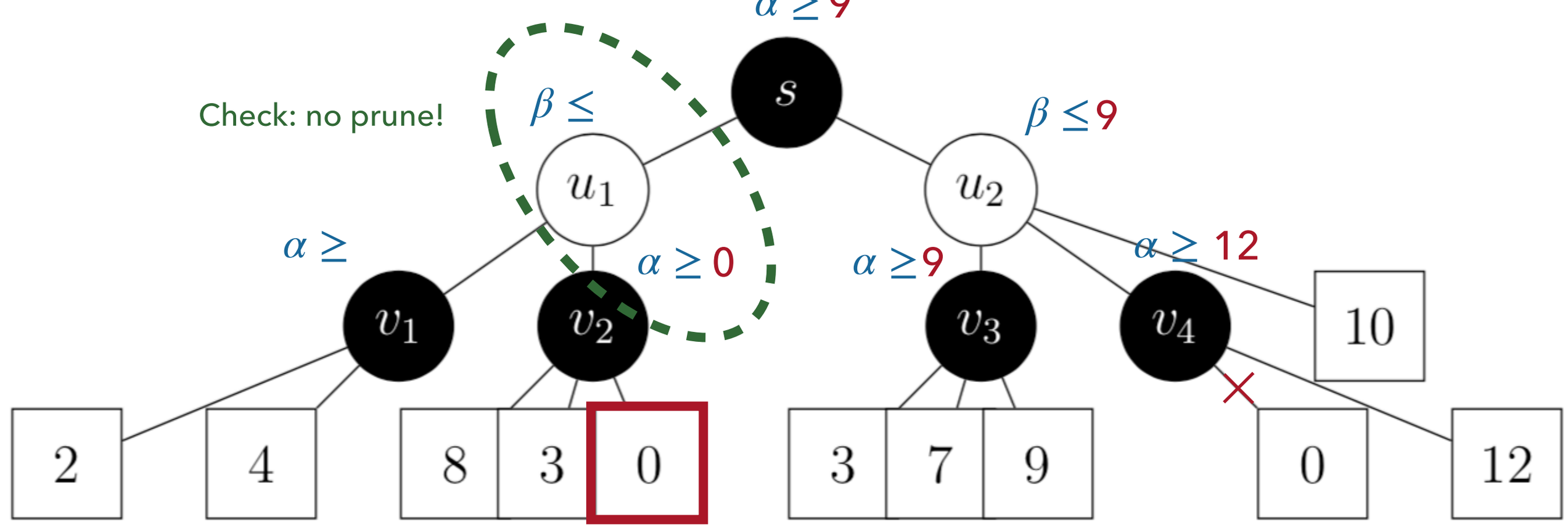
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

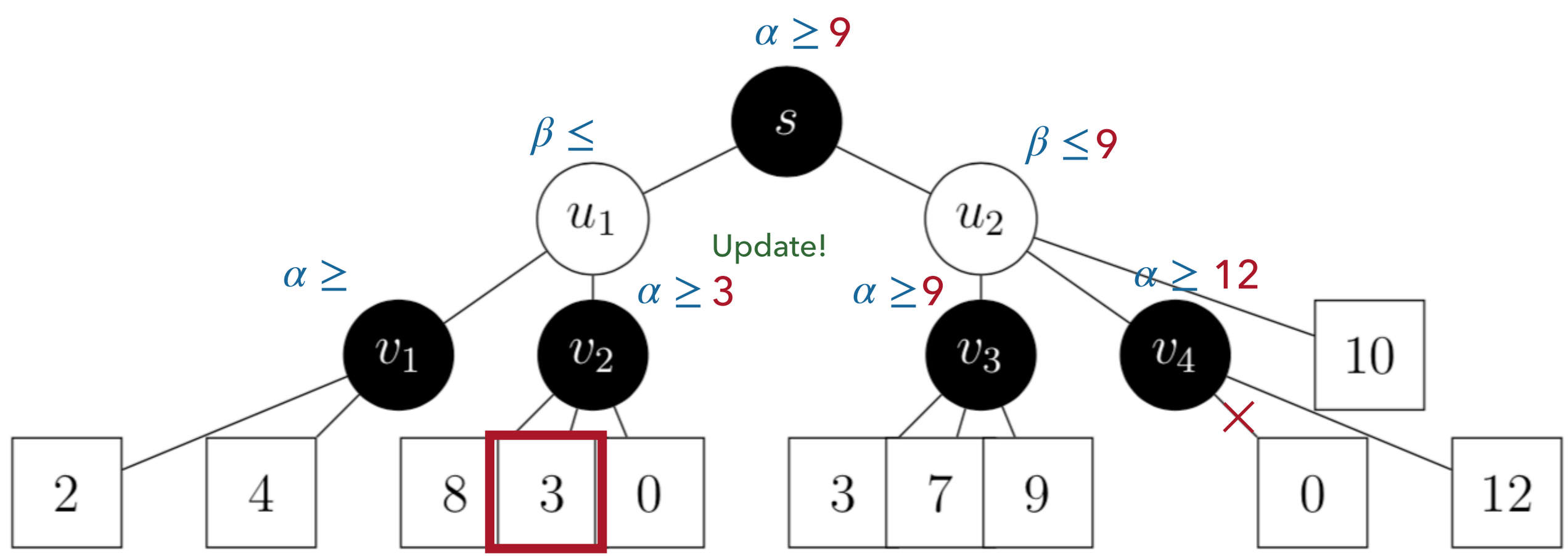
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

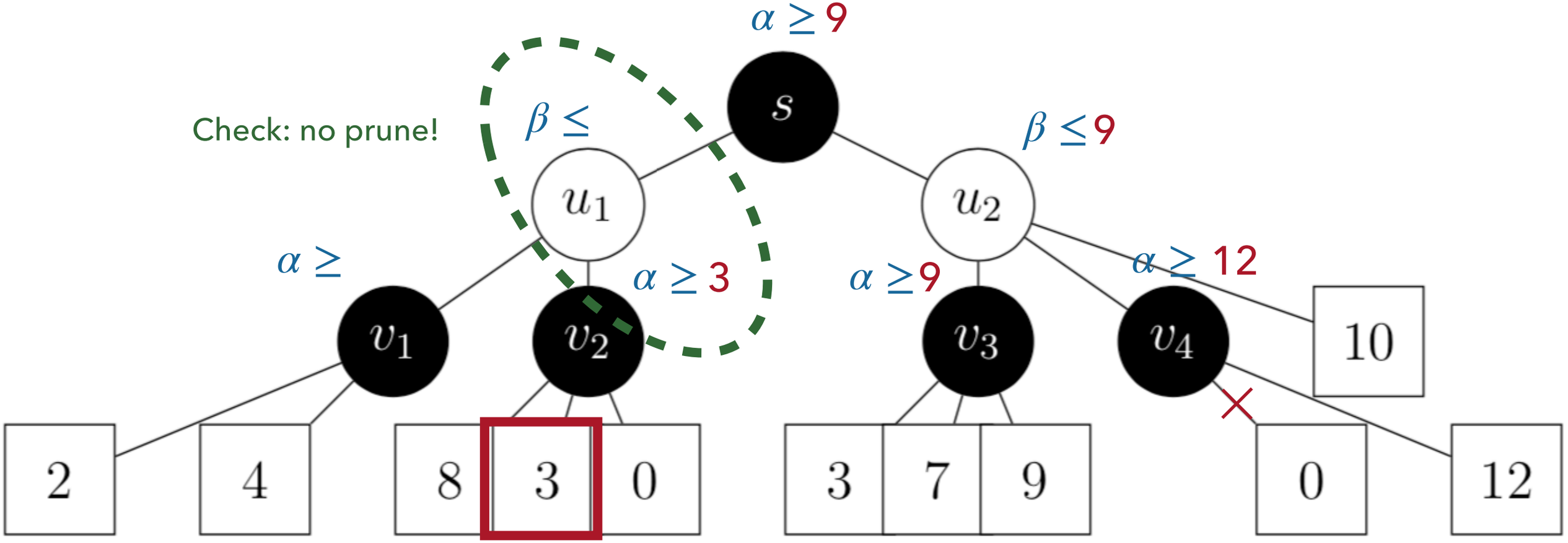
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

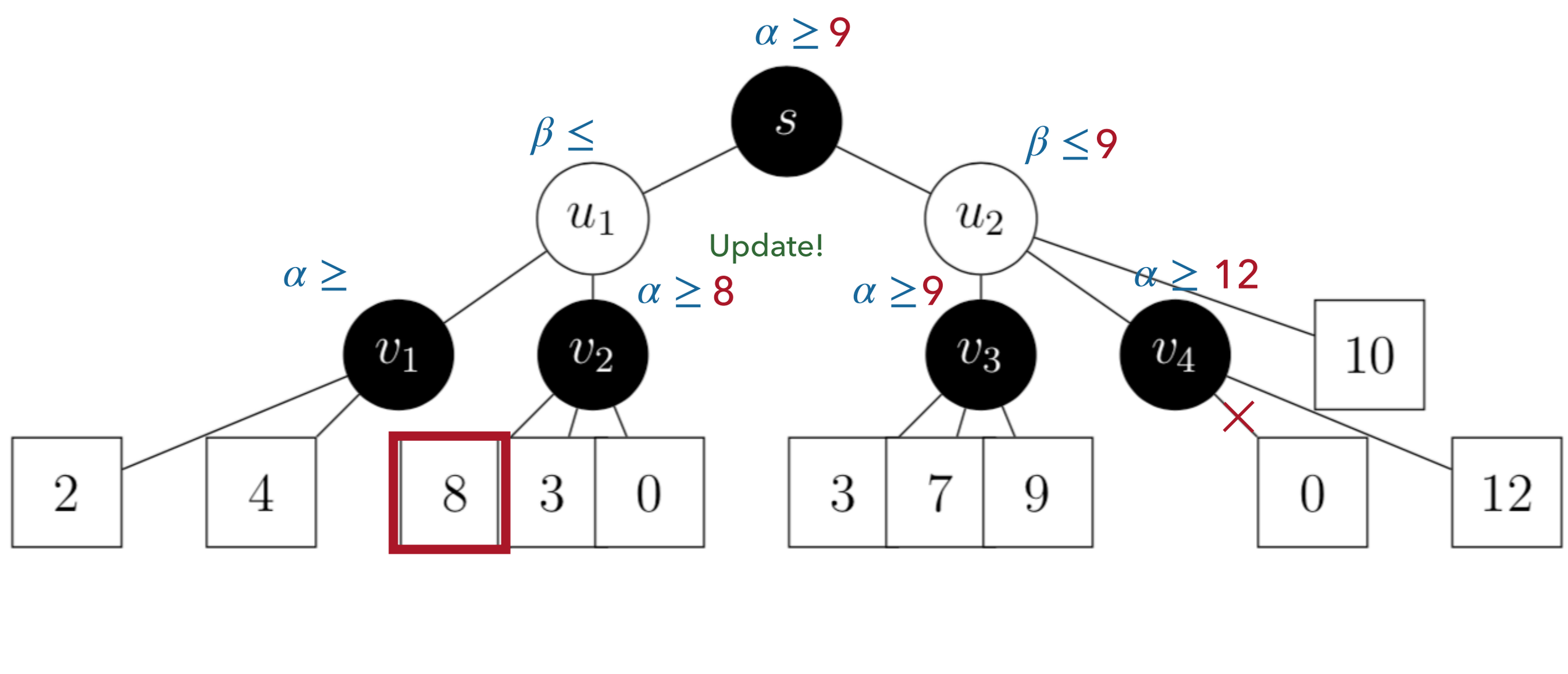
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

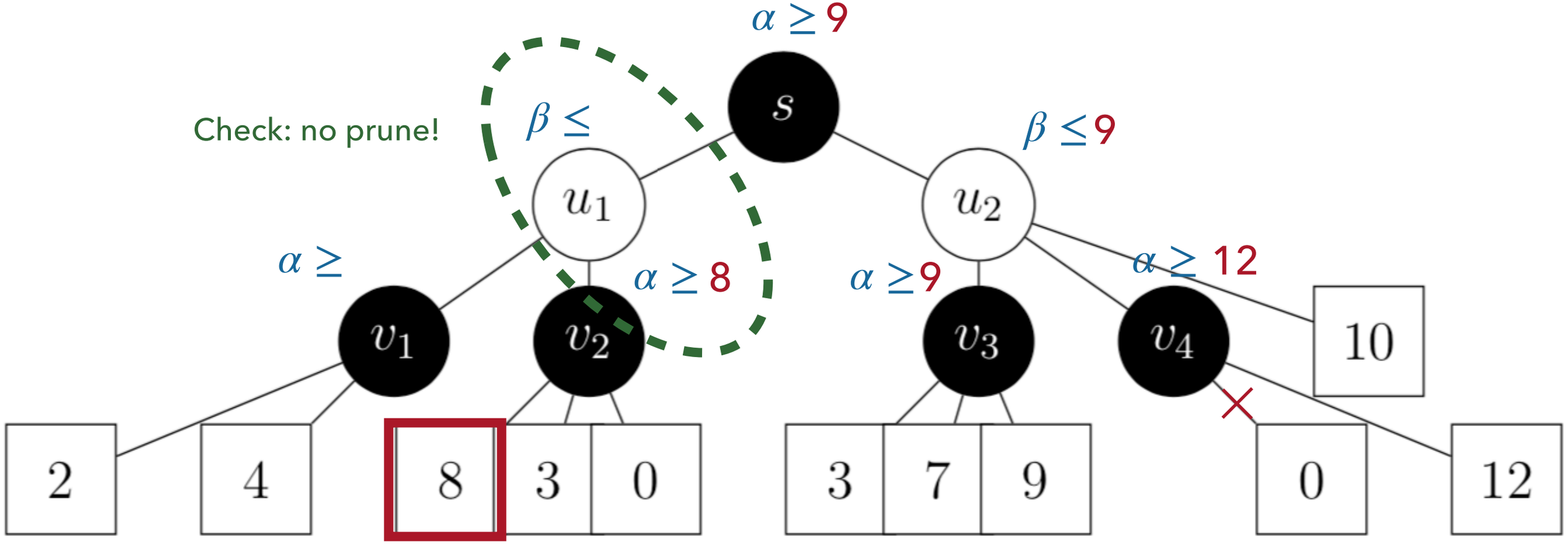
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

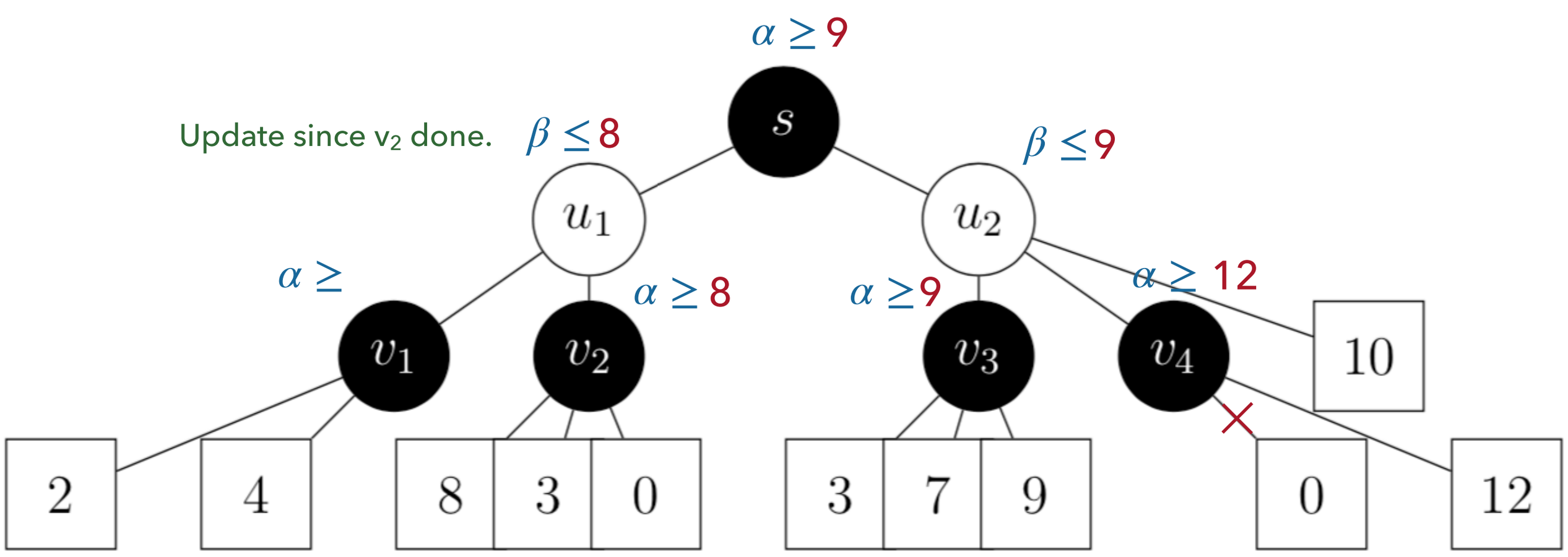
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

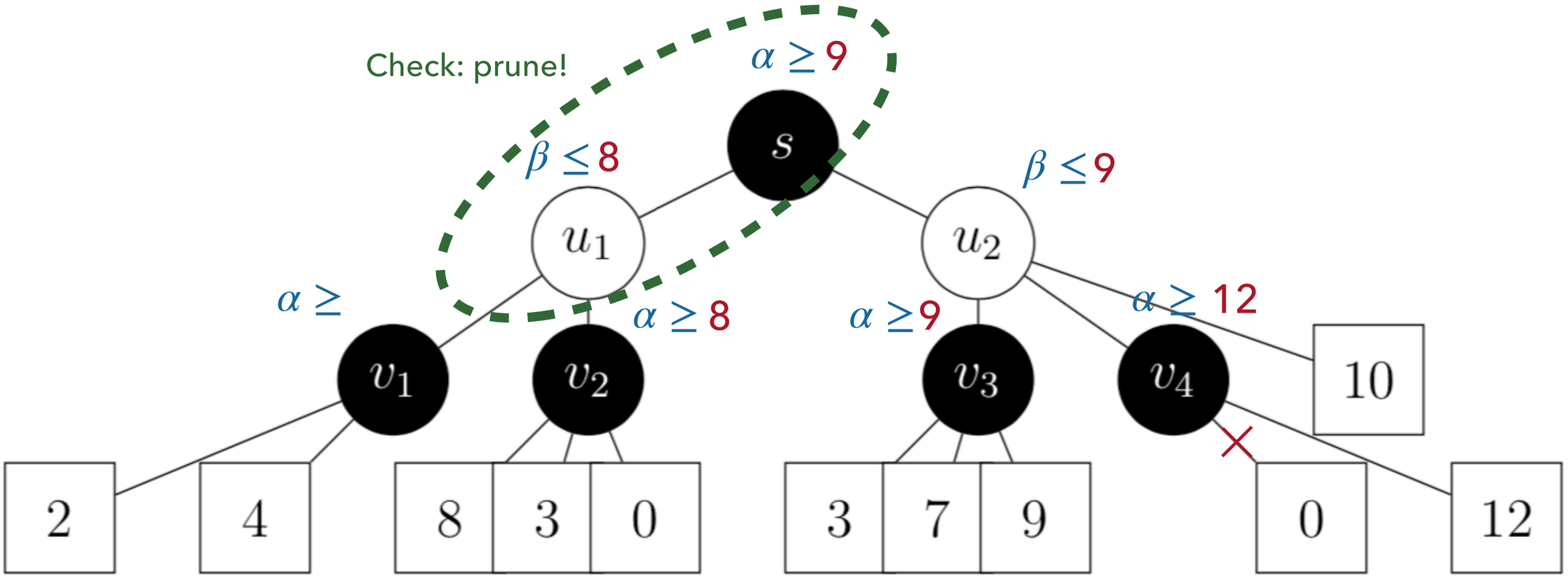
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

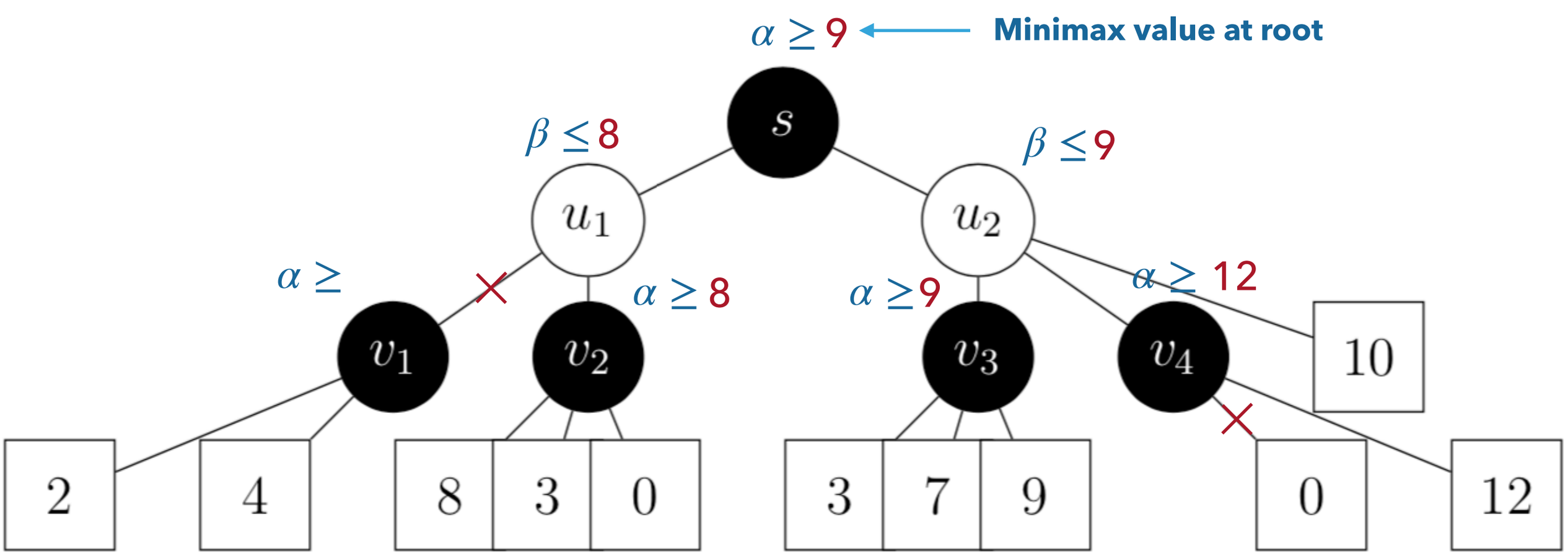
- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.



TUTORIAL 4 QUESTION 3

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; mark with an 'X' all ARCS that are pruned by α-β pruning, if any.

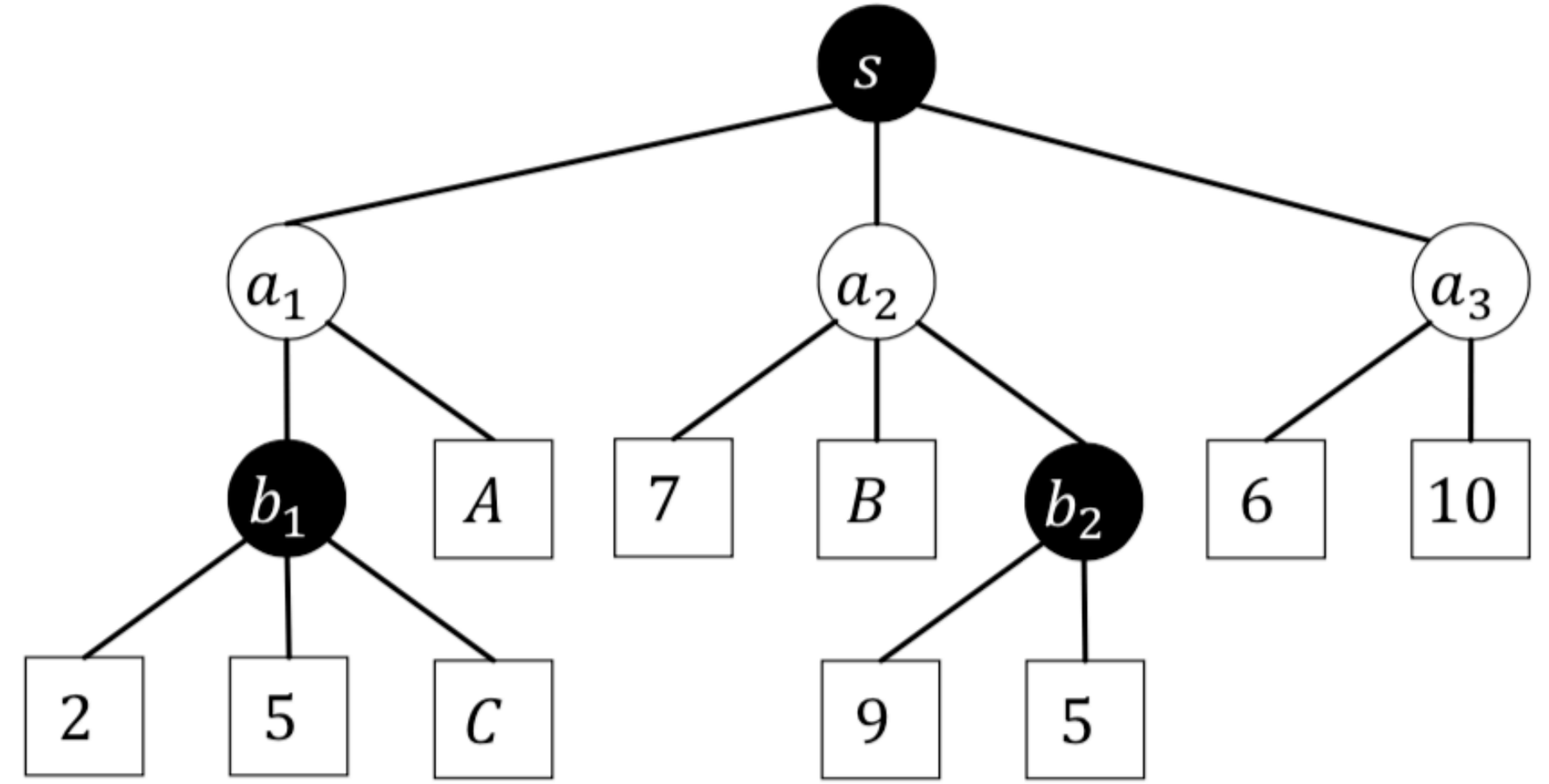


If question requires justifications, tell us the alpha/beta values of the immediate node will do.

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

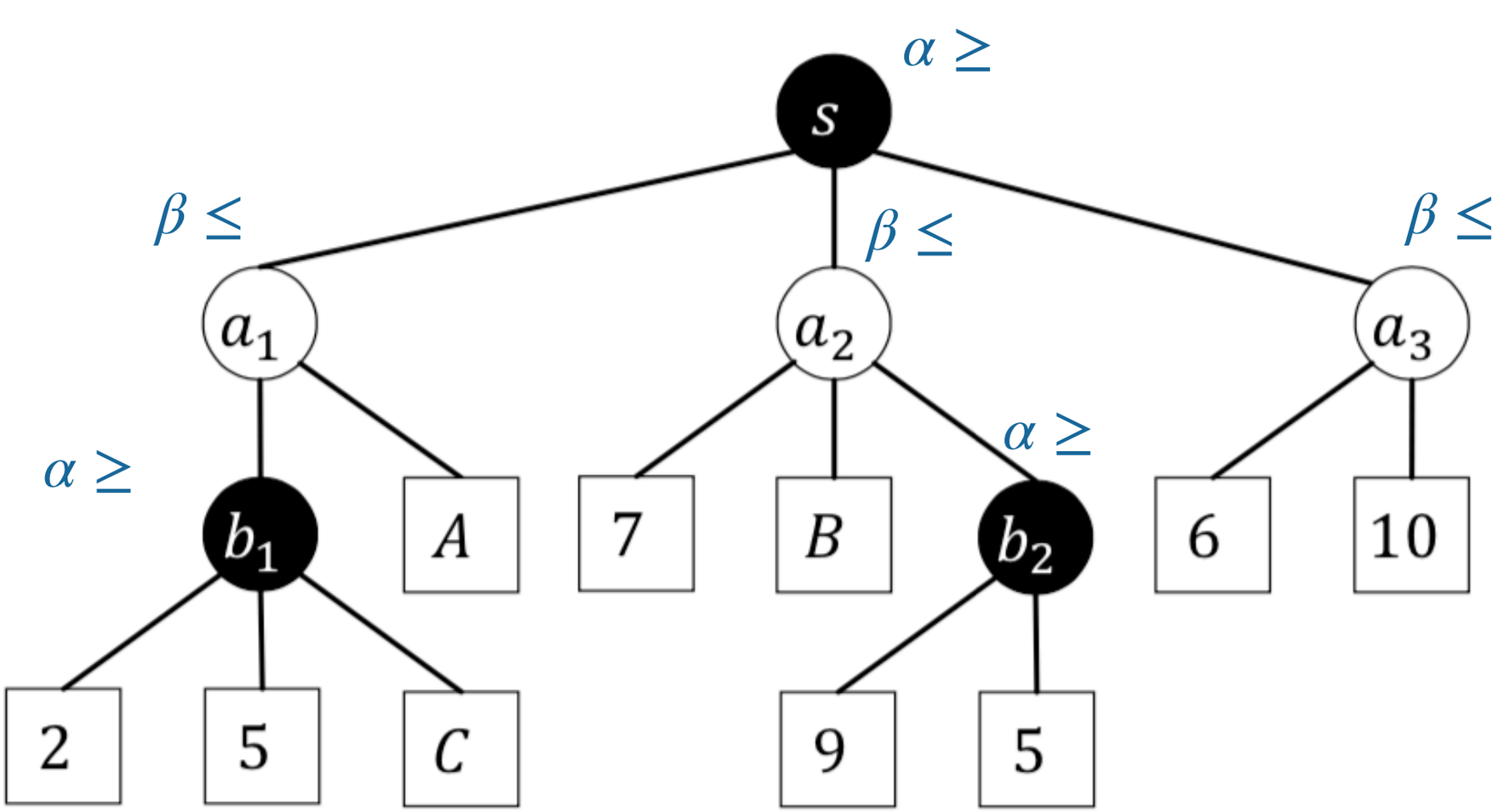
B > _____

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

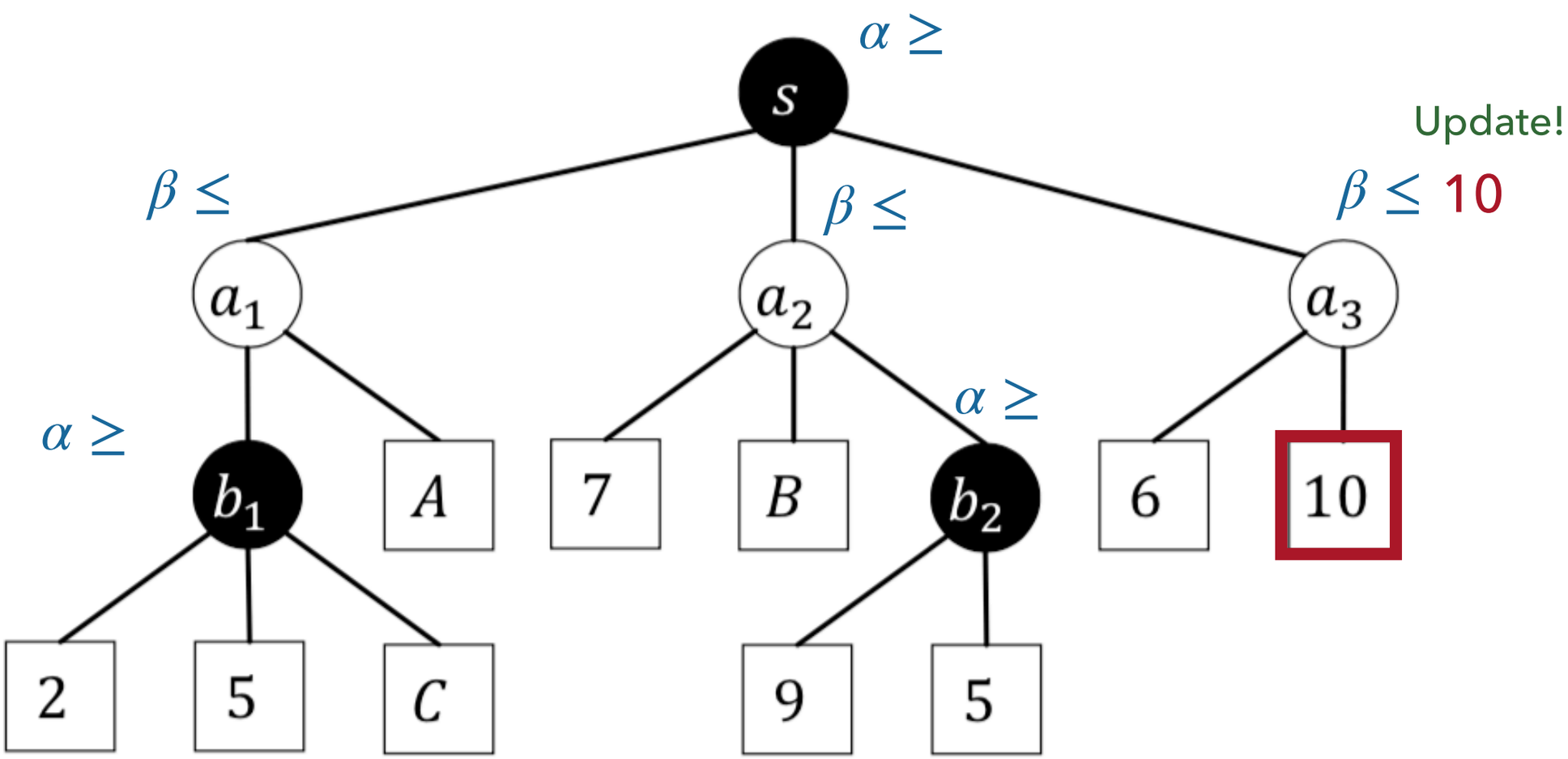
B > _____

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

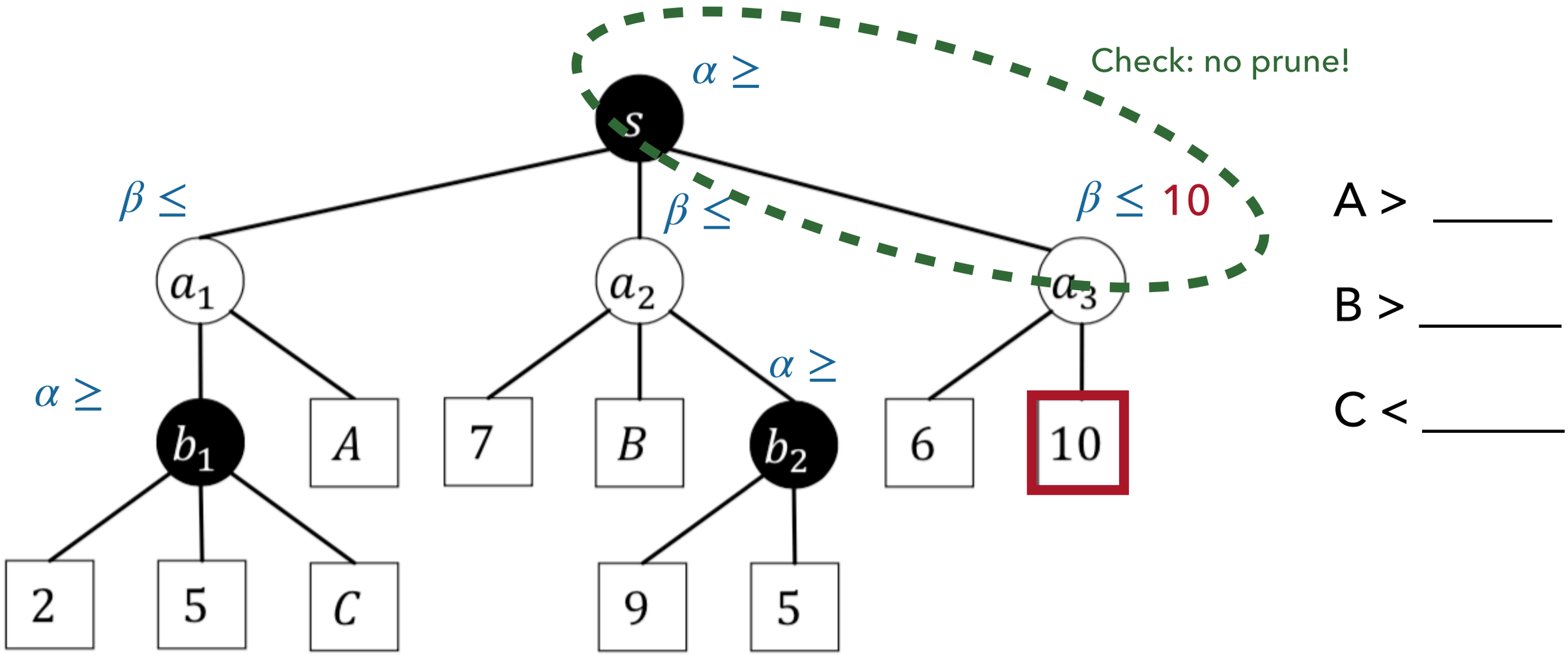
B > _____

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

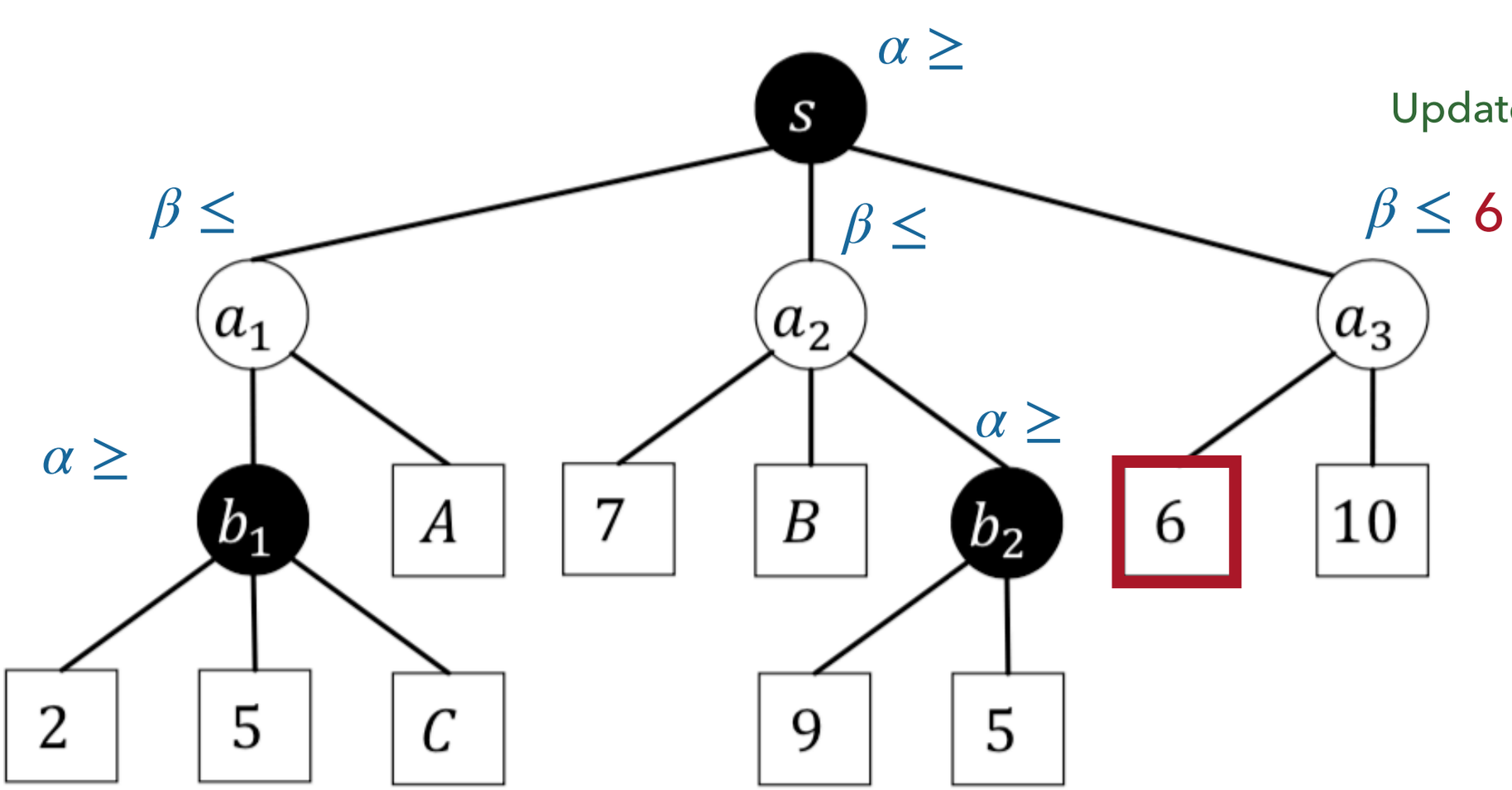
- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



Update!

A > _____

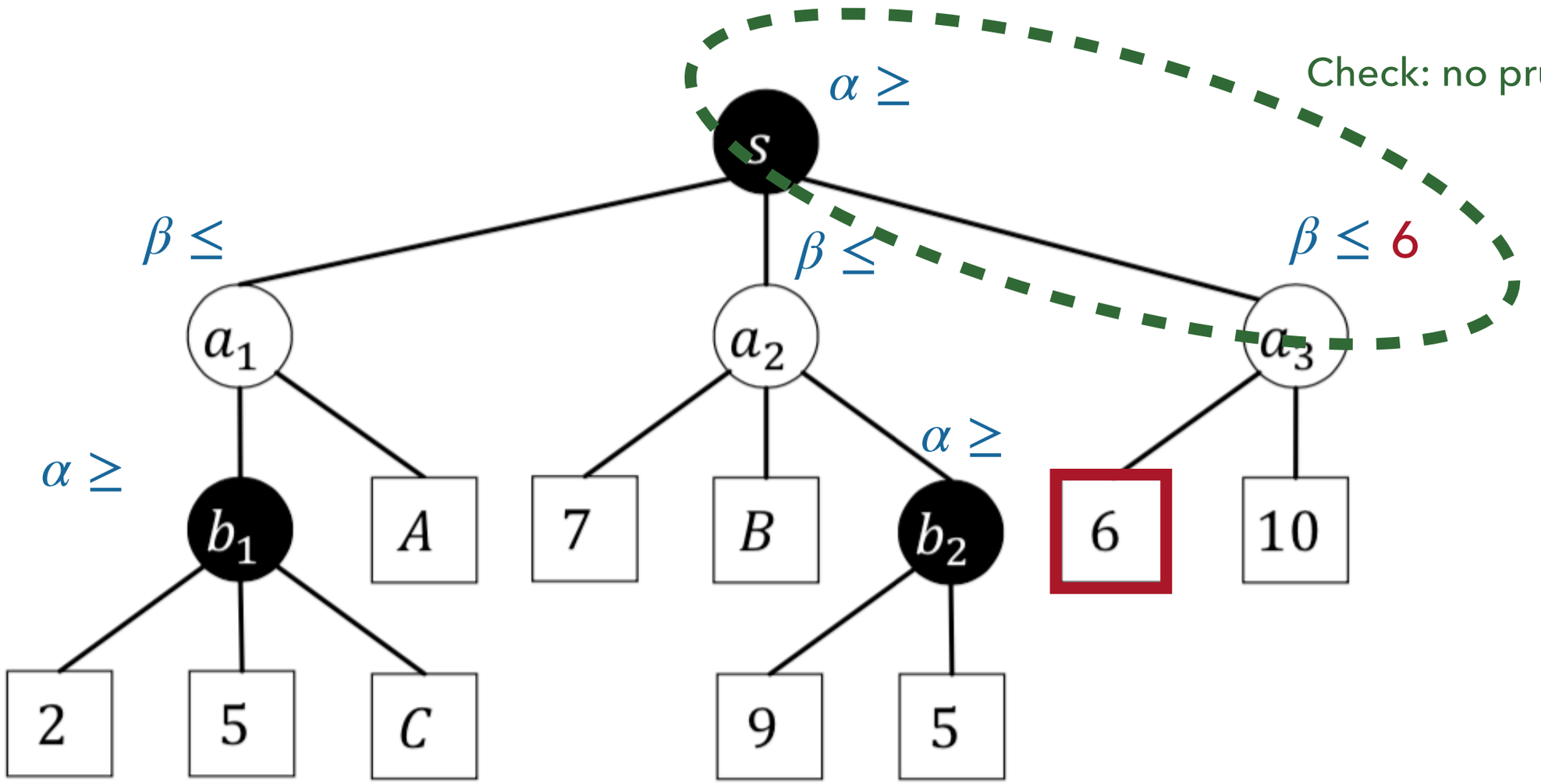
B > _____

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

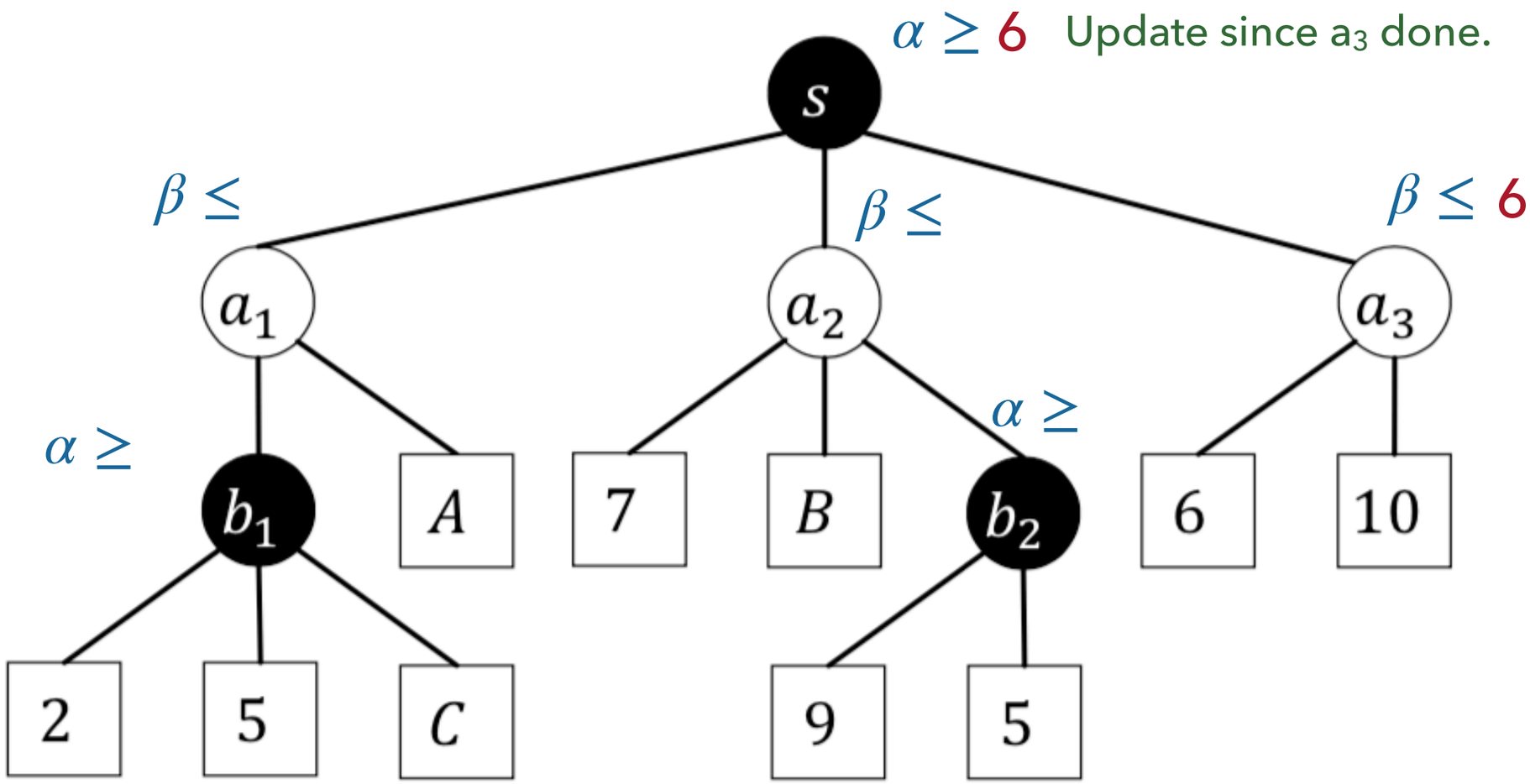
B > _____

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

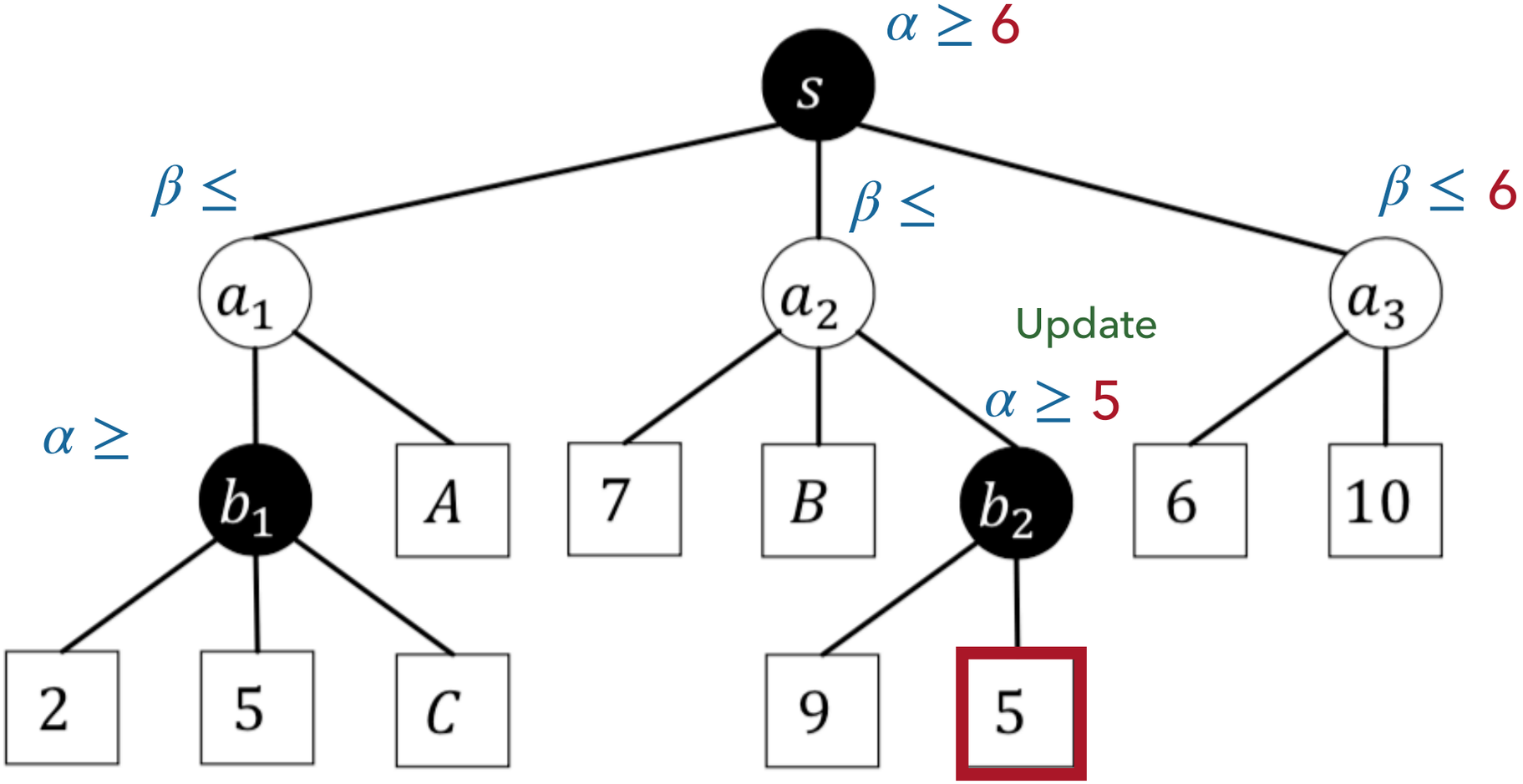
B > _____

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

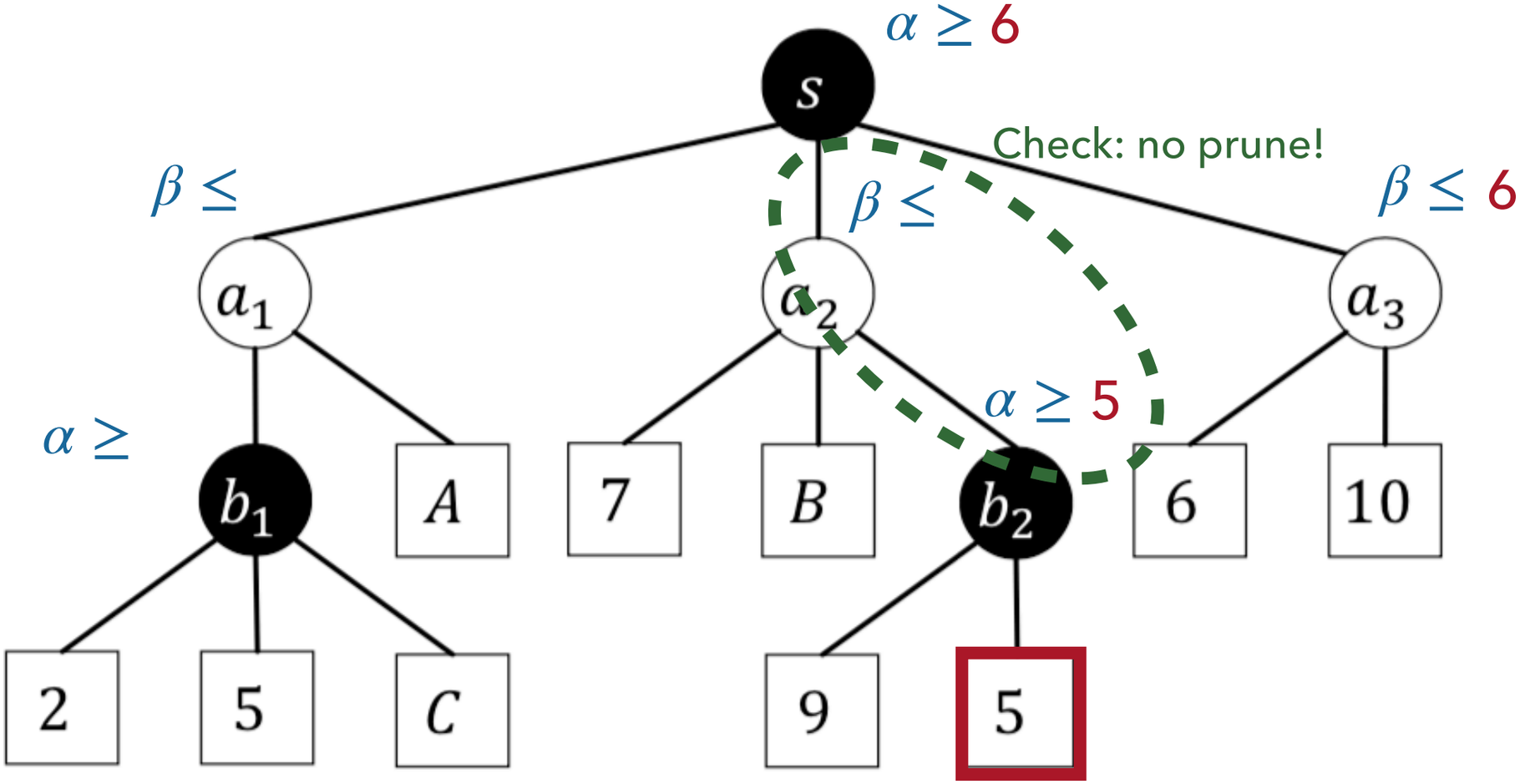
B > _____

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

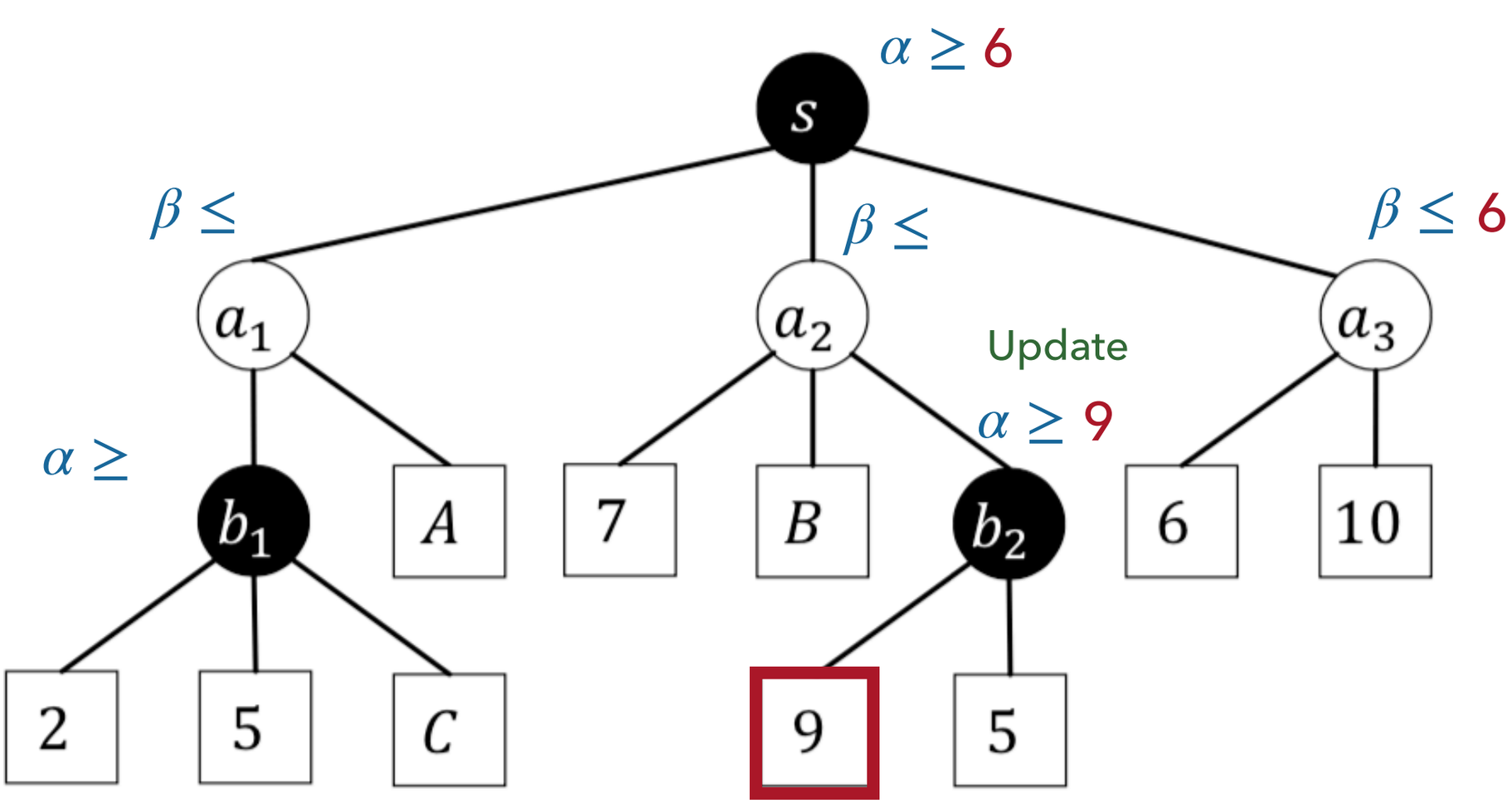
B > _____

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

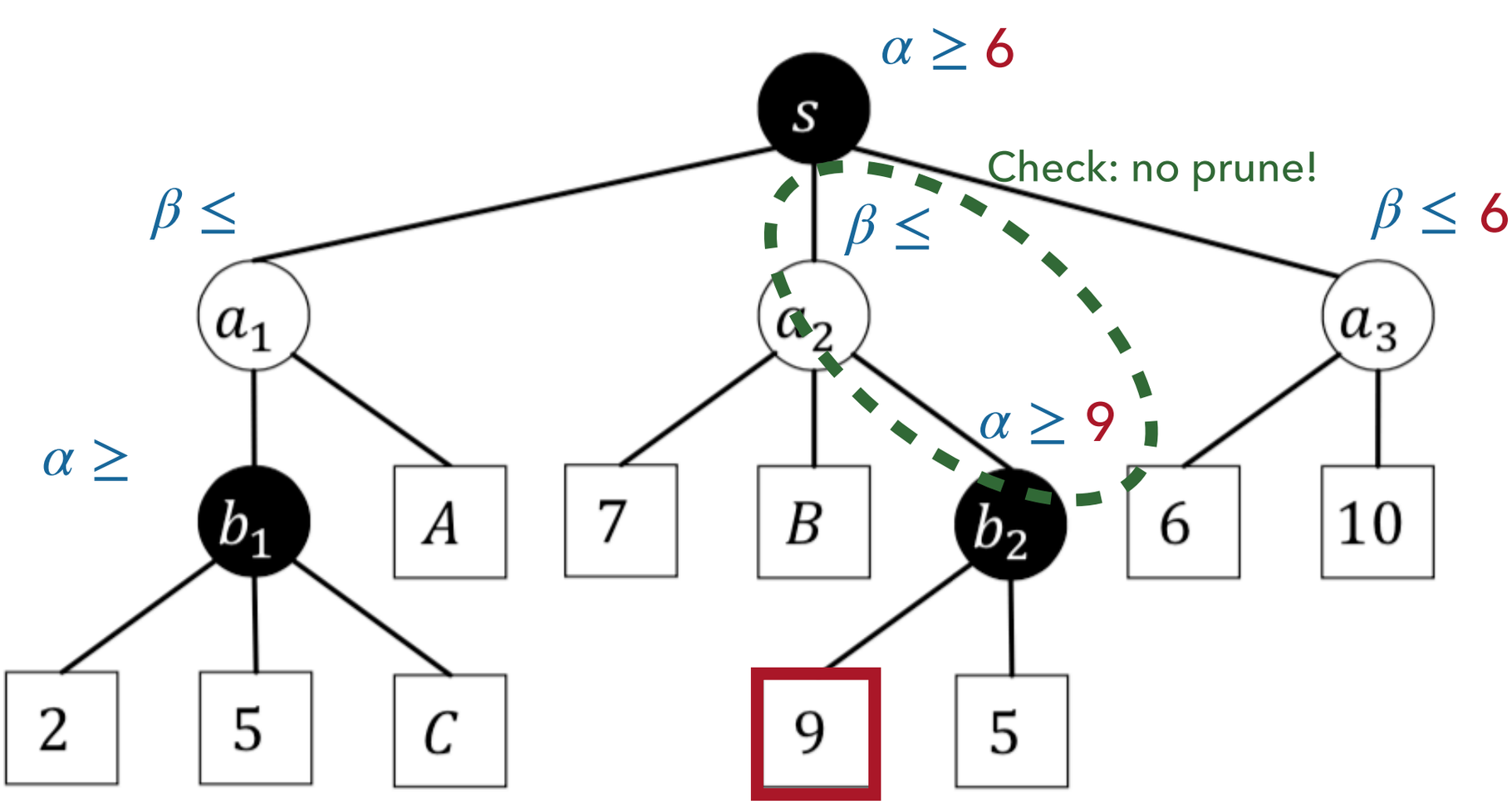
B > _____

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

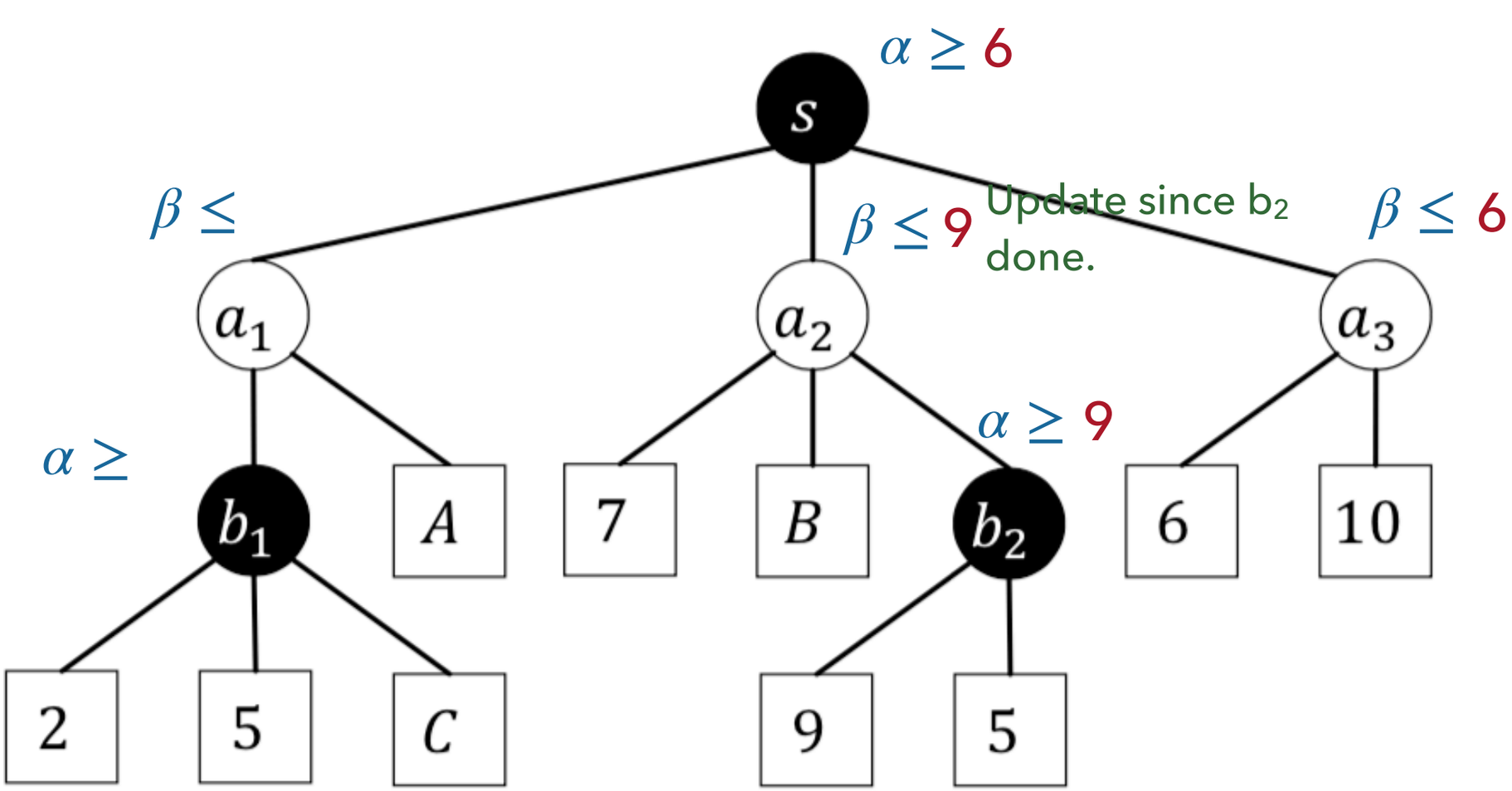
B > _____

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

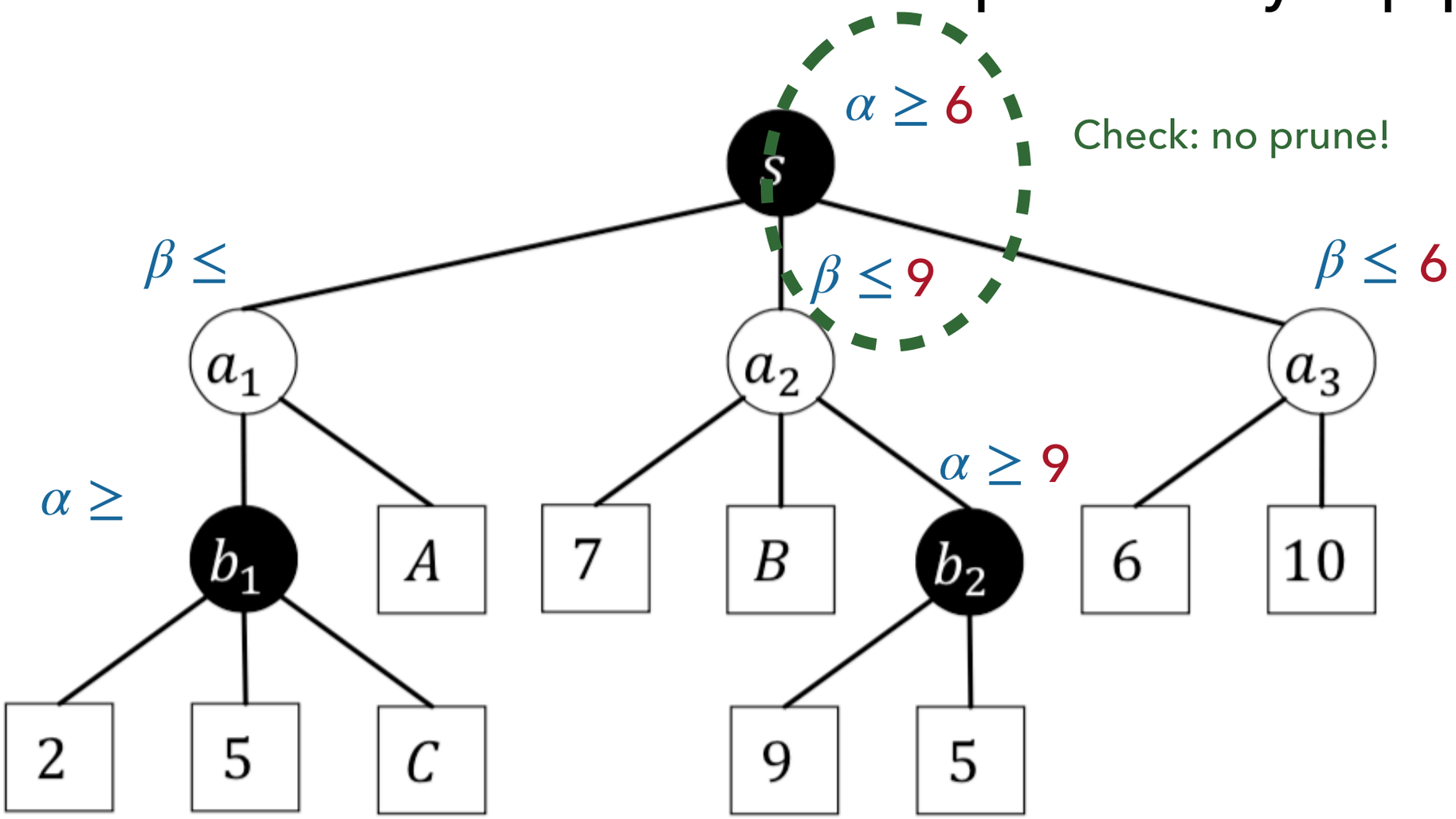
B > _____

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

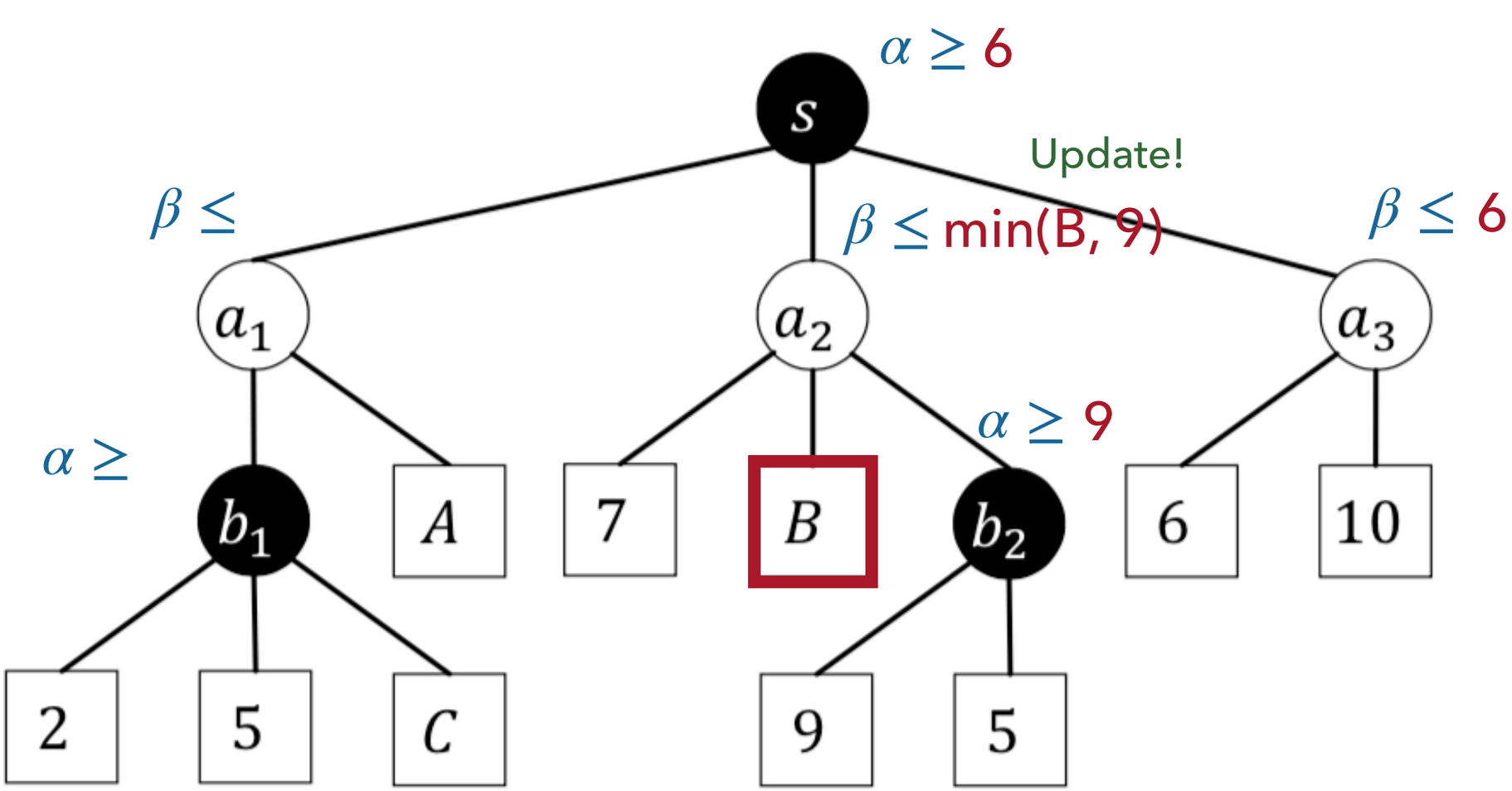
B > _____

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

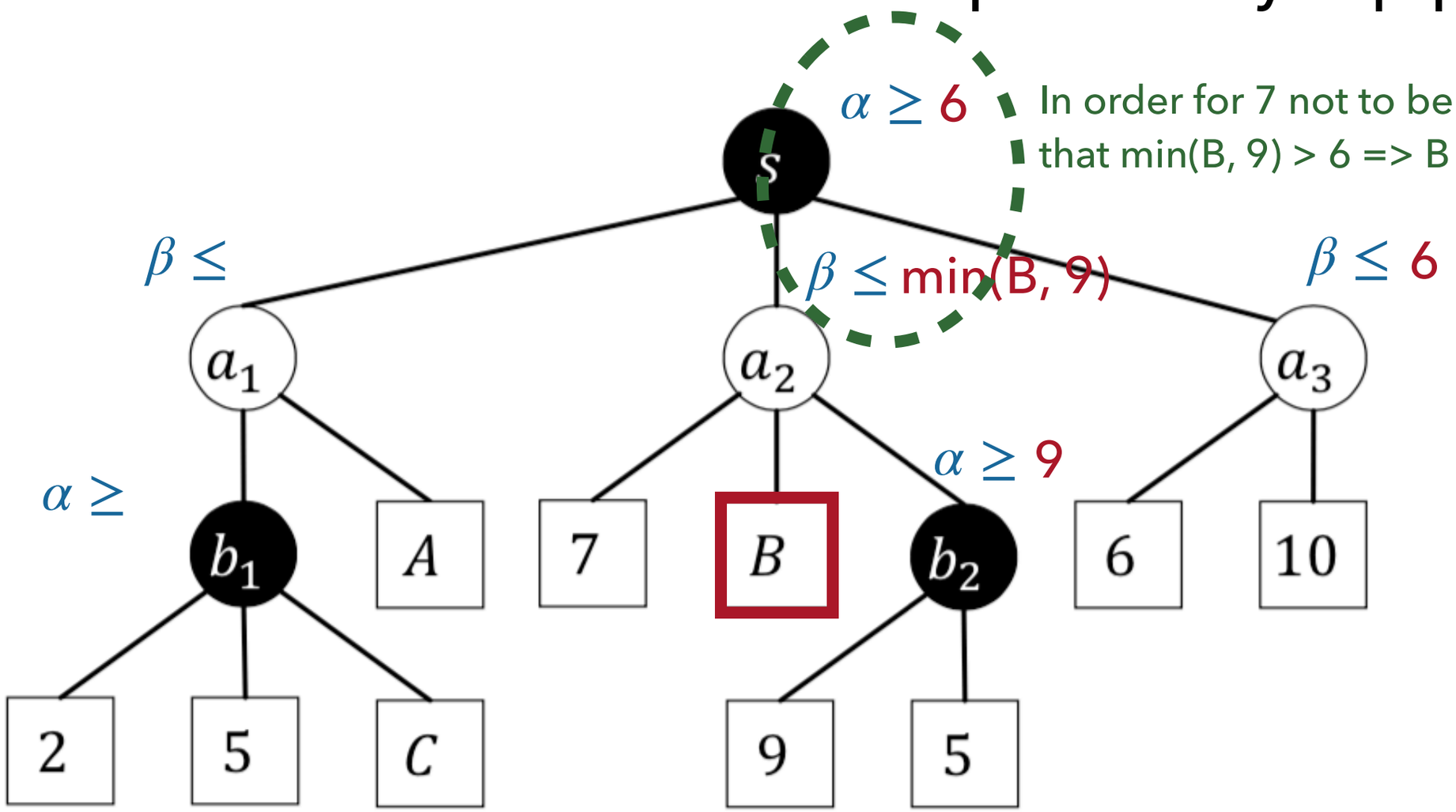
B > 6

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

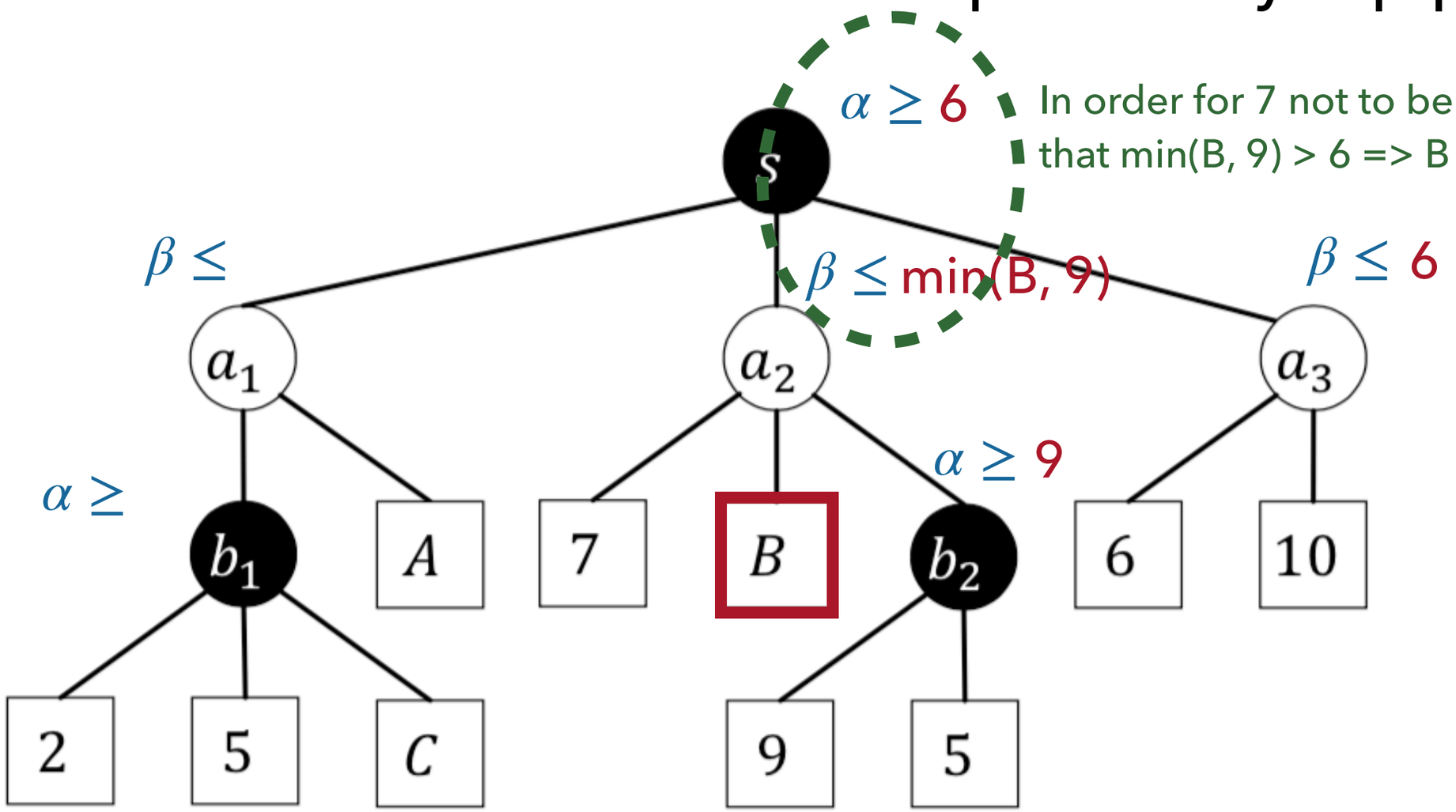
B > 6

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

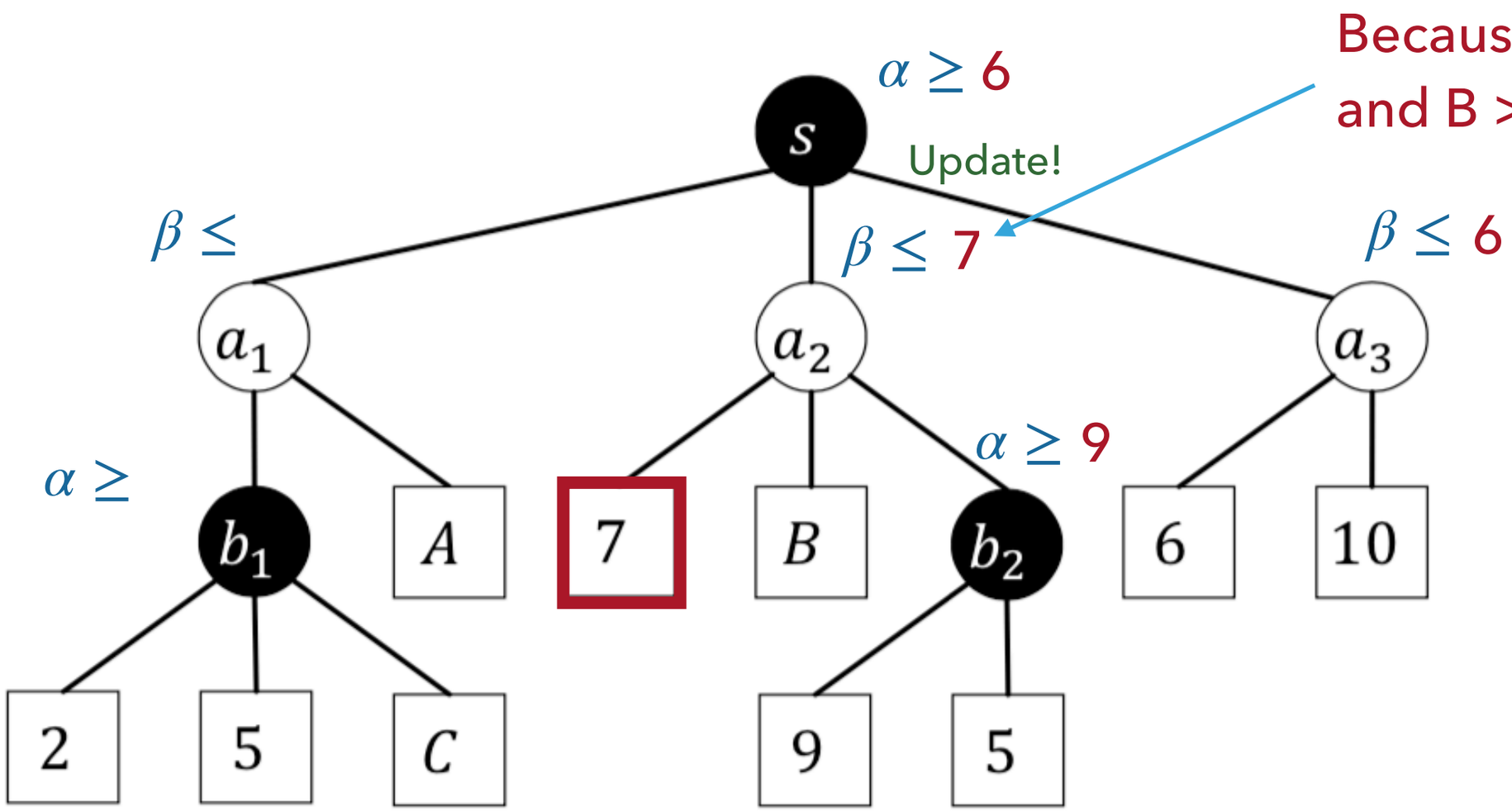
B > 6

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

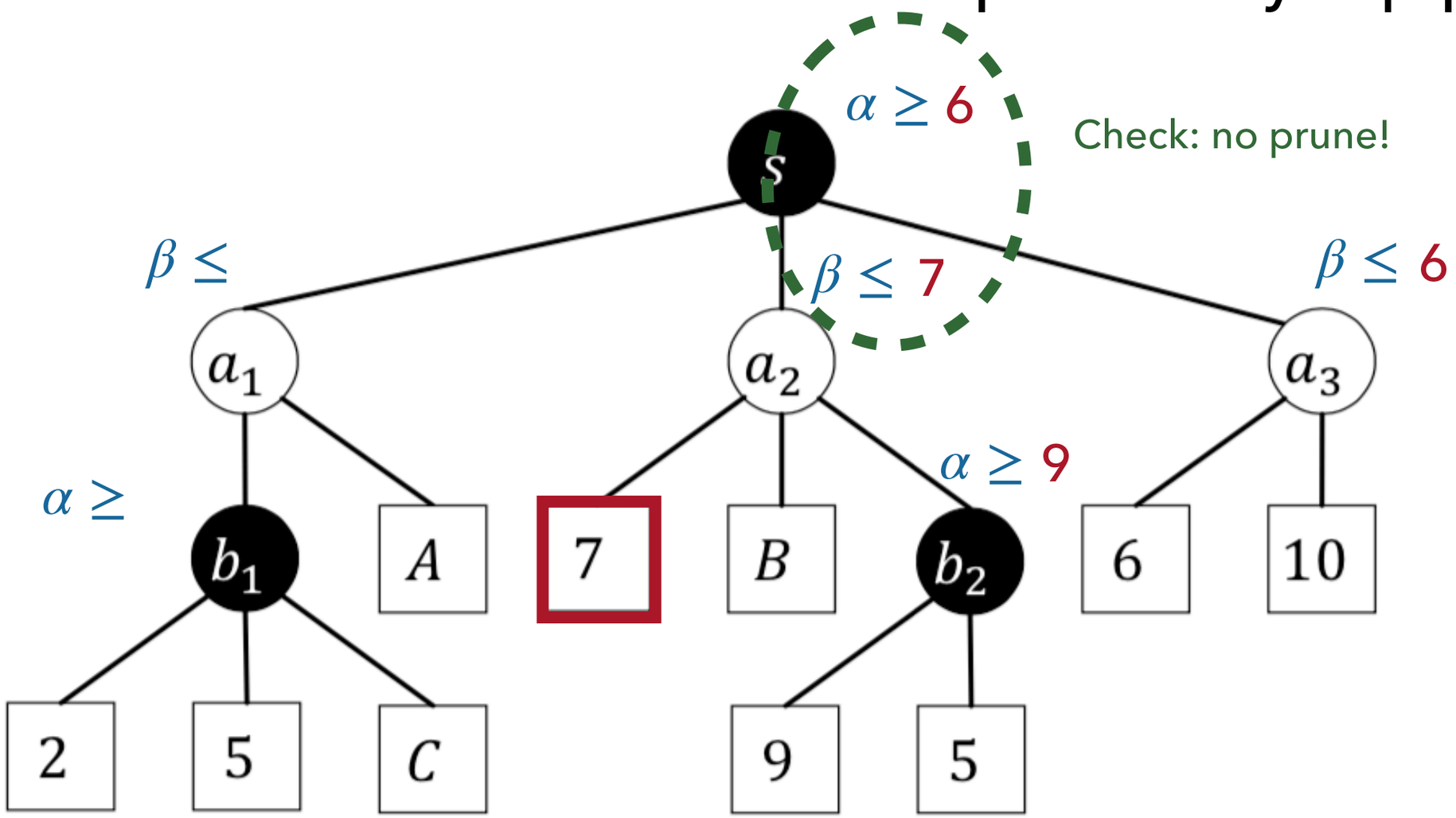
B > 6

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

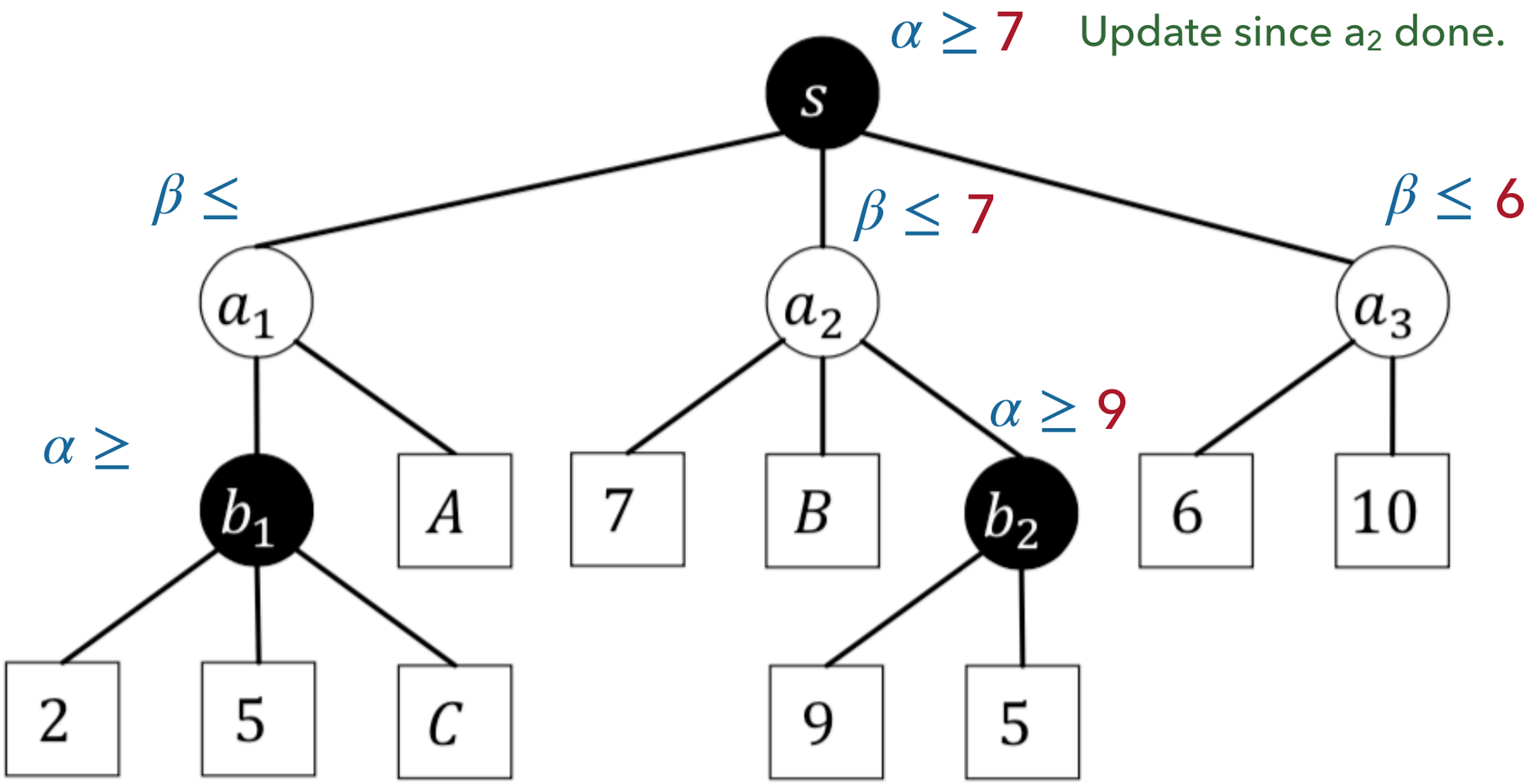
B > 6

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > _____

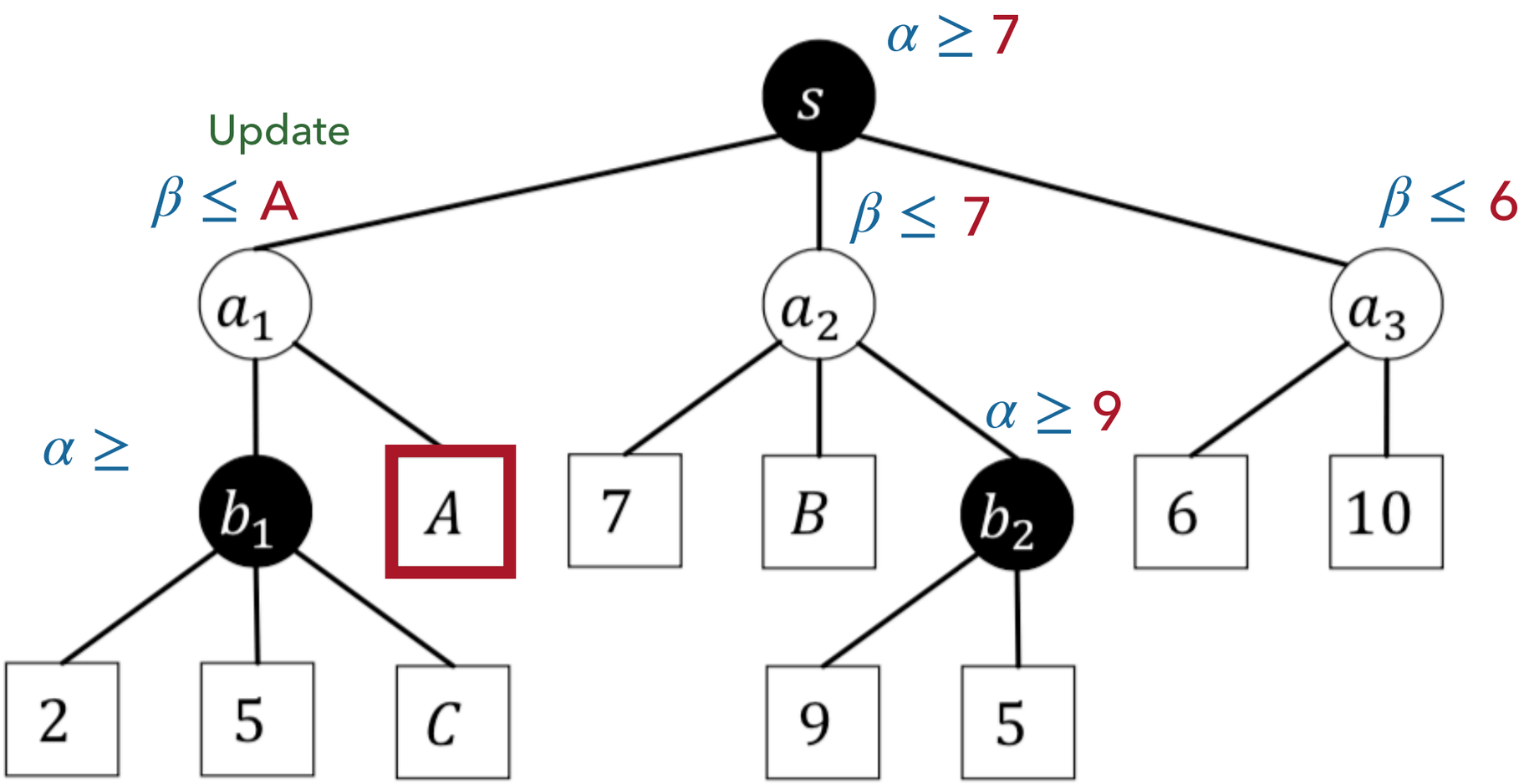
B > 6

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > 7

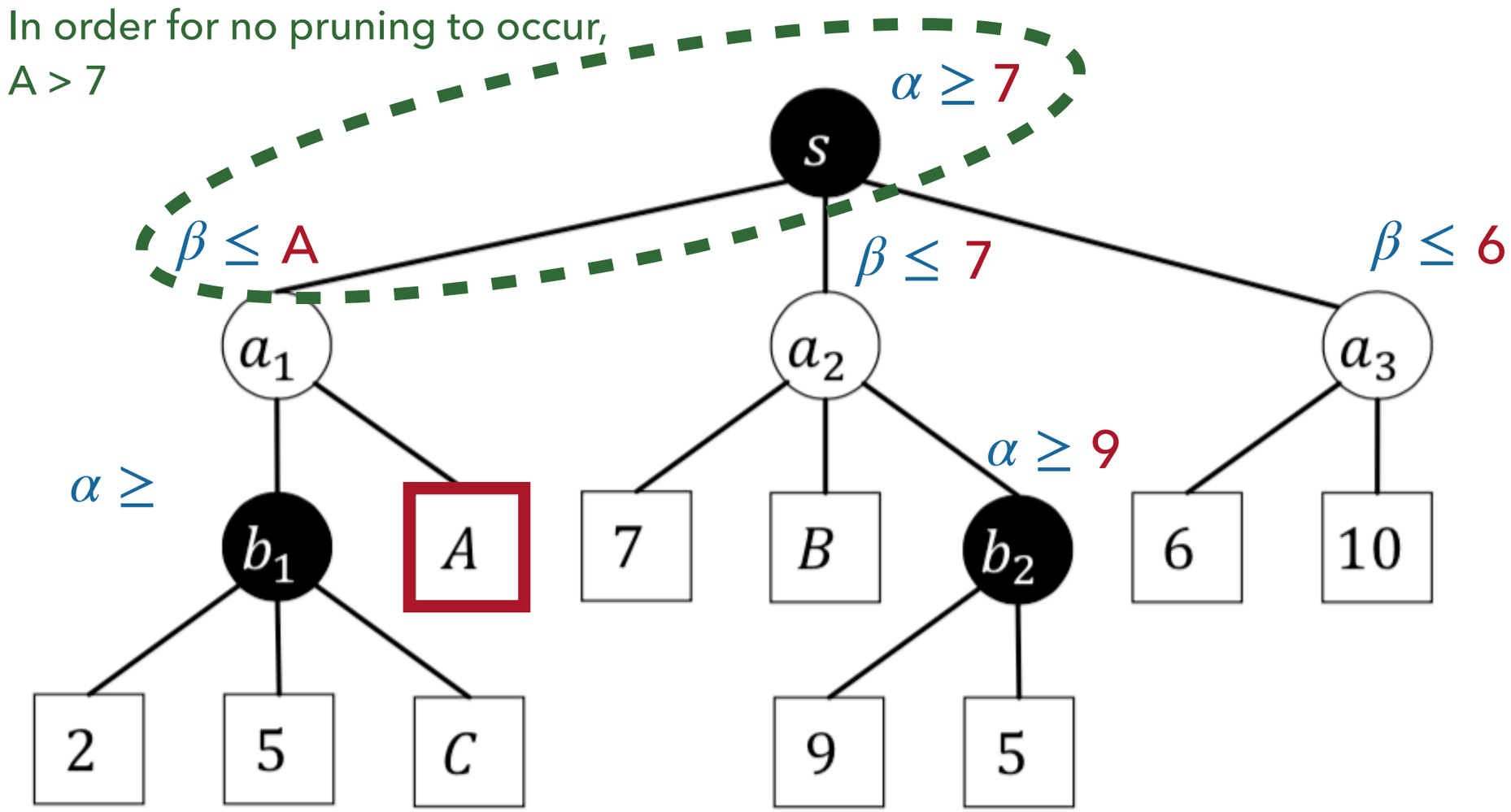
B > 6

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > 7

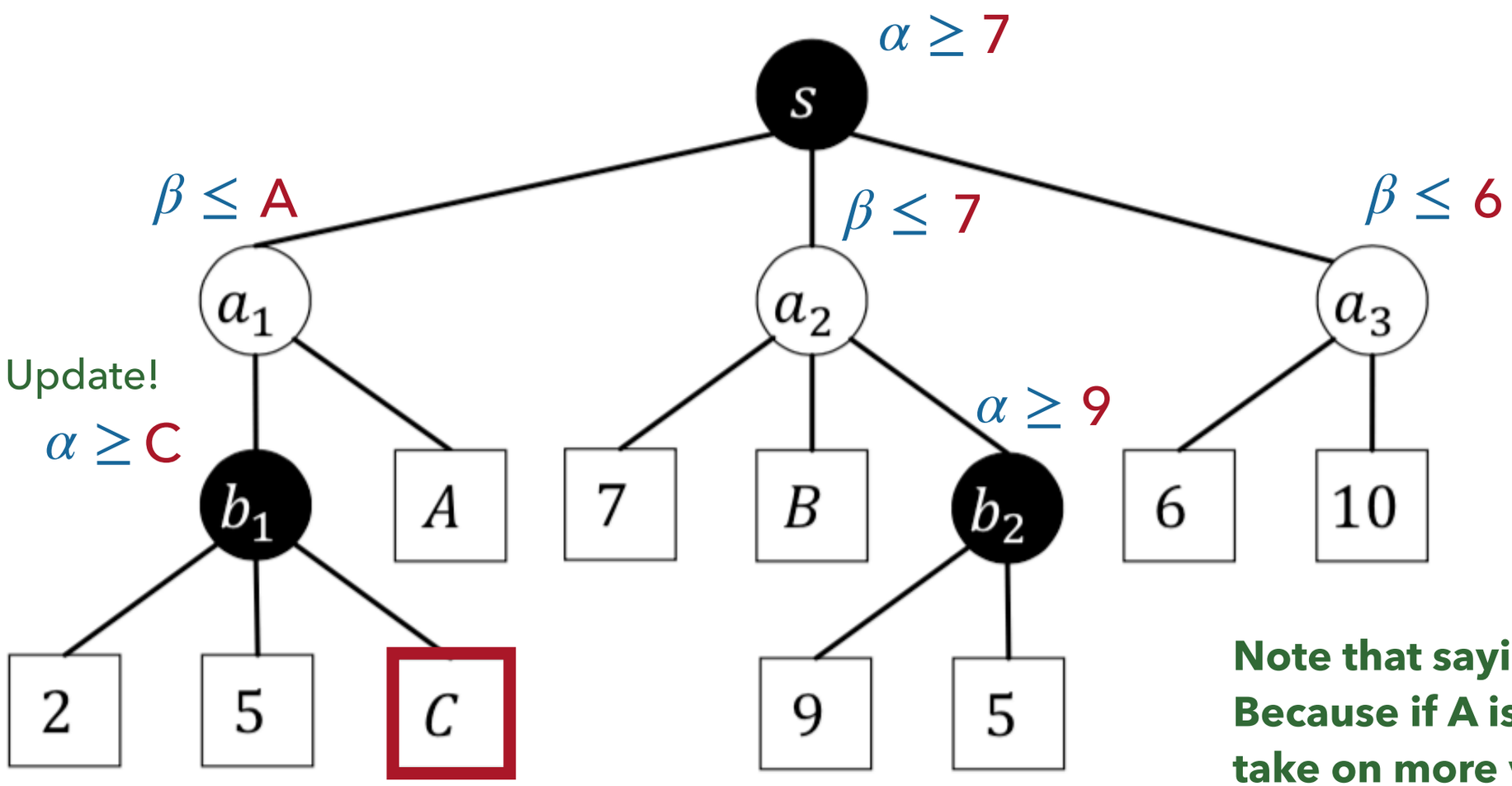
B > 6

C < _____

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > 7

B > 6

C < A

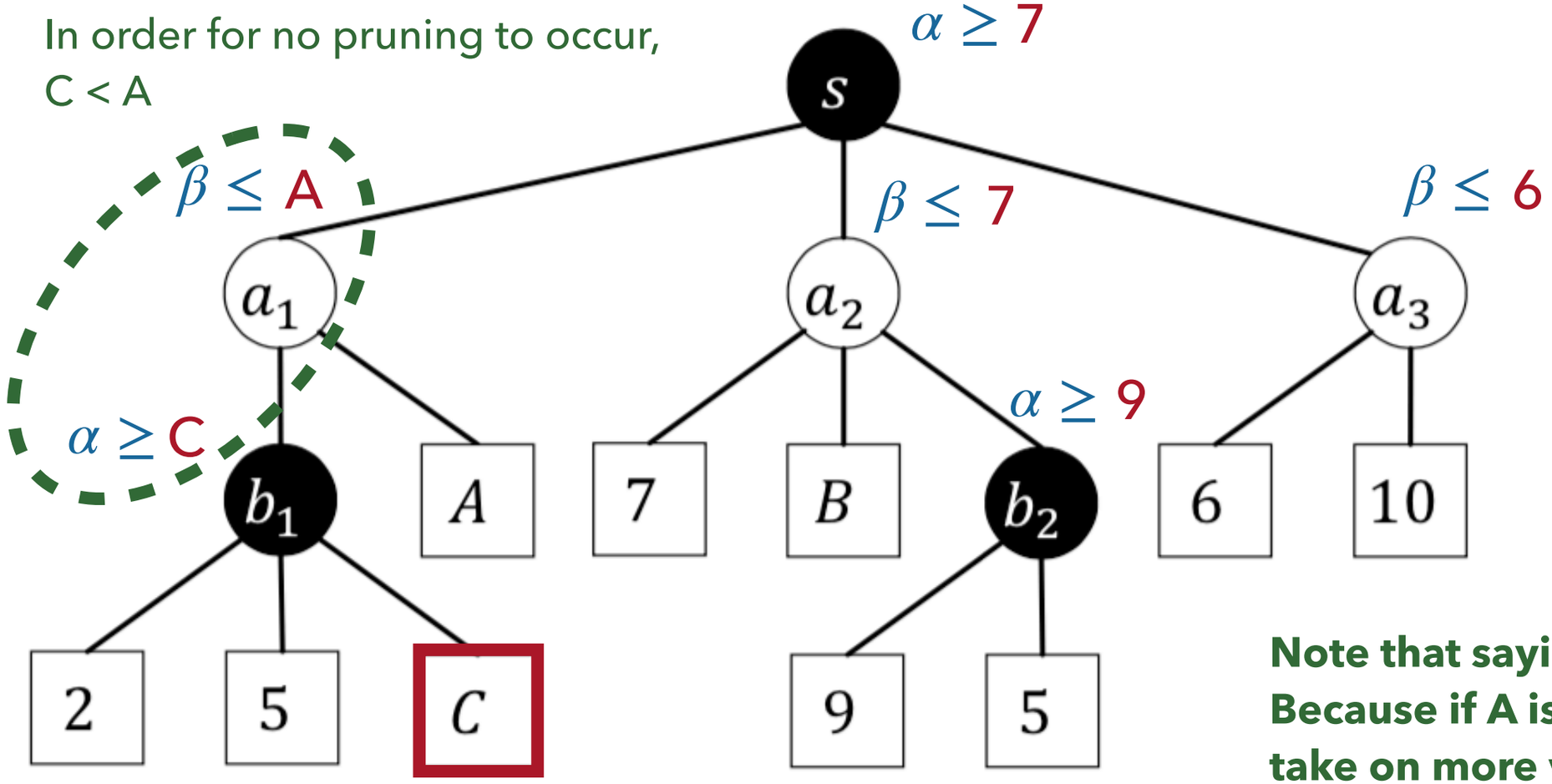
Note that saying C < 8 is insufficient. Because if A is very large, then C can take on more values than merely C < 8.

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.

In order for no pruning to occur,
C < A



A > 7

B > 6

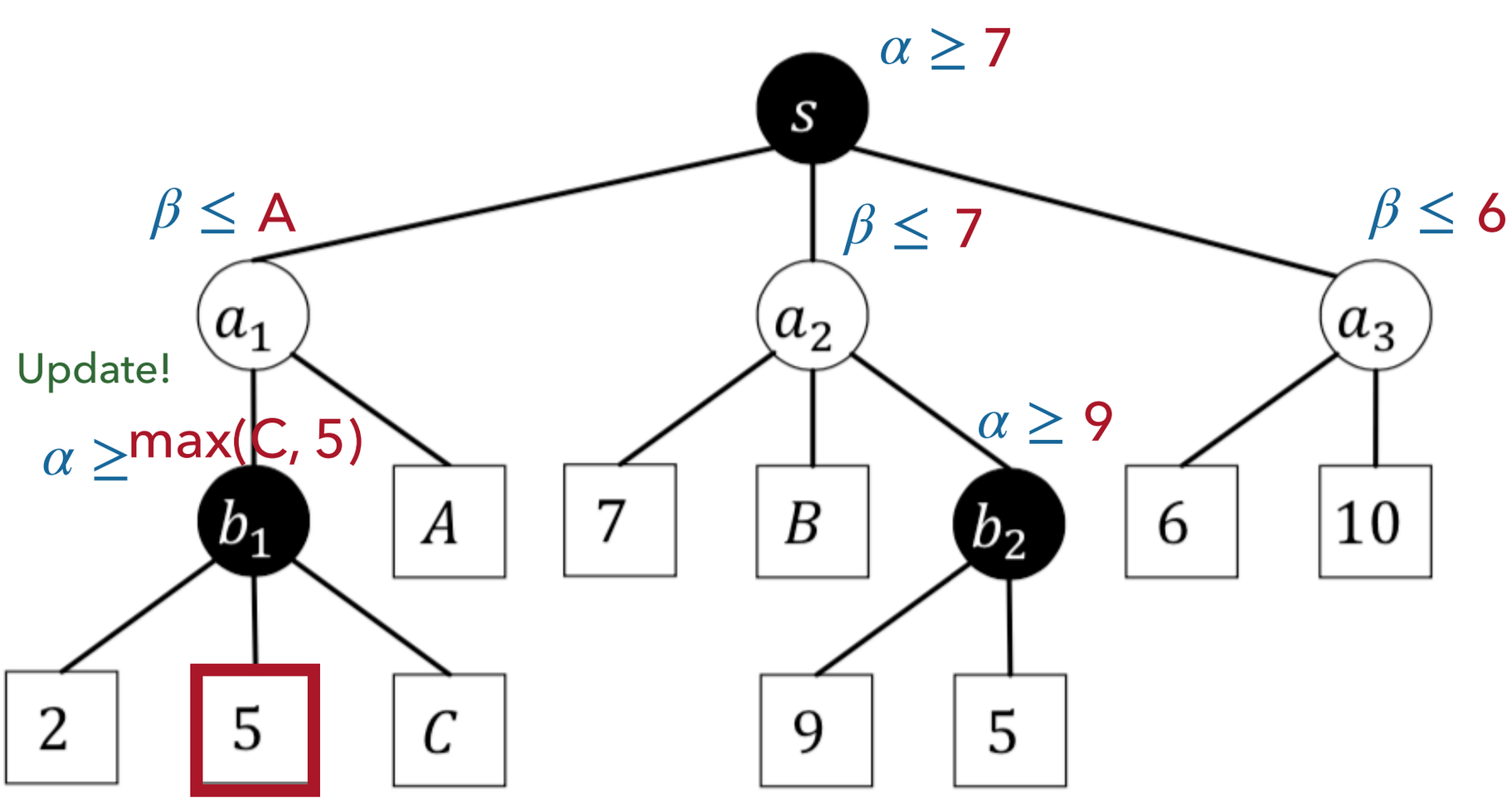
C < A

Note that saying C < 8 is insufficient. Because if A is very large, then C can take on more values than merely C < 8.

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.



A > 7

B > 6

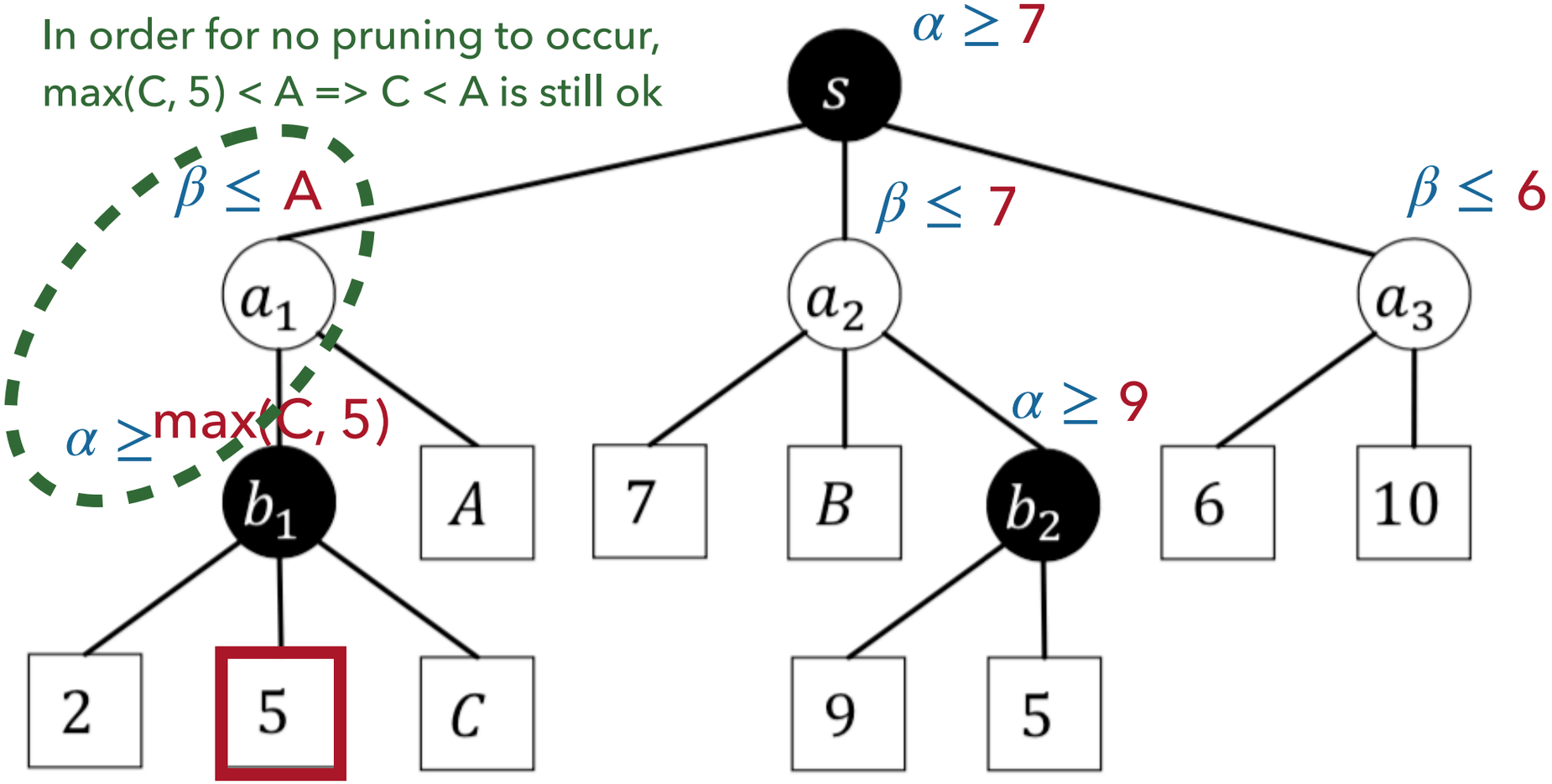
C < A

EXTRA QUESTION 1

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

- Assume that we iterate over nodes from **right to left**; find values to ensure no arcs are pruned by α-β pruning.

In order for no pruning to occur,
max(C, 5) < A => C < A is still ok



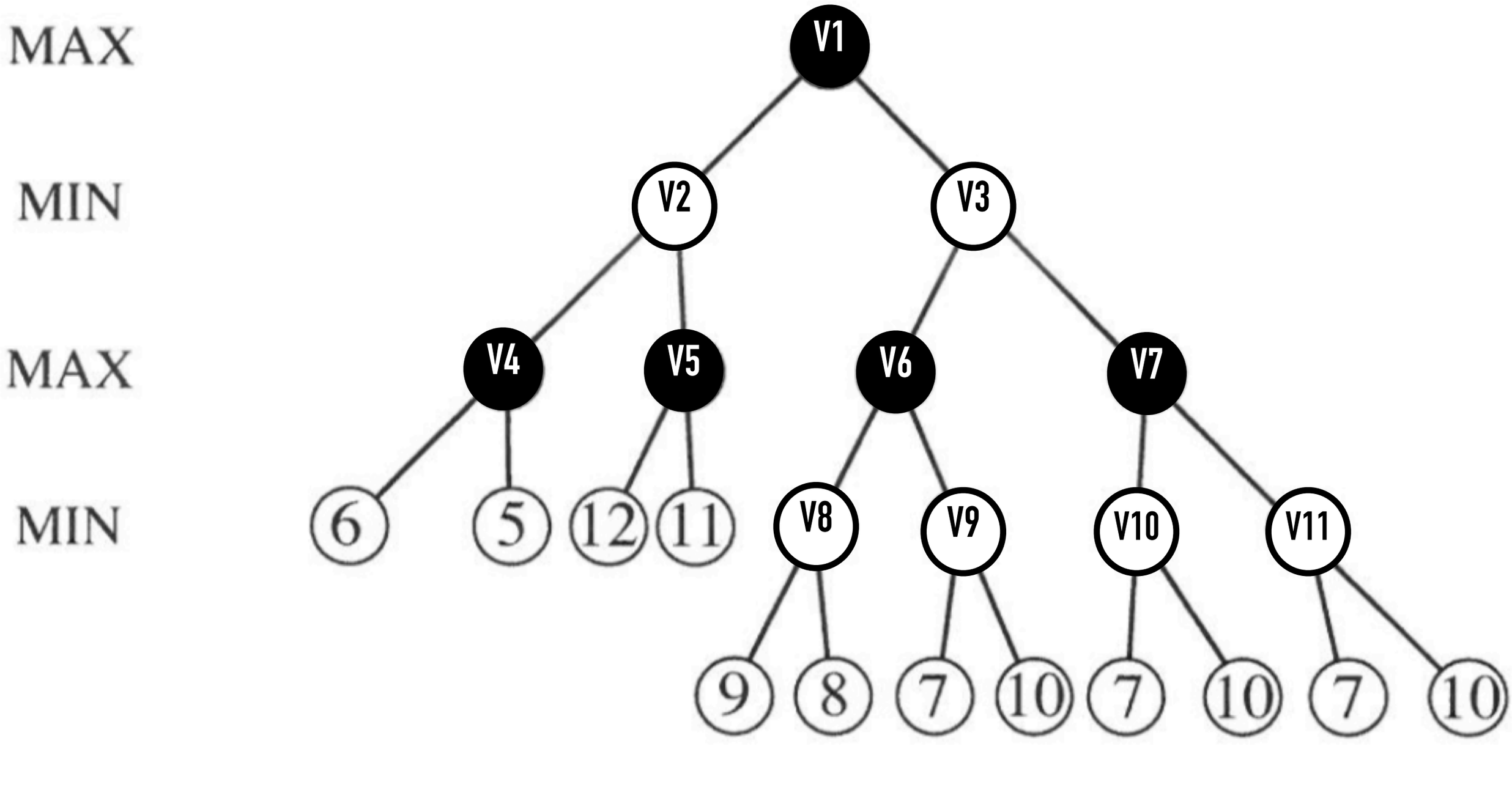
A > 7

B > 6

C < A

EXTRA QUESTION 2

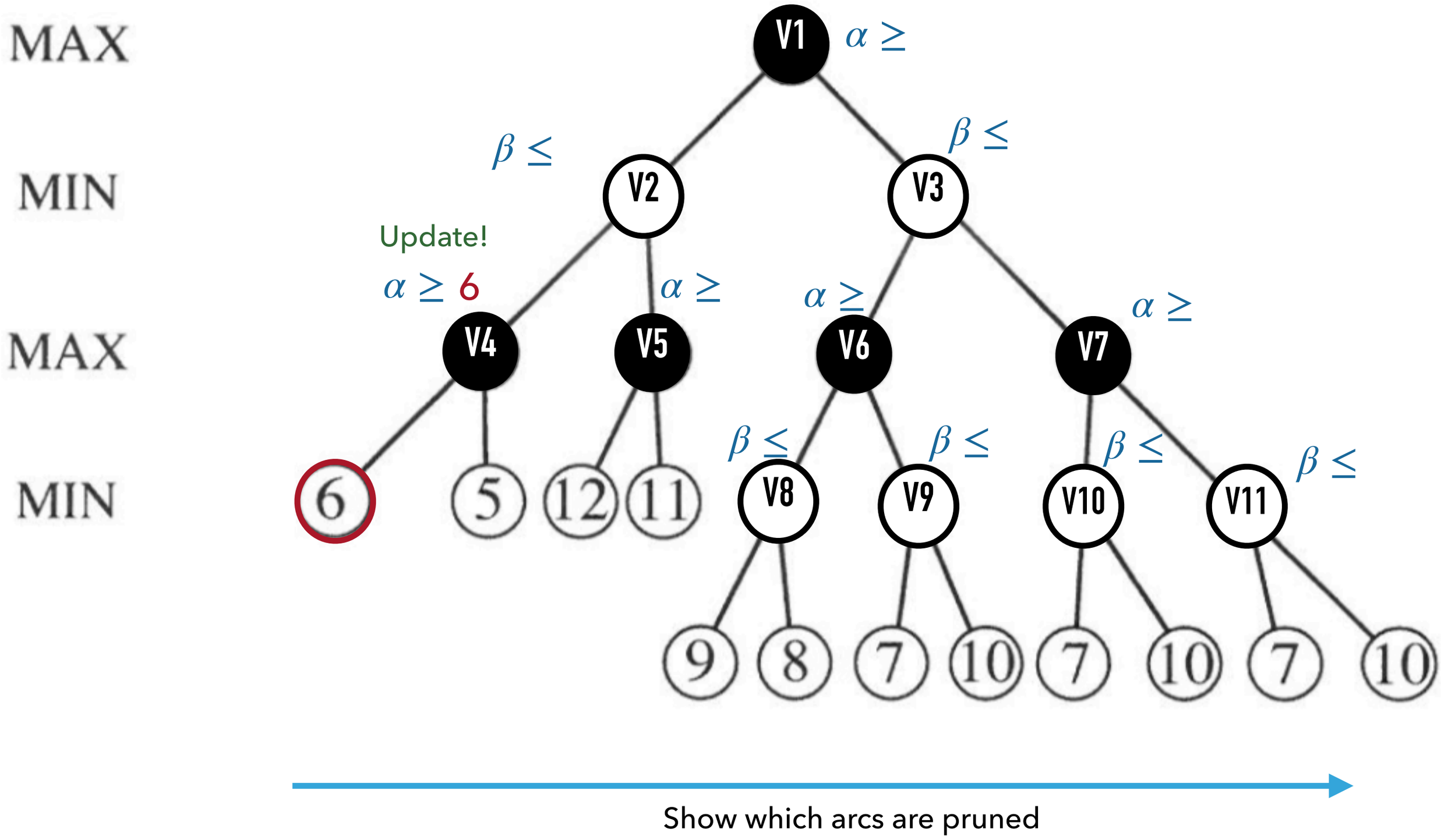
PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



Show which arcs are pruned

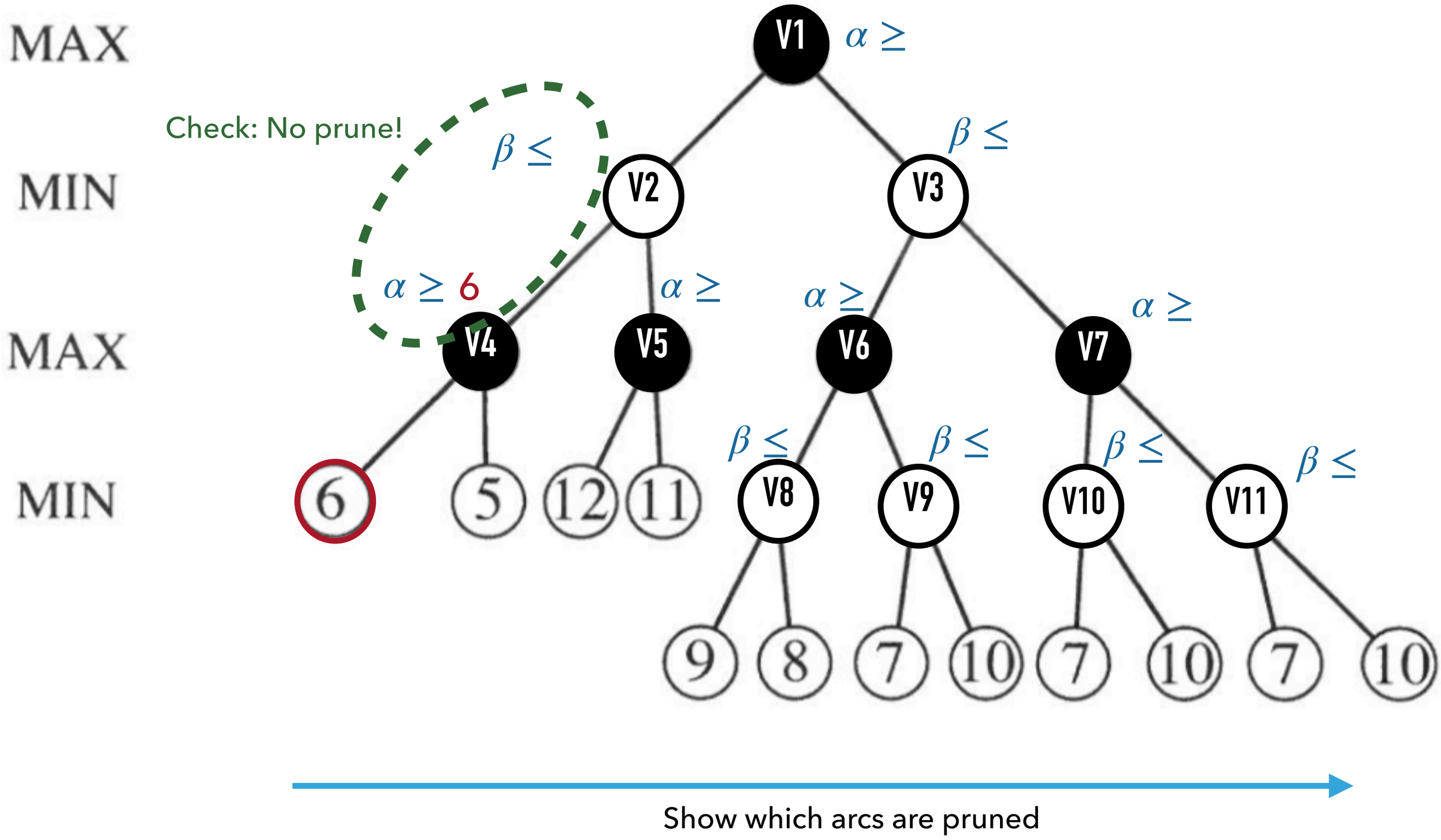
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



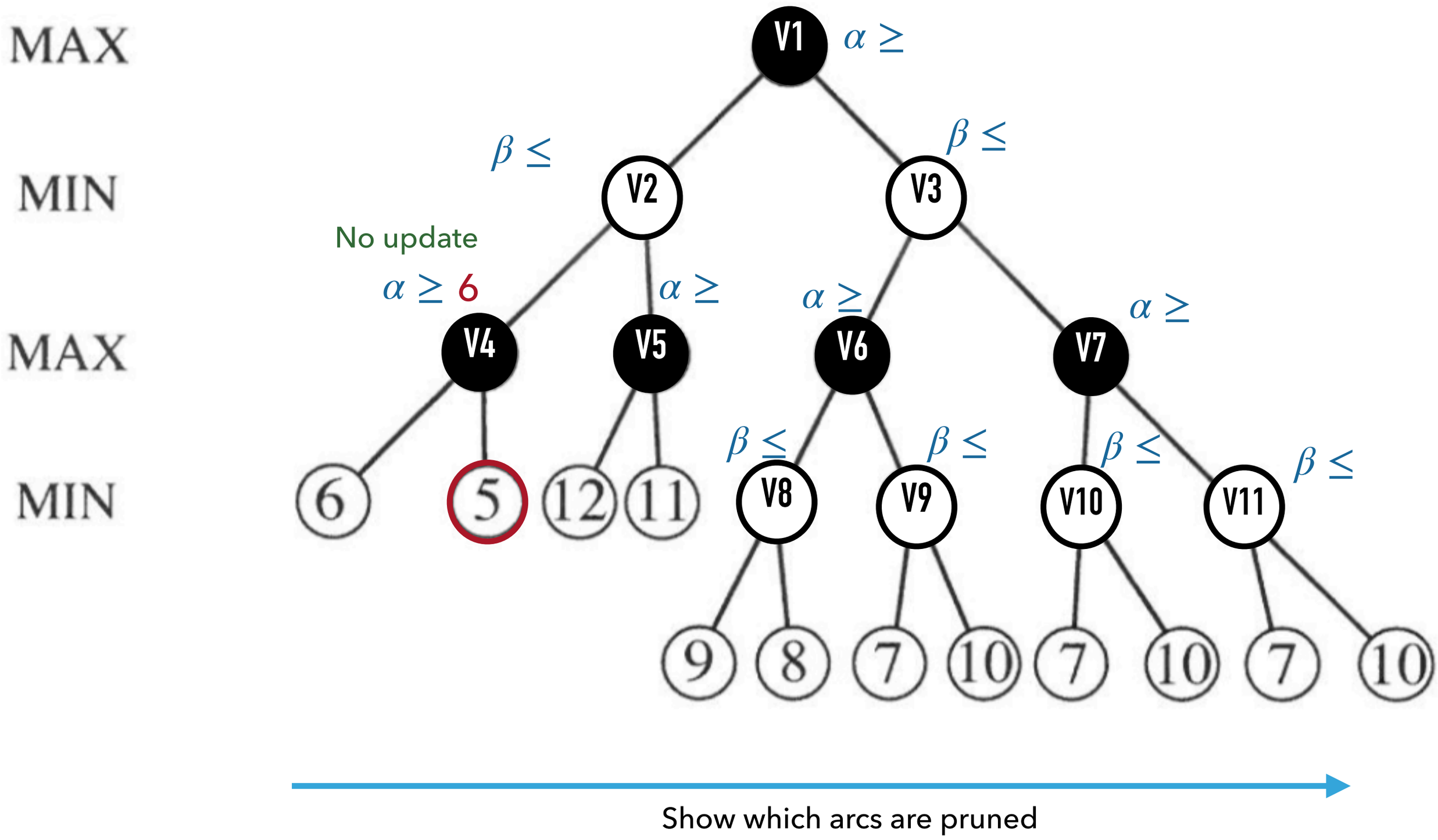
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



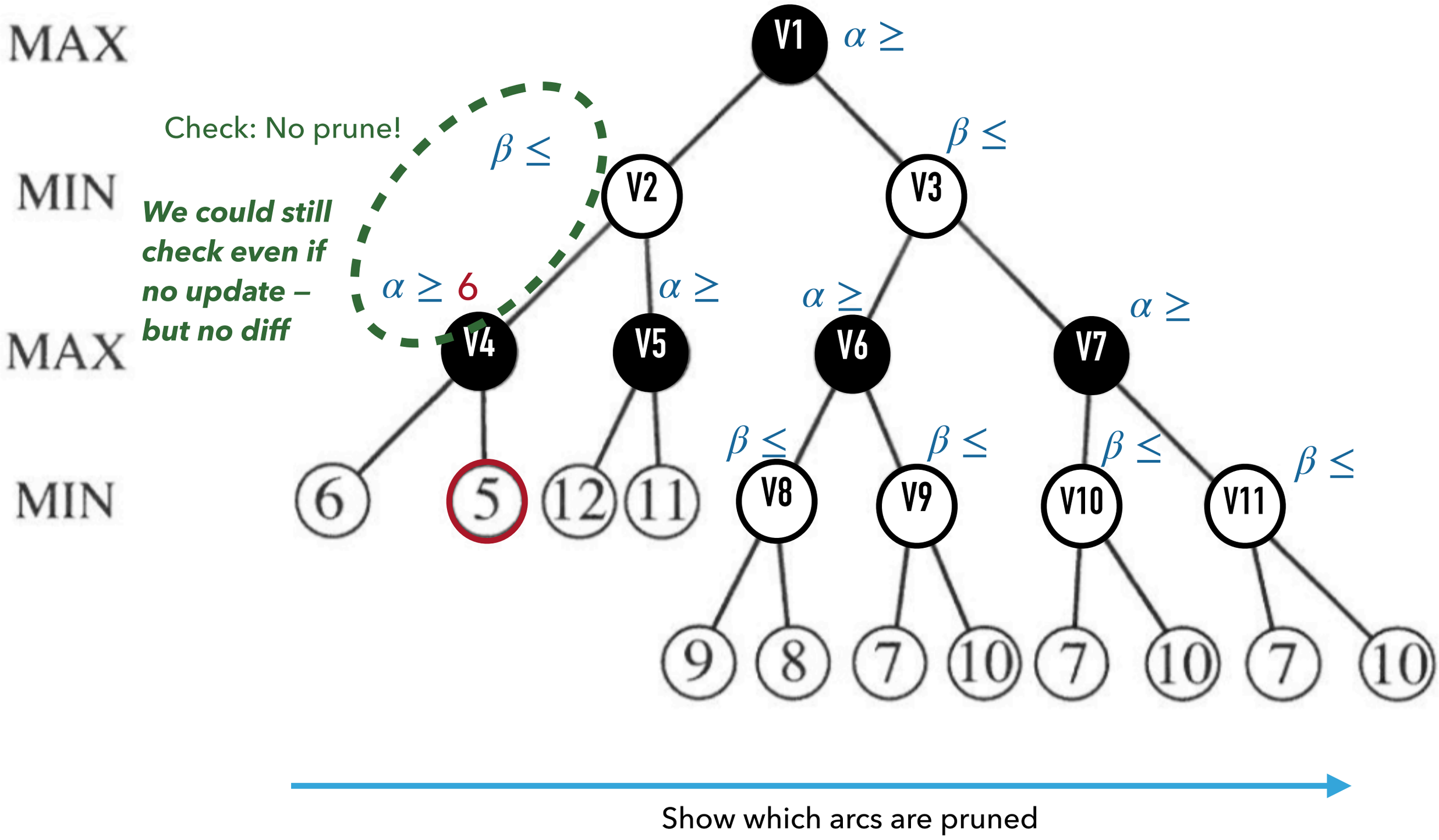
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



EXTRA QUESTION 2

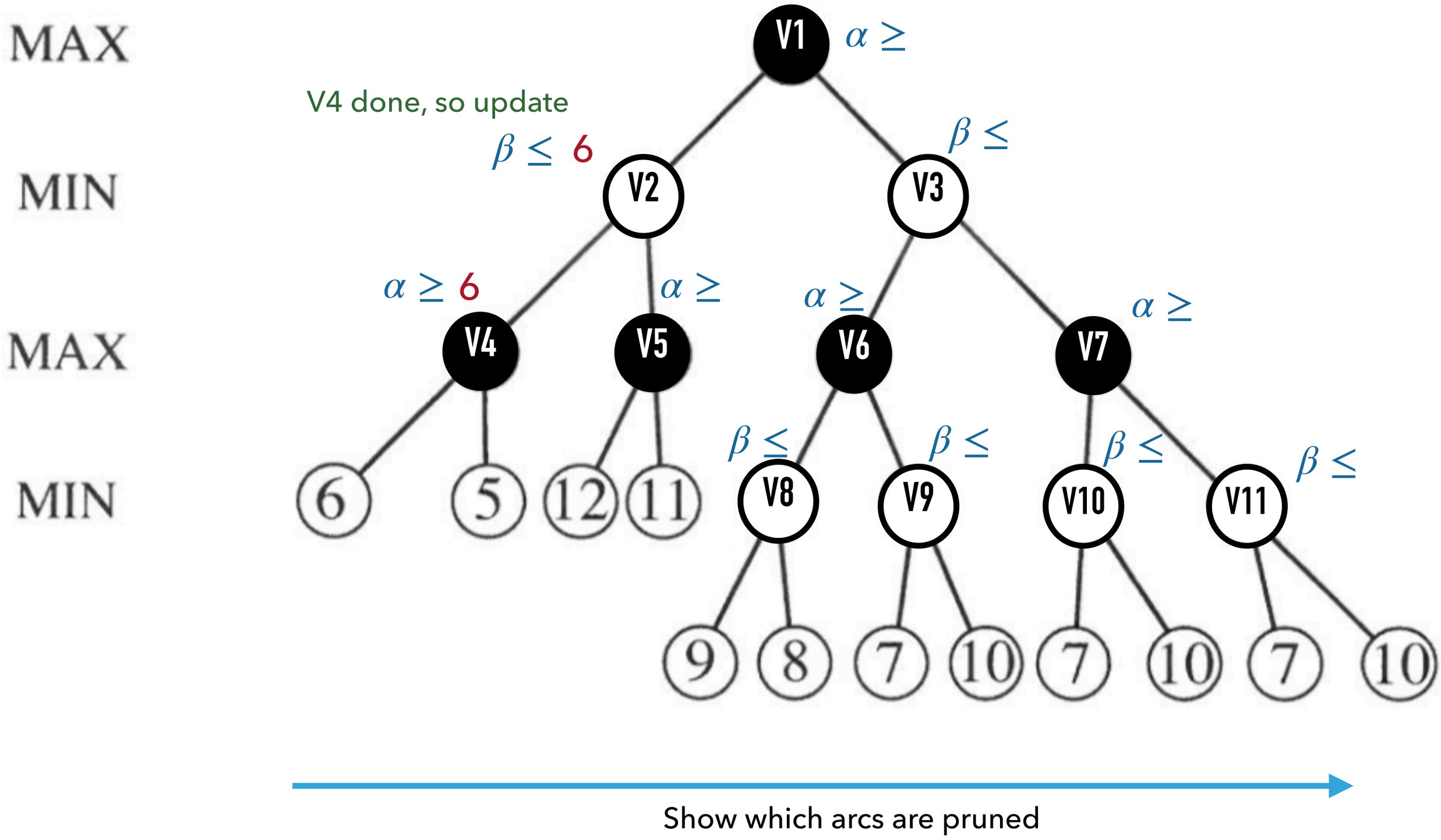
PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



Show which arcs are pruned

EXTRA QUESTION 2

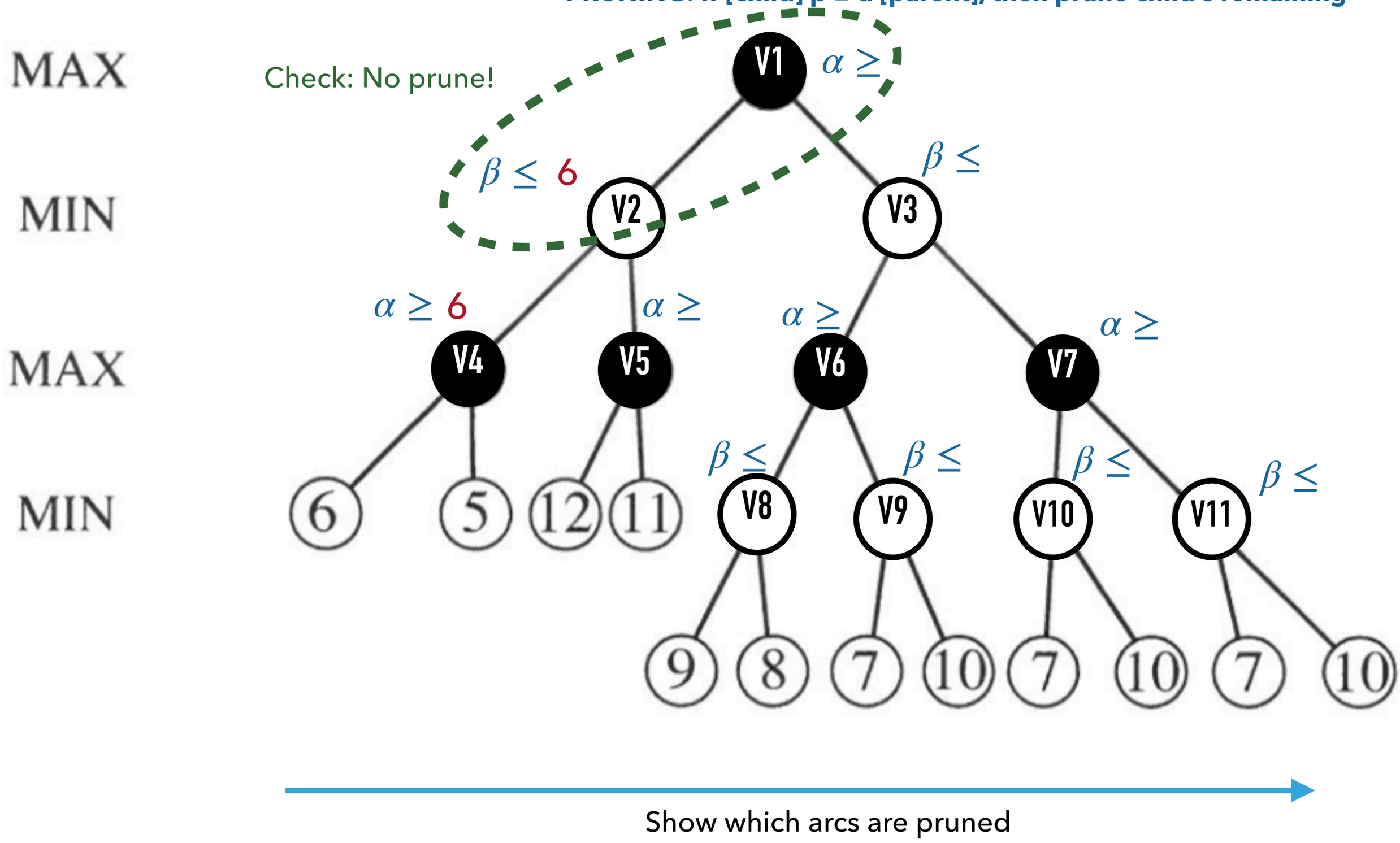
PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

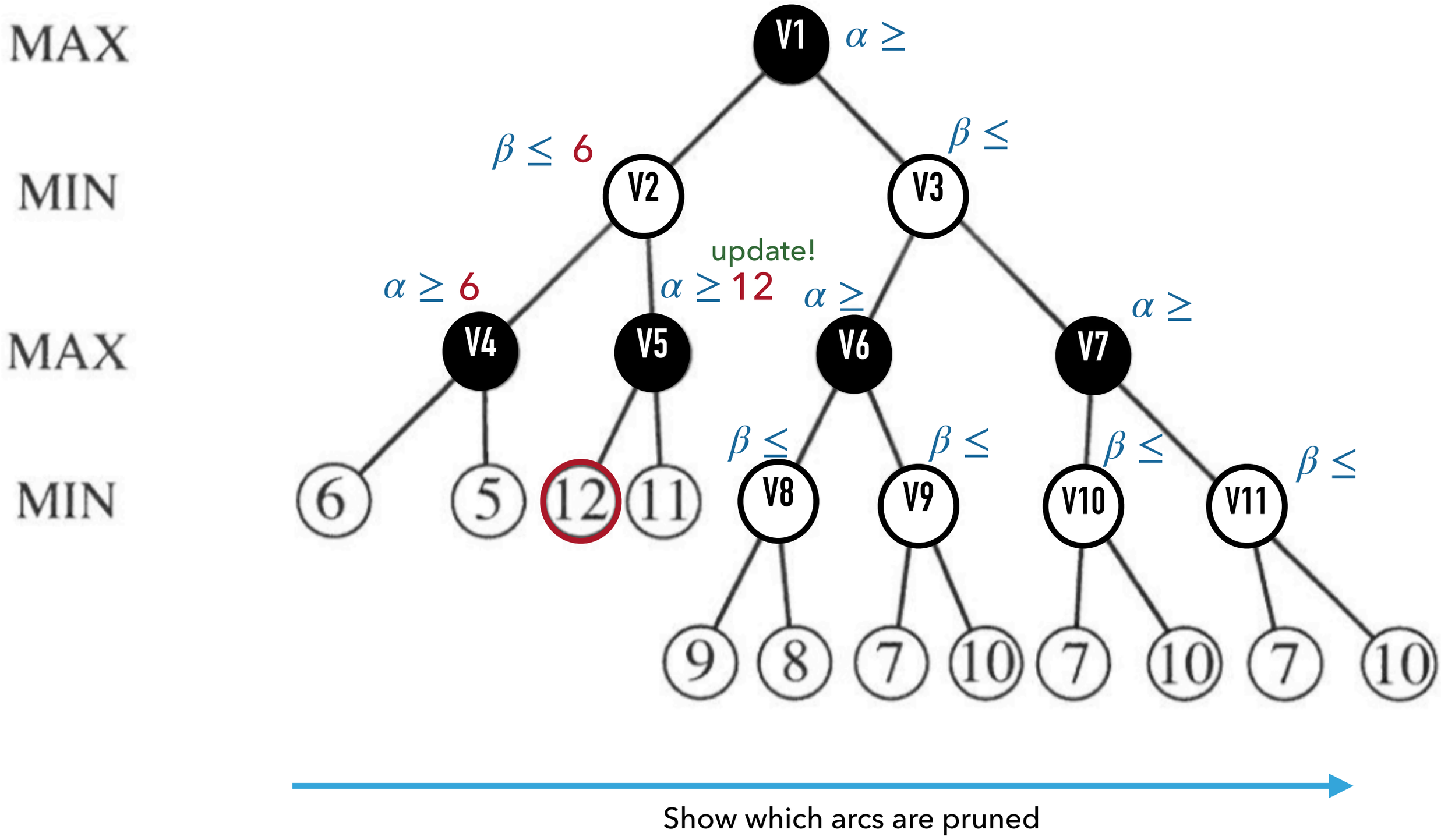
Check: No prune!



Show which arcs are pruned

EXTRA QUESTION 2

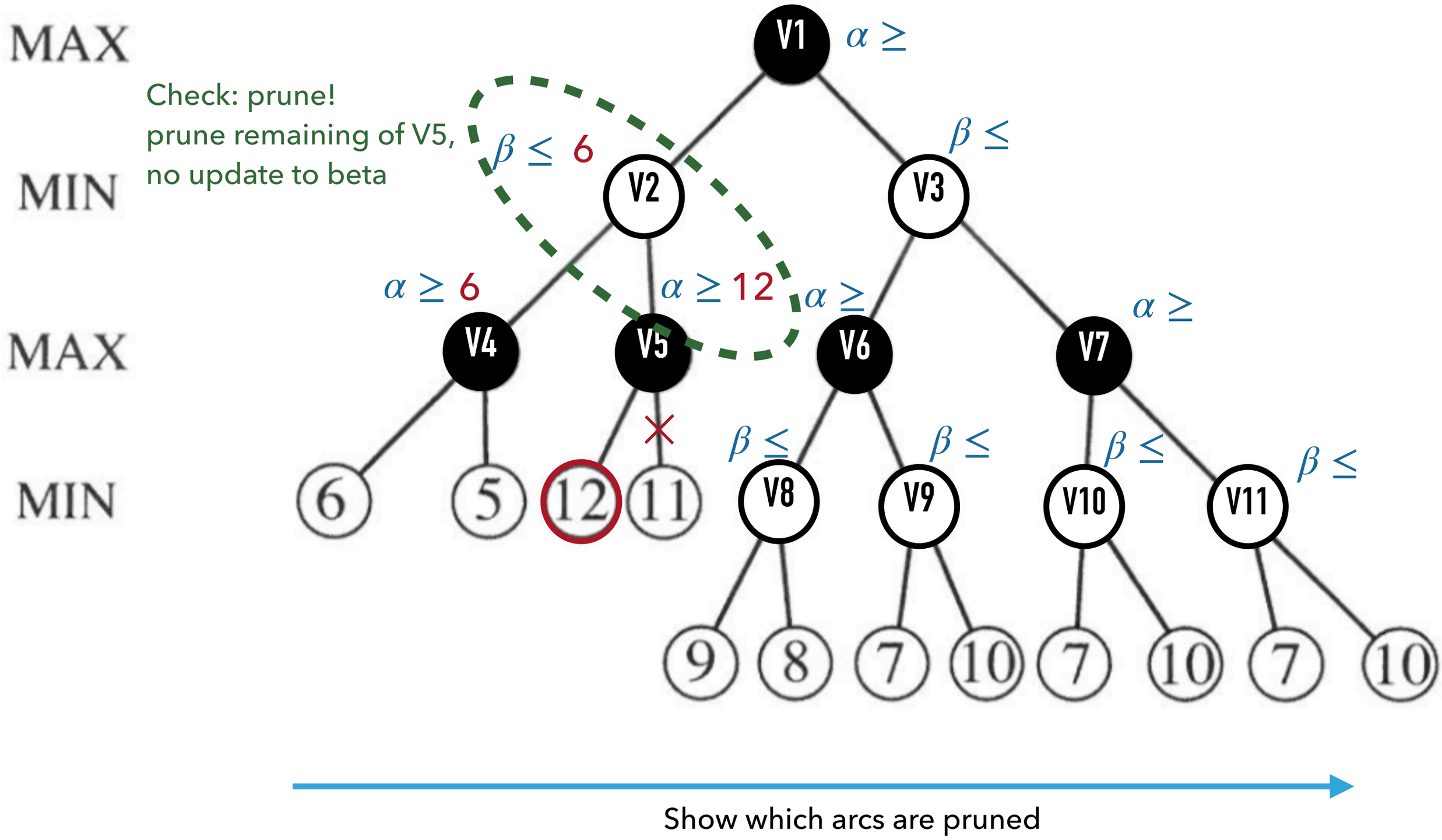
PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



Show which arcs are pruned

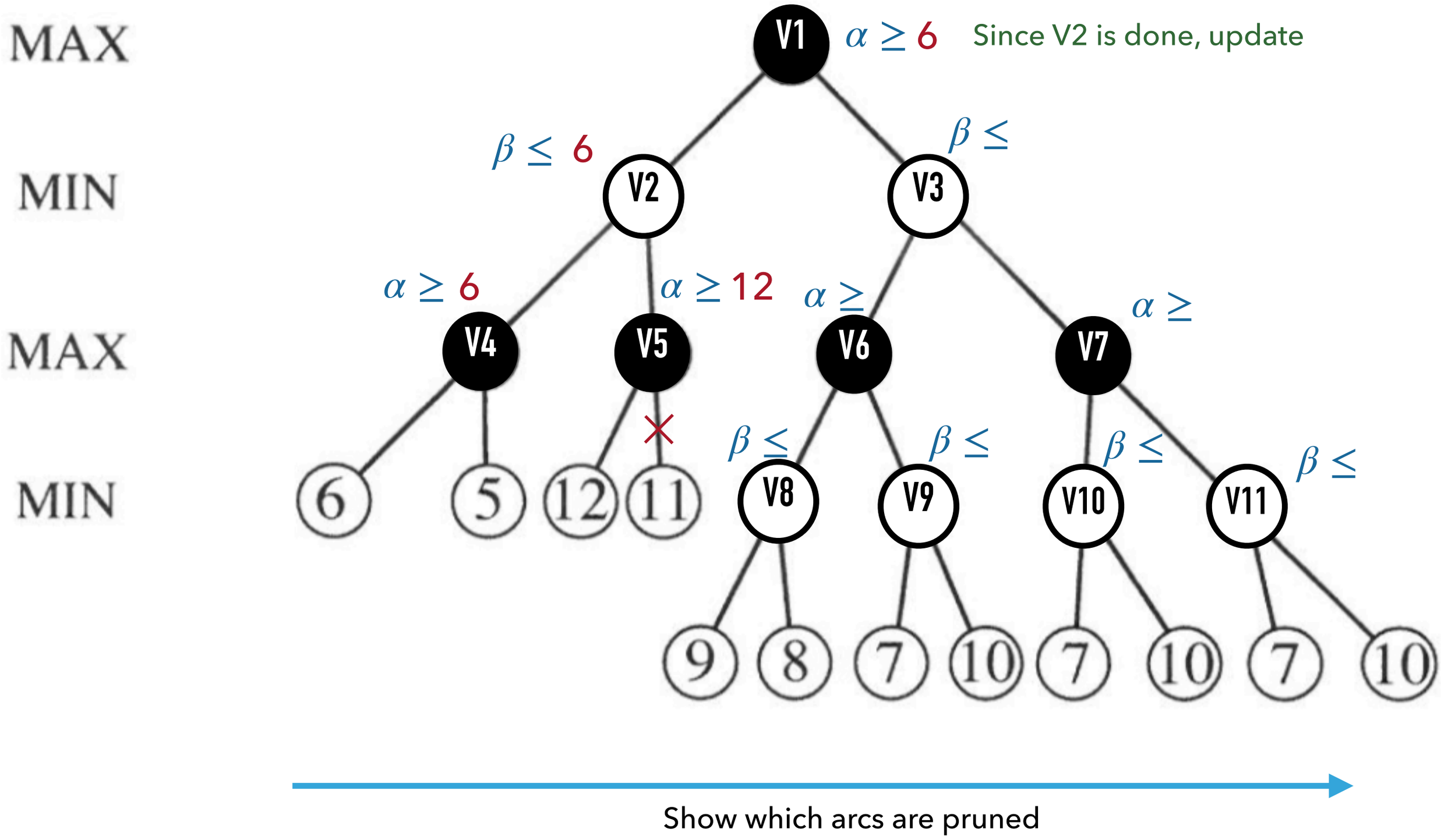
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



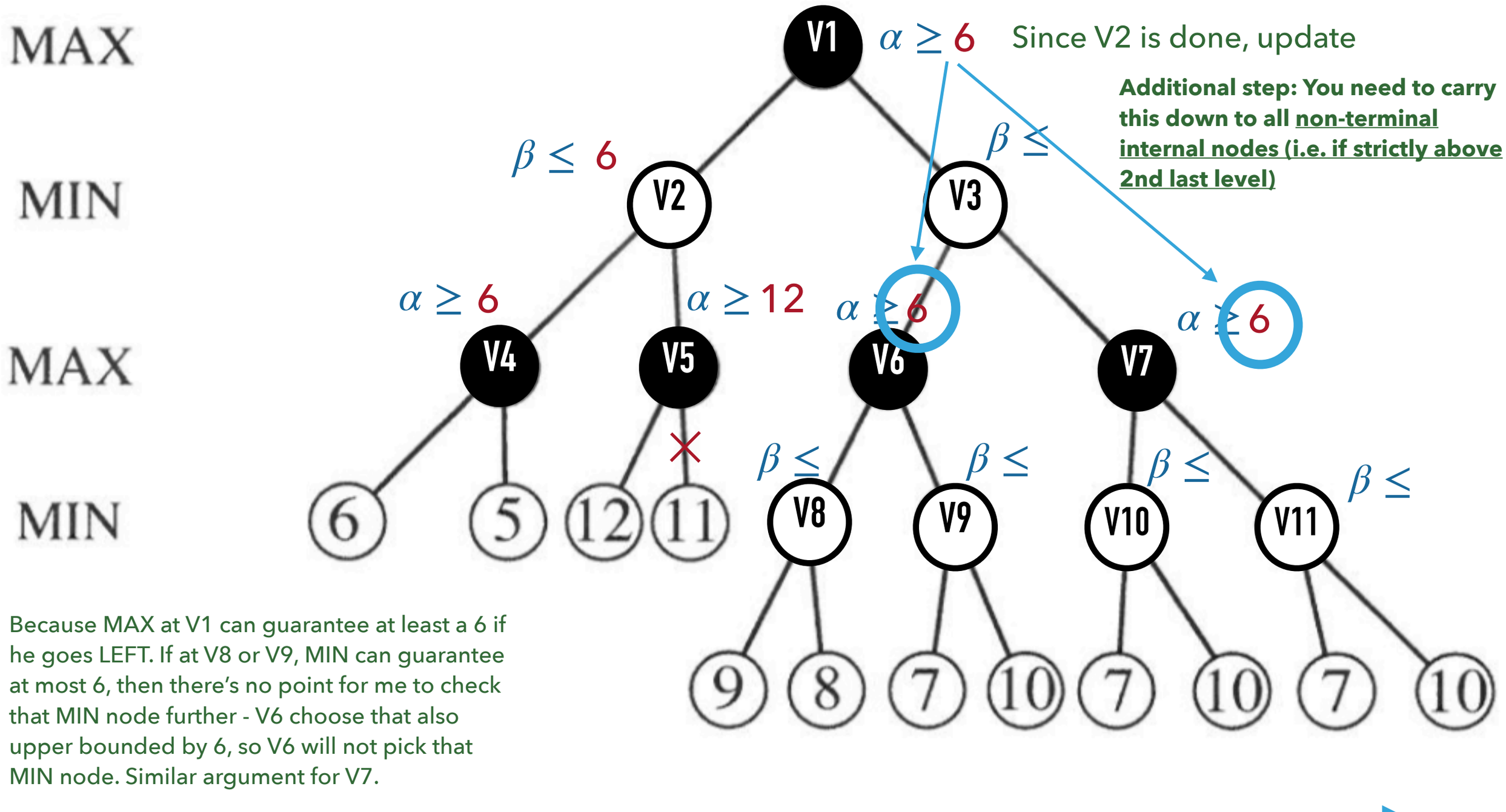
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

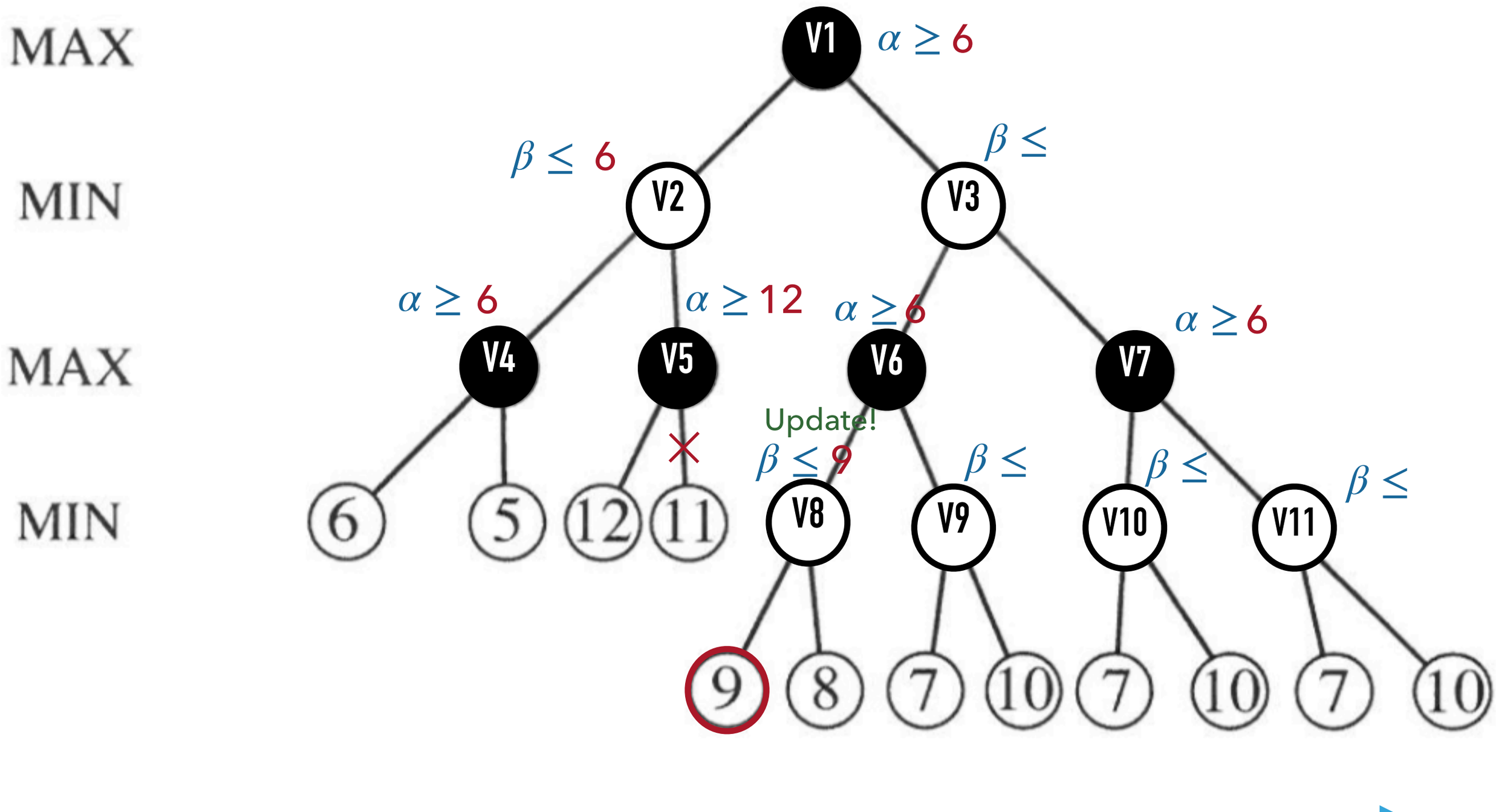


Because MAX at V1 can guarantee at least a 6 if he goes LEFT. If at V8 or V9, MIN can guarantee at most 6, then there's no point for me to check that MIN node further - V6 choose that also upper bounded by 6, so V6 will not pick that MIN node. Similar argument for V7.

Show which arcs are pruned

EXTRA QUESTION 2

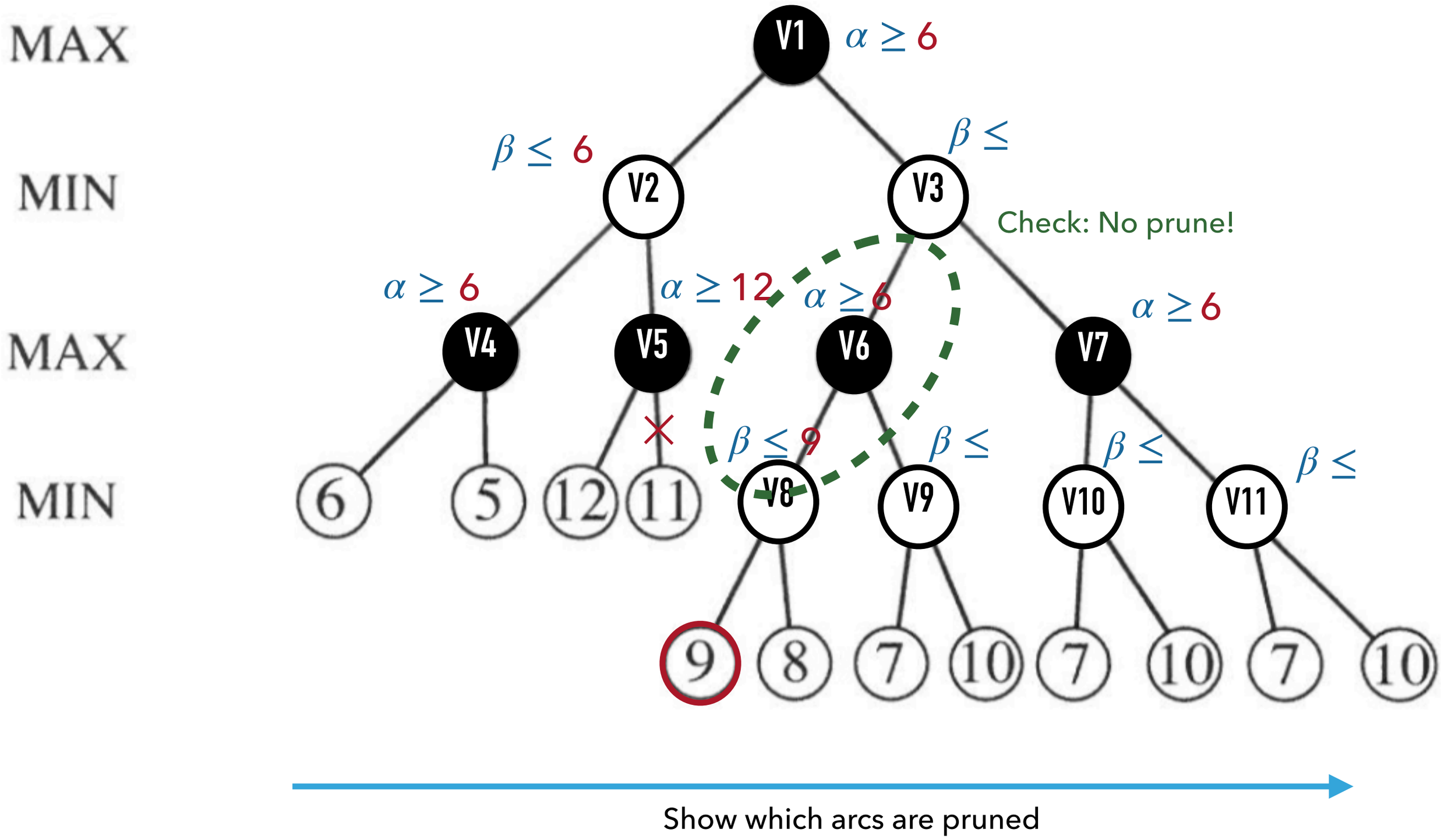
PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



Show which arcs are pruned

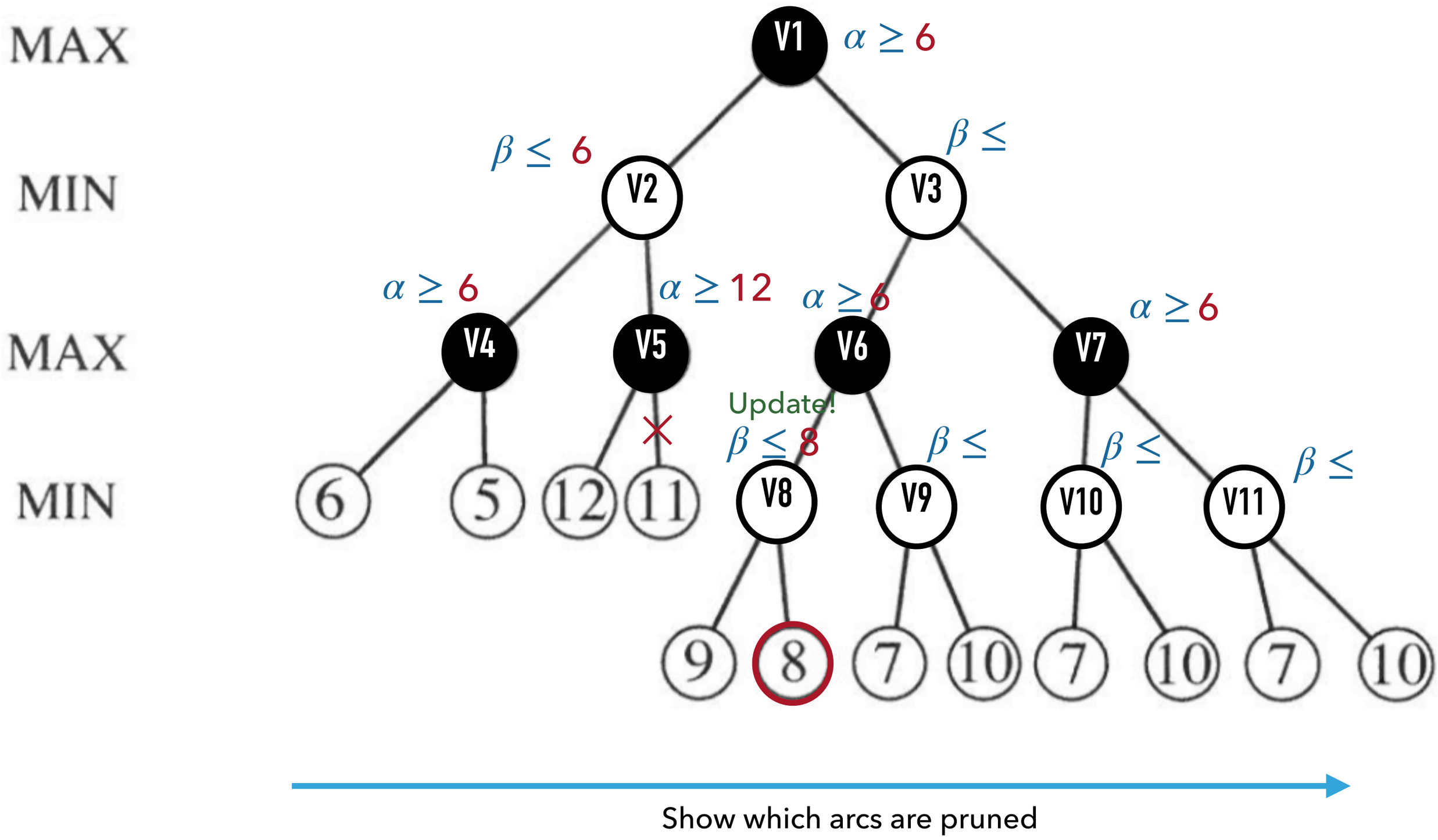
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



EXTRA QUESTION 2

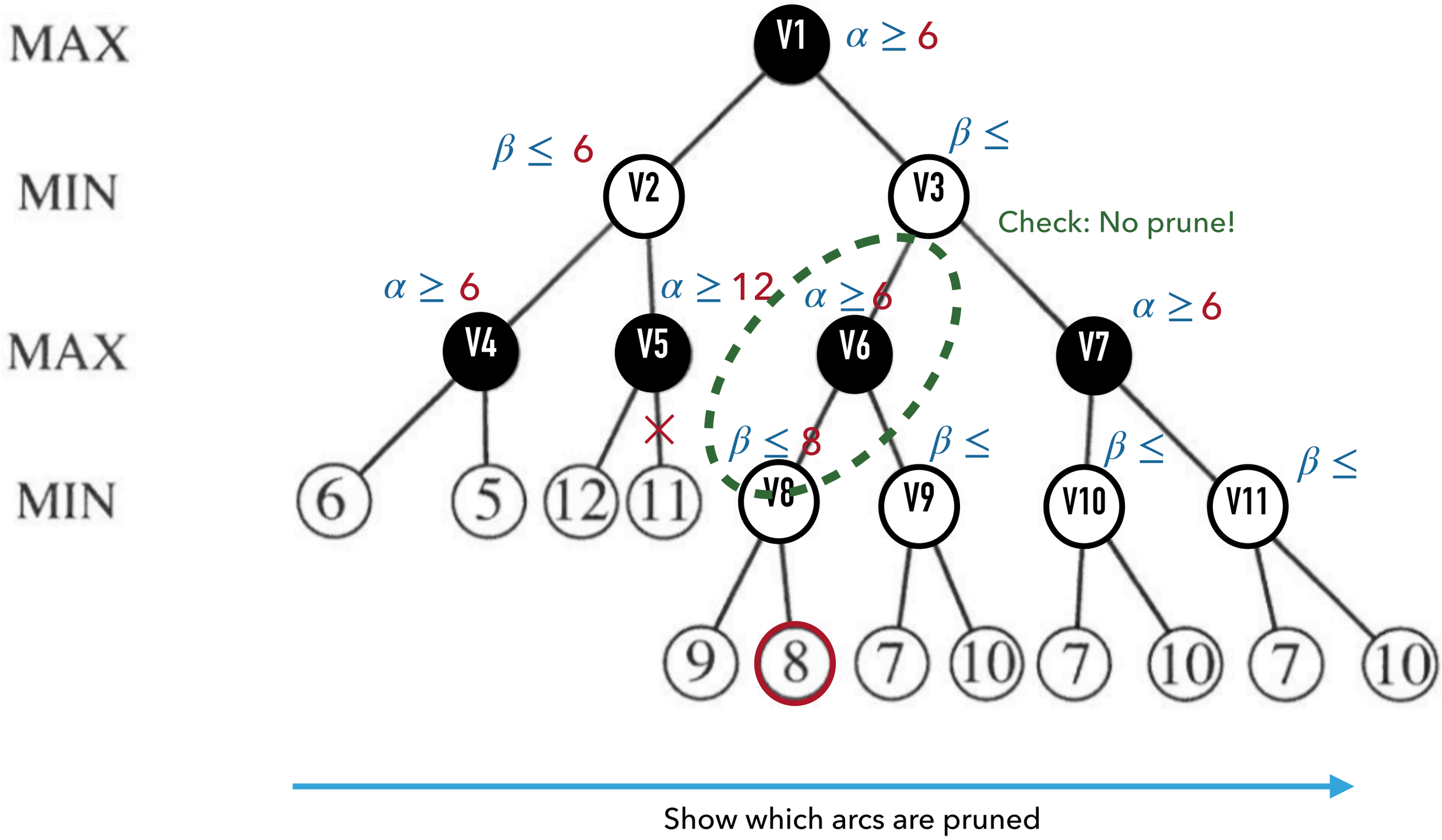
PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



Show which arcs are pruned

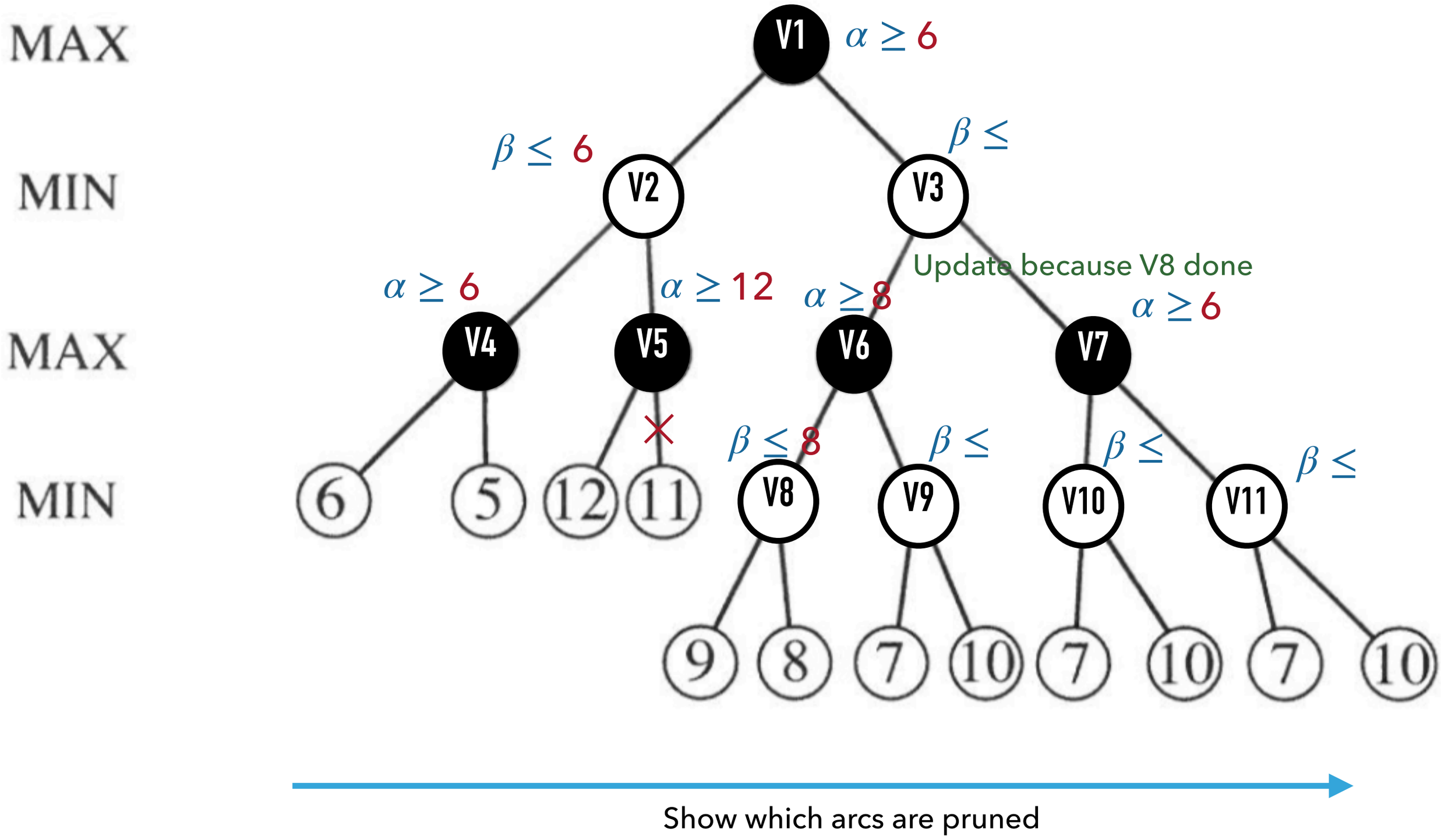
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



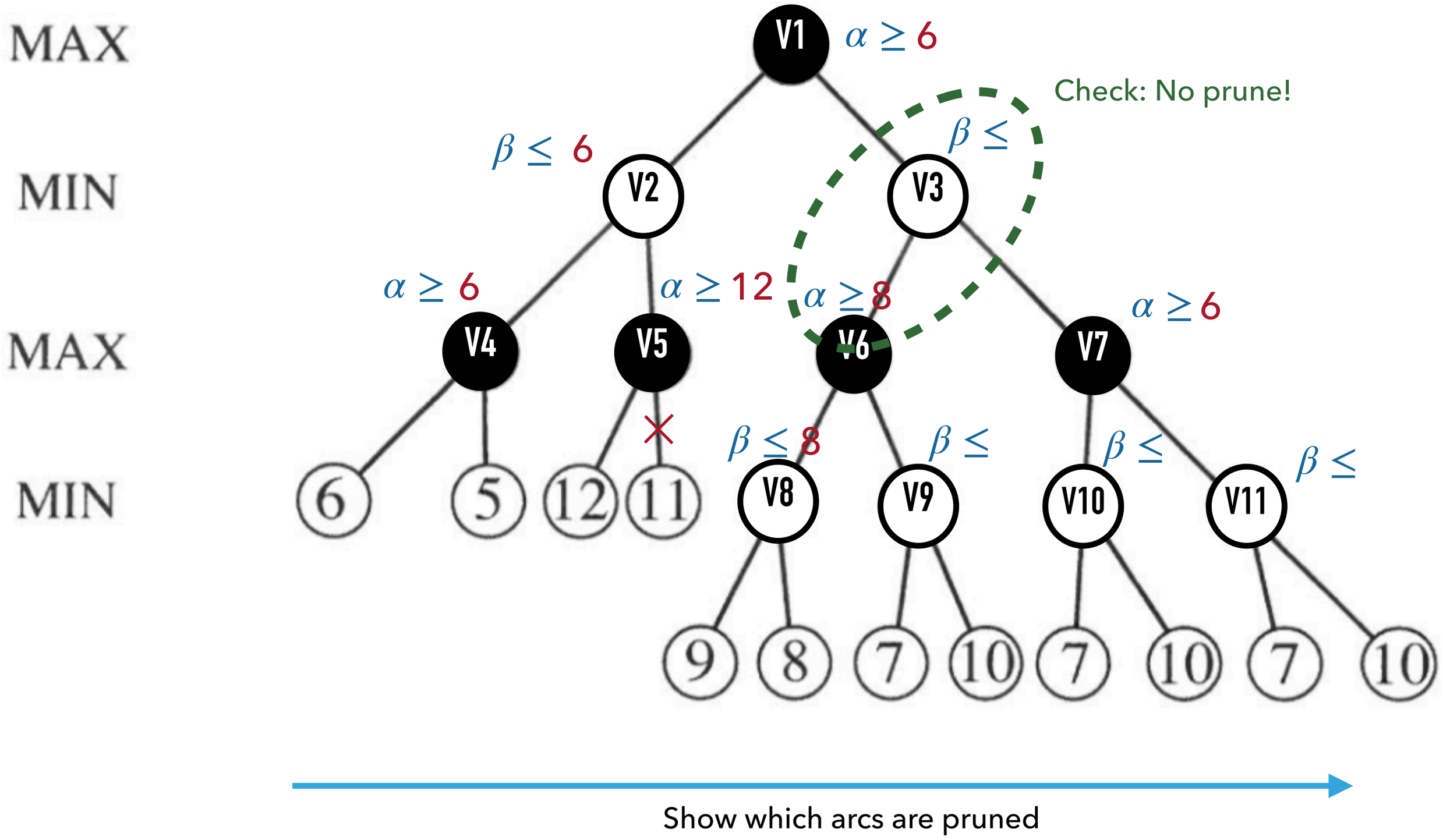
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



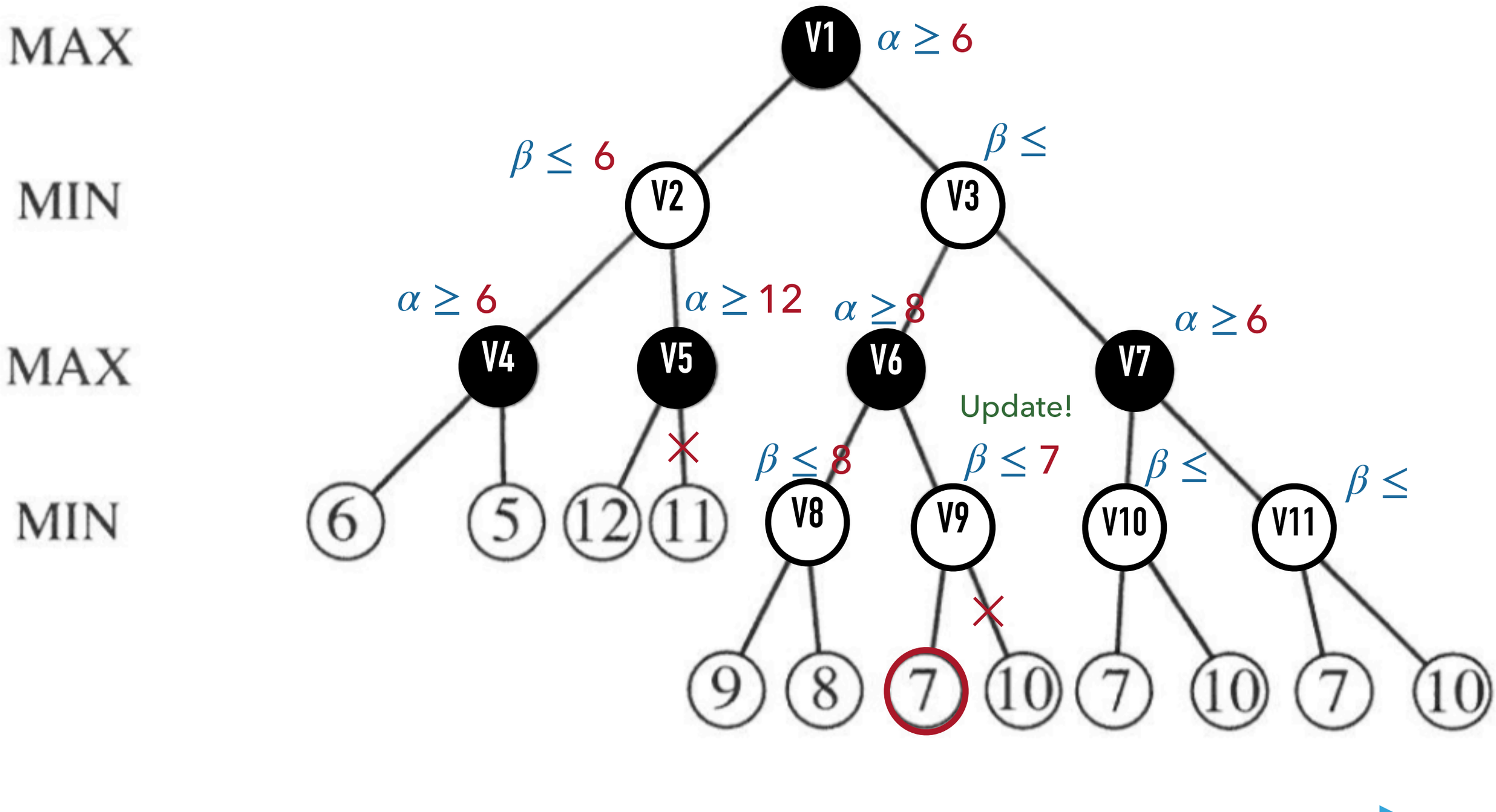
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



EXTRA QUESTION 2

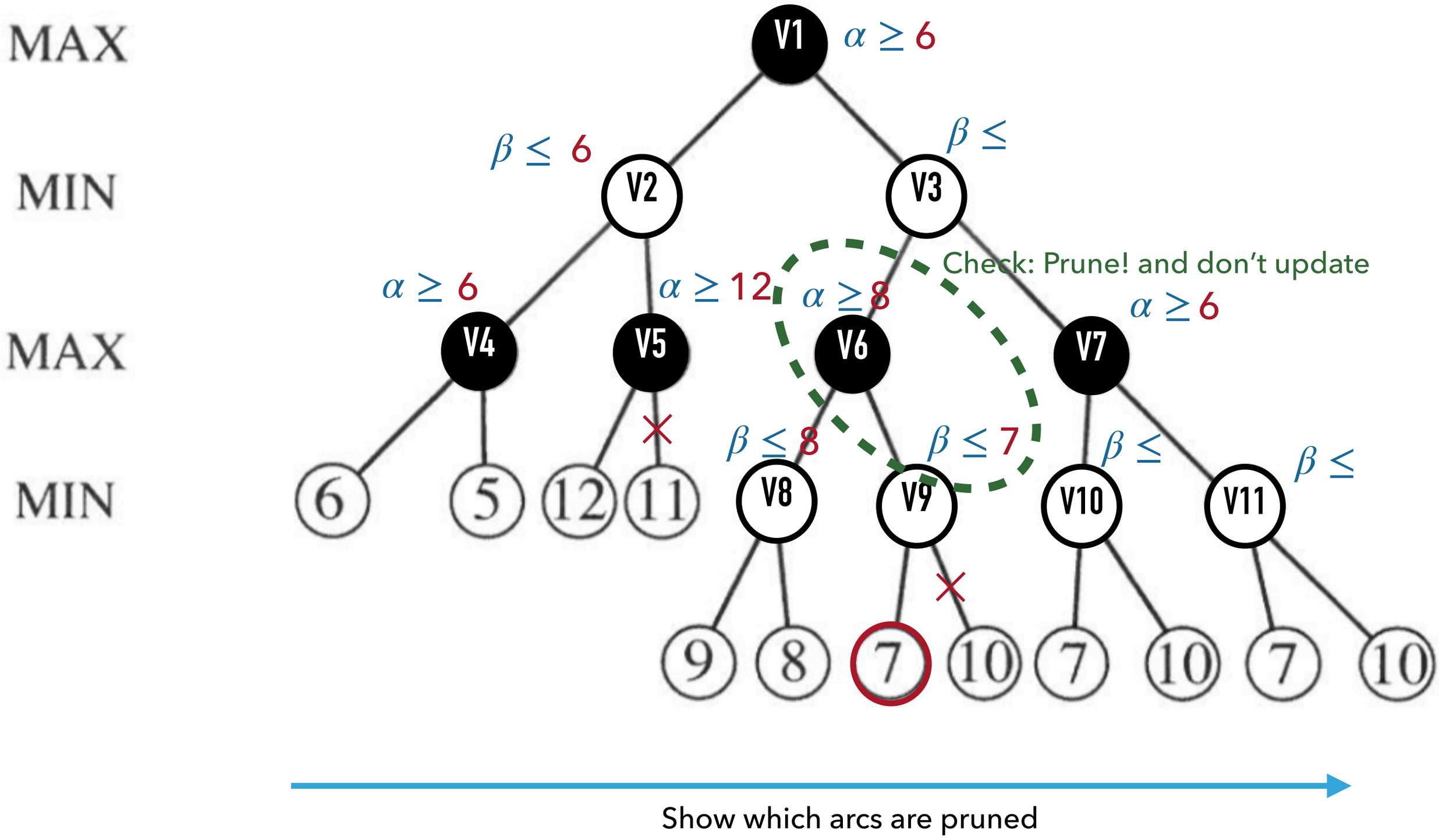
PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



Show which arcs are pruned

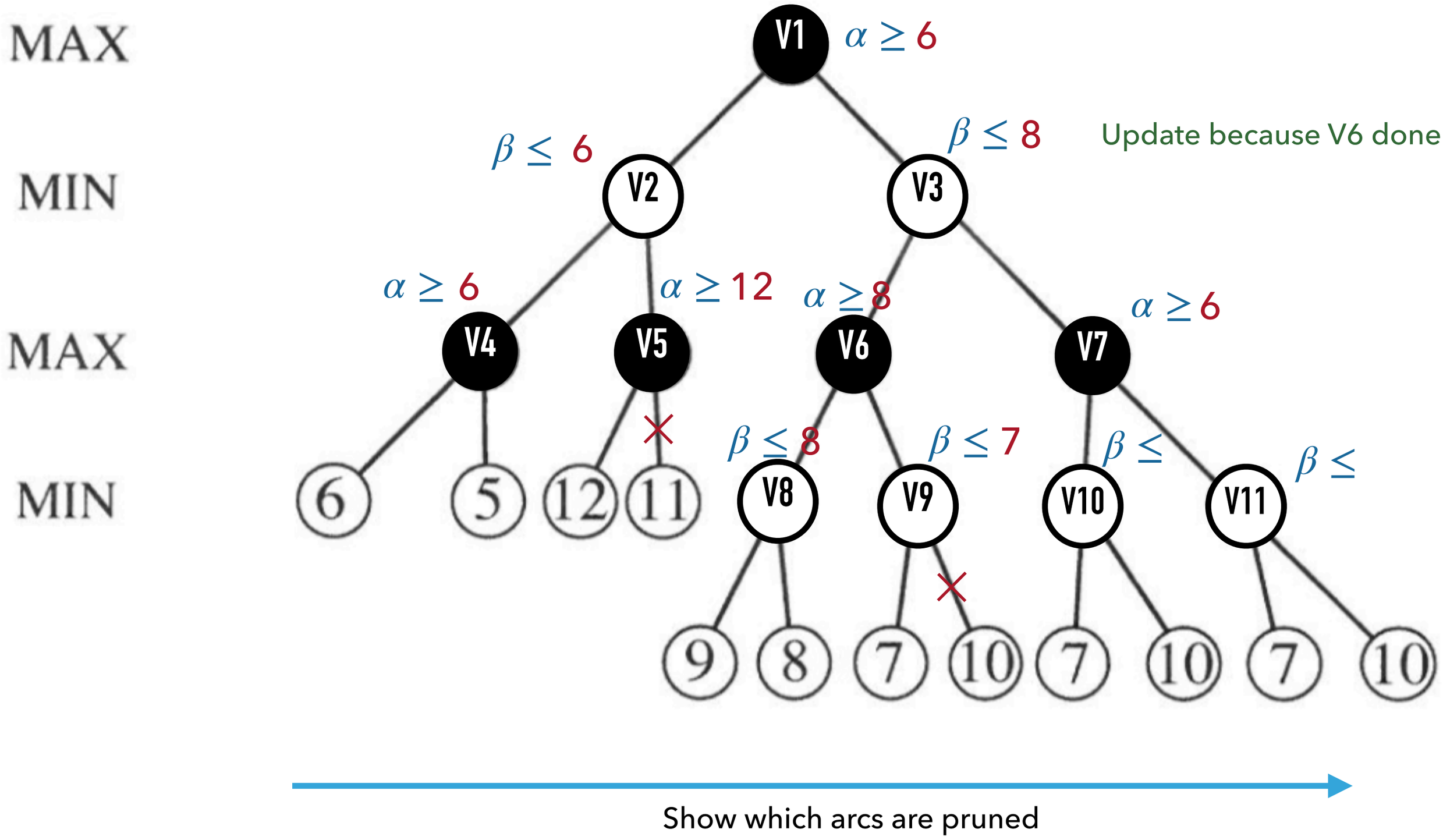
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining

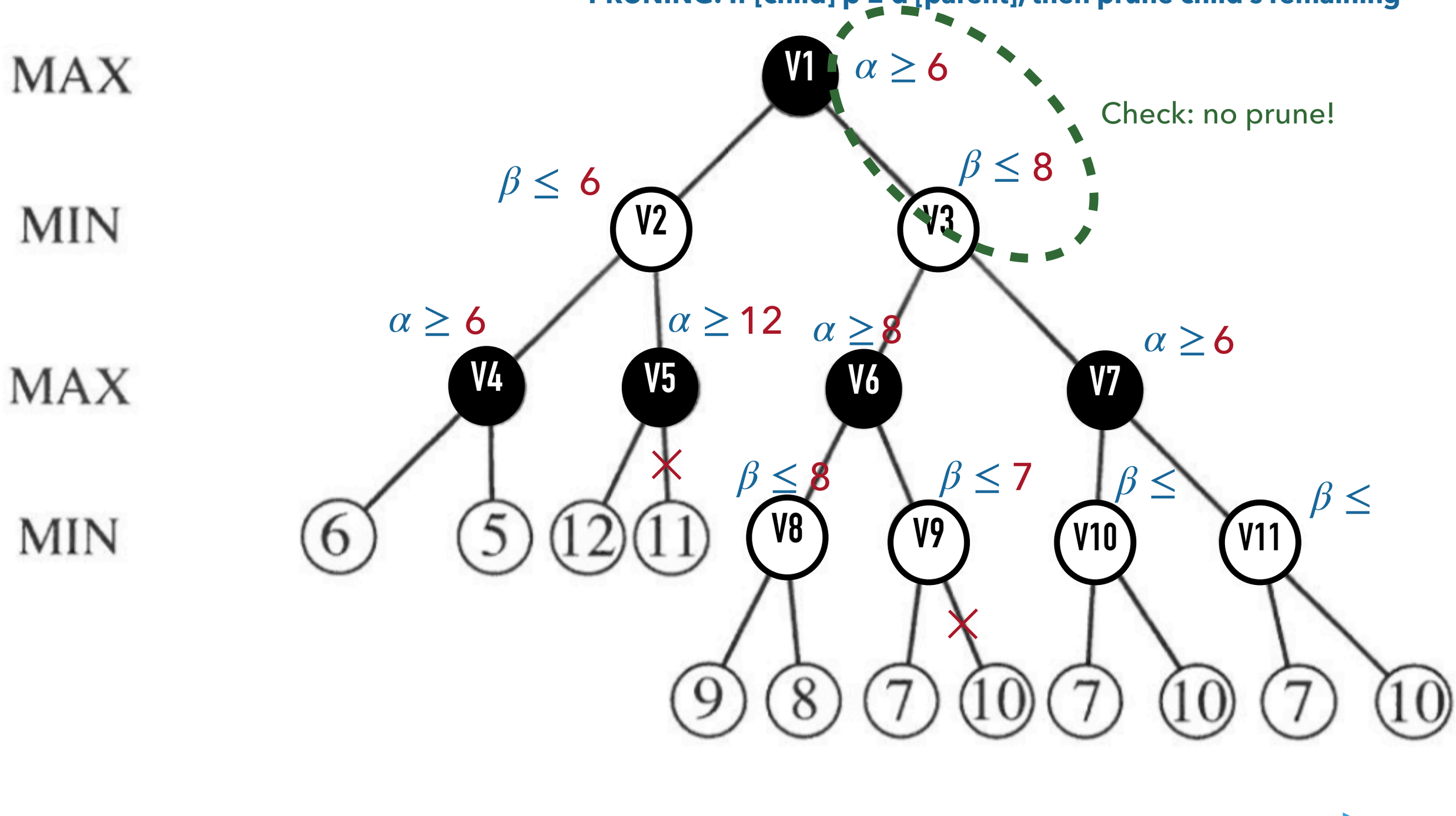


Update because V6 done

Show which arcs are pruned

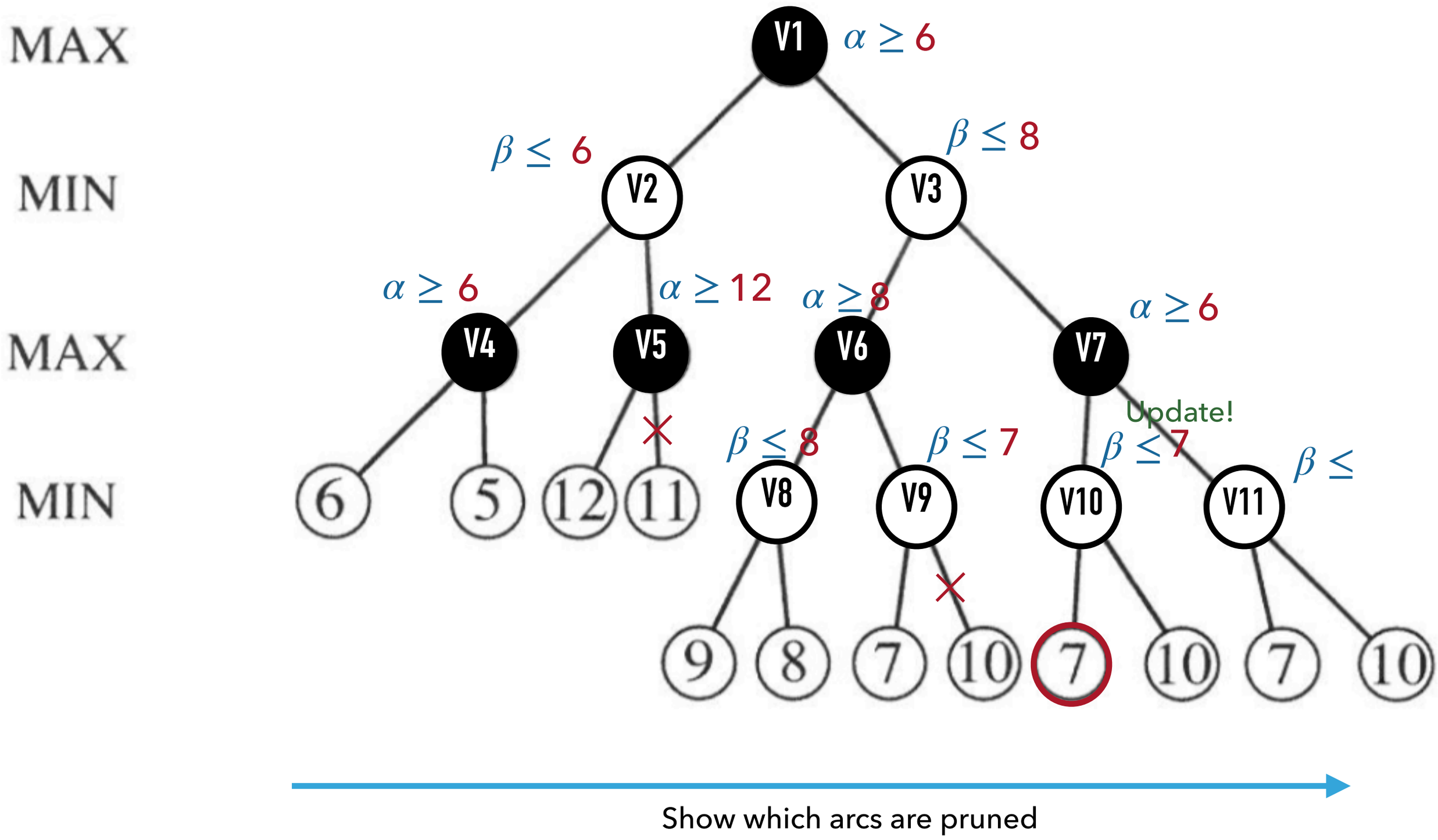
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



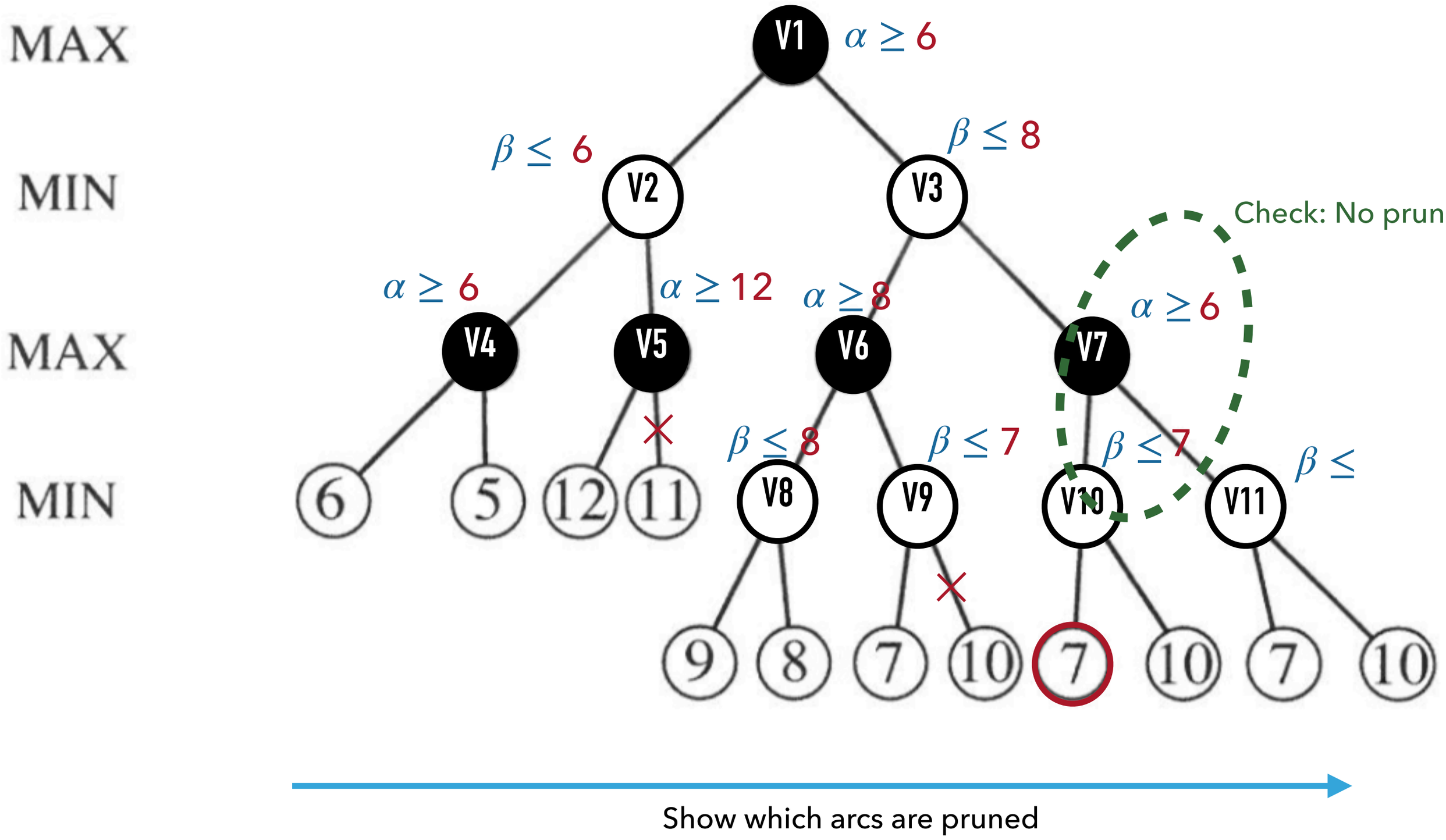
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



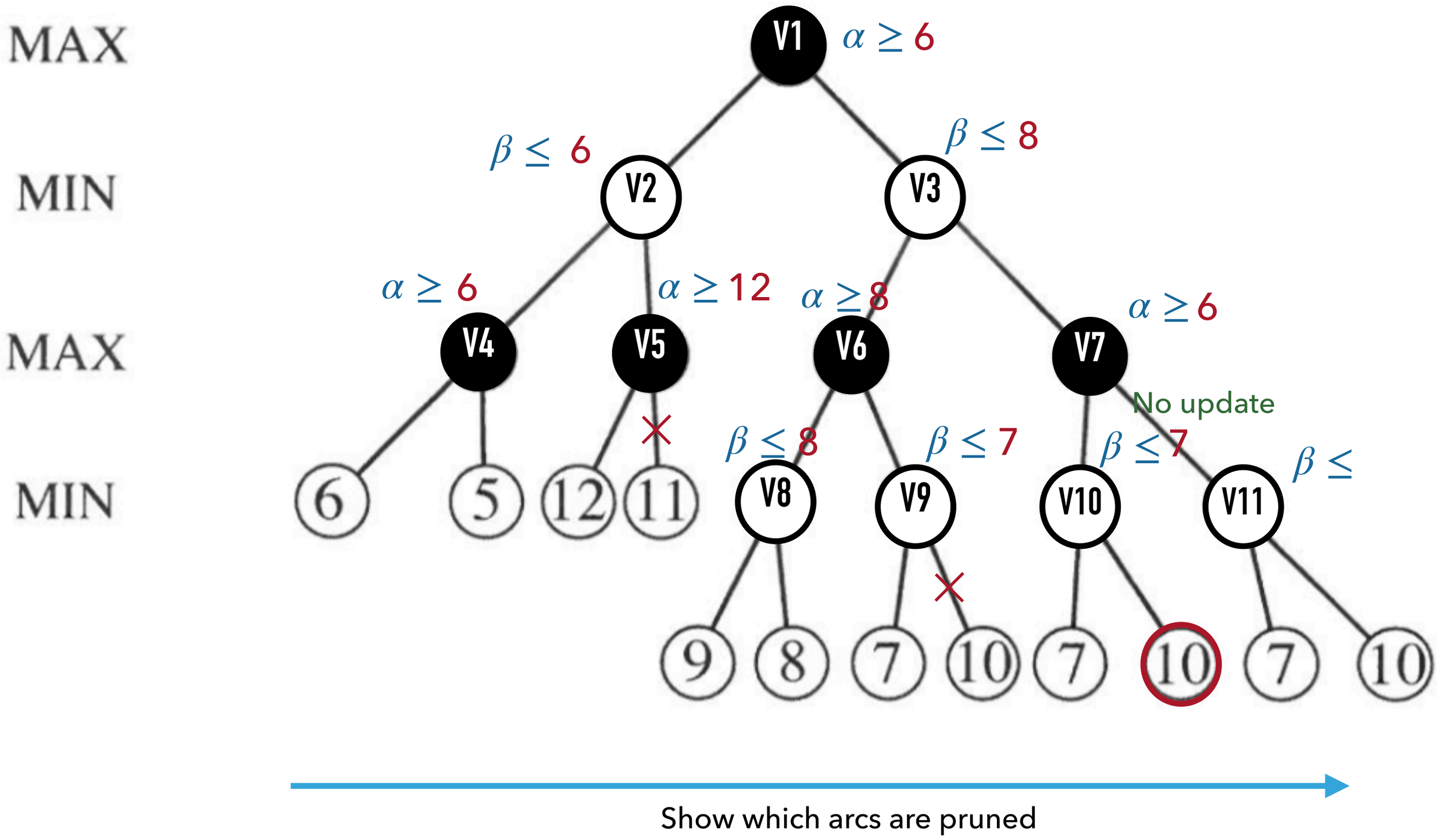
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



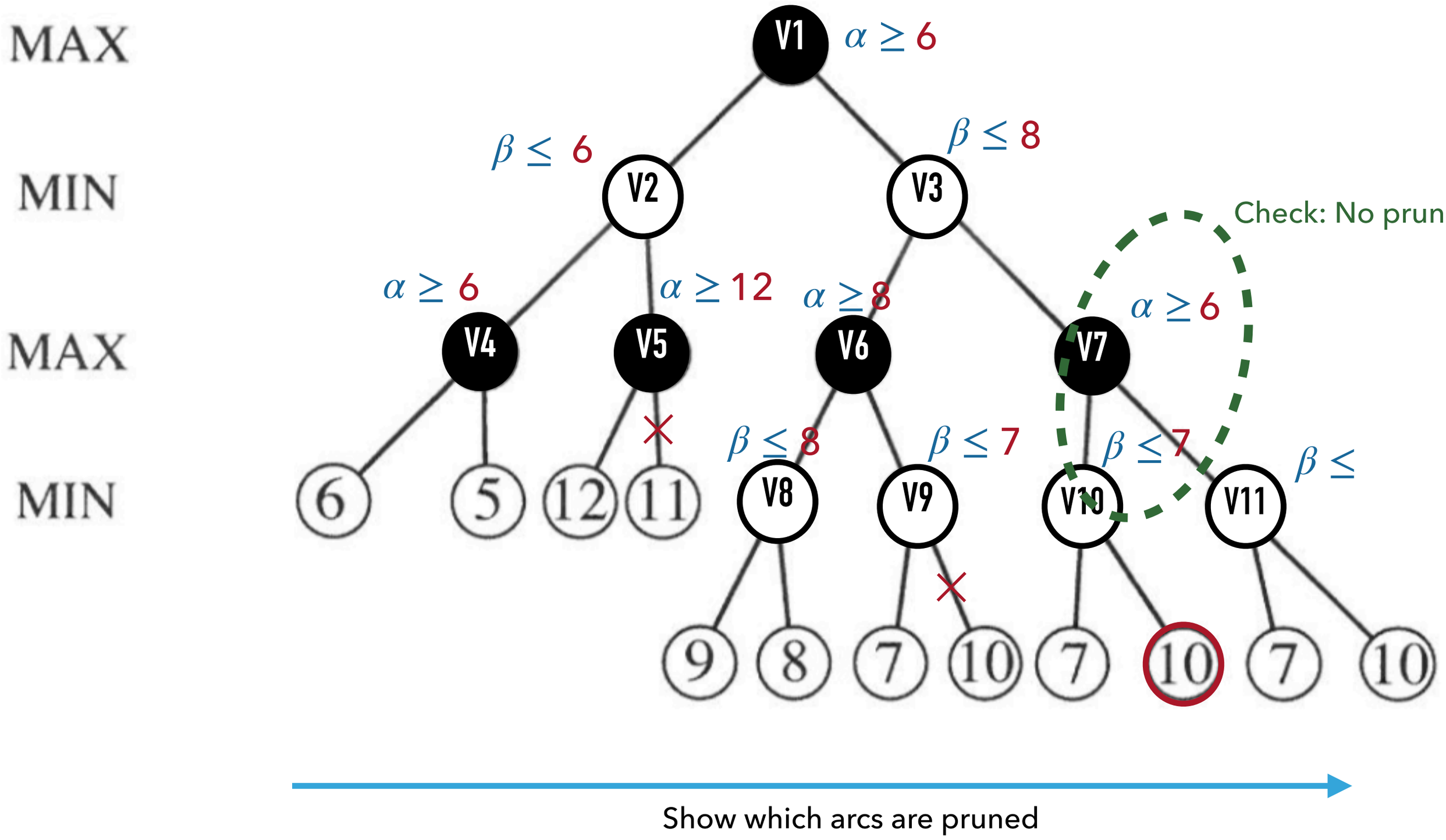
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



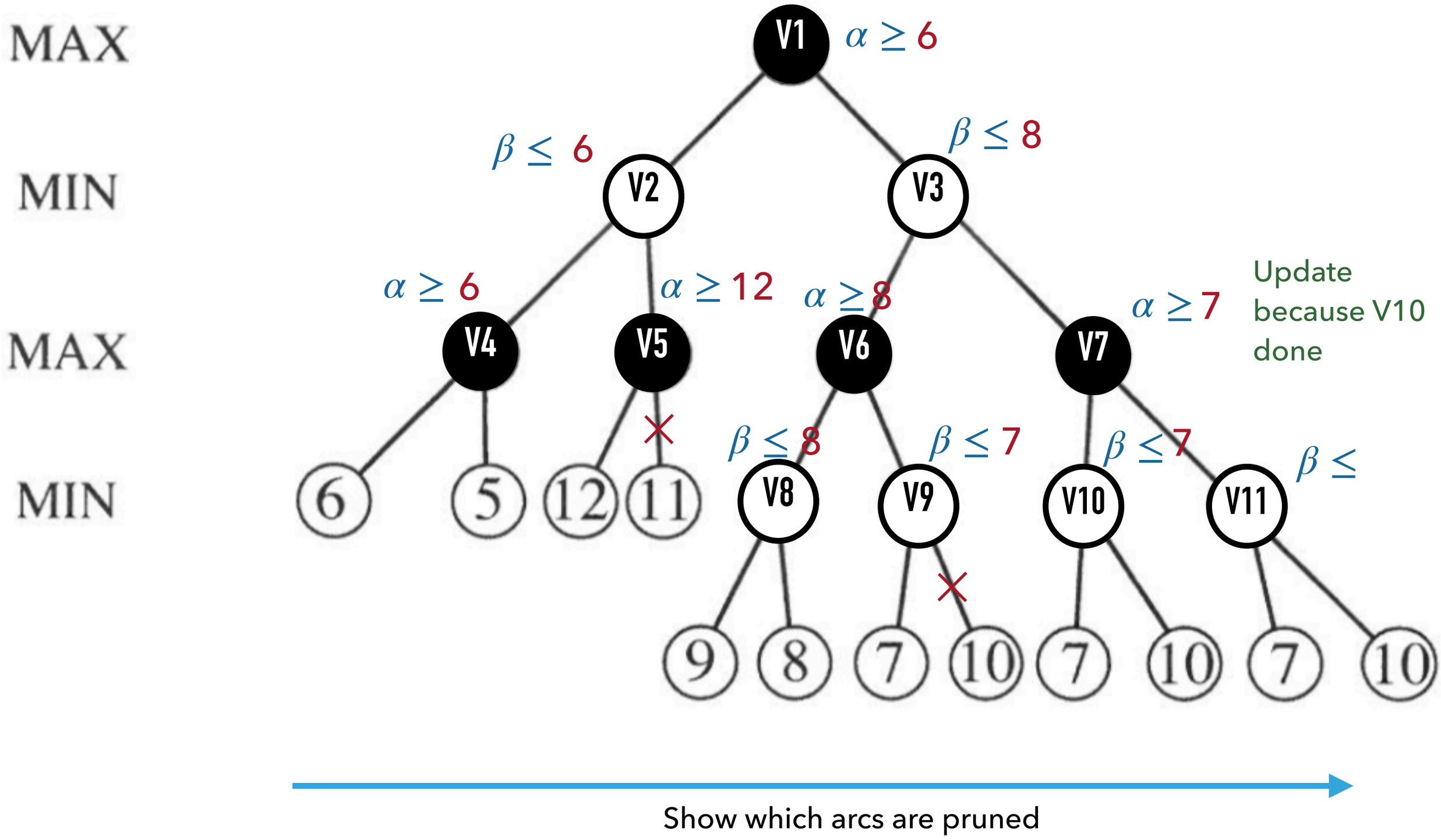
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



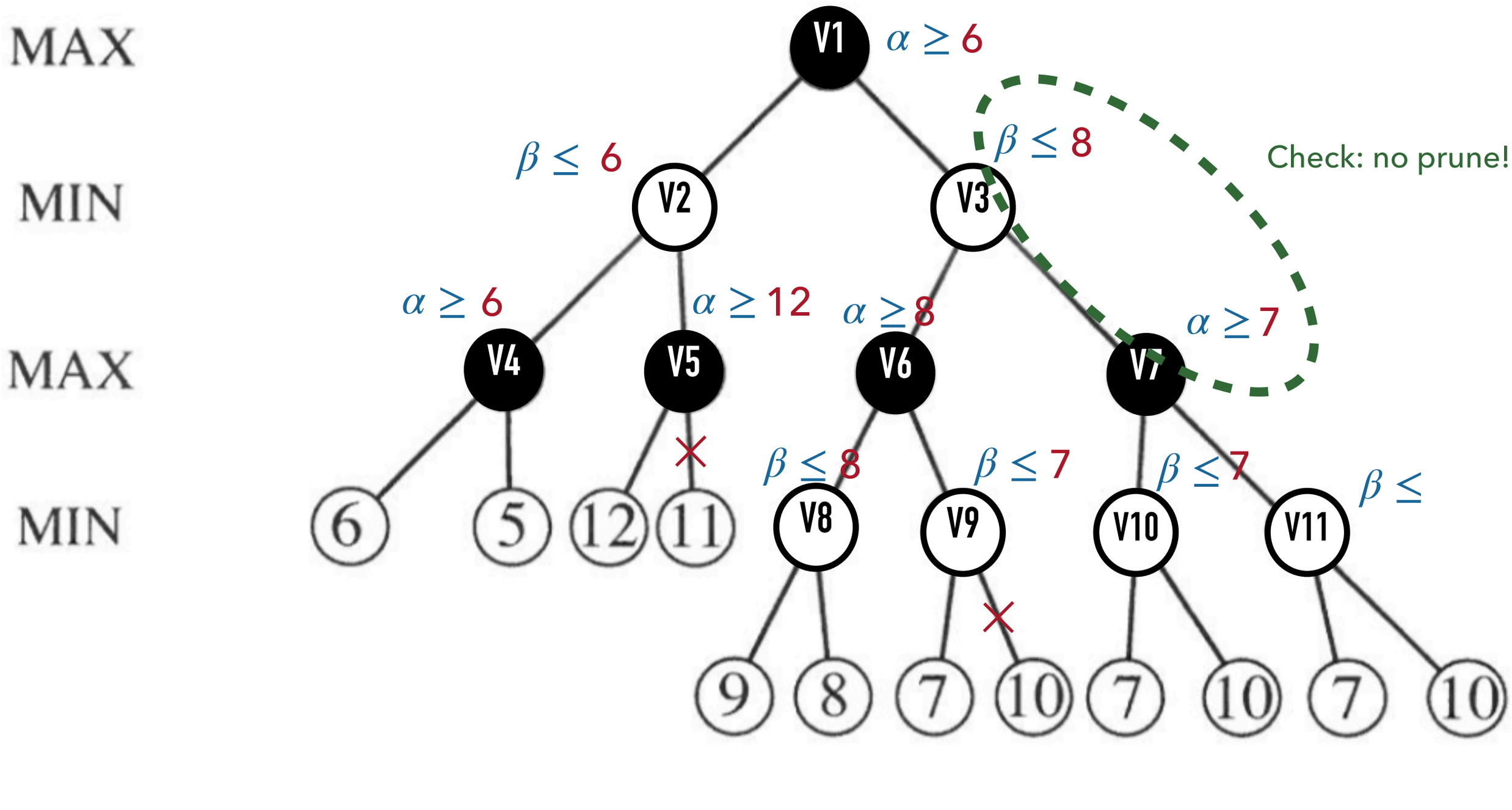
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



EXTRA QUESTION 2

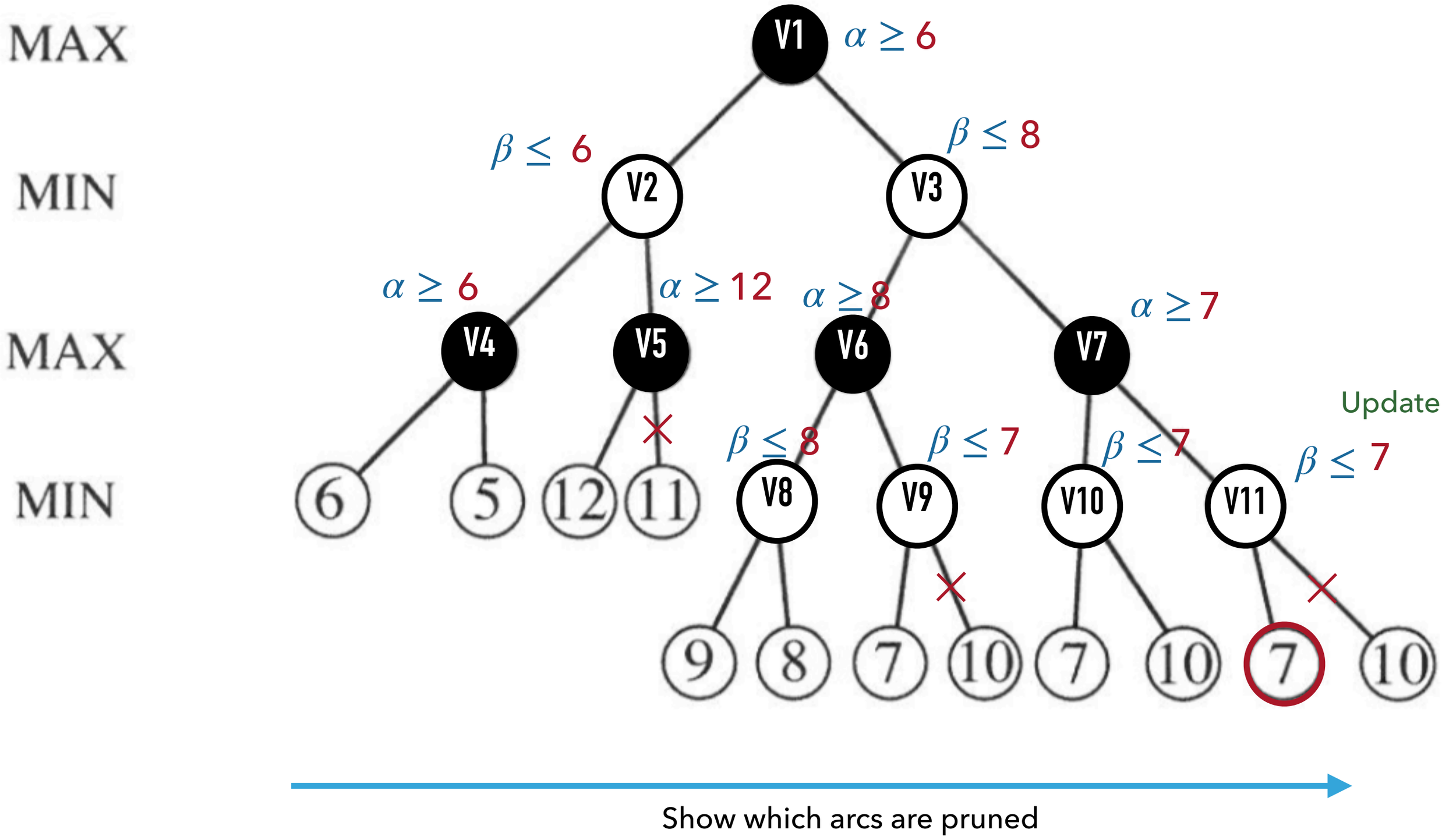
PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



Show which arcs are pruned

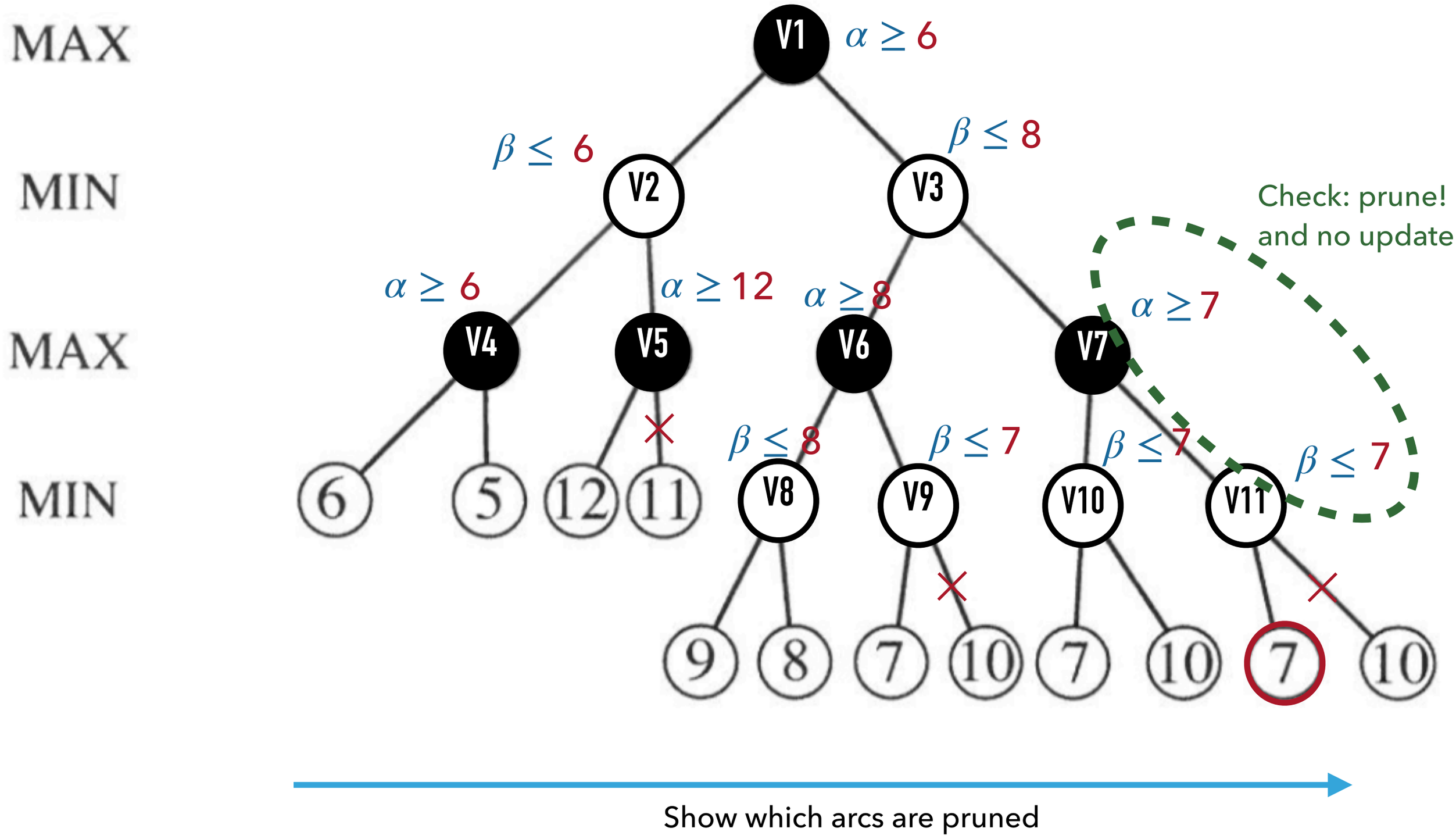
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



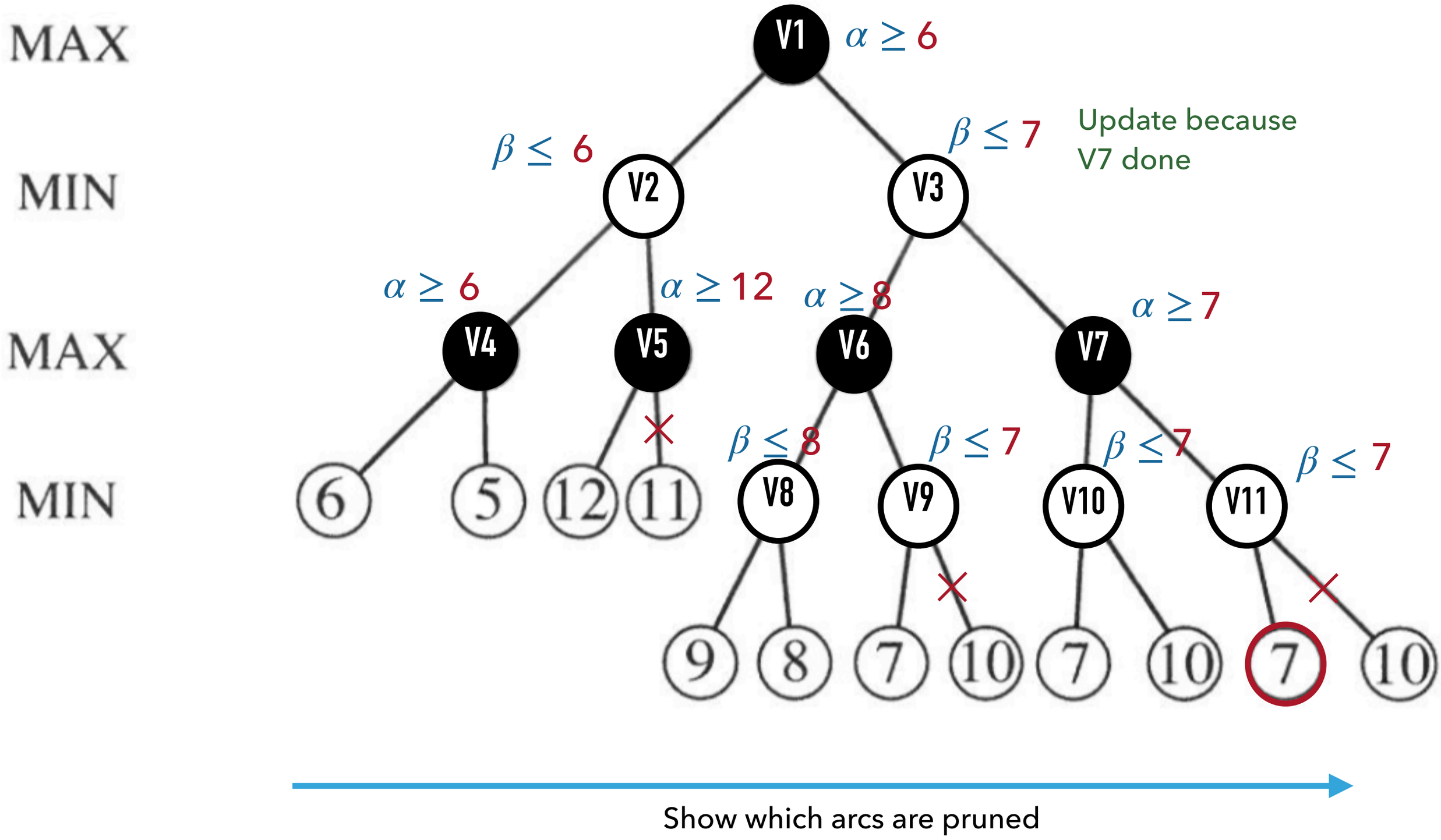
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



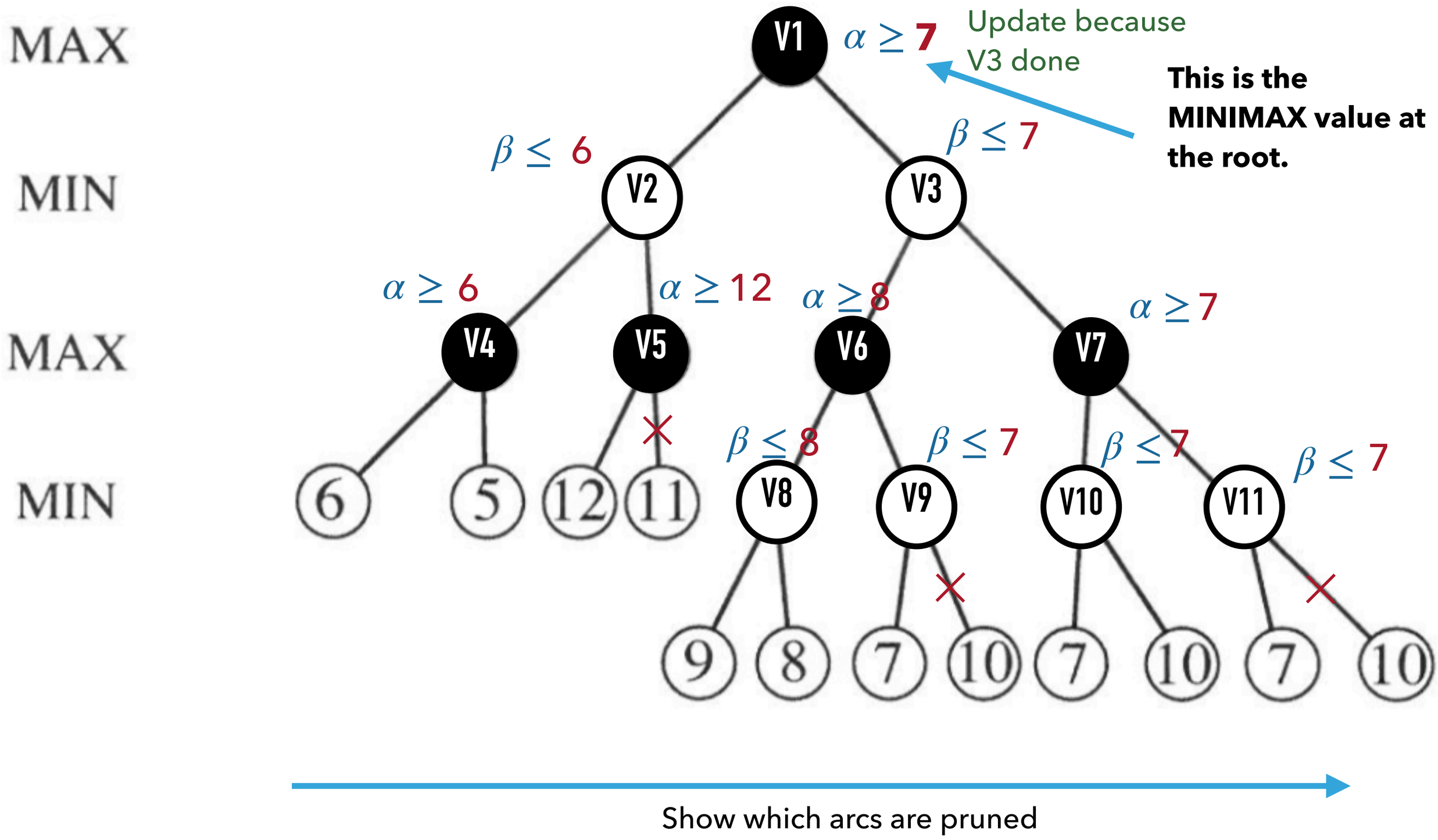
EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



EXTRA QUESTION 2

PRUNING: If [child] α ≥ β [parent], then prune child's remaining
PRUNING: If [child] β ≤ α [parent], then prune child's remaining



TUTORIAL 4 QUESTION 4

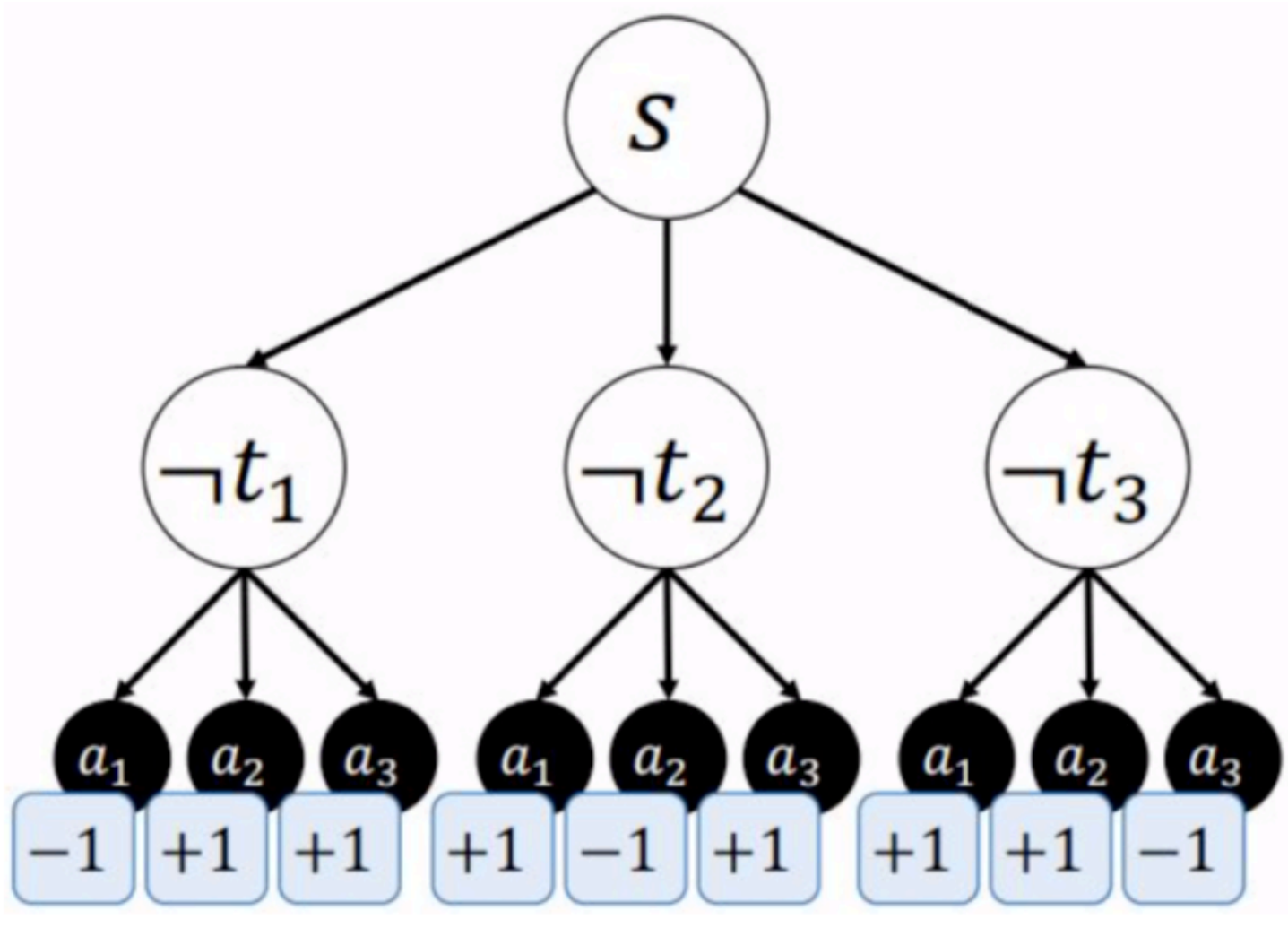
(Stackelberg Security Games)

- Consider the following game: we have an attacker looking at three targets: t_1, t_2 and t_3. A defender must choose which of the two targets it will guard; however, the attacker has an advantage: it can observe what the defender is doing before it chooses its move. If an attacker successfully attacks it receives a payoff of 1 and the defender gets a payoff of −1.

(a) Model this problem as a minimax search problem. Draw out the search tree. What is the defender's payoff in this game?

TUTORIAL 4 QUESTION 4

- **(a) Model this problem as a minimax search problem. Draw out the search tree. What is the defender's payoff in this game?**



TUTORIAL 4 QUESTION 4

- Consider the following game: we have an attacker looking at three targets: t_1, t_2 and t_3. A defender must choose which of the two targets it will guard; however, the attacker has an advantage: it can observe what the defender is doing before it chooses its move. If an attacker successfully attacks it receives a payoff of 1 and the defender gets a payoff of −1.

(b) Can the defender do better by randomizing? What is the defender's optimal strategy? Prove your claim.

TUTORIAL 4 QUESTION 4

- **(b) Can the defender do better by randomizing? What is the defender's optimal strategy? Prove your claim.**
- If the defender does not randomise, the attacker can know for sure which one the defender will defend, attack that. Defender's payoff: -1
- If defender randomises uniformly (1/3, 1/3, 1/3), defender's expected payoff is 1/3.
- If defender does any other assignment of probability that is not uniform, attacker attacks the one that defender does not defend with highest probability, and defender's expected payoff is < 1/3.
- So defender's OPTIMAL strategy is to **defend** uniformly at random.

EXTRA QUESTION 3

- **Construct an example where, should the MIN player play sub-optimally, the MINIMAX algorithm makes a sub-optimal move.**

- MINIMAX = making moves assuming other player plays optimally.

- What this questions wants is for you to come up with a simple game tree, whereby, if MIN does not play optimally (i.e. he does something irrational), then in retrospect, MAX COULD'VE gotten higher payoff by not playing MINIMAX in the first place.

EXTRA QUESTION 3

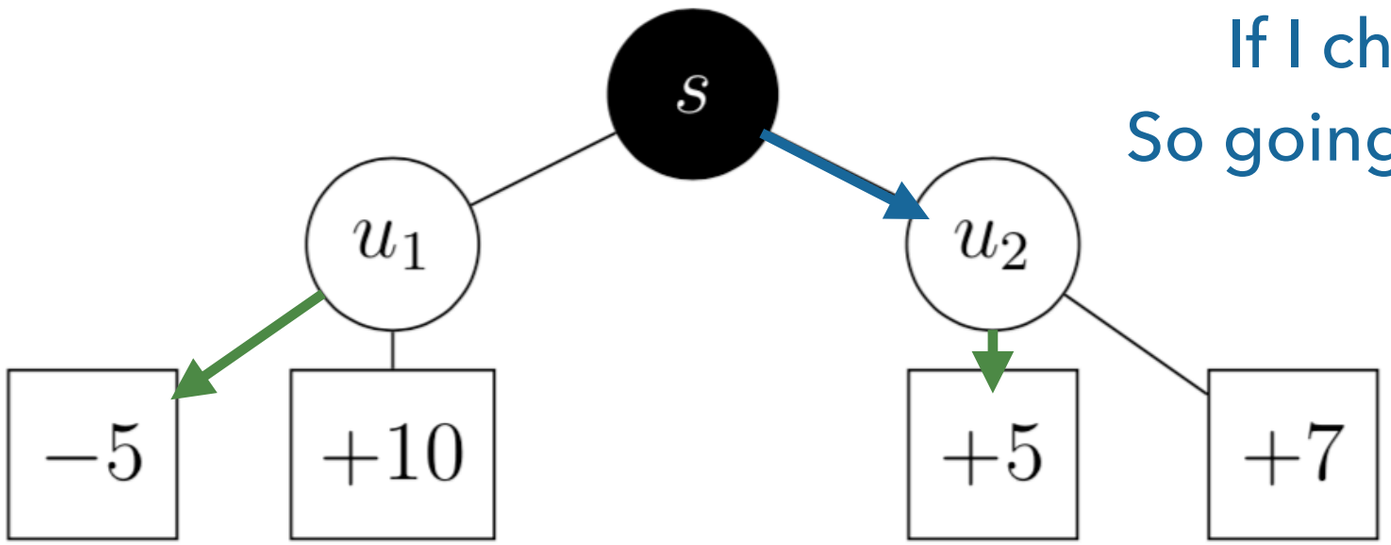
- **Construct an example where, should the MIN player play sub-optimally, the MINIMAX algorithm makes a sub-optimal move.**

MAX's POV: Assume MIN plays optimally.

i.e. If I choose u1, MIN will choose -5

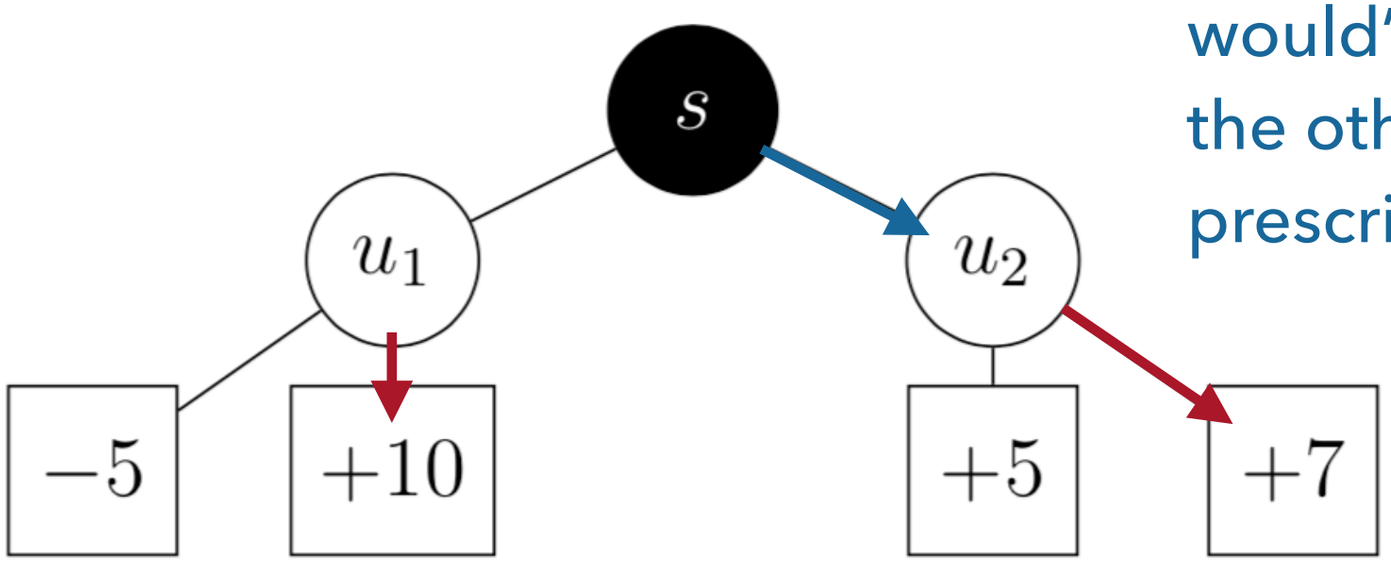
If I choose u2, MIN will choose +5

So going **u2** is better for me!



EXTRA QUESTION 3

- **Construct an example where, should the MIN player play sub-optimally, the MINIMAX algorithm makes a sub-optimal move.**



But in fact, when MIN plays **suboptimally**, MAX's payoff would've been better had he picked the other action u1 (instead of that prescribed by minimax)