

Αναζήτηση με Αντιπαλότητα

Μανόλης Κουμπάρκης

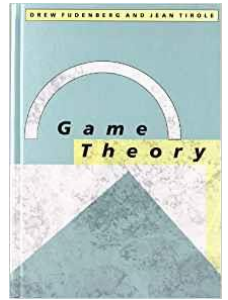
Αναζήτηση με Αντιπαλότητα

- Όταν σε ένα περιβάλλον έχουμε **περισσότερους από ένα πράκτορες**, τότε κάθε πράκτορας πρέπει να λαμβάνει υπόψη του τις ενέργειες των άλλων πρακτόρων όταν σχεδιάζει ένα πλάνο για να επιτύχει κάποιο στόχο του.



- Μπορούμε να διακρίνουμε ανάμεσα σε **συνεργατικά** και **ανταγωνιστικά** πολυπρακτορικά περιβάλλοντα.
- Τα ανταγωνιστικά περιβάλλοντα στα οποία οι στόχοι των πρακτόρων συγκρούονται, δημιουργούν τα **προβλήματα αναζήτησης με αντιπαλότητα (adversarial search problems)**, που είναι επίσης γνωστά ως **παιχνίδια (games)**.

Θεωρία Παιγνίων



- Η **θεωρία παιγνίων (game theory)** είναι ένας κλάδος των οικονομικών που αντιμετωπίζει ένα πολυπρακτορικό περιβάλλον ως **παιχνίδι**, με την προϋπόθεση ότι η επίδραση κάθε πράκτορα στους άλλους είναι σημαντική, ανεξάρτητα αν οι πράκτορες είναι συνεργατικοί ή ανταγωνιστικοί.
- Τα παιχνίδια που θα μελετήσουμε είναι ενός συγκεκριμένου τύπου: **αιτιοκρατικά, εκ περιτροπής, δύο παικτών, παιχνίδια μηδενικού αθροίσματος με τέλεια πληροφόρηση.**
- Αυτό σημαίνει ότι θα μελετήσουμε περιβάλλοντα που είναι αιτιοκρατικά και πλήρως παρατηρήσιμα, στα οποία υπάρχουν δύο πράκτορες των οποίων οι ενέργειες εναλλάσσονται, και οι **τιμές χρησιμότητας (utility values)** στο τέλος του παιχνιδιού είναι αντίθετες. Για παράδειγμα, αν ο ένας παίκτης νικήσει σε μια παρτίδα σκάκι, ο άλλος παίκτης αναγκαστικά χάνει.
- Αυτή ακριβώς η αντίθεση των τιμών χρησιμότητας είναι που κάνει αυτά τα προβλήματα να είναι προβλήματα αντιπαλότητας.



Ορισμοί

- Θα θεωρήσουμε παιχνίδια με **δύο παίκτες** που θα τους ονομάσουμε **MAX** και **MIN** για λόγους που θα γίνουν σύντομα κατανοητοί.
- **Ο MAX παίζει πρώτος** και μετά οι κινήσεις των παικτών εναλλάσσονται μέχρι το τέλος του παιχνιδιού.



Ορισμοί

- Ένα **παιχνίδι** ορίζεται τυπικά σαν ένα πρόβλημα αναζήτησης με τις παρακάτω συνιστώσες:
 - S_0 : Η **αρχική κατάσταση** η οποία καθορίζει πως ξεκινάει το παιχνίδι.
 - $\text{PLAYER}(s)$: Ορίζει ποιος παίκτης παίζει σε κάθε κατάσταση.
 - $\text{ACTIONS}(s)$: Επιστρέφει το σύνολο των νόμιμων κινήσεων για κάθε κατάσταση.
 - $\text{RESULT}(s,a)$: Ορίζει την κατάσταση που είναι το αποτέλεσμα κάθε κίνησης.
 - $\text{TERMINAL-TEST}(s)$: Ό έλεγχος τερματισμού που είναι αληθής για τις καταστάσεις στις οποίες το παιχνίδι τελειώνει (**τερματικές καταστάσεις**) αλλιώς είναι ψευδής.
 - $\text{UTILITY}(s,p)$: Η **συνάρτηση χρησιμότητας (utility function)** η οποία δίνει μια αριθμητική τιμή για κάθε παιχνίδι που τελειώνει στην κατάσταση s για τον παίκτη p .

Συναρτήσεις Χρησιμότητας

- Στο σκάκι, το αποτέλεσμα είναι νίκη, ήττα ή ισοπαλία με τιμές της συνάρτησης χρησιμότητας $+1$, 0 και $\frac{1}{2}$.



- Στο τάβλι, η συνάρτηση χρησιμότητας κυμαίνεται από 0 έως $+192$.

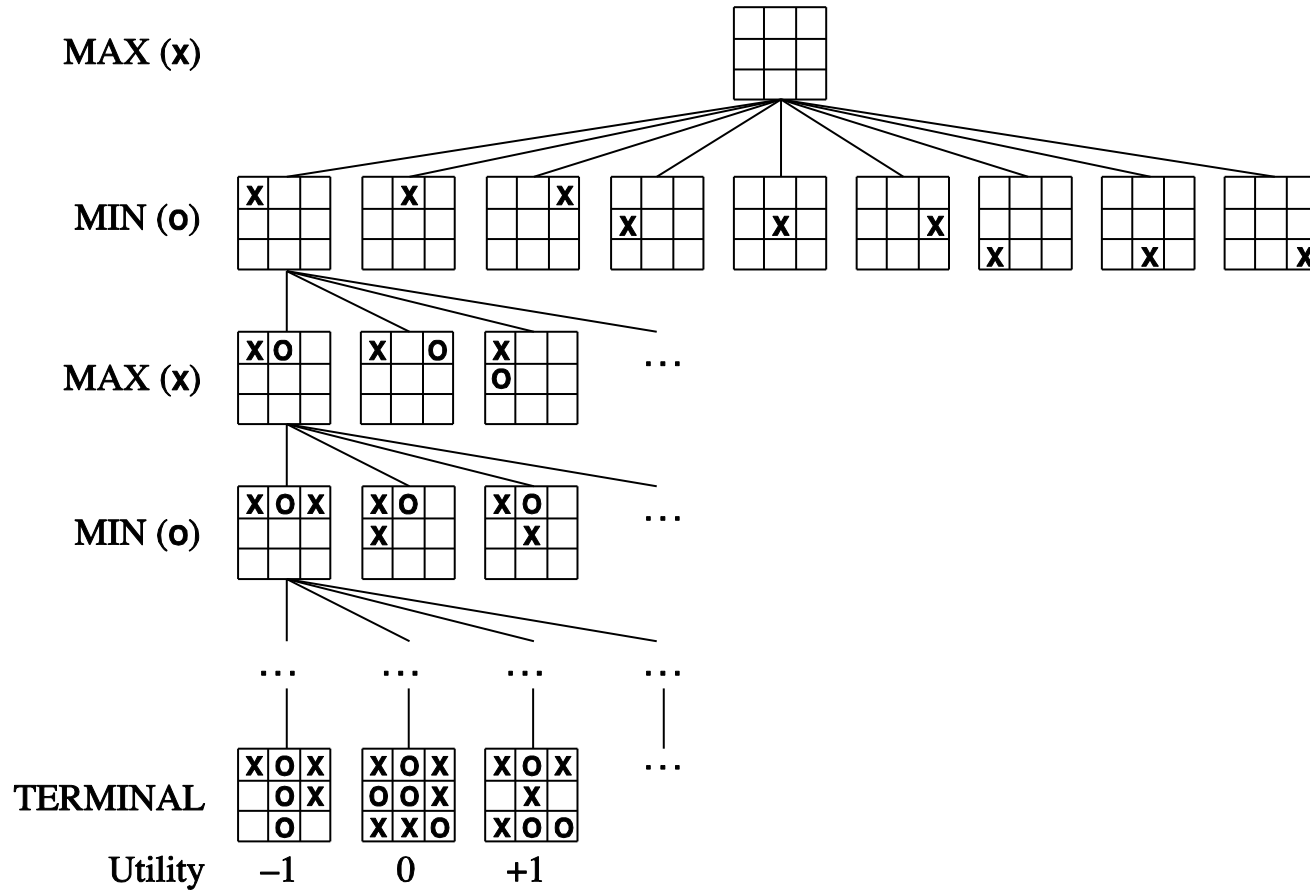


- Ένα **παιχνίδι μηδενικού αθροίσματος (zero-sum game)** είναι τέτοιο ώστε το άθροισμα των συναρτήσεων χρησιμότητας για όλους του παίκτες είναι σταθερό (ίσως ένα καλύτερο όνομα θα ήταν «σταθερού αθροίσματος»).
- Για παράδειγμα, στο σκάκι το άθροισμα είναι $0 + 1$, $1 + 0$ ή $\frac{1}{2} + \frac{1}{2}$.

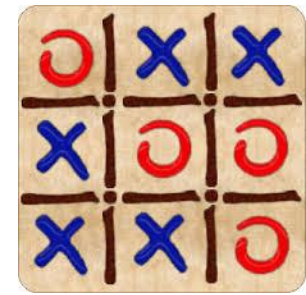
Δένδρο Παιχνιδιού

- Η αρχική κατάσταση, η συνάρτηση ACTIONS και η συνάρτηση RESULT για ένα παιχνίδι ορίζουν το **δένδρο παιχνιδιού (game tree)**.
- Στο δένδρο παιχνιδιού οι κόμβοι είναι οι καταστάσεις και οι ακμές είναι οι κινήσεις των παικτών.

Το Δένδρο Παιχνιδιού για την Τρίλιζα



Παρατηρήσεις



- Η τρίλιζα προχωράει με τον παίκτη MAX να βάζει ένα X σε ένα τετράγωνο και τον MIN να βάζει ένα O, μέχρι να φτάσουμε στους κόμβους φύλλα που αντιστοιχούν σε τερματικές καταστάσεις όπου ένας παίκτης έχει τρία σύμβολα στη σειρά ή όλα τα τετράγωνα είναι γεμάτα.
- Ο αριθμός σε κάθε φύλλο δείχνει την τιμή της συνάρτησης χρησιμότητας από την σκοπιά του MAX. Οι υψηλές τιμές θεωρούνται καλές για τον MAX και κακές για τον MIN (**έτσι παίρνουν οι παίκτες το όνομα τους**).
- Το δένδρο παιχνιδιού για την τρίλιζα είναι σχετικά μικρό: λιγότερες από $3^9 = 19683$ τερματικές καταστάσεις.
- Για το σκάκι, ο μέσος παράγοντας διακλάδωσης είναι περίπου 35 και κάθε παιχνίδι έχει συνήθως μέχρι 50 κινήσεις για κάθε παίκτη, οπότε το δένδρο παιχνιδιού έχει περίπου 35^{100} ή 10^{154} **κόμβους**. Άρα το δένδρο παιχνιδιού σε αυτή την περίπτωση είναι ένα θεωρητικό κατασκεύασμα που δεν μπορούμε να υλοποιήσουμε στον πραγματικό κόσμο.
- Θα χρησιμοποιήσουμε τον όρο **δένδρο αναζήτησης (search tree)** για ένα δένδρο το οποίο τοποθετείται πάνω από το πλήρες δένδρο παιχνιδιού και το οποίο εξετάζει αρκετούς κόμβους ώστε να επιτρέπει σε κάποιο παίκτη να αποφασίζει την επόμενη κίνηση του.

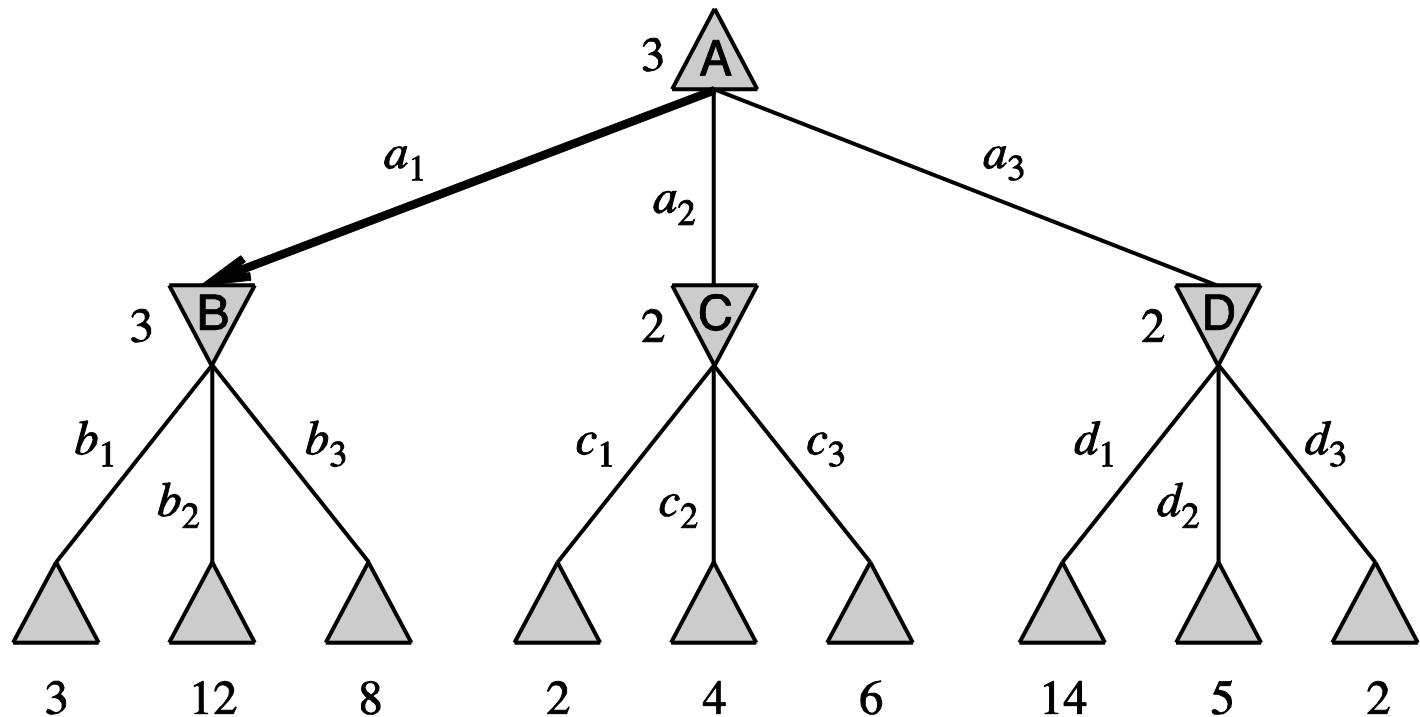
Βέλτιστες Στρατηγικές

- Σε ένα παιχνίδι, ο MAX πρέπει να βρει μια **στρατηγική** η οποία προσδιορίζει την κίνηση του στην αρχική κατάσταση, έπειτα τις κινήσεις του στις καταστάσεις που προκύπτουν από κάθε δυνατή απόκριση του MIN κ.ο.κ.
- Μια **βέλτιστη στρατηγική** μας οδηγεί σε αποτελέσματα τουλάχιστον το ίδιο καλά με οποιαδήποτε άλλη στρατηγική όταν παίζει κανείς εναντίον ενός αλάνθαστου αντιπάλου.
- Θα μελετήσουμε **πως μπορούμε να βρούμε τη βέλτιστη στρατηγική** αν και είναι ανέφικτο για τον MAX να την υπολογίσει για παιχνίδια πιο πολύπλοκα από την τρίλιζα.

Ένα Δένδρο Παιχνιδιού Δύο Στρώσεων

MAX

MIN



Παρατηρήσεις

- Το απλοϊκό αυτό παιχνίδι τελειώνει αφού οι MAX και MIN κάνουν από μία κίνηση ο καθένας.
- Στη διάλεκτο της θεωρίας παιγνίων, λέμε ότι αυτό το δένδρο έχει βάθος **μία κίνηση**, που αποτελείται από **δύο μισές κινήσεις**, κάθε μια από τις οποίες ονομάζεται **στρώση (ply)**.

Τιμές Minimax

- Η βέλτιστη κίνηση σε ένα κόμβο του δένδρου παιχνιδιού μπορεί να υπολογιστεί με την εξέταση της **τιμής minimax** του κόμβου.
- Για ένα κόμβο n , η τιμή αυτή συμβολίζεται με $\text{MINIMAX}(n)$.
- Η **τιμή minimax ενός κόμβου** είναι η **καλύτερη τιμή χρησιμότητας** που μπορεί να επιτευχθεί από τον παίκτη στον κόμβο αυτό, με την προϋπόθεση ότι και οι δύο παίκτες παίζουν βέλτιστα από εκείνο το σημείο μέχρι το τέλος του παιχνιδιού.
- Η τιμή minimax για τις **τερματικές καταστάσεις** είναι ίση με την χρησιμότητα τους και είναι **γνωστή** κατά τη διάρκεια του παιχνιδιού.
- Ο MAX προτιμάει να μετακινείται σε καταστάσεις **μέγιστης τιμής** ενώ ο MIN σε καταστάσεις **ελάχιστης τιμής** (**έτσι παίρνουν οι παίκτες το όνομα τους**).

Τιμές Minimax

- Έτσι έχουμε την ακόλουθη ισότητα:

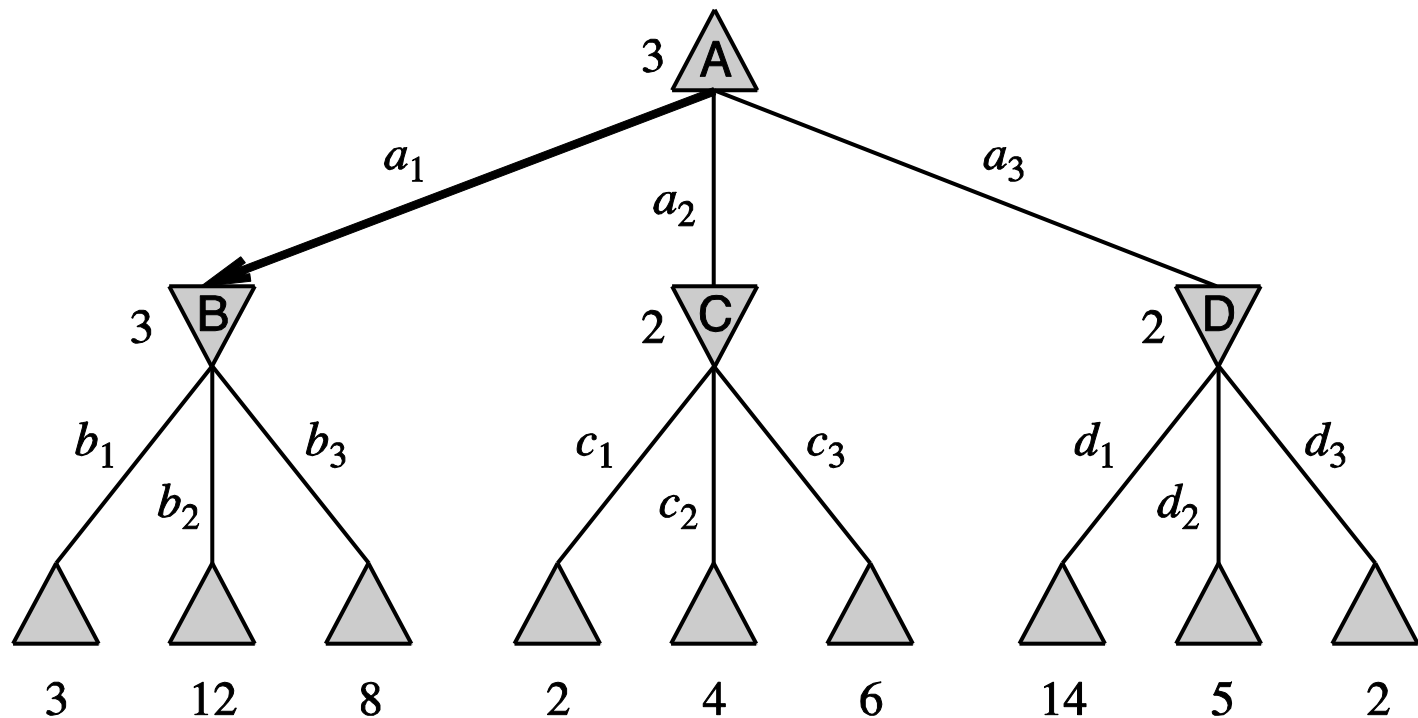
$$\begin{aligned} \text{MINIMAX}(s) &= \\ &= \begin{cases} \text{UTILITY}(s) & \text{if } \text{TERMINAL-TEST}(s) \\ \max_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if } \text{PLAYER}(s) = \text{MAX} \\ \min_{a \in \text{ACTIONS}(s)} \text{MINIMAX}(\text{RESULT}(s, a)) & \text{if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

- Μπορούμε να εφαρμόσουμε την παραπάνω ισότητα στο προηγούμενο δένδρο παιχνιδιού και να υπολογίσουμε τις minimax τιμές για κάθε κατάσταση.

Παράδειγμα

MAX

MIN



Minimax Απόφαση

- Μπορούμε να υπολογίσουμε την **minimax απόφαση** σε κάθε κόμβο του δένδρου.
- Για παράδειγμα, η minimax απόφαση στη ρίζα του προηγούμενου δένδρου είναι η ενέργεια a_1 για τον MAX επειδή οδηγεί στην κατάσταση με την υψηλότερη minimax τιμή.

Γιατί η Minimax Απόφαση είναι η Σωστή για τον MAX;

- Οι τιμές στα φύλλα του δένδρου μας δίνουν τη **χρησιμότητα** της αντίστοιχης κατάστασης για κάθε παίκτη.
- Υποθέτουμε ότι οι χρησιμότητες ορίζονται με τέτοιο τρόπο ώστε τον MAX τον συμφέρουν οι μεγάλες τιμές και τον MIN οι μικρές. Γιατί; Έτσι μοντελοποιούμε ένα παιχνίδι **μηδενικού αθροίσματος** – όταν ο ένας παίκτης κερδίζει, ο άλλος χάνει!
- Για να αποφασίσει πως θα παίξει στον κόμβο A, ο MAX μπορεί να διασχίσει όλο το δένδρο παιχνιδιού.
- Οι κόμβοι του δένδρου που αντιστοιχούν στον MAX είναι υπό τον έλεγχο του, ενώ οι υπόλοιποι είναι υπό τον έλεγχο του MIN.
- Έτσι, για το προηγούμενο δένδρο, ο MAX βλέπει ότι στον κόμβο B, ο MIN (που ο MAX υποθέτει ότι παίζει **βέλτιστα**!) θα επιλέξει το κλαδί που οδηγεί σε χρησιμότητα 3, στον κόμβο C το κλαδί που οδηγεί σε χρησιμότητα 2 και στον κόμβο D το κλαδί που οδηγεί σε χρησιμότητα 2.
- Άρα, ο καλύτερος τρόπος για να παίξει ο MAX στον κόμβο A είναι να επιλέξει το κλαδί με χρησιμότητα 3, δηλαδή να κάνει την κίνηση a_1 .

Παρατήρηση

- Θα ήταν ίσως καλύτερα να μην έχουμε τριγωνάκι στο τελευταίο επίπεδο ενός δένδρου παιχνιδιού (στα φύλλα).
- Σ' αυτό το επίπεδο απλά δίνουμε τις χρησιμότητες των αντιστοίχων καταστάσεων - δεν παίζει κάποιος από τους παίκτες.
- Θα μπορούσαμε να χρησιμοποιήσουμε ένα άλλο σύμβολο (π.χ., τετραγωνάκι) ή να μην έχουμε σύμβολο σε κείνο το επίπεδο.

Ο Αλγόριθμος Minimax

function MINIMAX-DECISION(*state*) **returns** an action
 return $\arg \max_{a \in \text{ACTIONS}(state)} \text{MIN-VALUE}(\text{RESULT}(state, a))$

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**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(state, a)))$

return *v*

Ο Αλγόριθμος Minimax

```
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
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(\text{state}, a)))$ 
  return  $v$ 
```

Σχόλια

- Ο αλγόριθμος MINIMAX-DECISION υπολογίζει **την minimax απόφαση για τον παίκτη MAX.**
- Υλοποιεί ένα απλό αναδρομικό υπολογισμό των τιμών minimax για κάθε διάδοχη κατάσταση χρησιμοποιώντας τις εξισώσεις που τις ορίζουν.
- Η αναδρομή κατεβαίνει όλη τη διαδρομή μέχρι τα φύλλα του δένδρου παιχνιδιού και μετά οι τιμές minimax **αντιγράφονται προς τα πίσω** μέσω του δένδρου όπως εκτυλίσσεται ή αναδρομή.

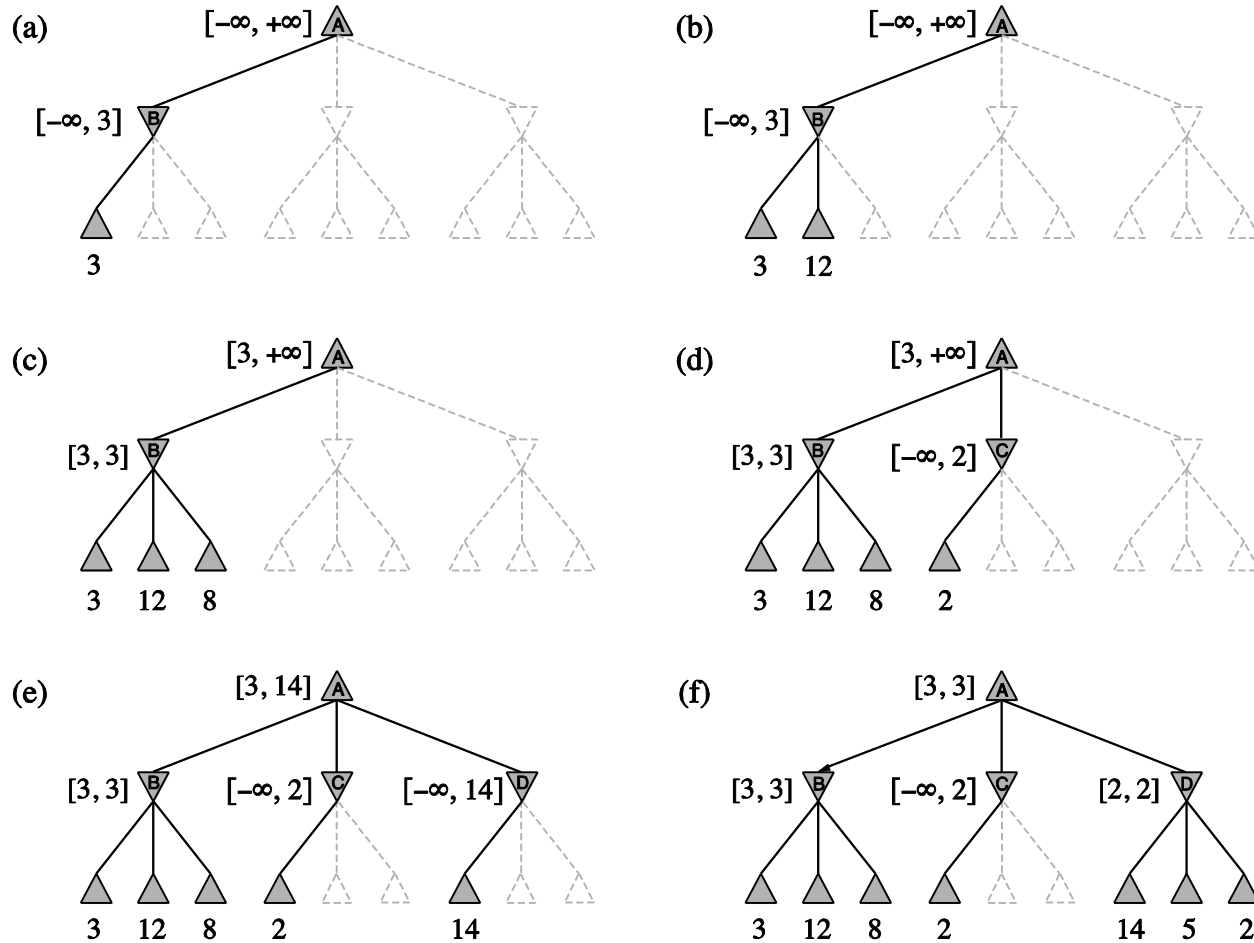
Υπολογιστική Πολυπλοκότητα

- Ο αλγόριθμος minimax υλοποιεί μια πλήρη **αναζήτηση πρώτα κατά βάθος** στο δένδρο παιχνιδιού.
- Αν το βάθος του δένδρου είναι m και υπάρχουν το πολύ b νόμιμες κινήσεις σε κάθε σημείο, τότε η **πολυπλοκότητα χρόνου** του αλγόριθμου minimax είναι $O(b^m)$.
- Η **πολυπλοκότητα χώρου** είναι $O(bm)$ αν ο αλγόριθμος παράγει όλες τις διάδοχες καταστάσεις μαζί ή $O(m)$ αν οι διάδοχες καταστάσεις παράγονται μία-μία.
- Για πραγματικά παιχνίδια αυτή η υπολογιστική πολυπλοκότητα χρόνου είναι απαράδεκτη, όμως ο αλγόριθμος minimax είναι χρήσιμος για τη μαθηματική ανάλυση παιχνιδιών και την ανάπτυξη αποδοτικότερων αλγορίθμων.

Κλάδεμα Άλφα-Βήτα

- Το πρόβλημα με τον αλγόριθμο minimax είναι ότι ο αριθμός των καταστάσεων που πρέπει να εξετάσουμε είναι εκθετικός ως προς το βάθος του δένδρου παιχνιδιού.
- Δεν μπορούμε να αποφύγουμε τον εκθέτη αυτό αλλά **μπορούμε να τον μειώσουμε στο μισό** με το να μην εξετάζουμε όλους τους κόμβους του δένδρου.
- Ο αλγόριθμος που θα χρησιμοποιήσουμε λέγεται **κλάδεμα άλφα-βήτα (alpha-beta pruning)**.

Παράδειγμα



Σχόλια

- Το πρώτο φύλλο κάτω από τον κόμβο B έχει τιμή 3. Επομένως ο B, που είναι κόμβος MIN, έχει τιμή **το πολύ 3**.
- Εξετάζοντας το δεύτερο και το τρίτο φύλλο κάτω από τον B, βλέπουμε ότι αυτός έχει τιμή **ακριβώς 3**.
- Τώρα μπορούμε να συμπεράνουμε ότι η τιμή της ρίζας είναι **τουλάχιστον 3**, επειδή η ρίζα είναι κόμβος MAX.

Σχόλια

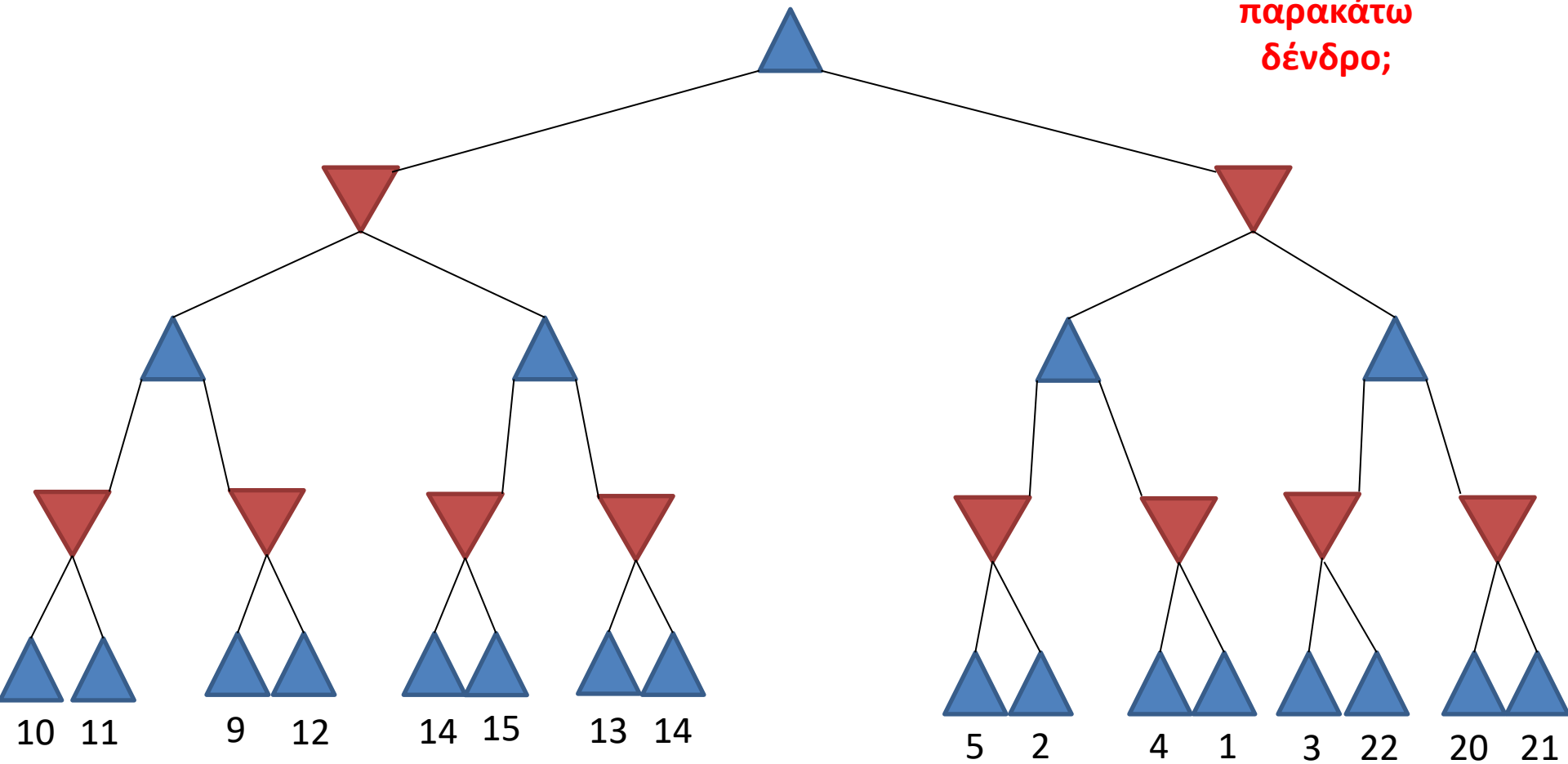
- Το πρώτο φύλλο κάτω από τον κόμβο C έχει τιμή 2. Επομένως ο C, που είναι κόμβος MIN, έχει τιμή **το πολύ 2**.
- Όμως ο B έχει τιμή 3 άρα ο MAX στη ρίζα δεν θα διάλεγε ποτέ τον C. Επομένως, δεν έχει νόημα να εξετάσουμε τα άλλα παιδιά του C. Αυτό είναι ένα **παράδειγμα κλαδέματος άλφα-βήτα**.

Σχόλια

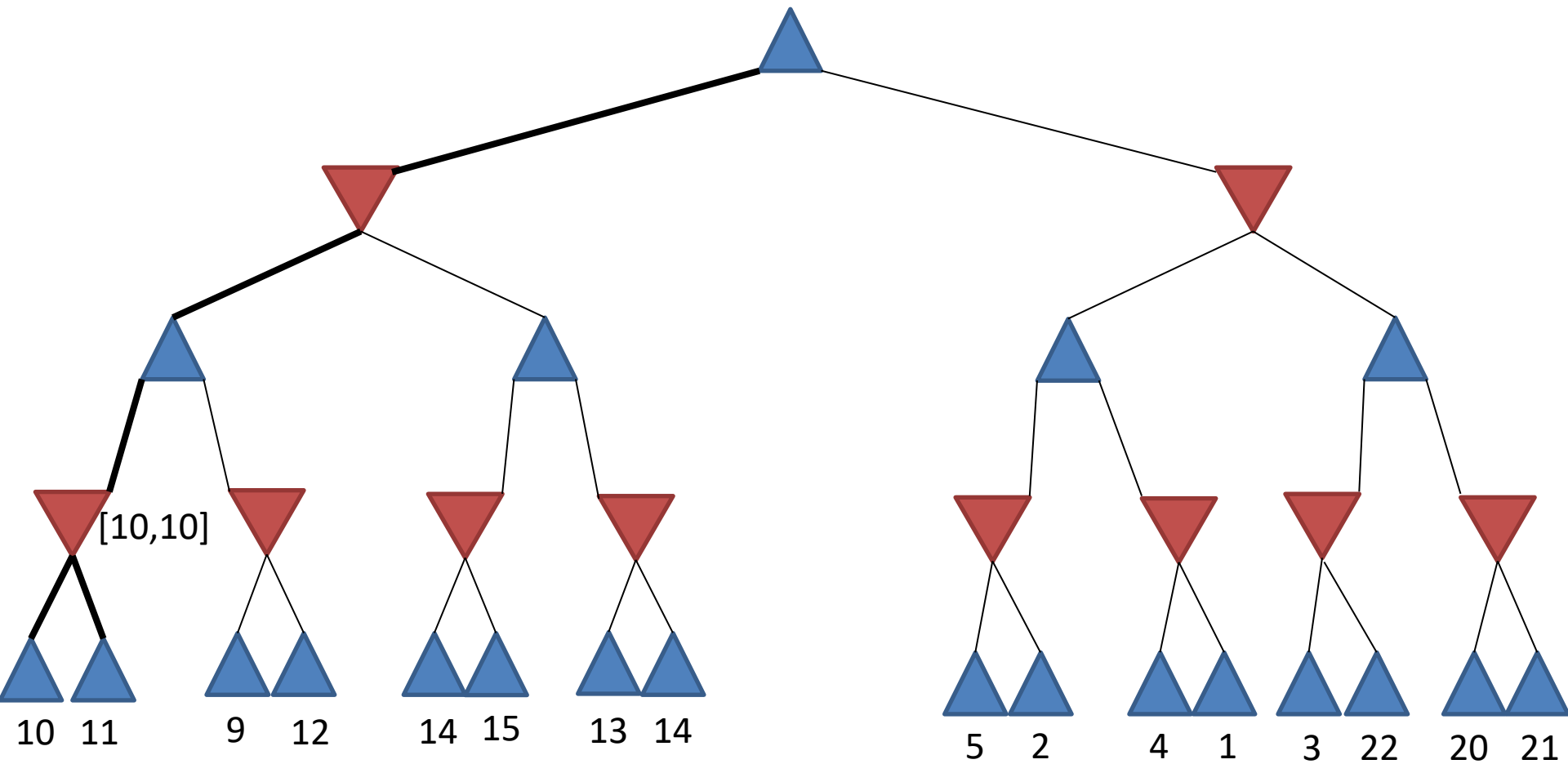
- Το πρώτο φύλλο κάτω από τον κόμβο D έχει την τιμή 14, γι αυτό η τιμή του D είναι το πολύ 14.
- Η τιμή αυτή είναι μεγαλύτερη από την καλύτερη εναλλακτική επιλογή του MAX (δηλαδή την τιμή 3) και άρα πρέπει να συνεχίζουμε να εξετάζουμε τα φύλλα κάτω από τον D.
- Εξετάζοντας τα άλλα δύο φύλλα βρίσκουμε ότι η τιμή του D είναι 2 και άρα η τιμή της ρίζας είναι 3 (ο MAX επιλέγει τον B).

Παράδειγμα

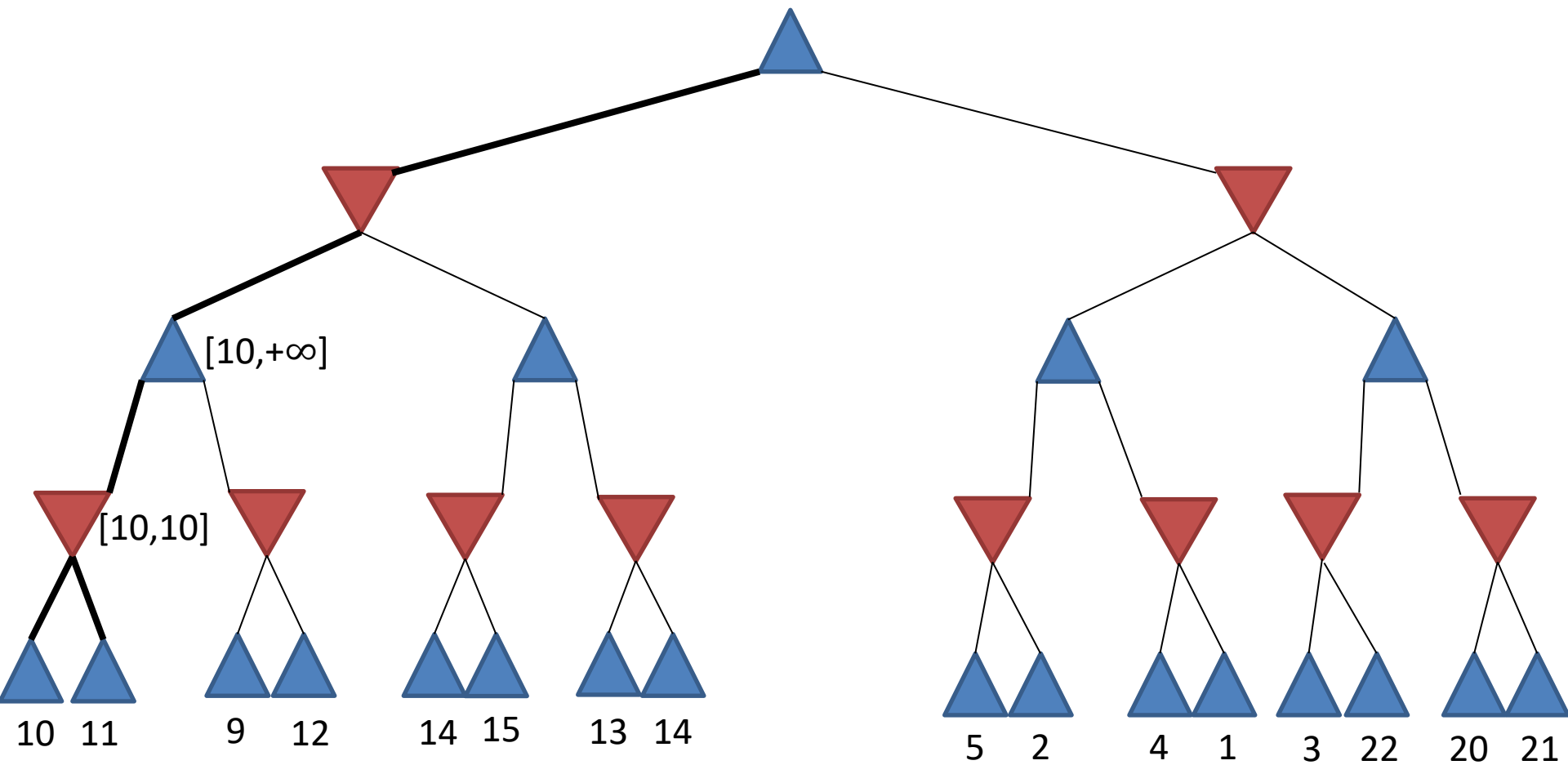
Ποιο είναι το
αποτέλεσμα του
κλαδέματος άλφα-
βήτα στο
παρακάτω
δένδρο;



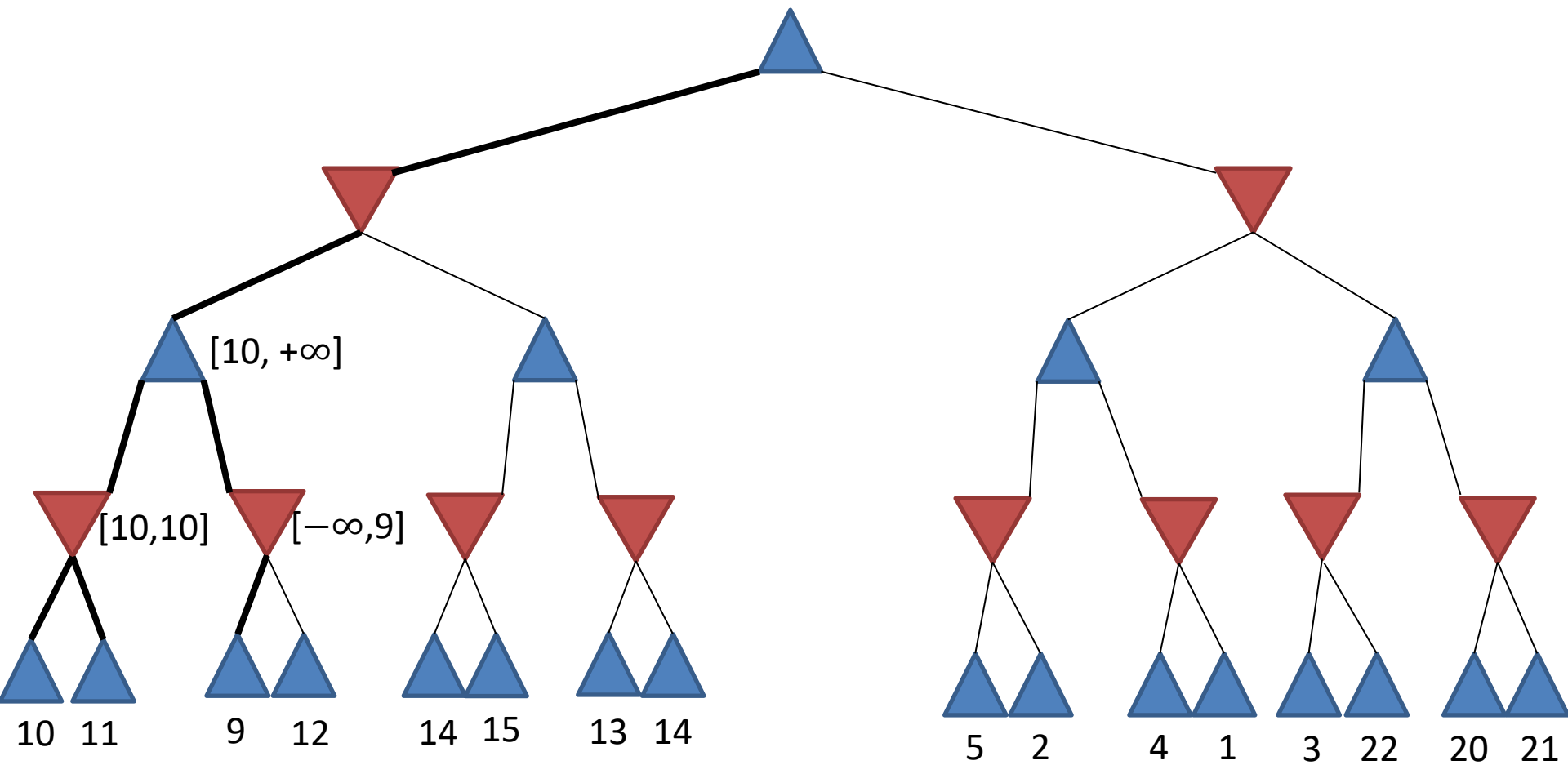
Παράδειγμα



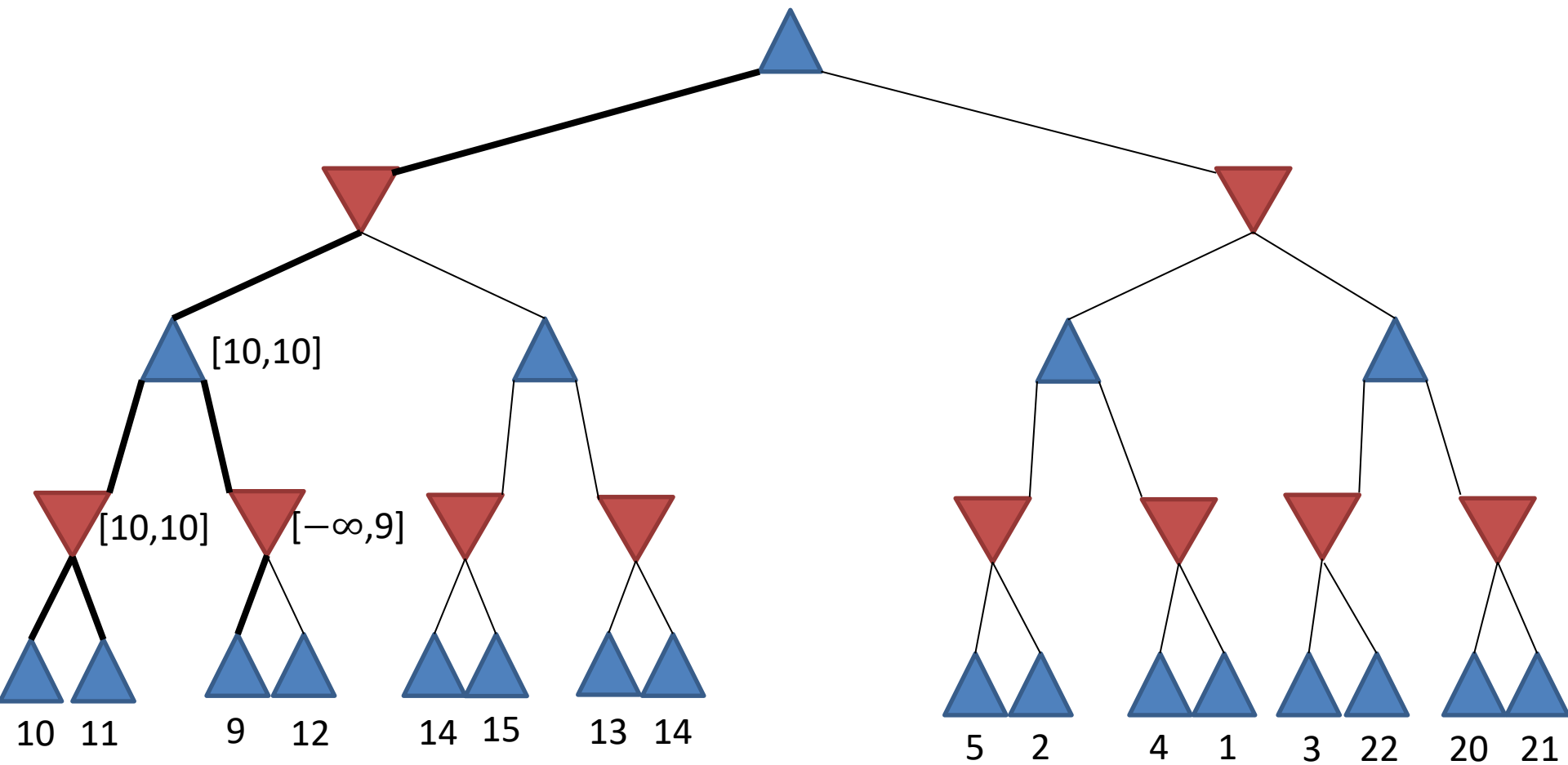
Παράδειγμα



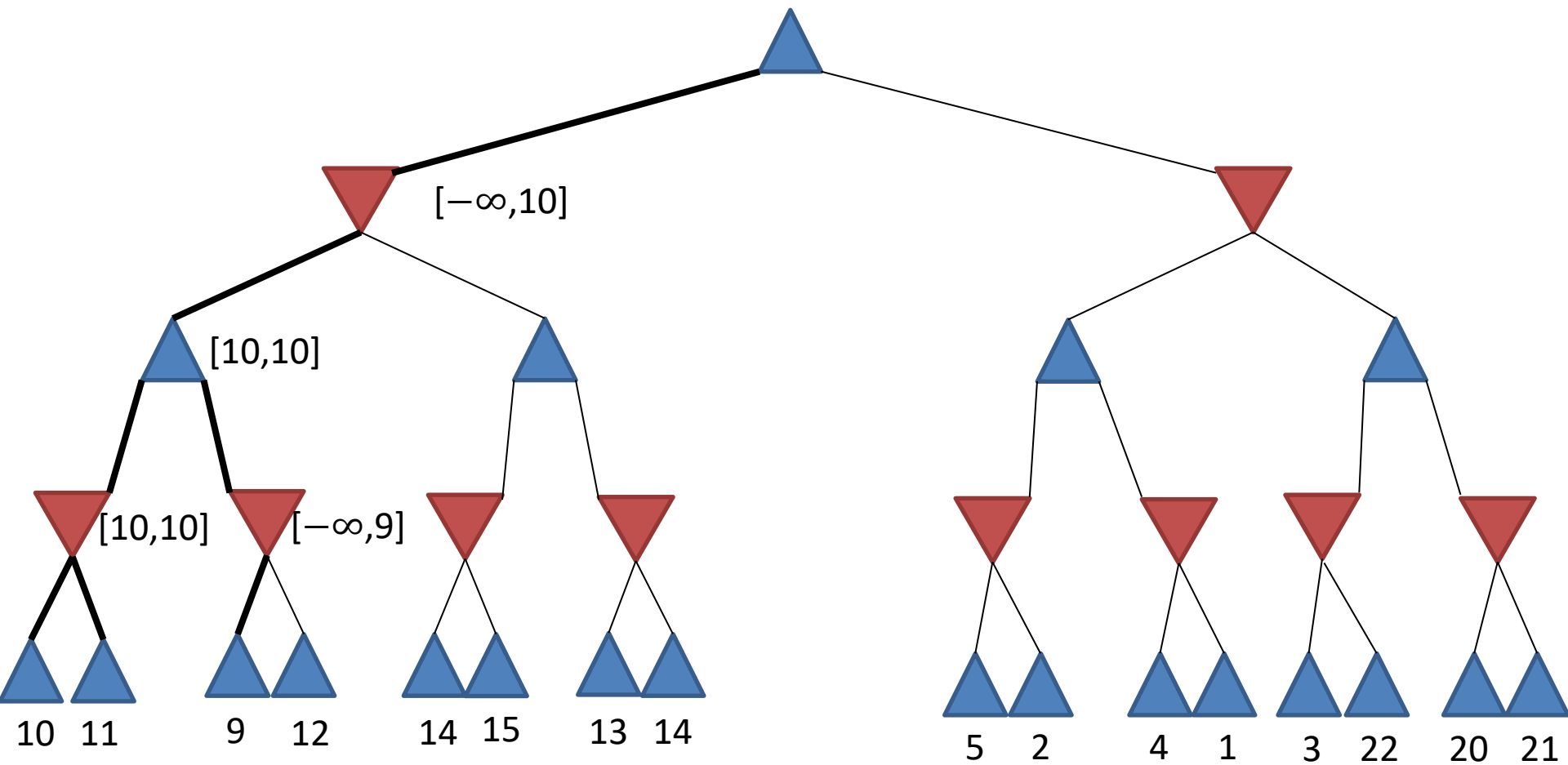
Παράδειγμα



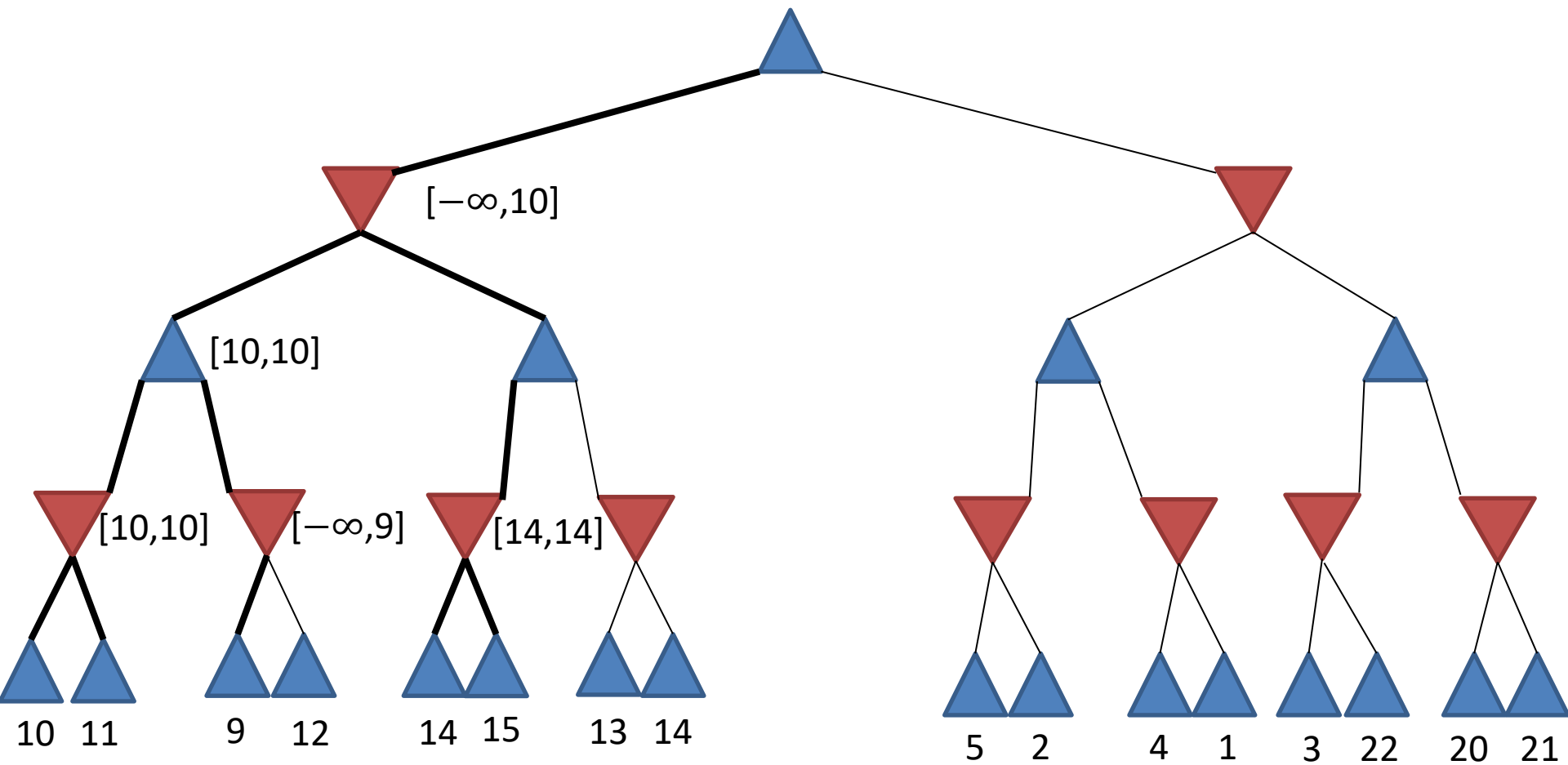
Παράδειγμα



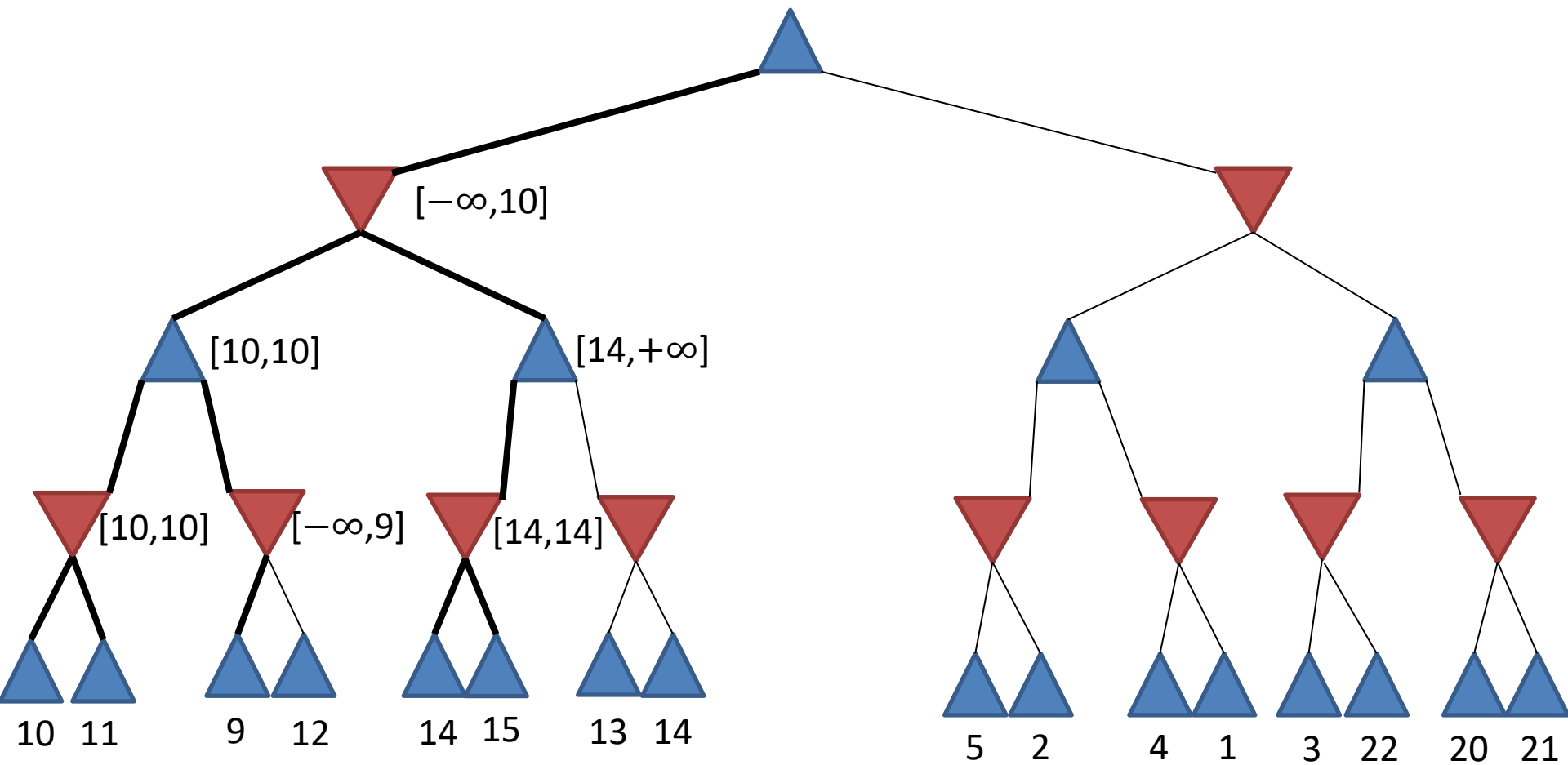
Παράδειγμα



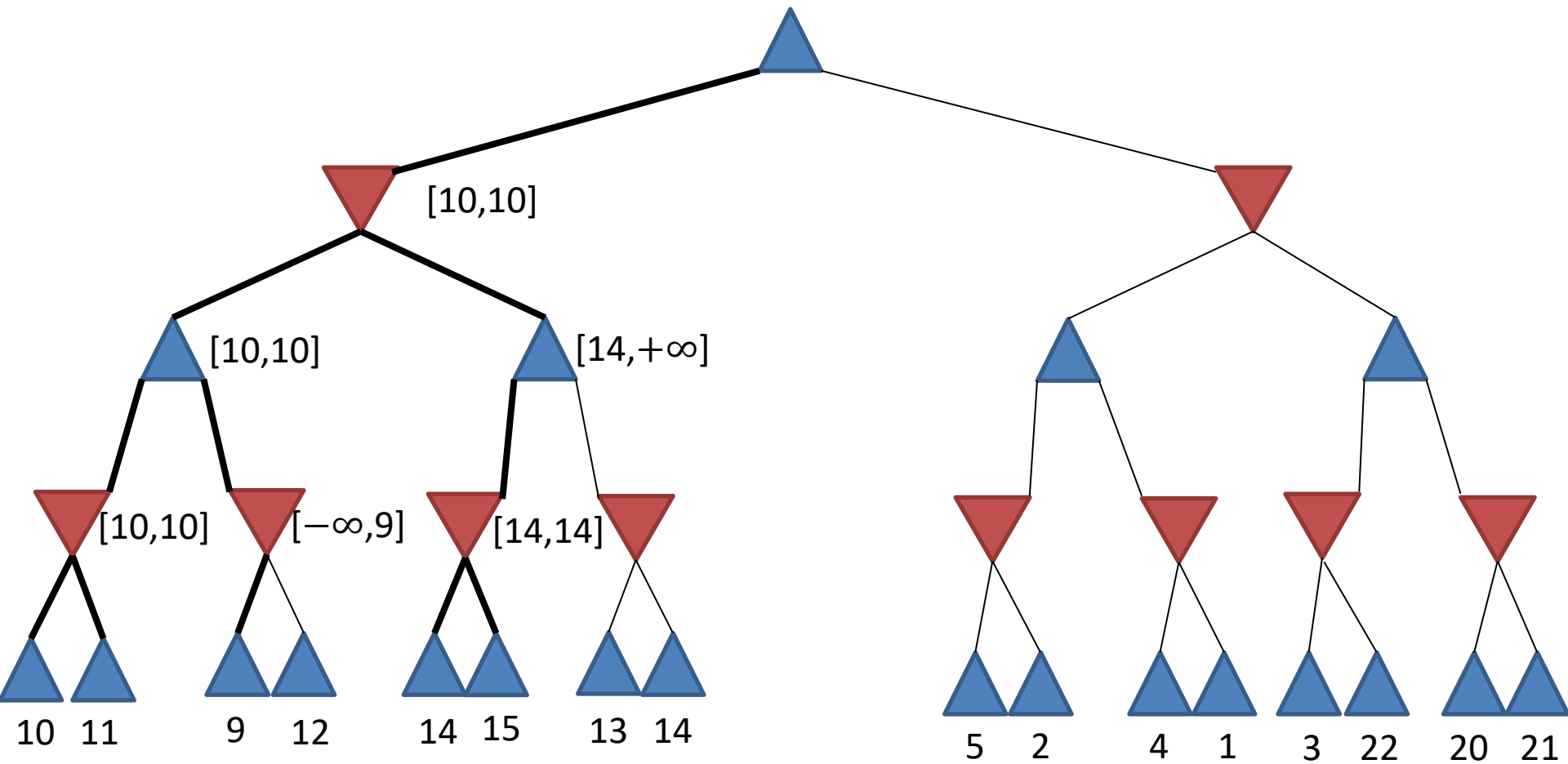
Παράδειγμα



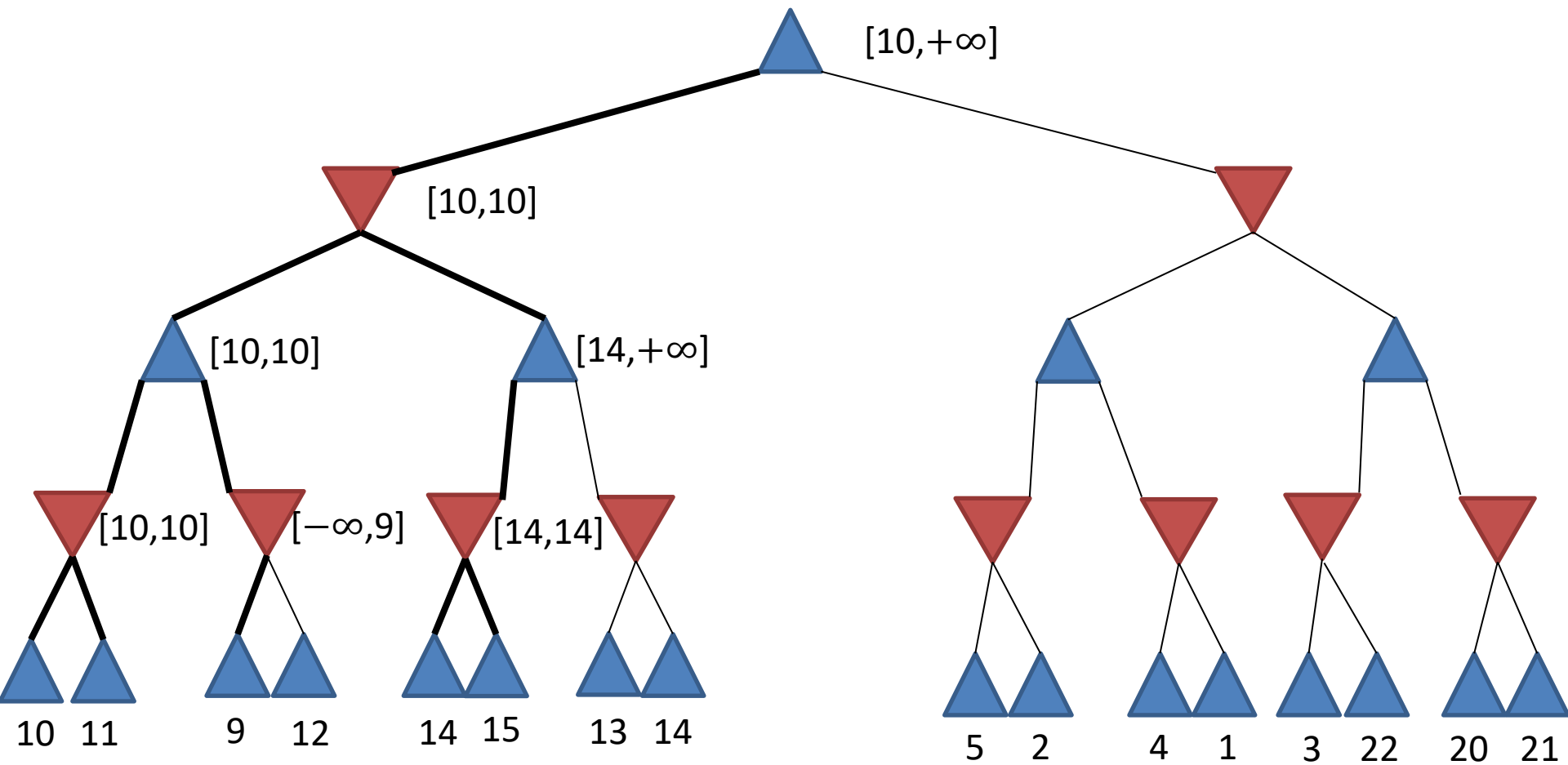
Παράδειγμα



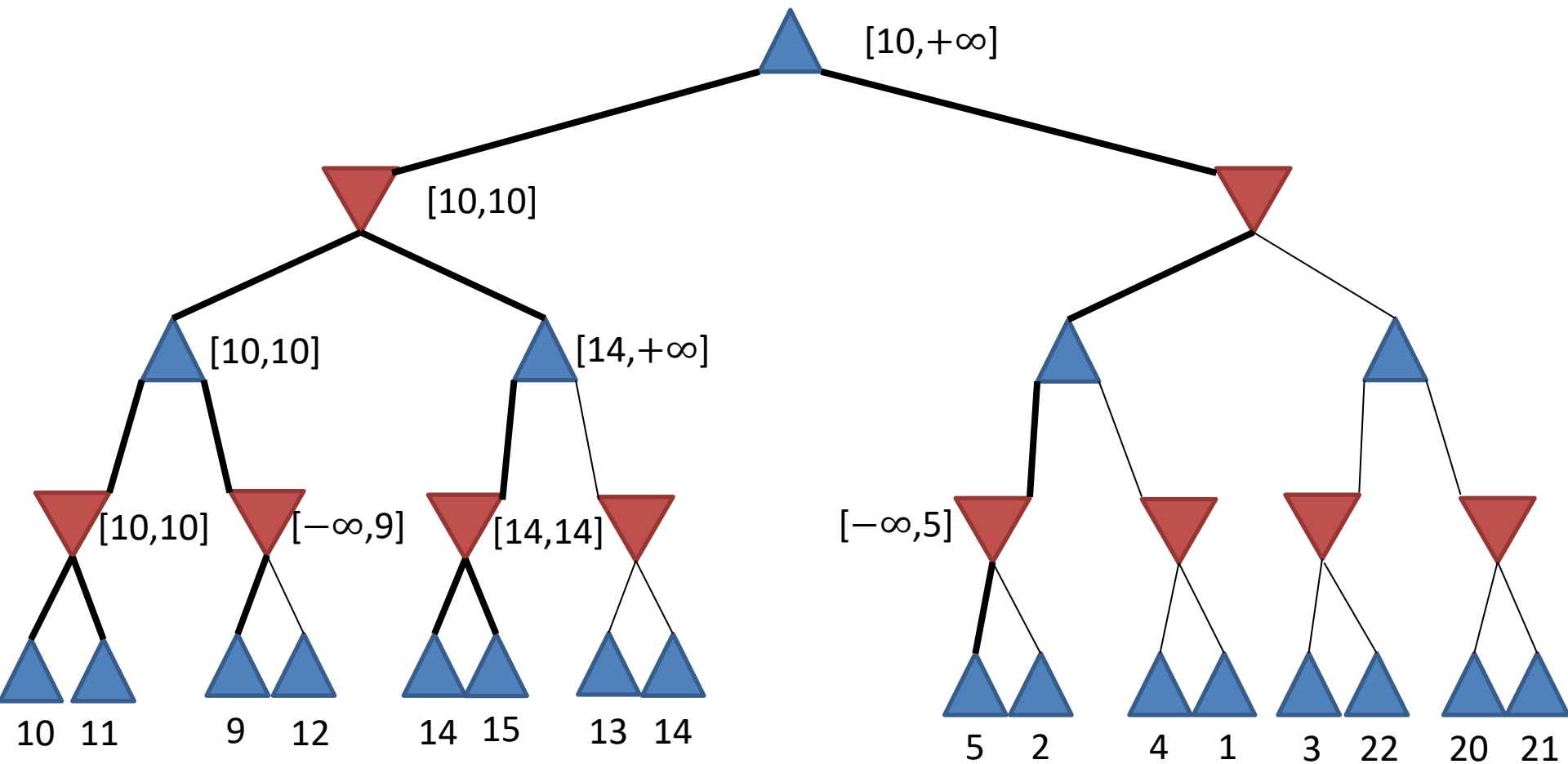
Παράδειγμα



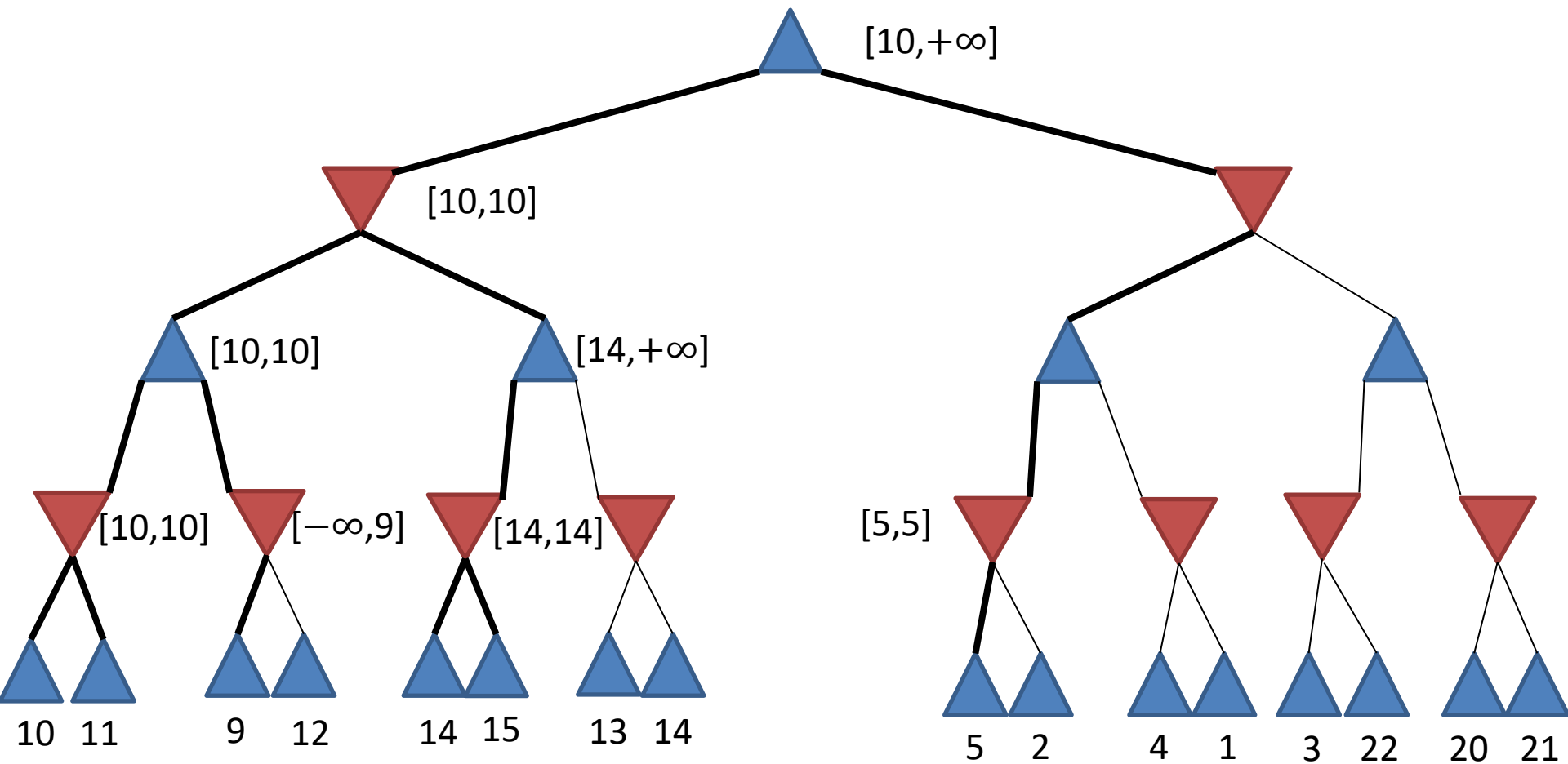
Παράδειγμα



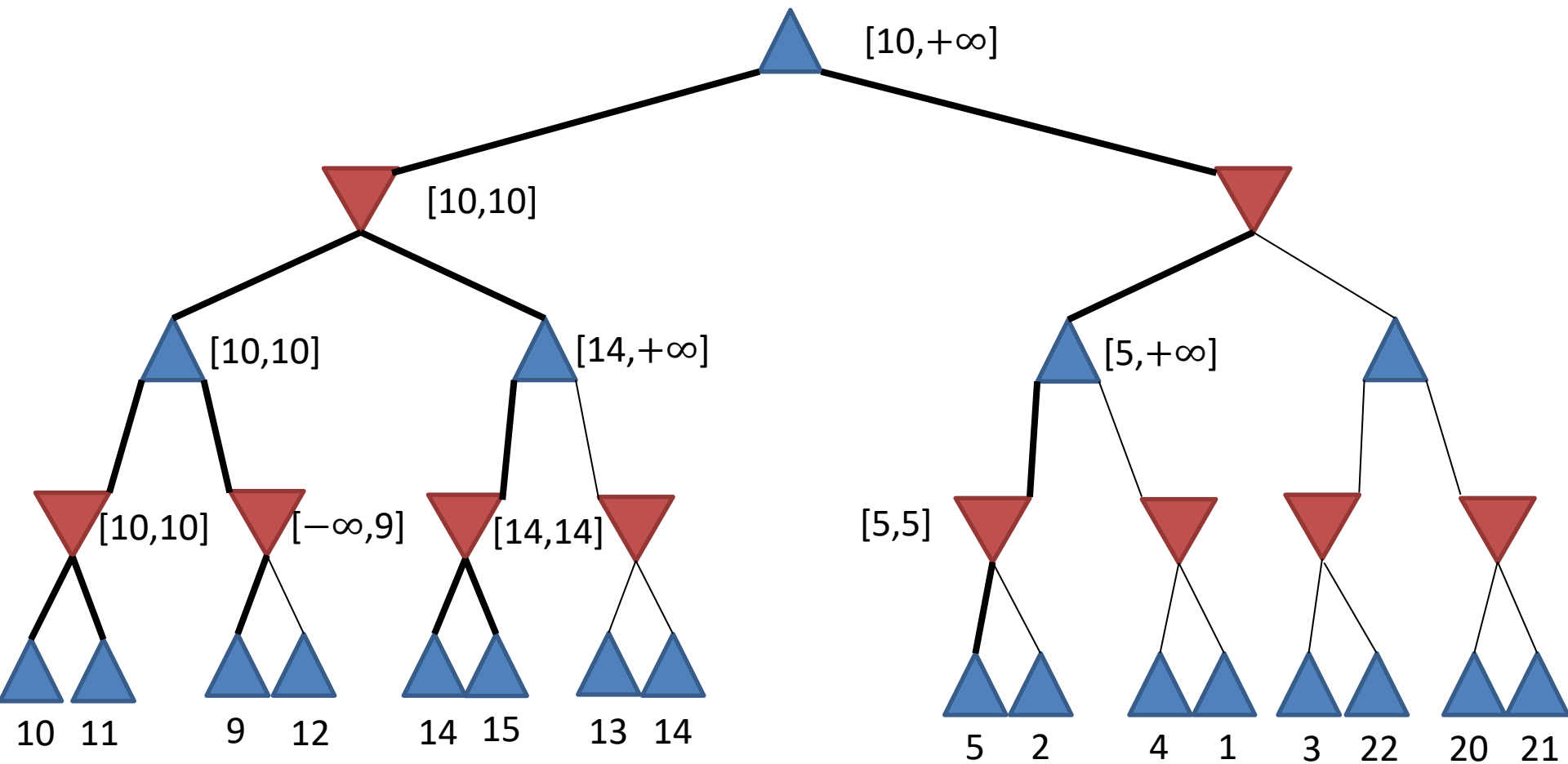
Παράδειγμα



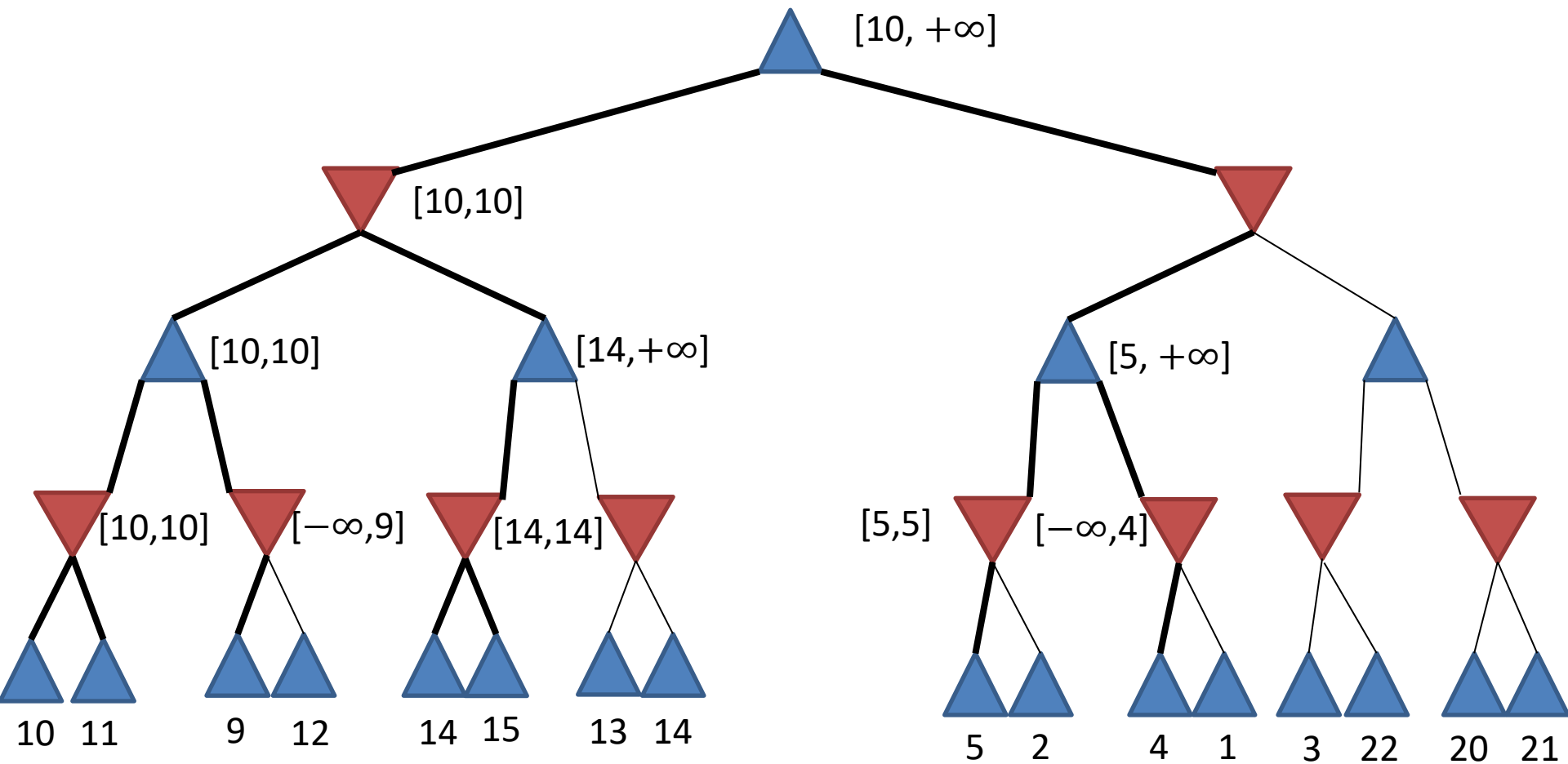
Παράδειγμα



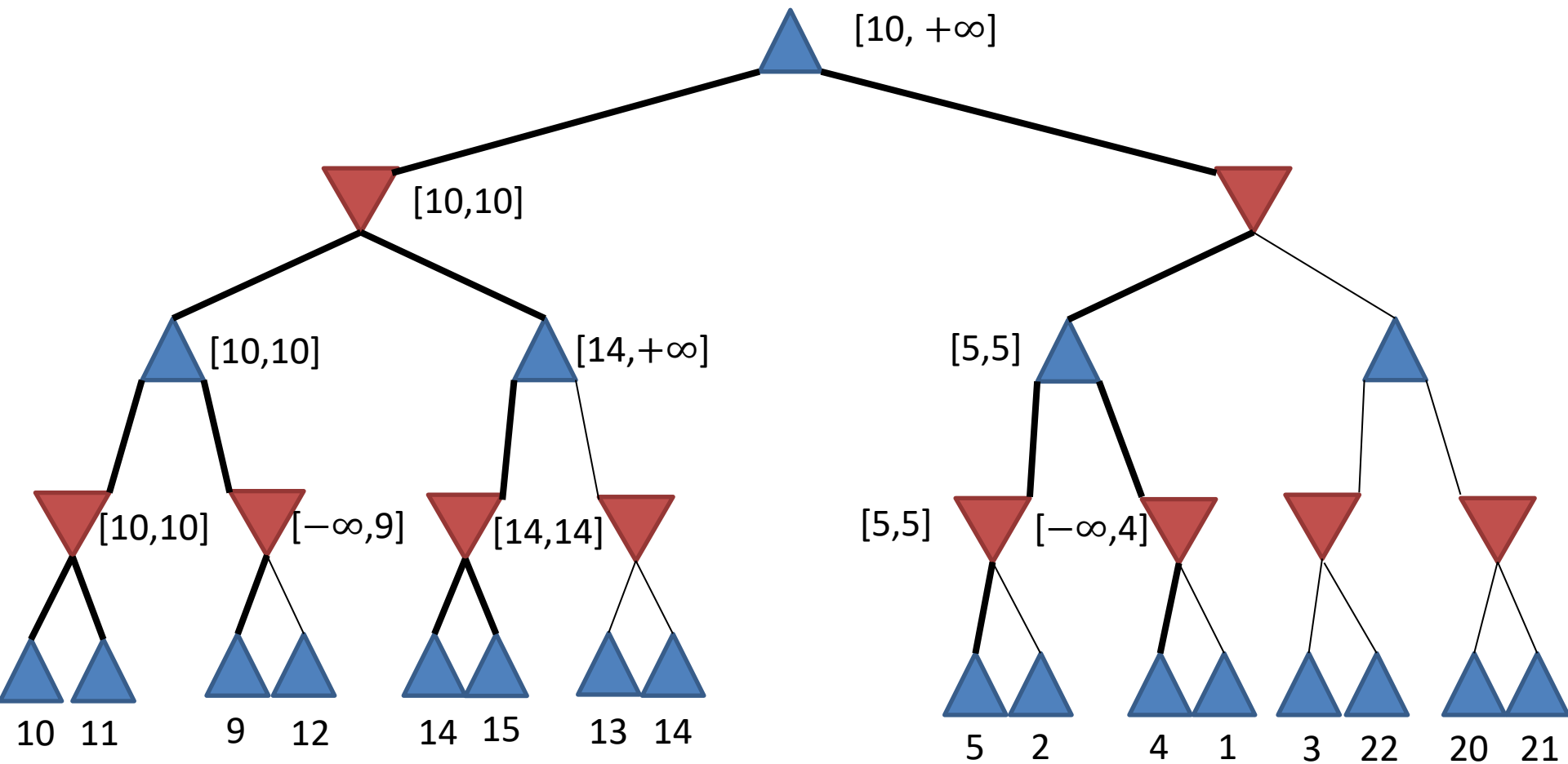
Παράδειγμα



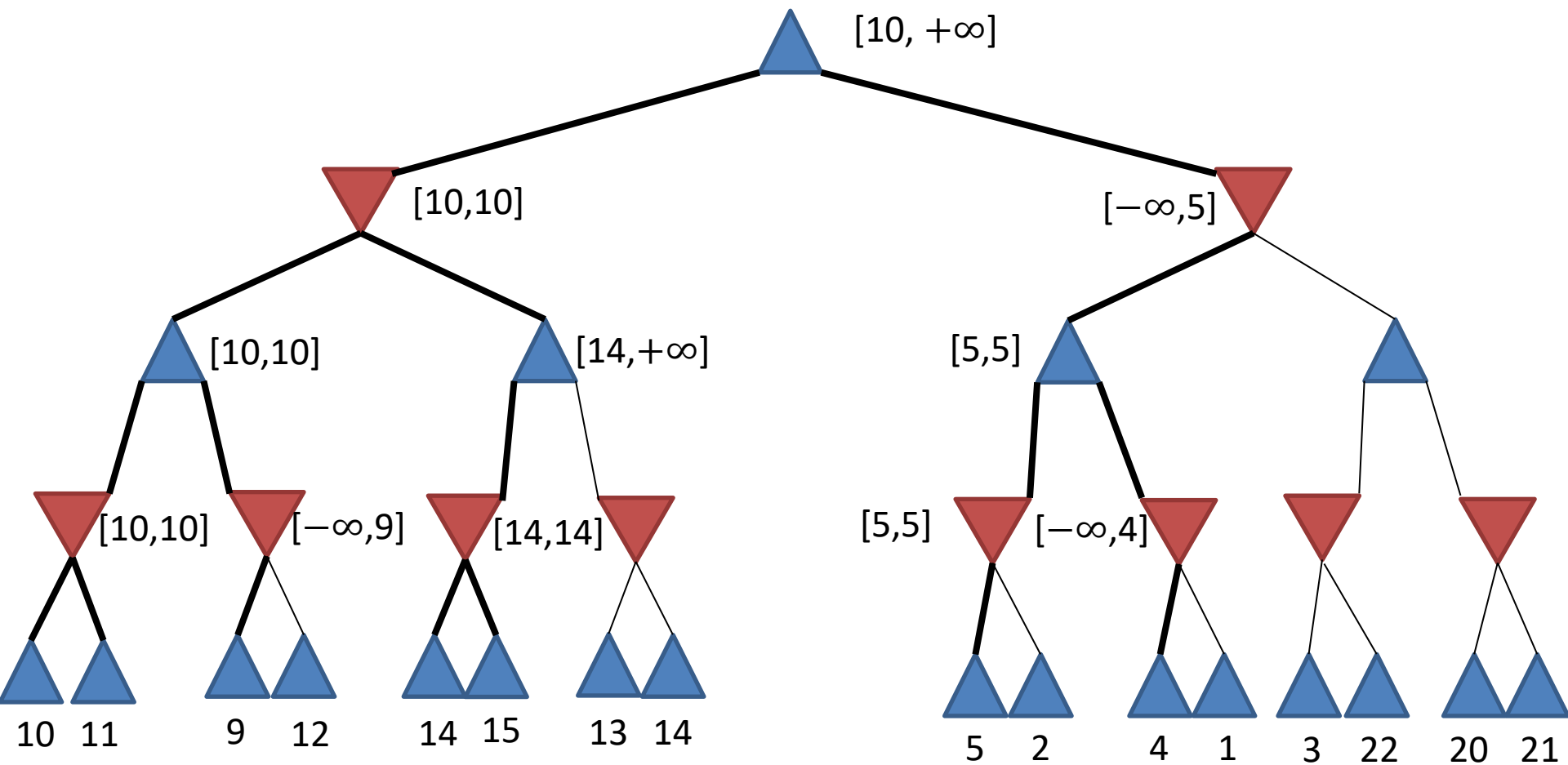
Παράδειγμα



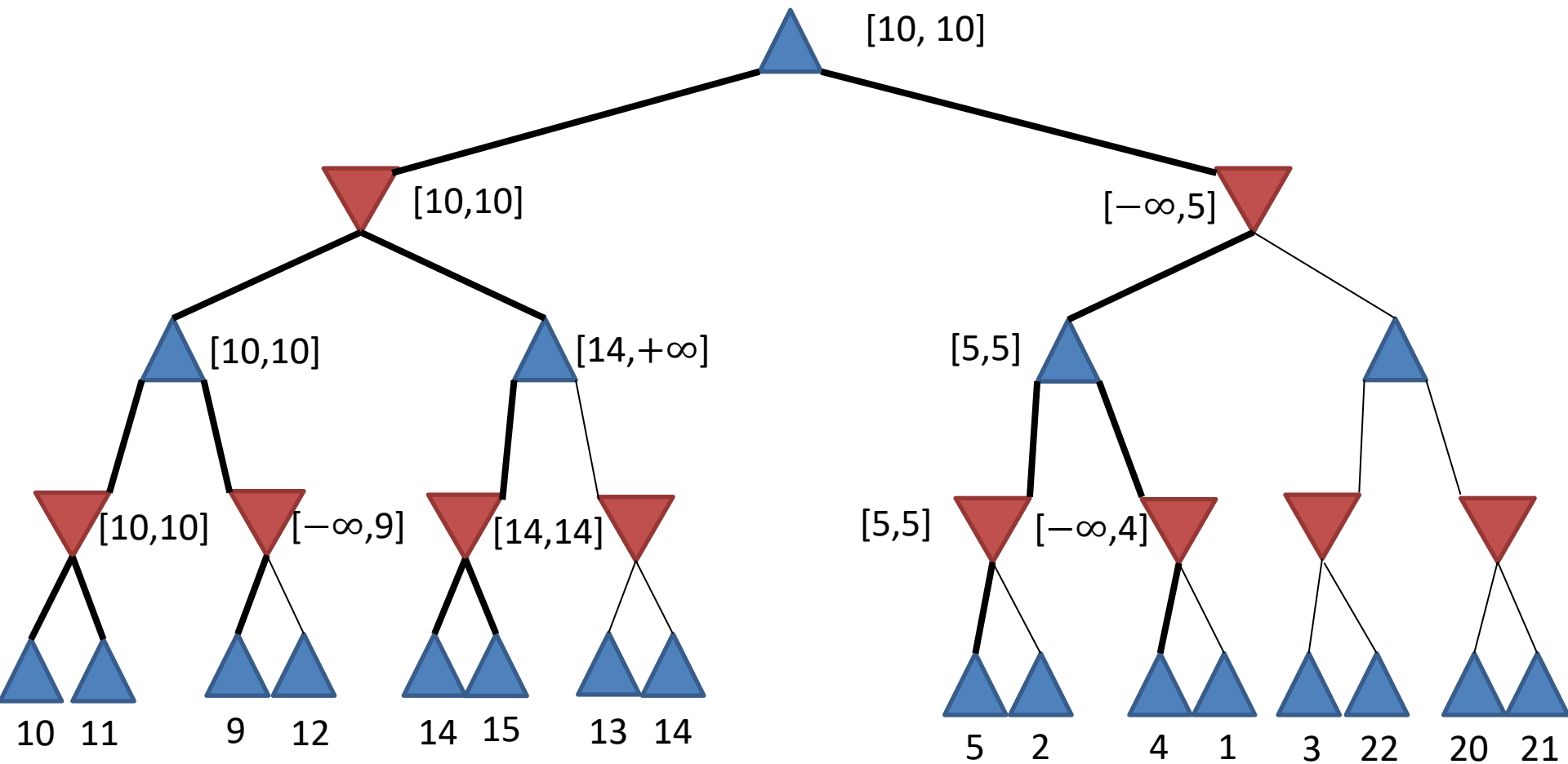
Παράδειγμα



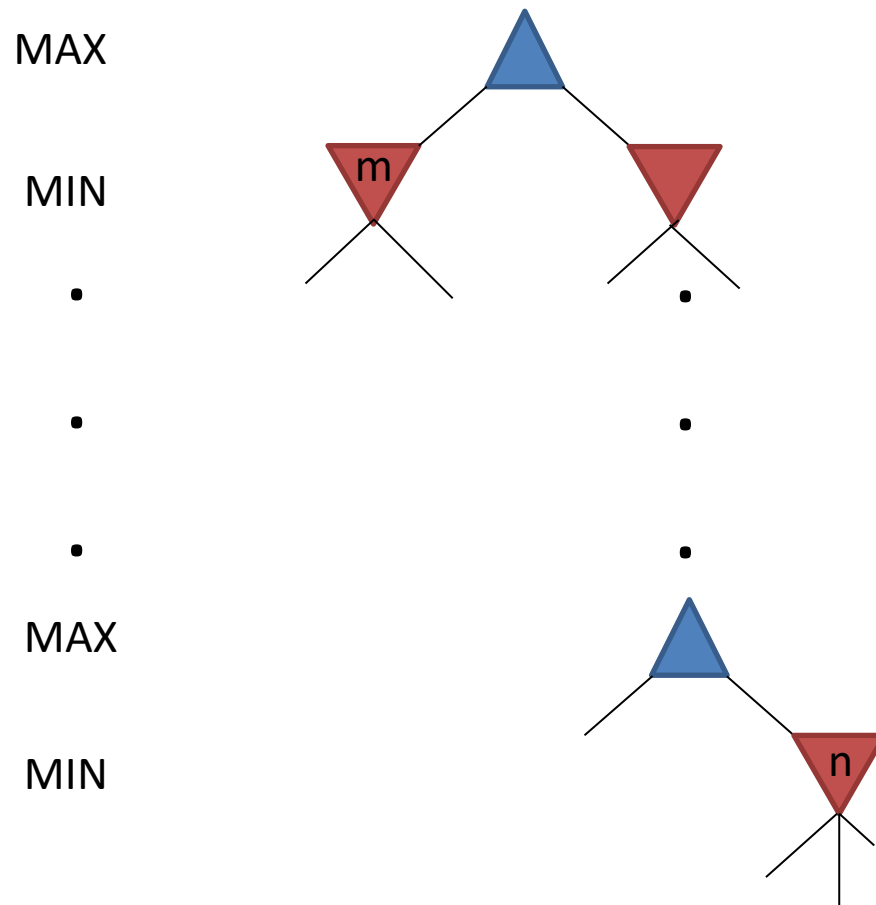
Παράδειγμα



Παράδειγμα



Κλάδεμα Άλφα-Βήτα: Η Γενική Περίπτωση



Η Γενική Περίπτωση για τον MIN

- Ας υποθέσουμε ότι υπολογίζουμε την τιμή MIN σε κάποιο κόμβο n .
- Εξετάζουμε όλα τα παιδιά του n το ένα μετά το άλλο.
- Η εκτίμηση της τιμής MIN ελαττώνεται καθώς προχωράμε.
- Ποιος ενδιαφέρεται για την τιμή του n ; Ο MAX!
- Έστω α η μικρότερη τιμή που μπορεί να πάρει ο MAX σε οποιοδήποτε σημείο επιλογής του τωρινού μονοπατιού από την ρίζα (π.χ., στο m).
- Αν η τιμή του n γίνει μικρότερη από το α , **μπορούμε να σταματήσουμε να εξετάζουμε τα επόμενα παιδιά του n** (η τιμή του n είναι ήδη χαμηλή και δεν θα φτάσουμε εκεί στο πραγματικό παιχνίδι).
- Η γενική περίπτωση για τον MAX είναι συμμετρική.

Οι Παράμετροι α και β



- Το **κλάδεμα άλφα-βήτα** παίρνει το όνομα του από τις δύο παρακάτω παραμέτρους που προσδιορίζουν φράγματα για τις τιμές χρησιμότητας που αντιγράφονται κατά μήκος μιας διαδρομής στο δένδρο παιχνιδιού:
 - **α** : η τιμή της καλύτερης επιλογής για τον MAX (η **μεγαλύτερη** τιμή) που έχουμε βρει μέχρι στιγμής σε οποιοδήποτε κόμβο κατά μήκος της διαδρομής.
 - **β** : η τιμή της καλύτερης επιλογής για τον MIN (η **μικρότερη** τιμή) που έχουμε βρει μέχρι στιγμής σε οποιοδήποτε κόμβο κατά μήκος της διαδρομής.
- Η αναζήτηση άλφα-βήτα ενημερώνει τις τιμές των α και β καθώς προχωρά, και κλαδεύει τα υπόλοιπα κλαδιά σε ένα κόμβο (δηλαδή τερματίζει την αναδρομική κλήση) μόλις γίνει γνωστό ότι η τιμή του τρέχοντος κόμβου είναι χειρότερη από την τρέχουσα τιμή των α και β για τον MAX και τον MIN αντίστοιχα.

Ο Αλγόριθμος Αναζήτησης Άλφα-Βήτα

function ALPHA-BETA-SEARCH($state$) **returns** an action

$v \leftarrow \text{MAX-VALUE}(state, -\infty, +\infty)$

return the *action* in $\text{ACTIONS}(state)$ with value v

function MAX-VALUE($state, \alpha, \beta$) **returns** a utility value

if $\text{TERMINAL-TEST}(state)$ **then return** $\text{UTILITY}(state)$

$v \leftarrow -\infty$

for each a **in** $\text{ACTIONS}(state)$ **do**

$v \leftarrow \text{MAX}(v, \text{MIN-VALUE}(\text{RESULT}(state, a), \alpha, \beta))$

if $v \geq \beta$ **then return** v

$\alpha \leftarrow \text{MAX}(\alpha, v)$

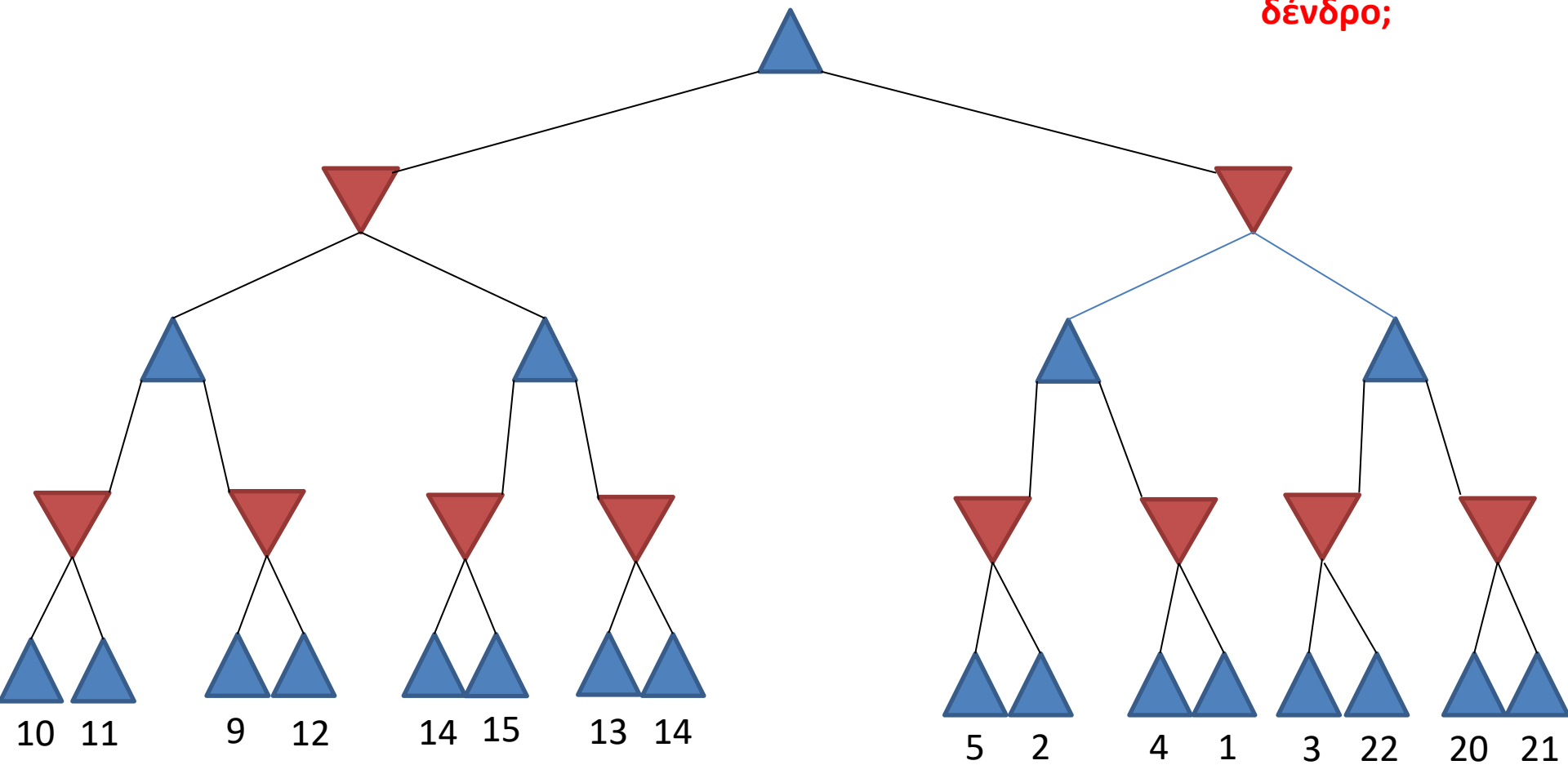
return v

Ο Αλγόριθμος Αναζήτησης Άλφα-Βήτα

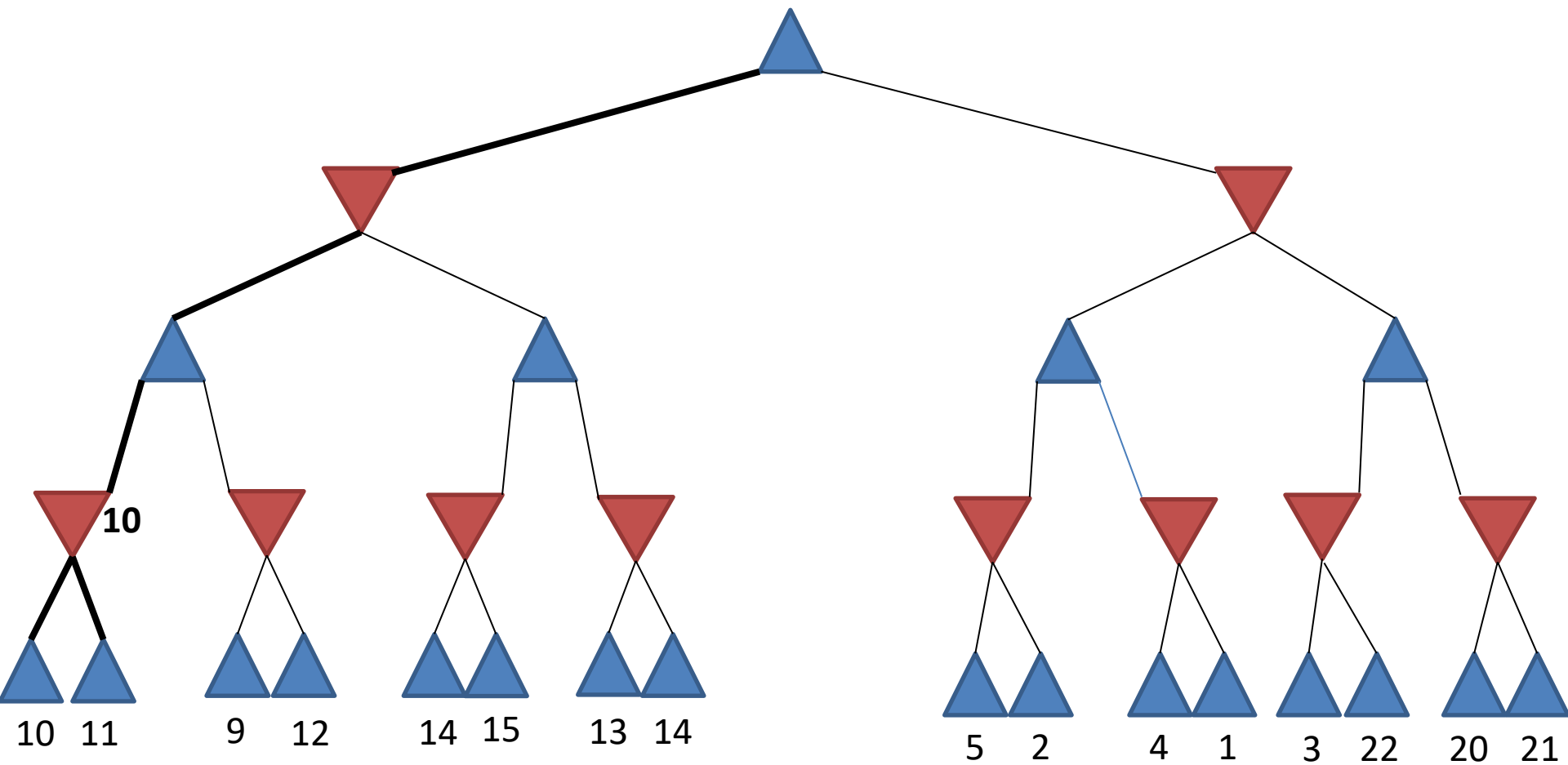
```
function MIN-VALUE(state,  $\alpha$ ,  $\beta$ ) returns a utility value
  if TERMINAL-TEST(state) then return UTILITY(state)
   $v \leftarrow +\infty$ 
  for each  $a$  in ACTIONS(state) do
     $v \leftarrow \text{MIN}(v, \text{MAX-VALUE}(\text{RESULT}(\text{state}, a), \alpha, \beta))$ 
    if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

Παράδειγμα

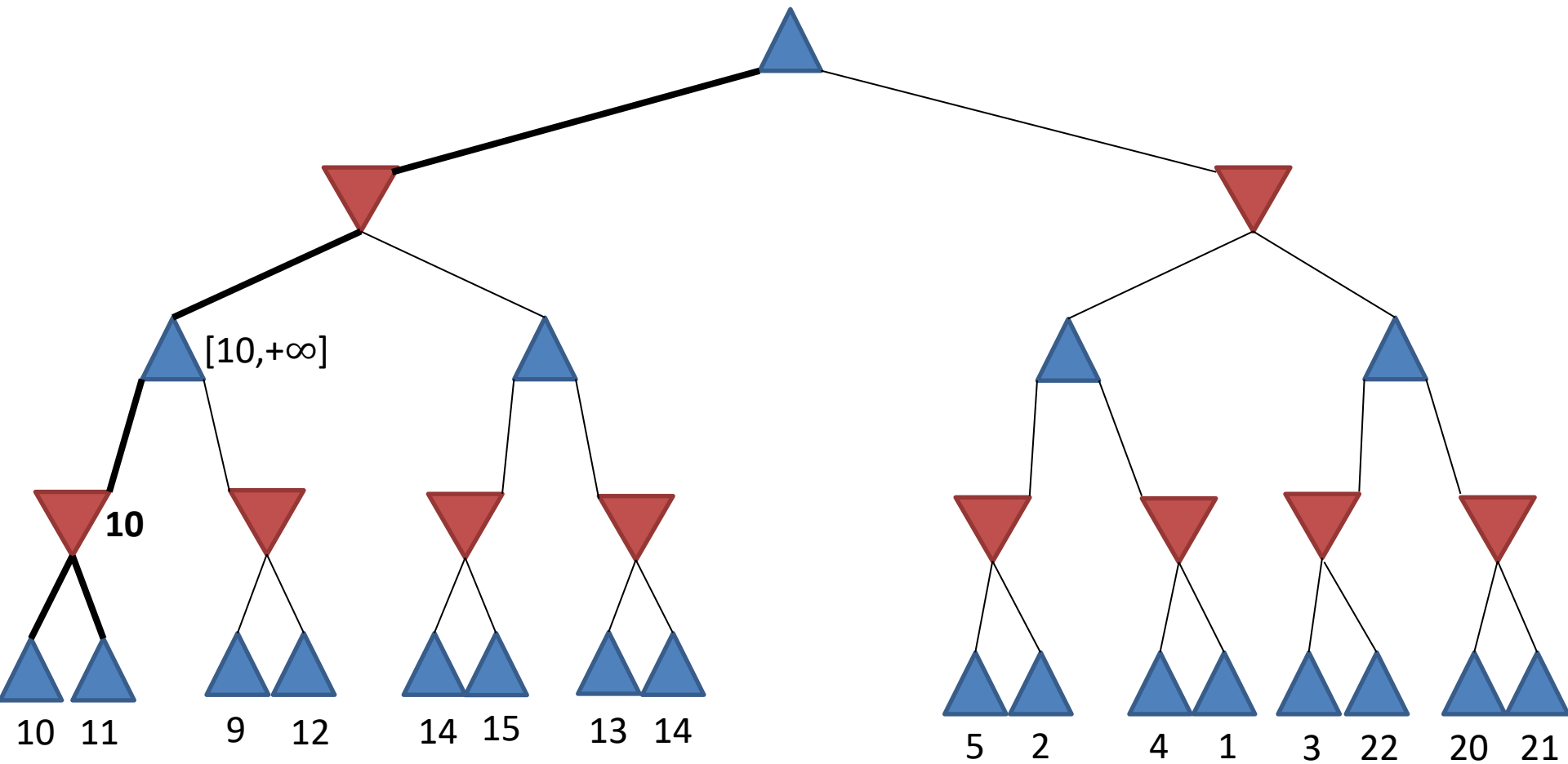
**Ποιο είναι το
αποτέλεσμα του
αλγόριθμου στο
παρακάτω
δένδρο;**



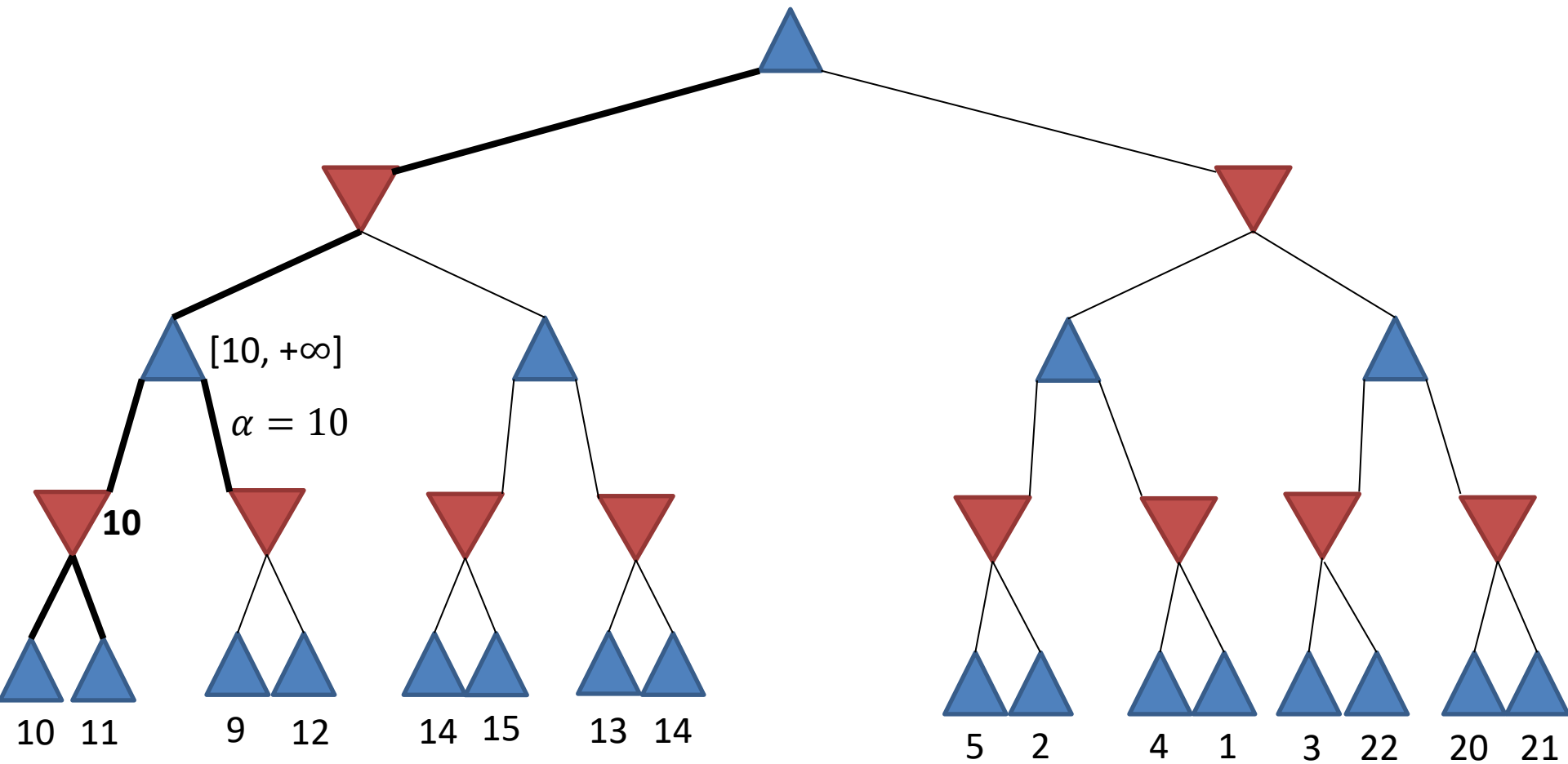
Παράδειγμα



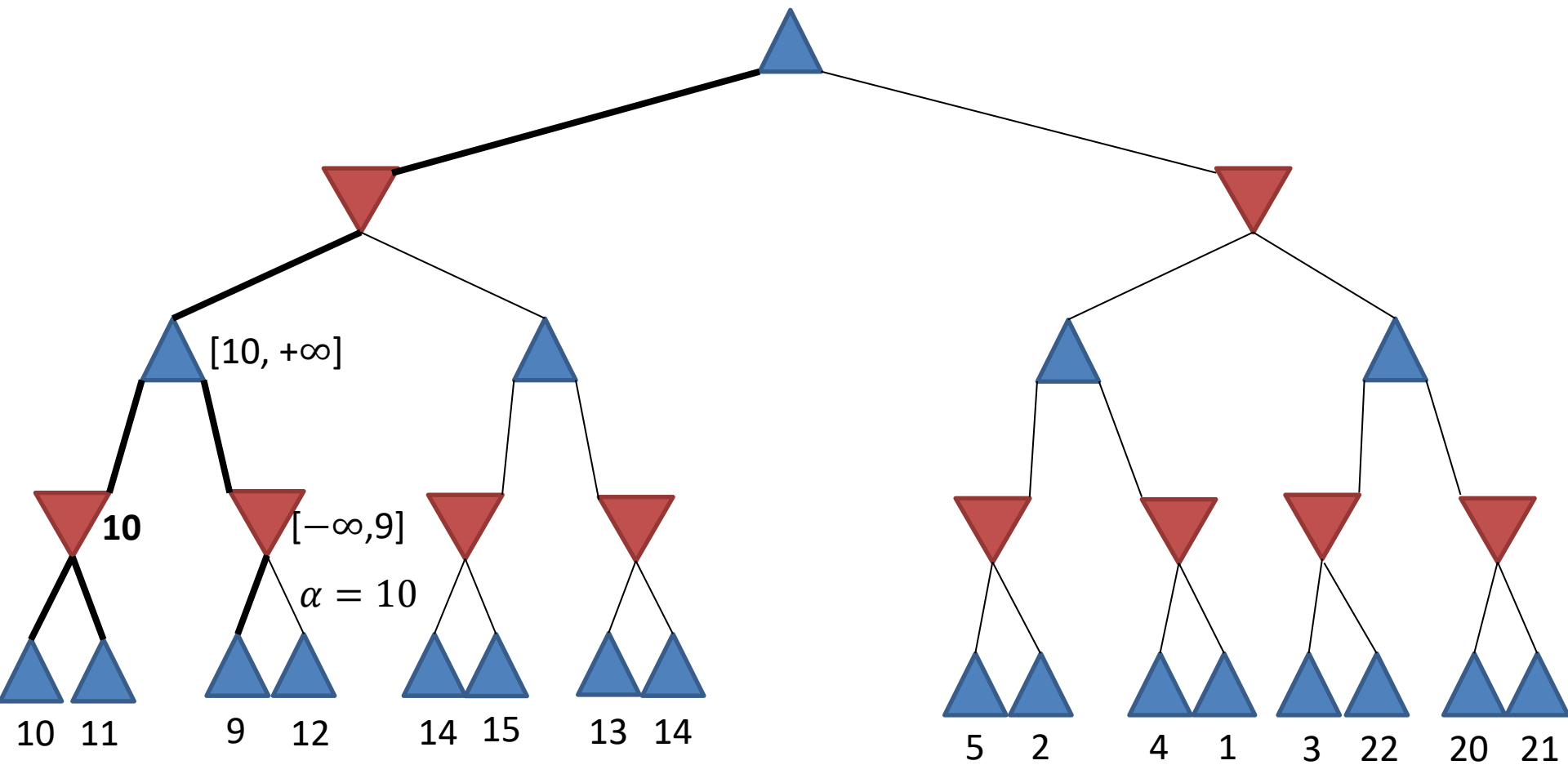
Παράδειγμα



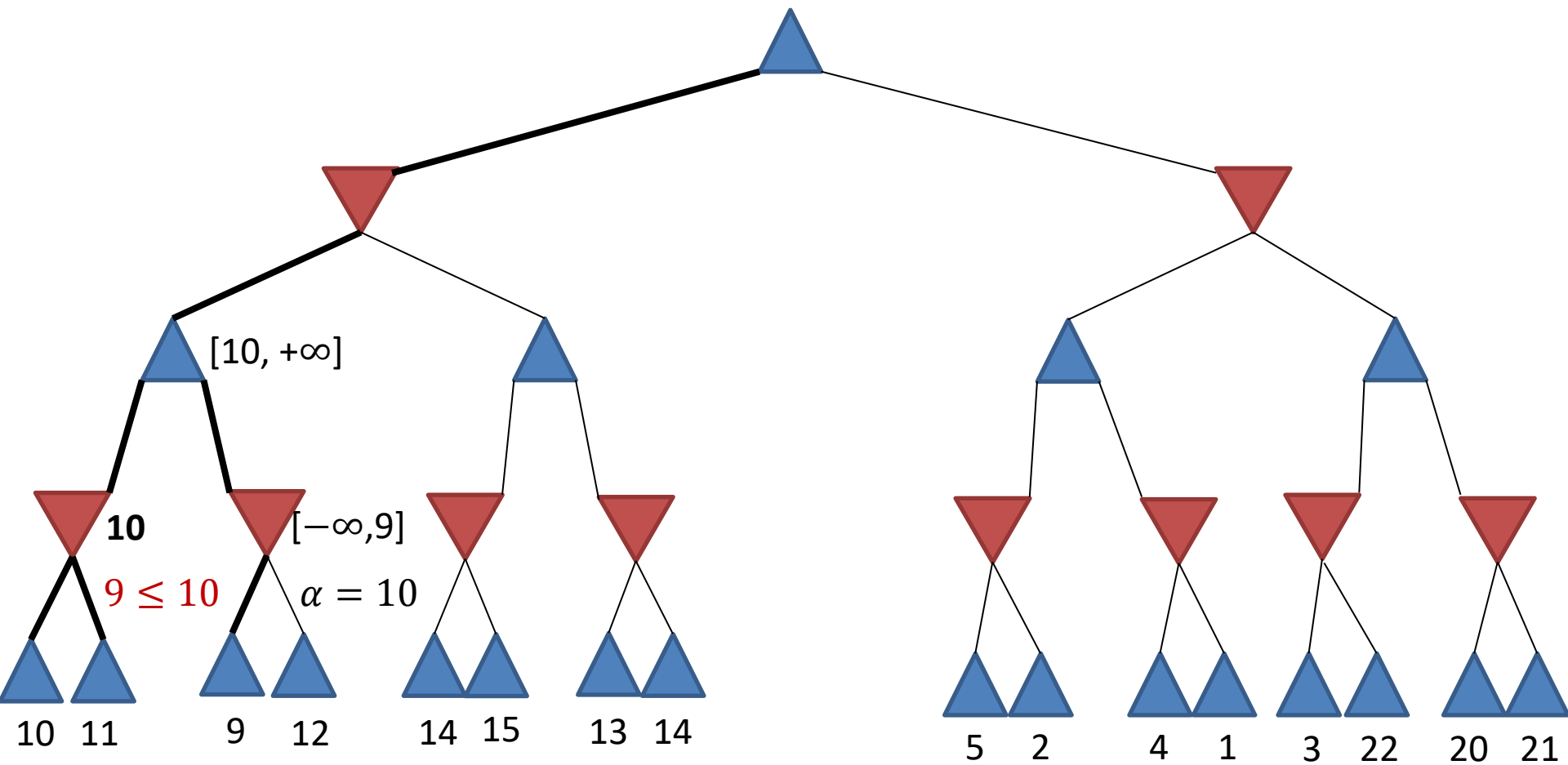
Παράδειγμα



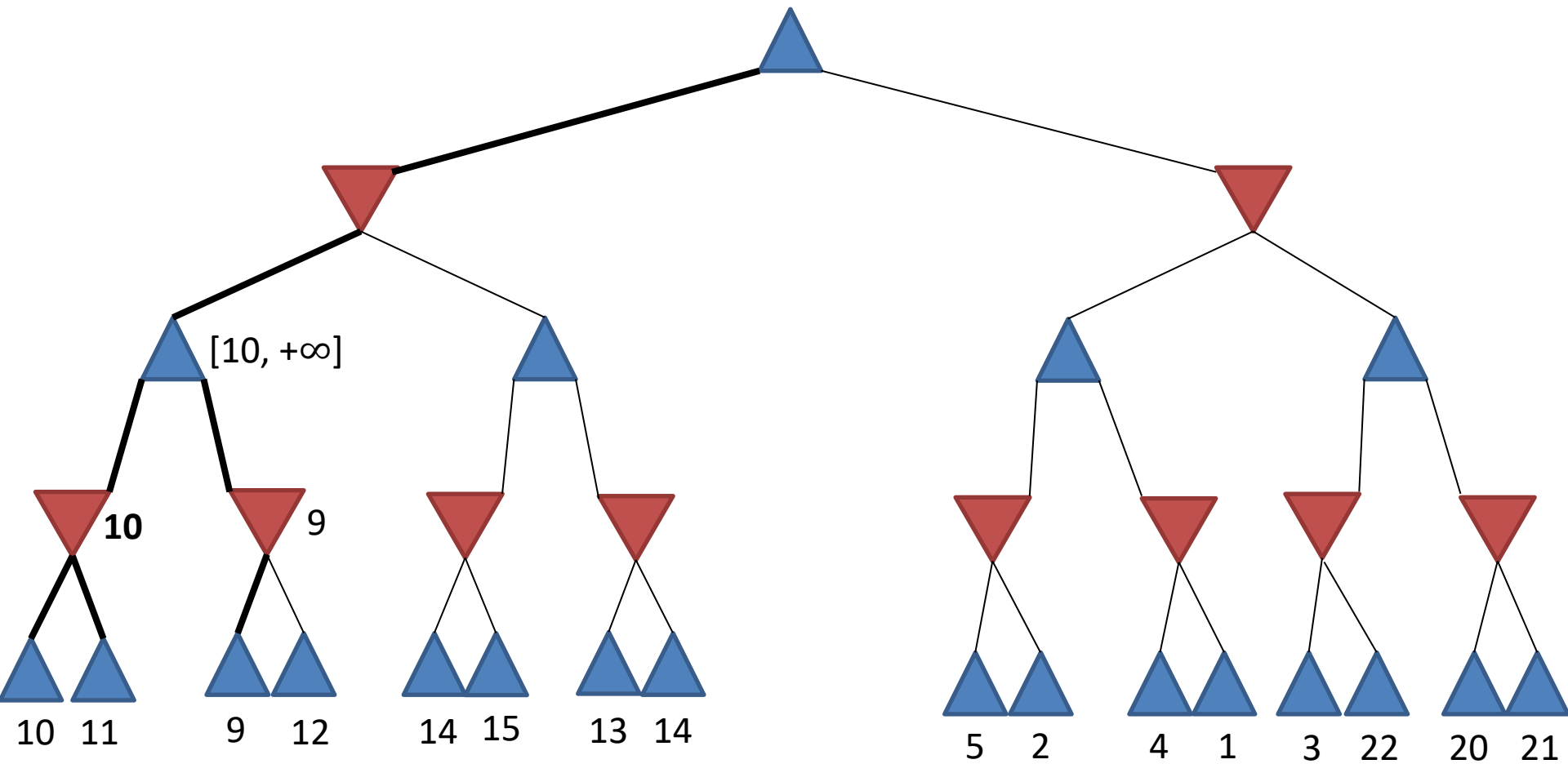
Παράδειγμα



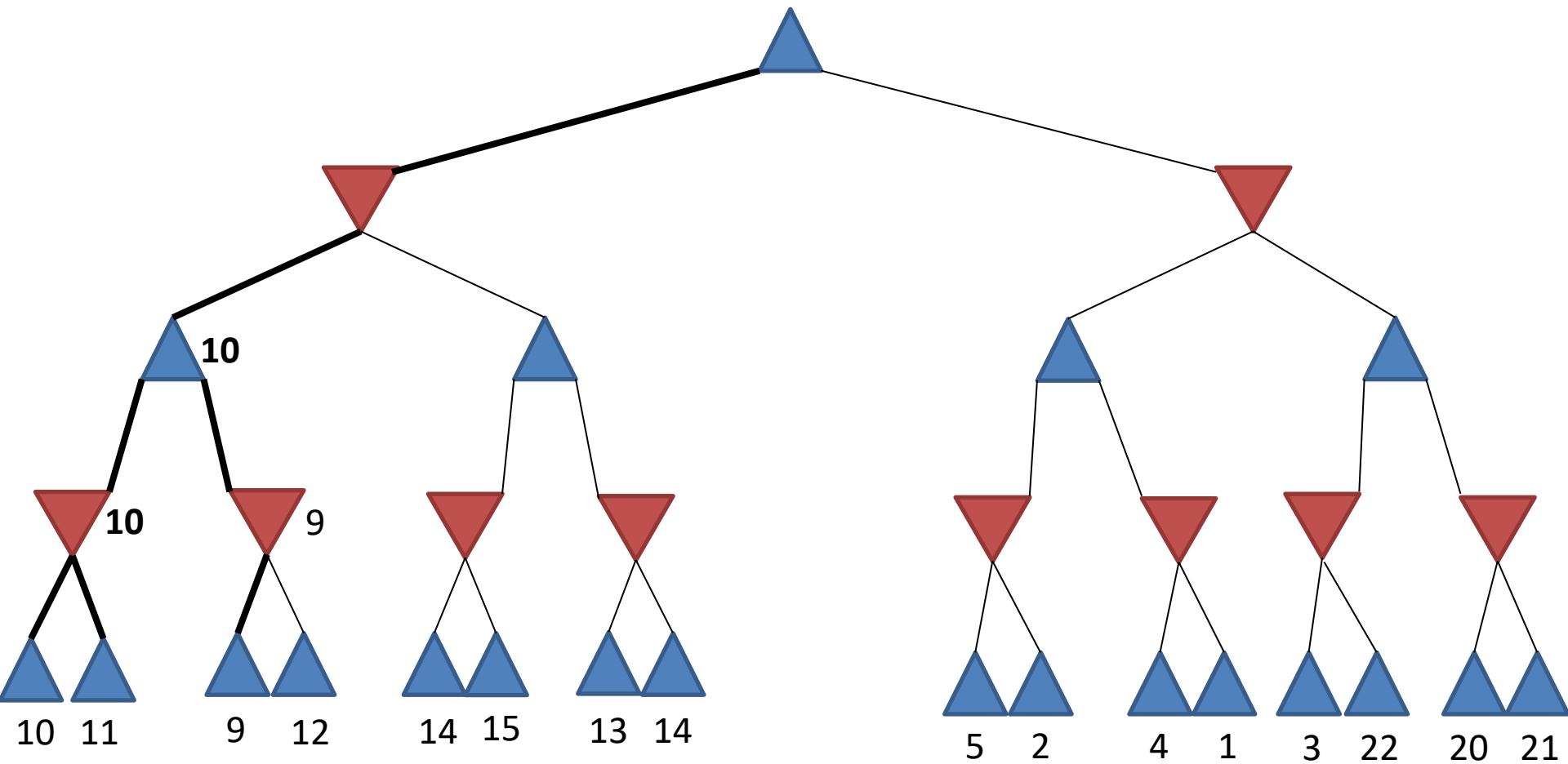
Παράδειγμα



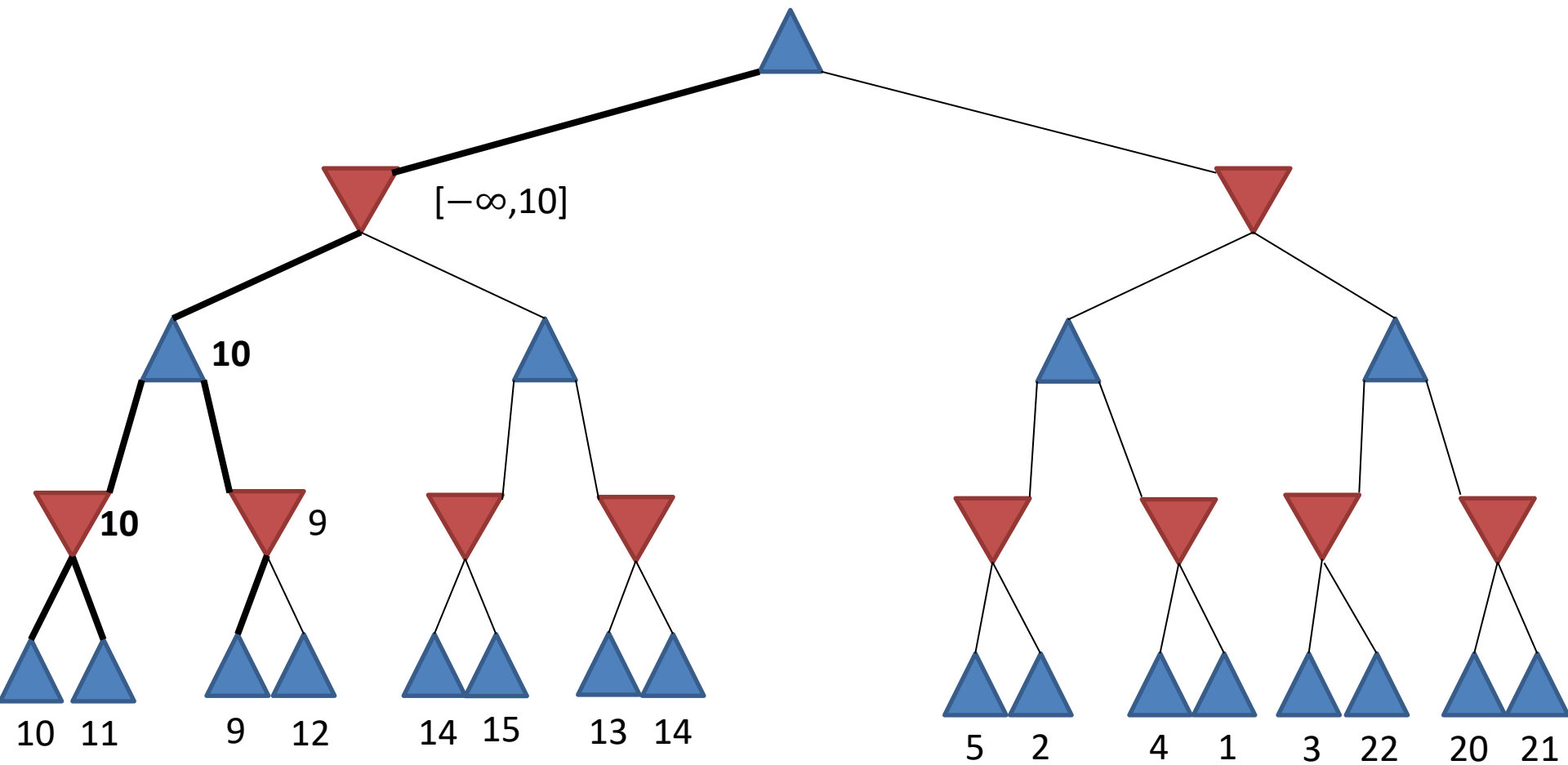
Παράδειγμα



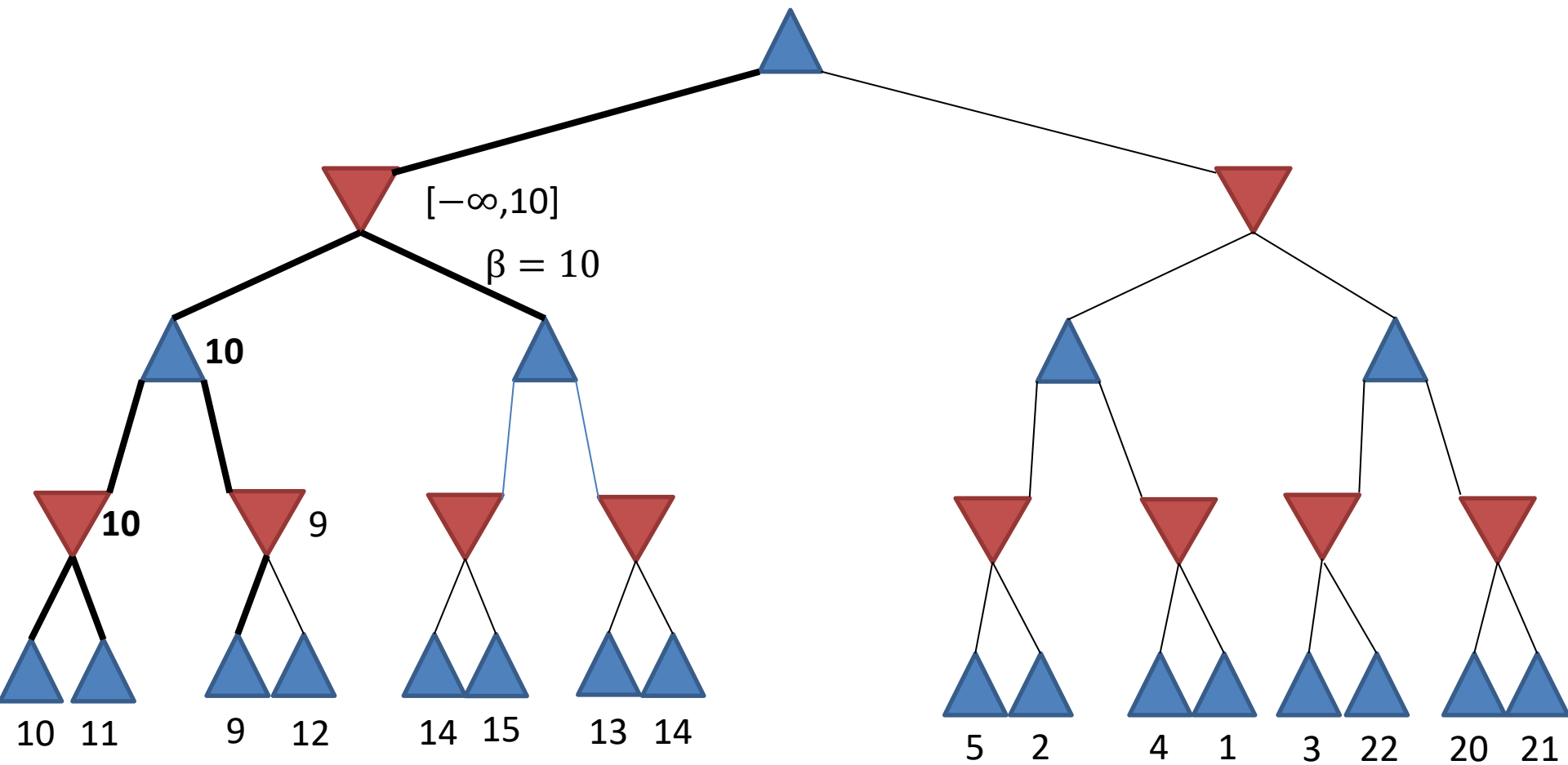
Παράδειγμα



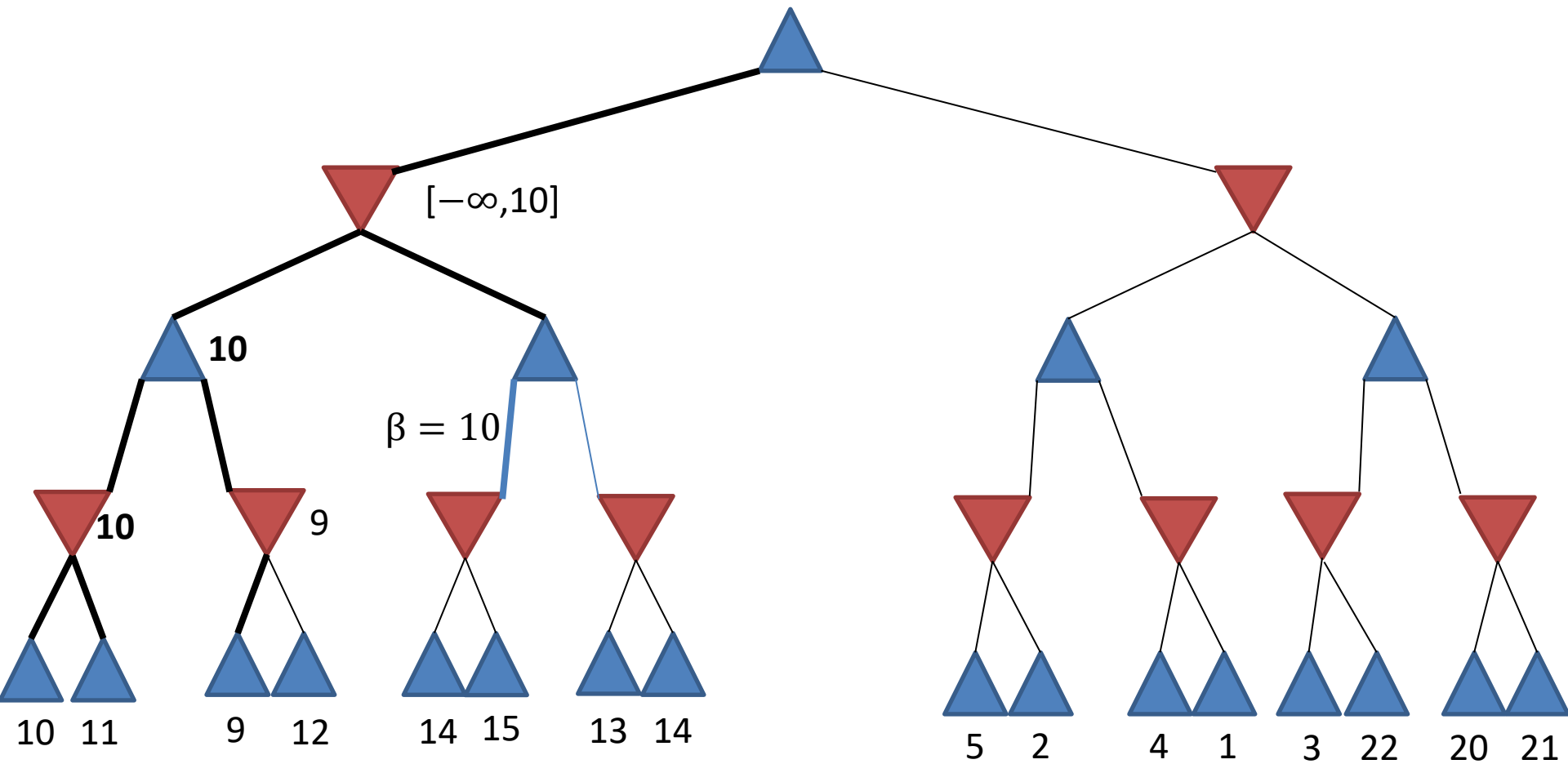
Παράδειγμα



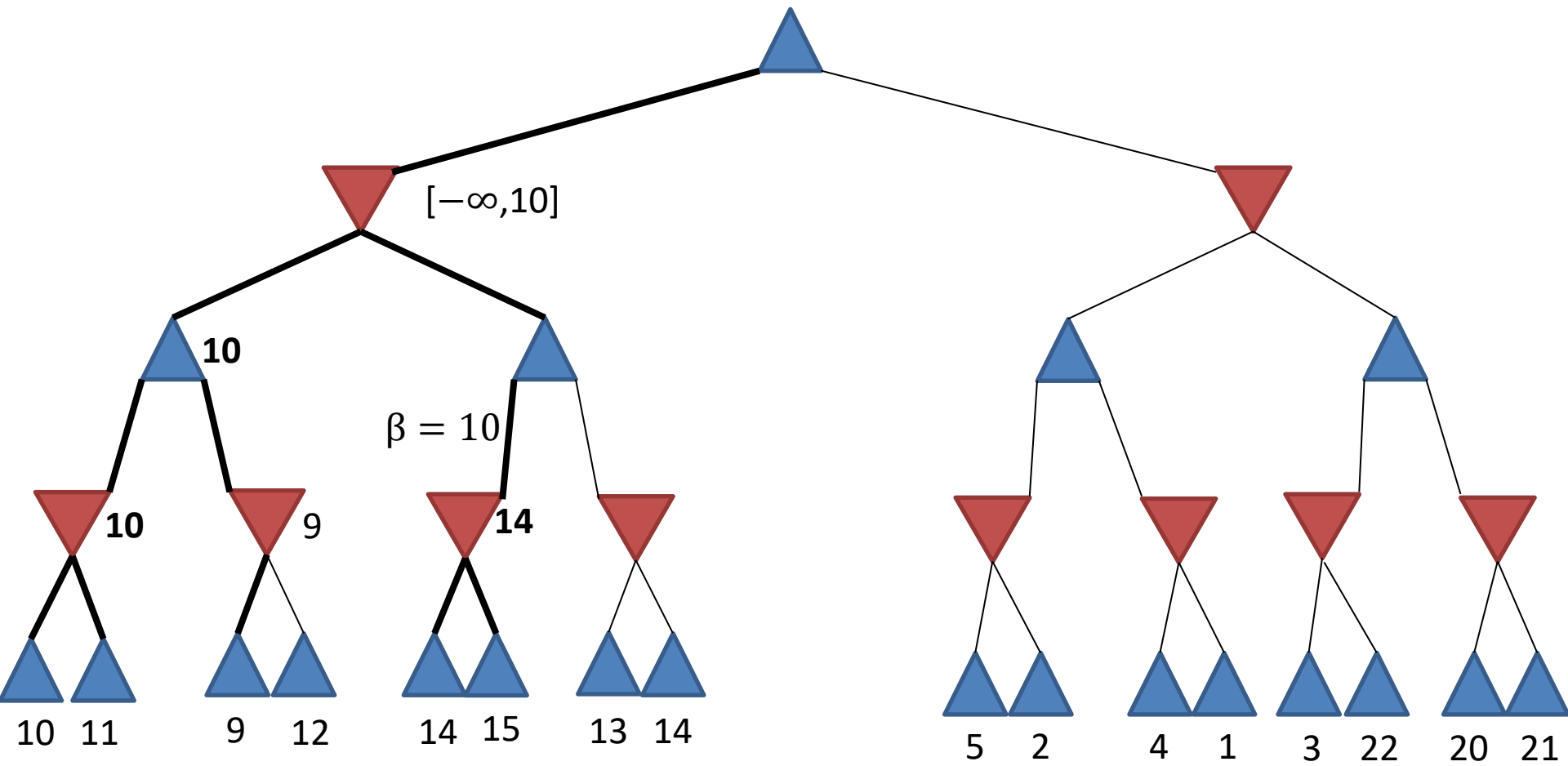
Παράδειγμα



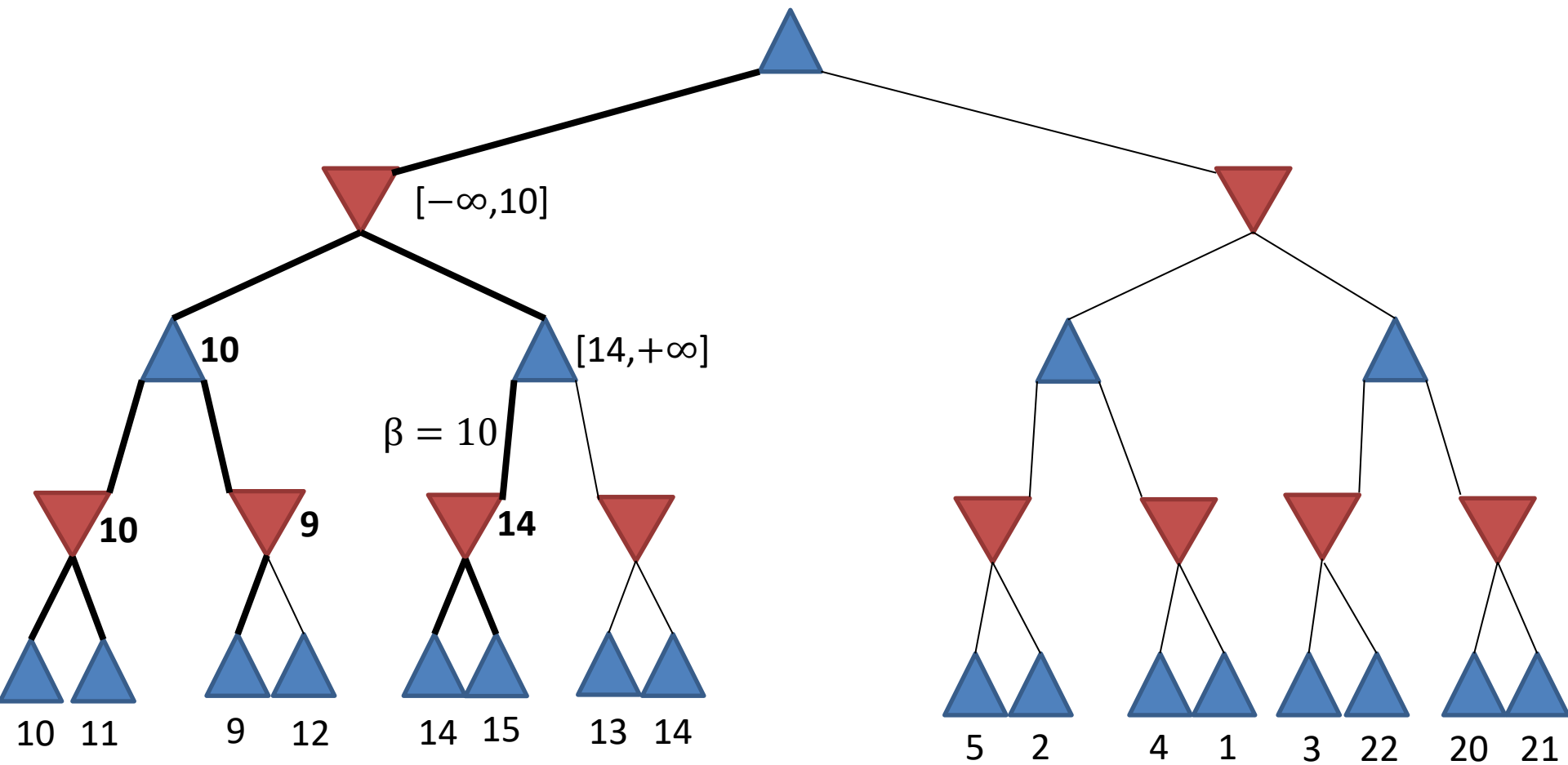
Παράδειγμα



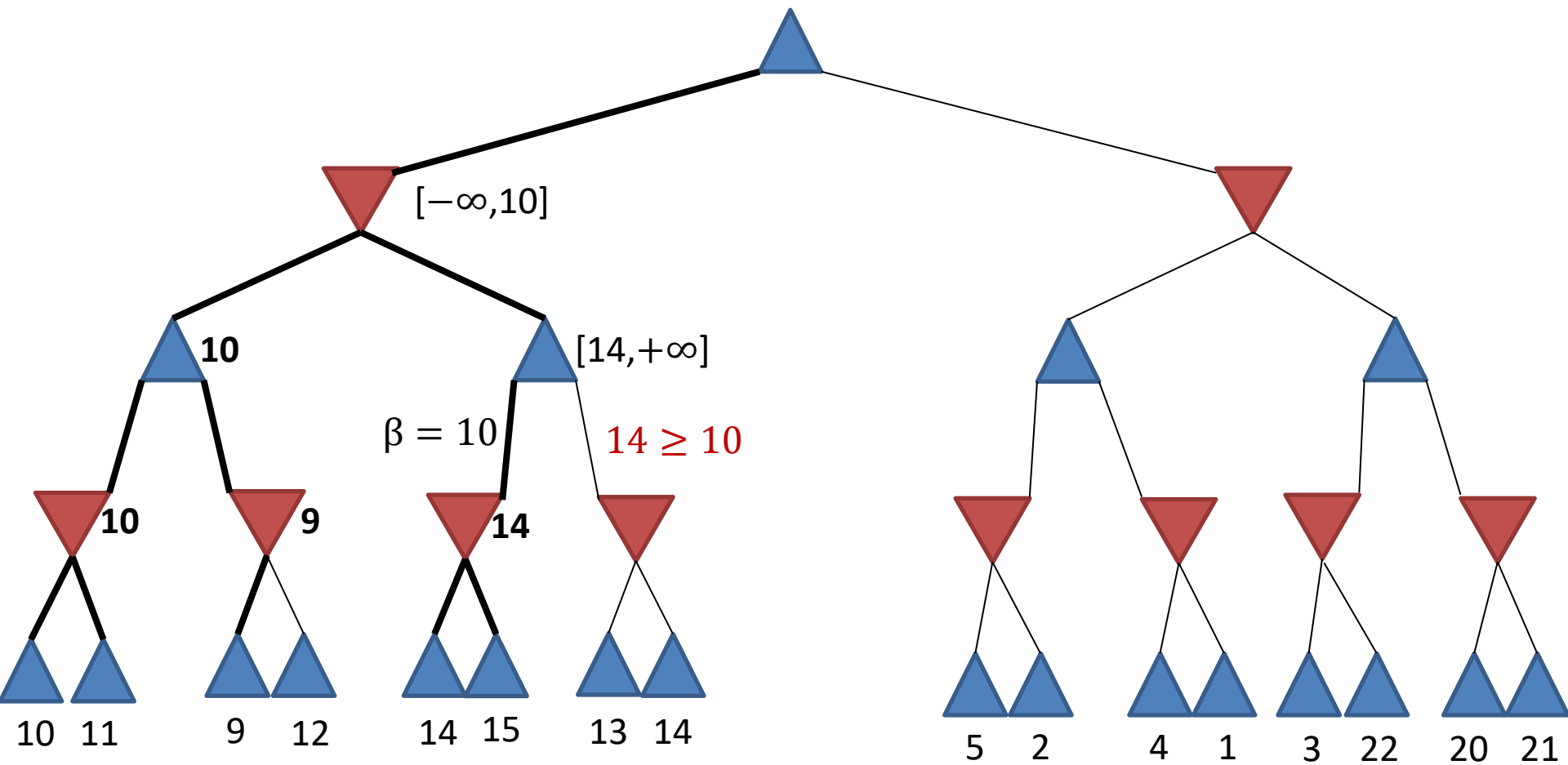
Παράδειγμα



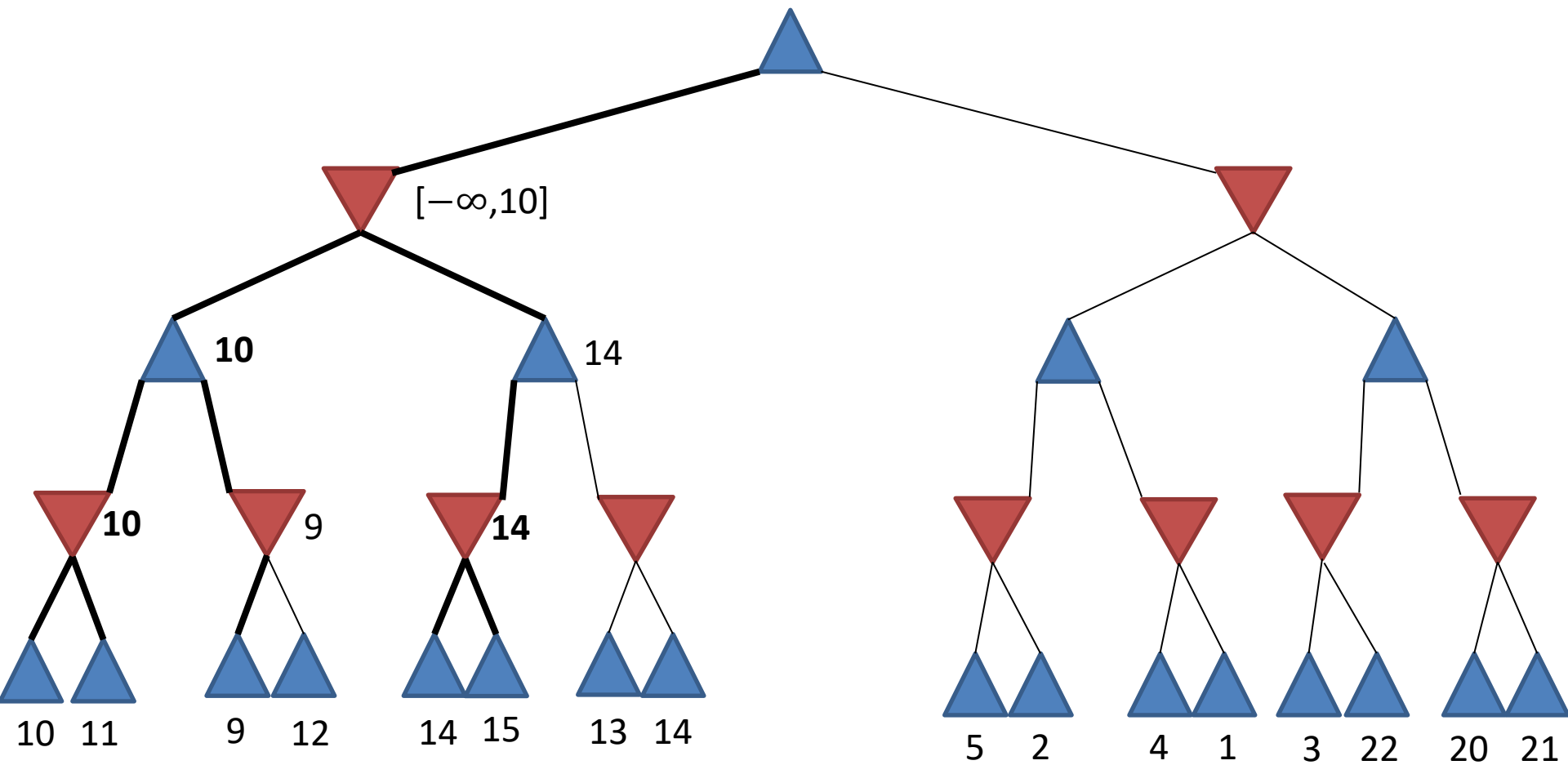
Παράδειγμα



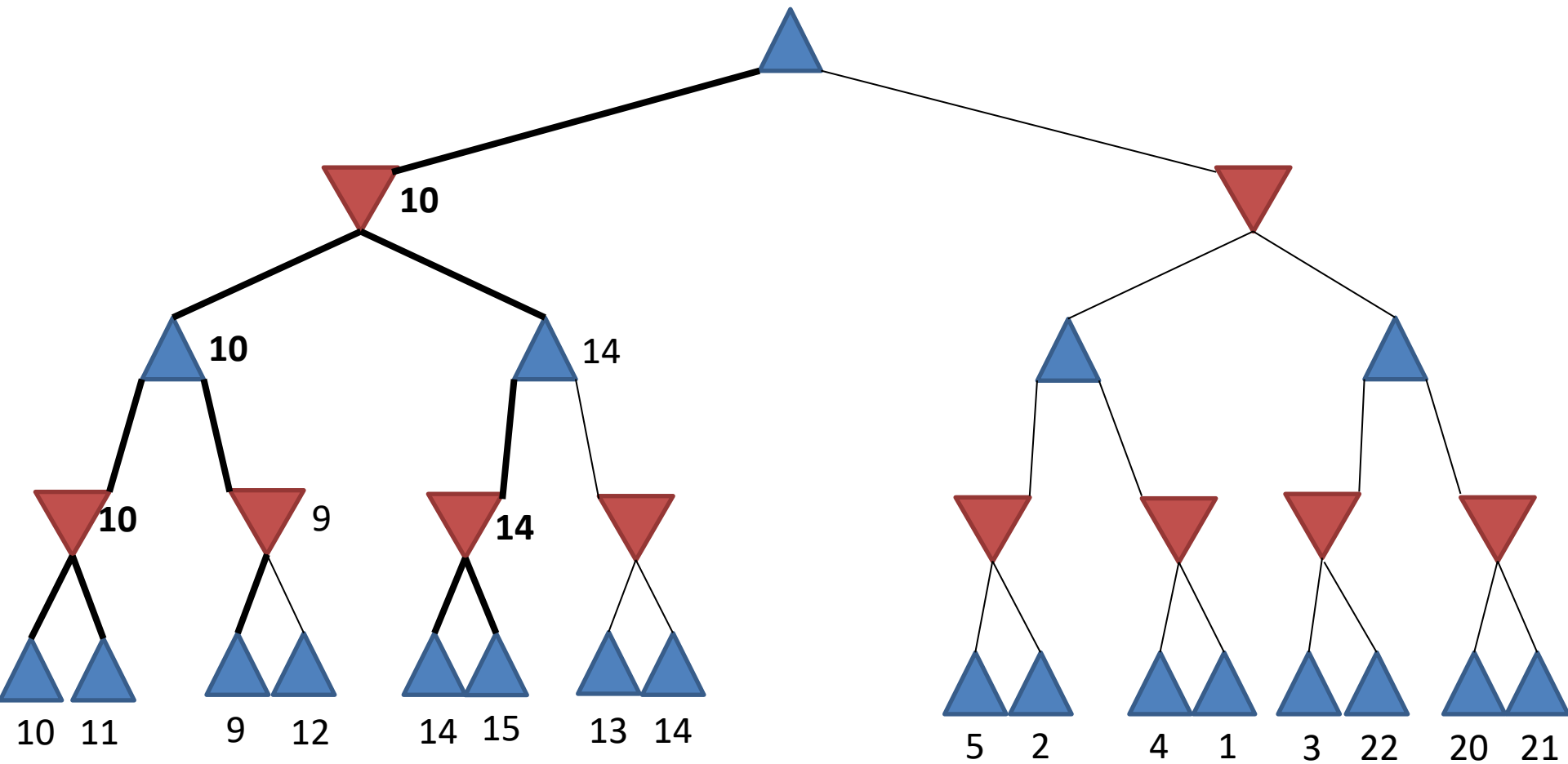
Παράδειγμα



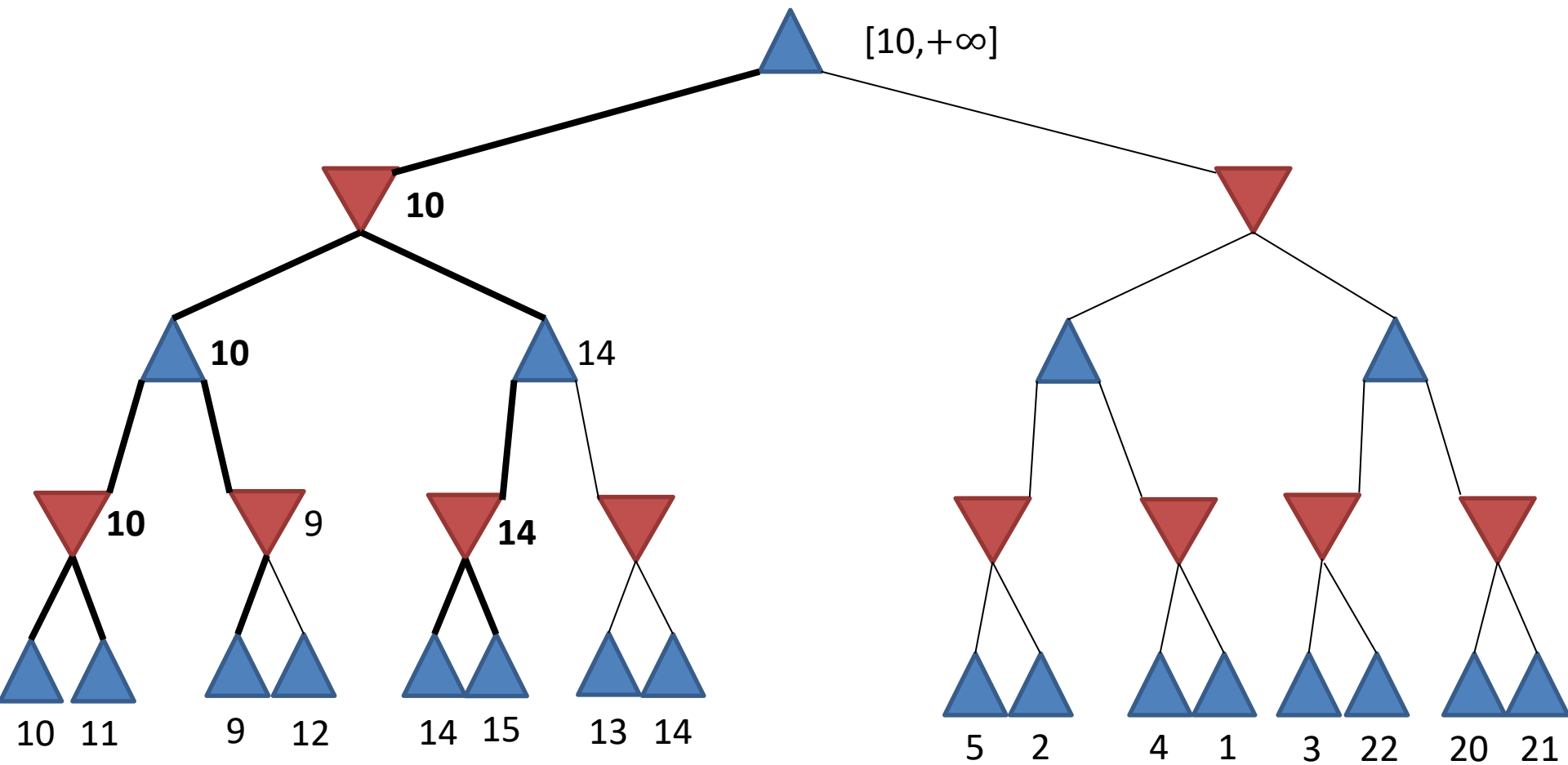
Παράδειγμα



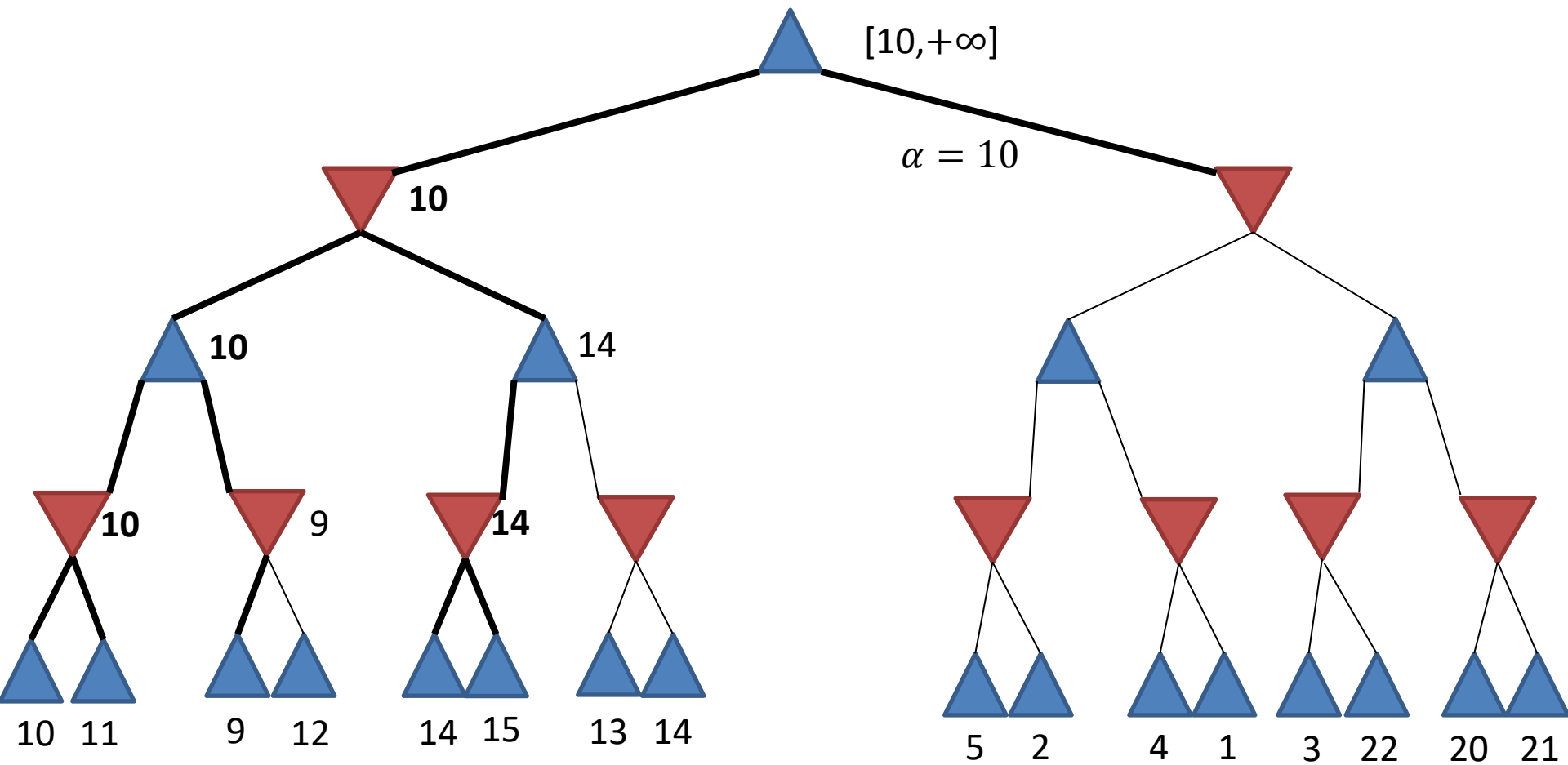
Παράδειγμα



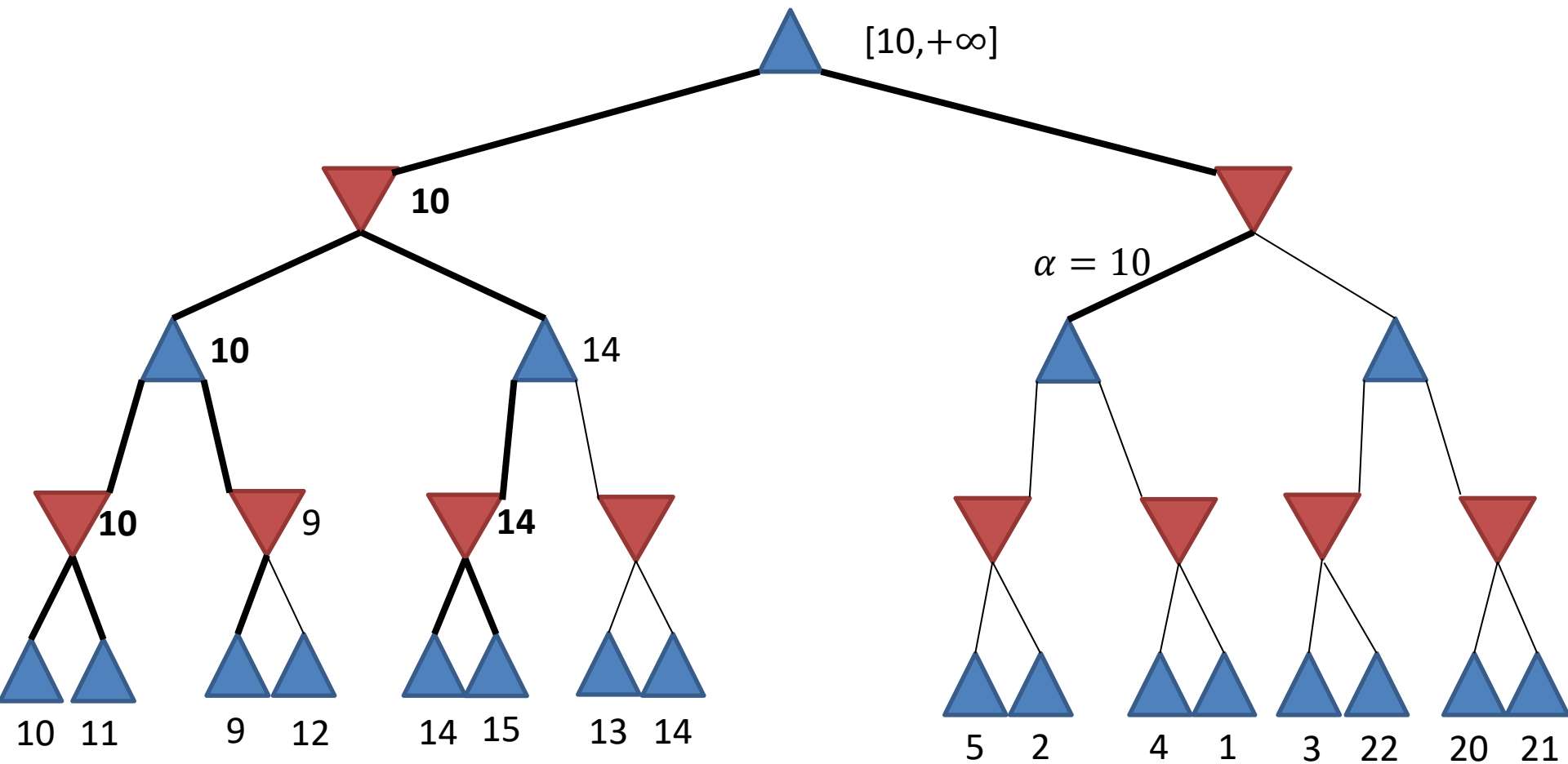
Παράδειγμα



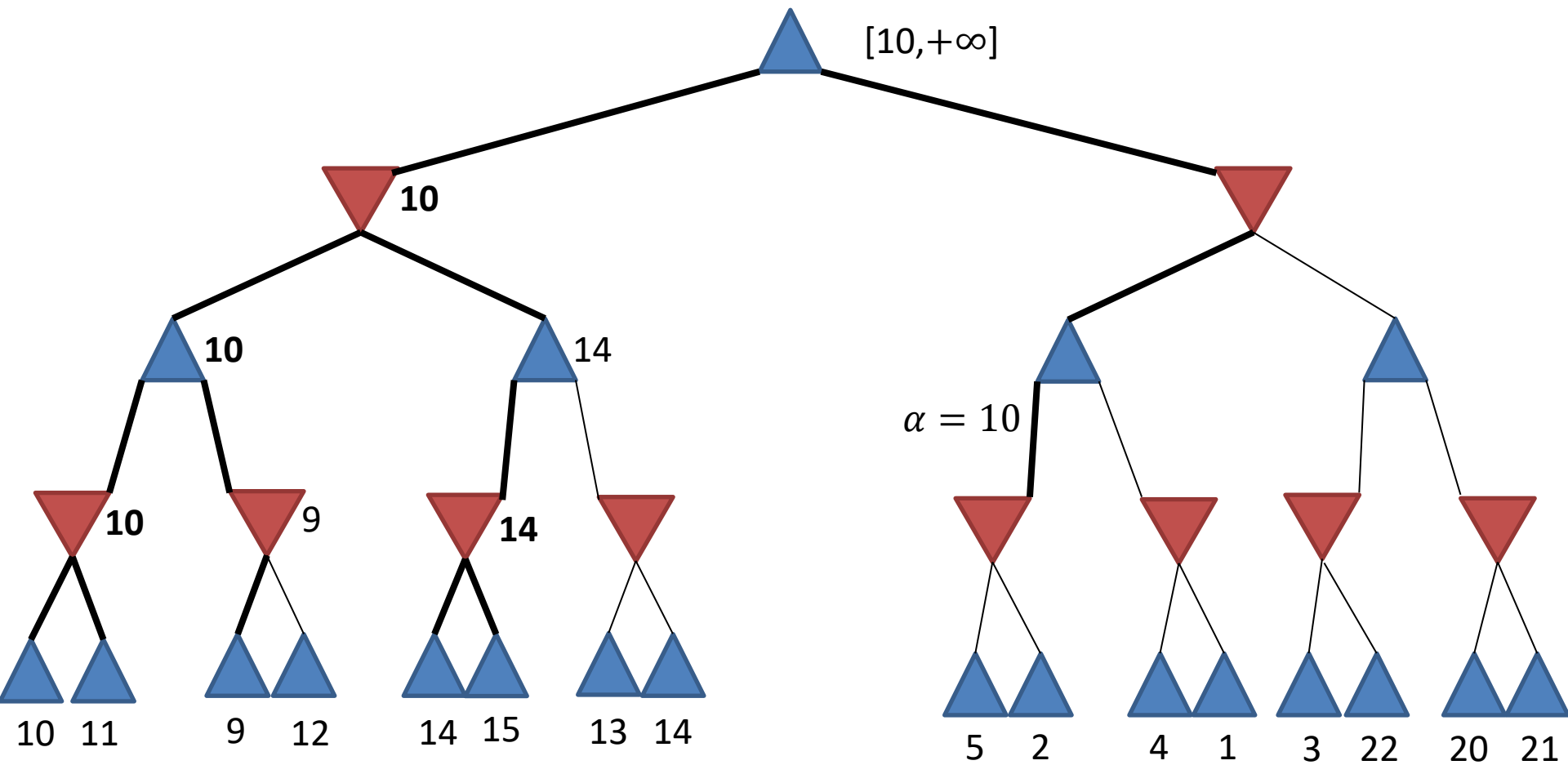
Παράδειγμα



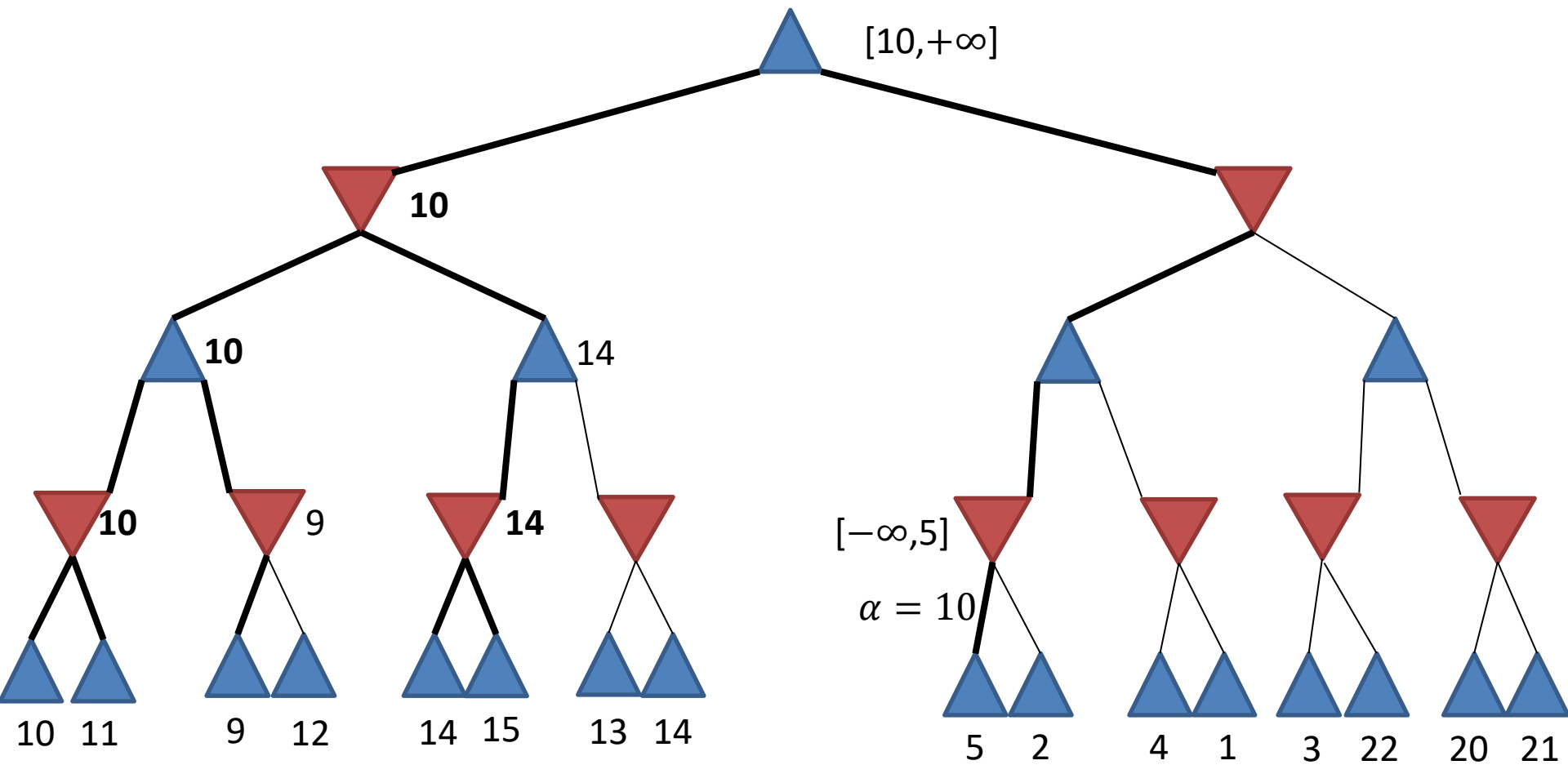
Παράδειγμα



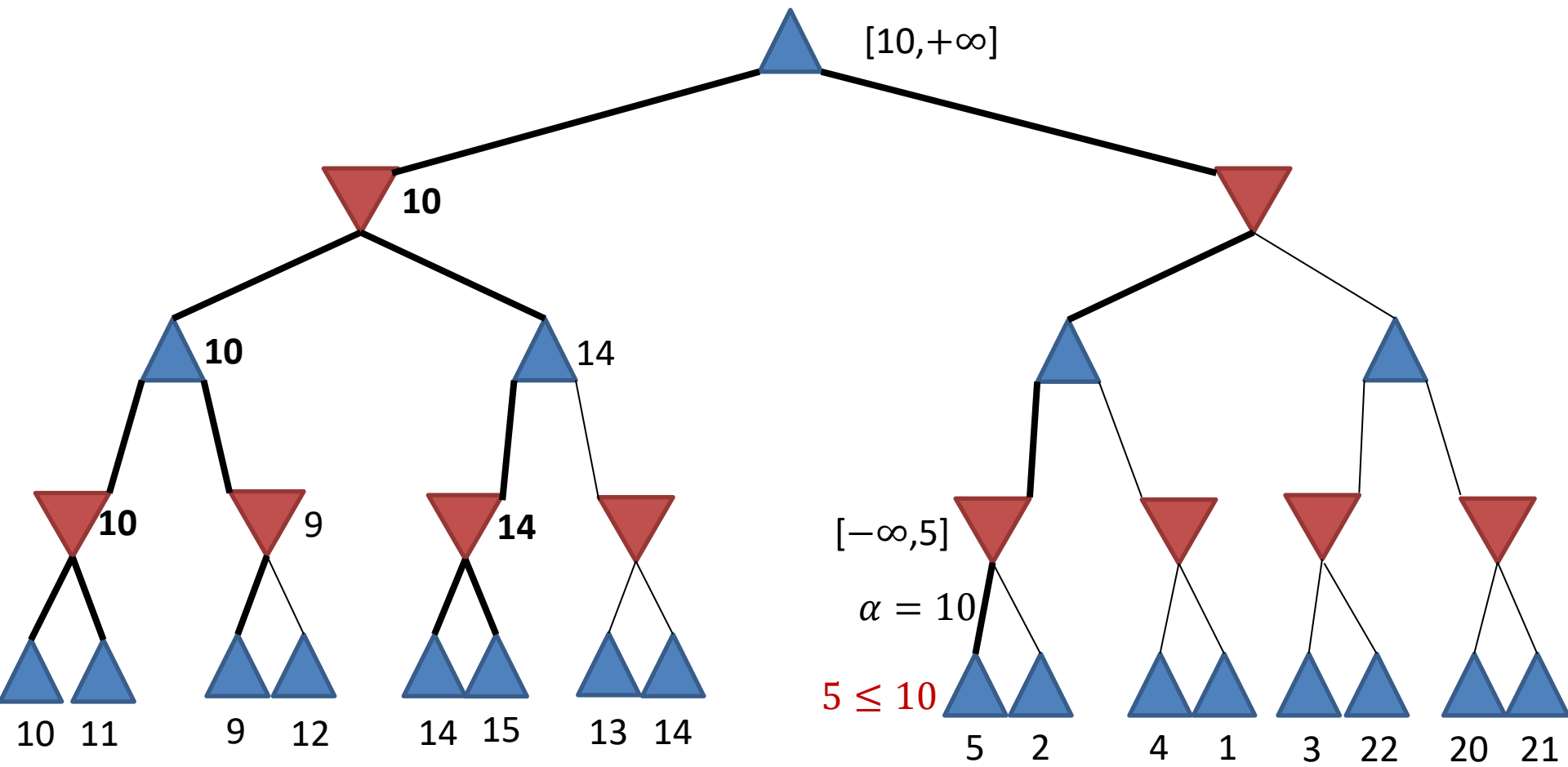
Παράδειγμα



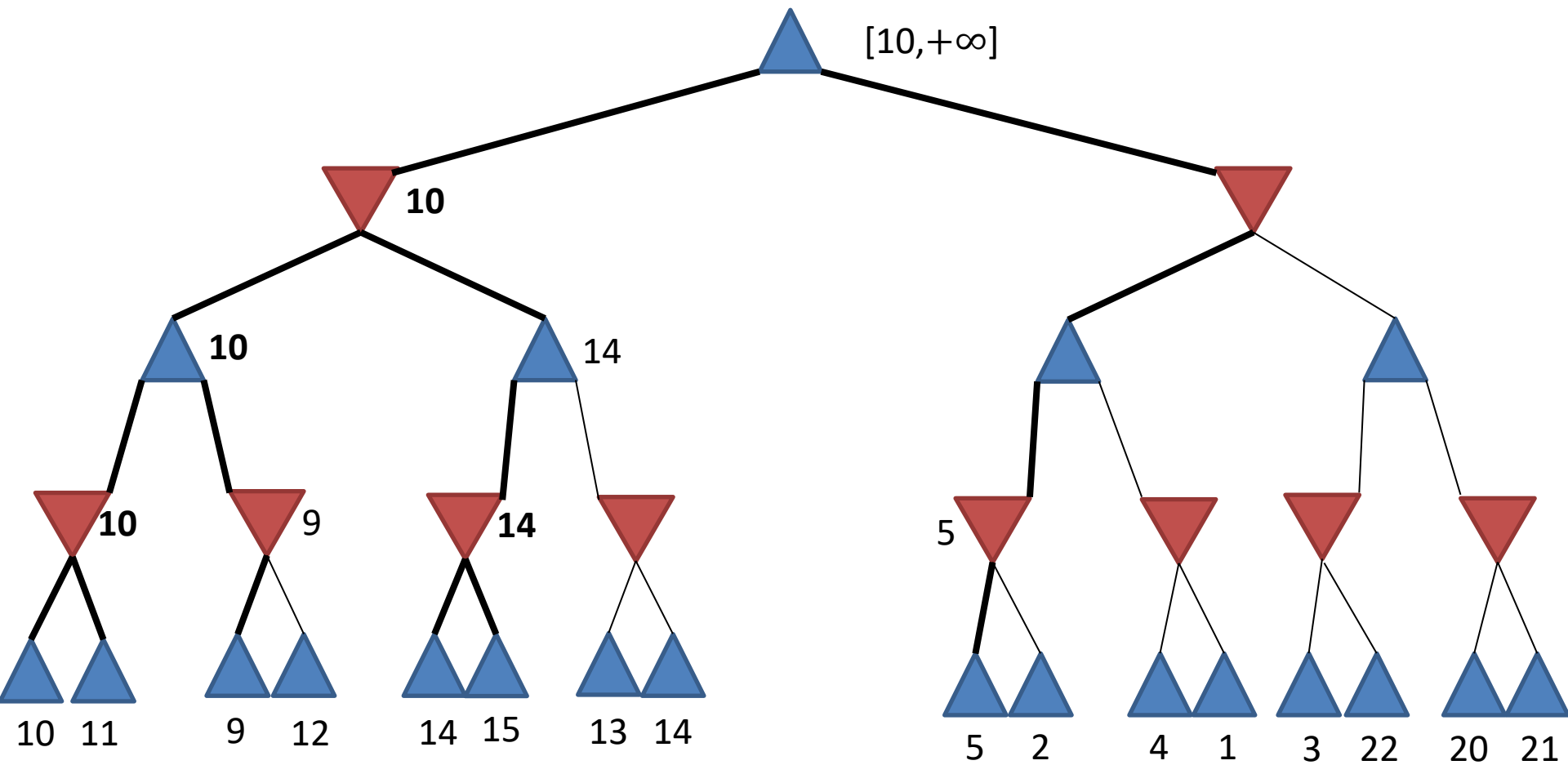
Παράδειγμα



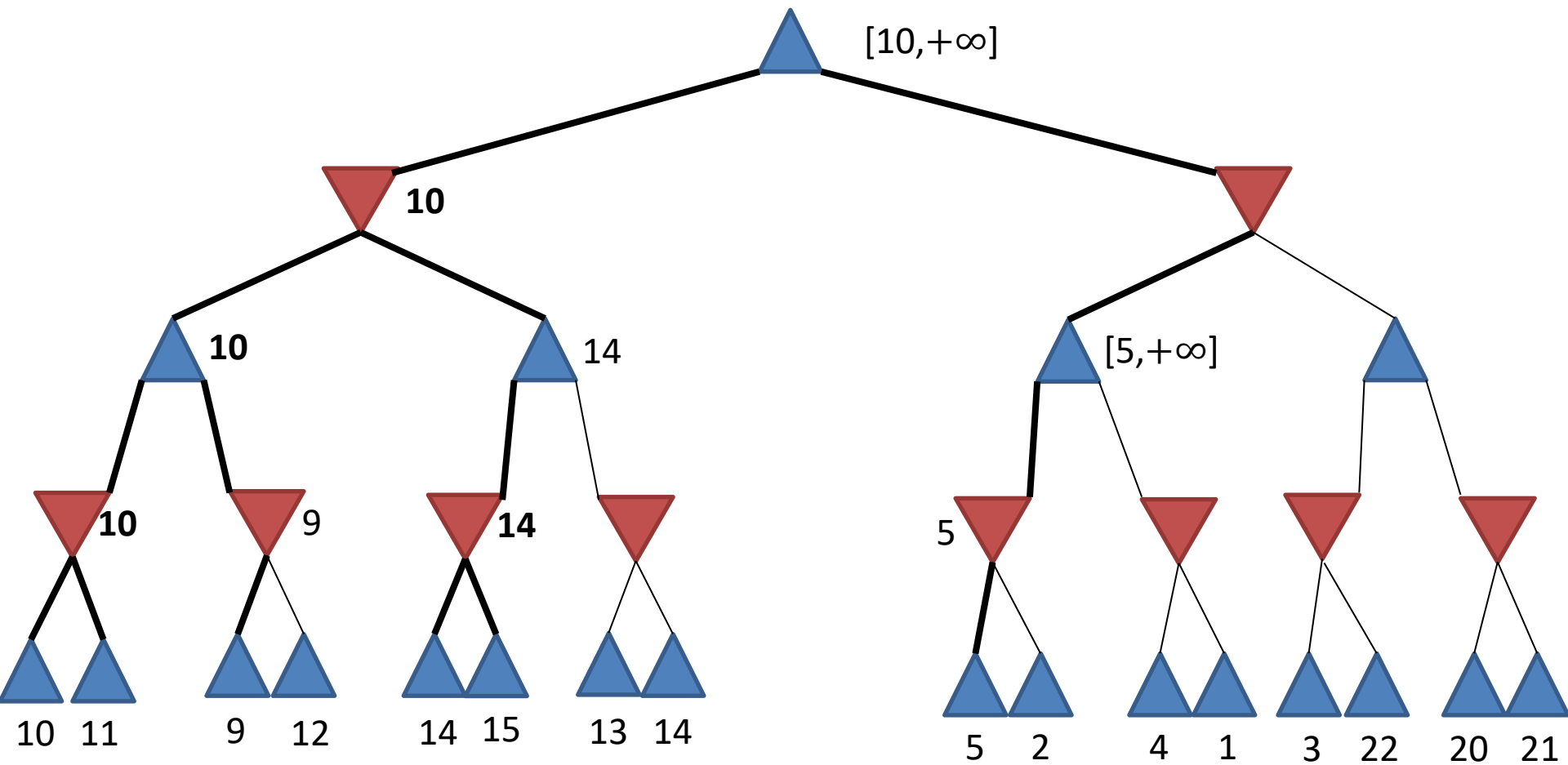
Παράδειγμα



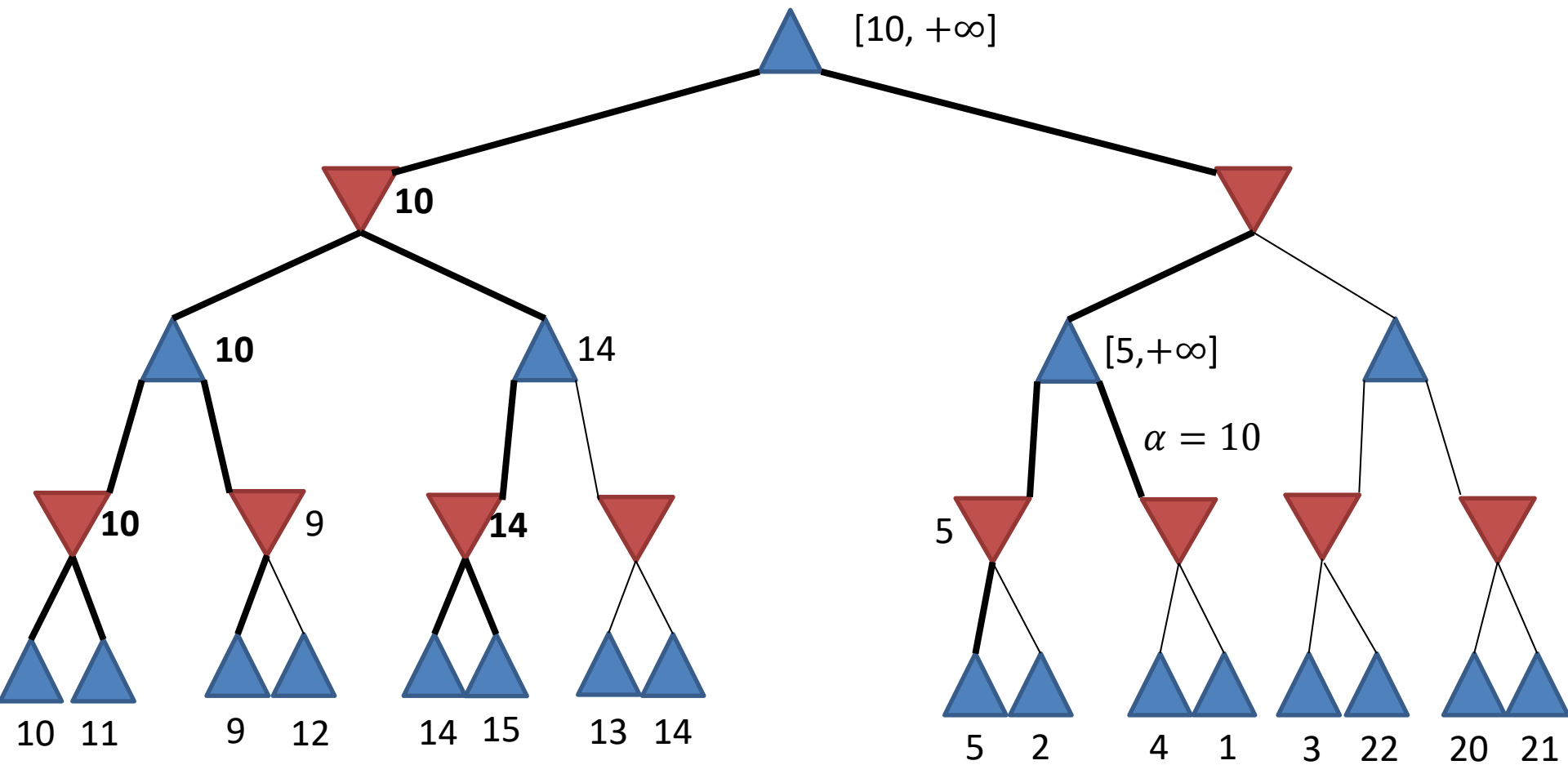
Παράδειγμα



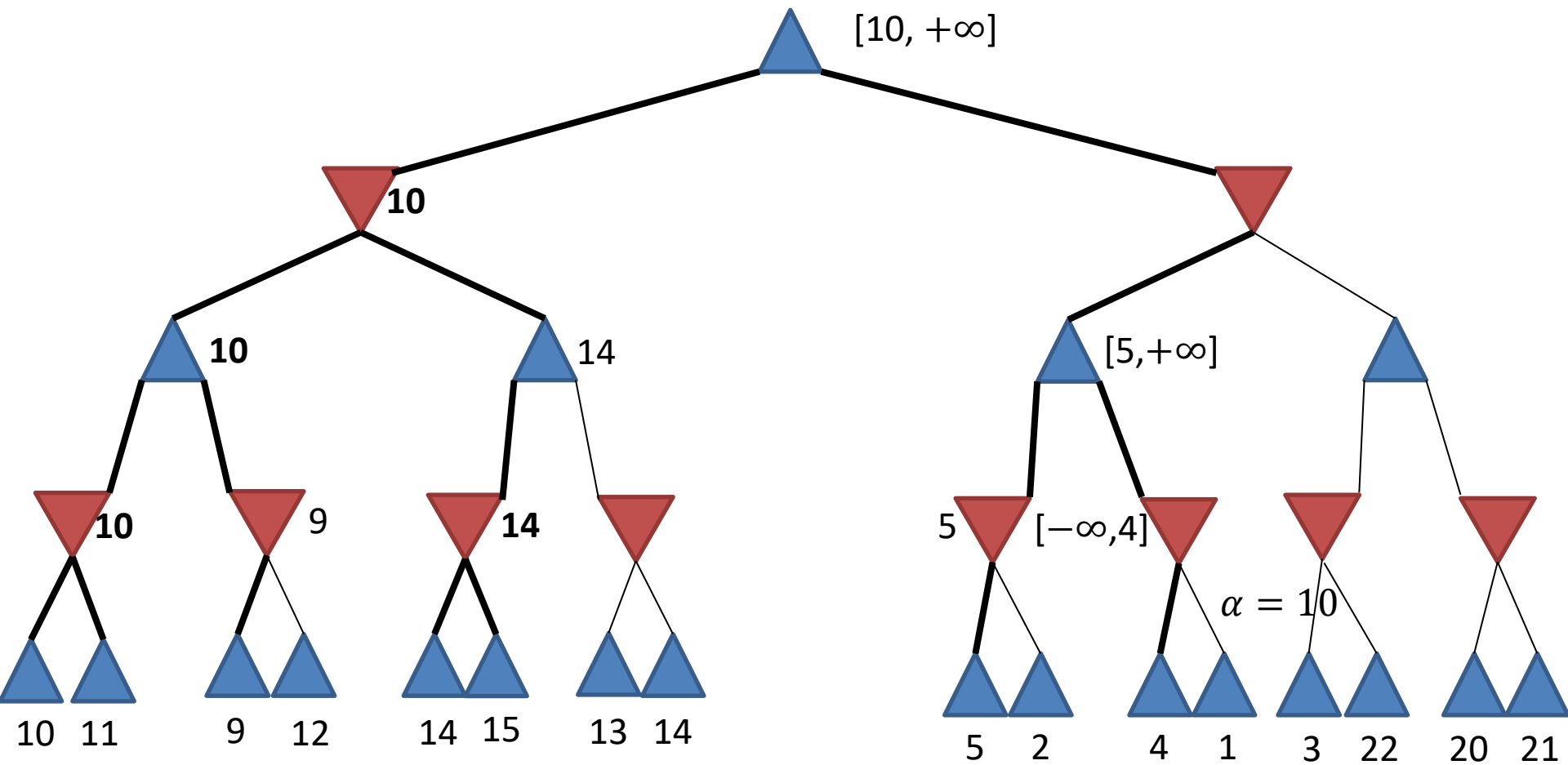
Παράδειγμα



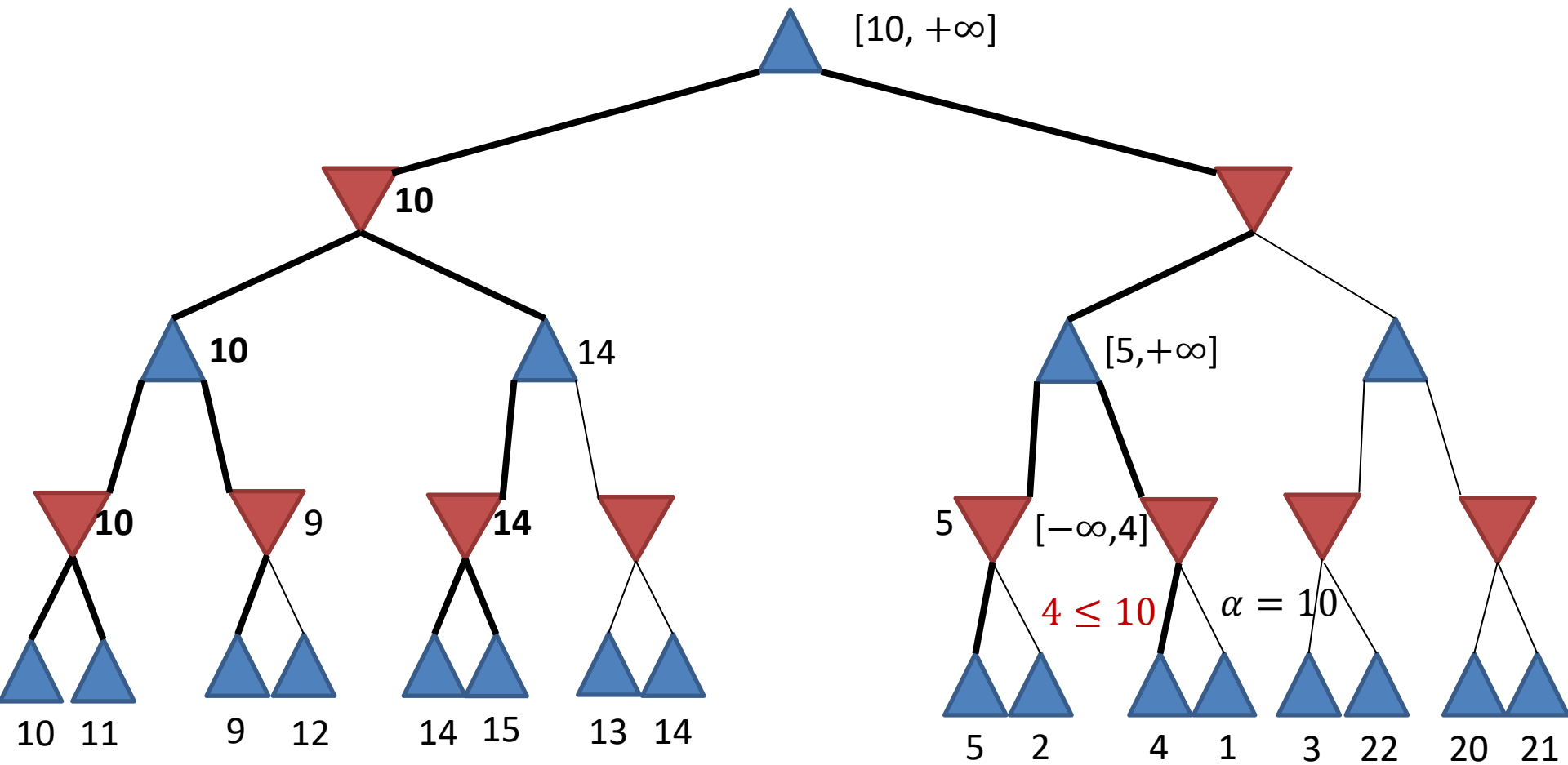
Παράδειγμα



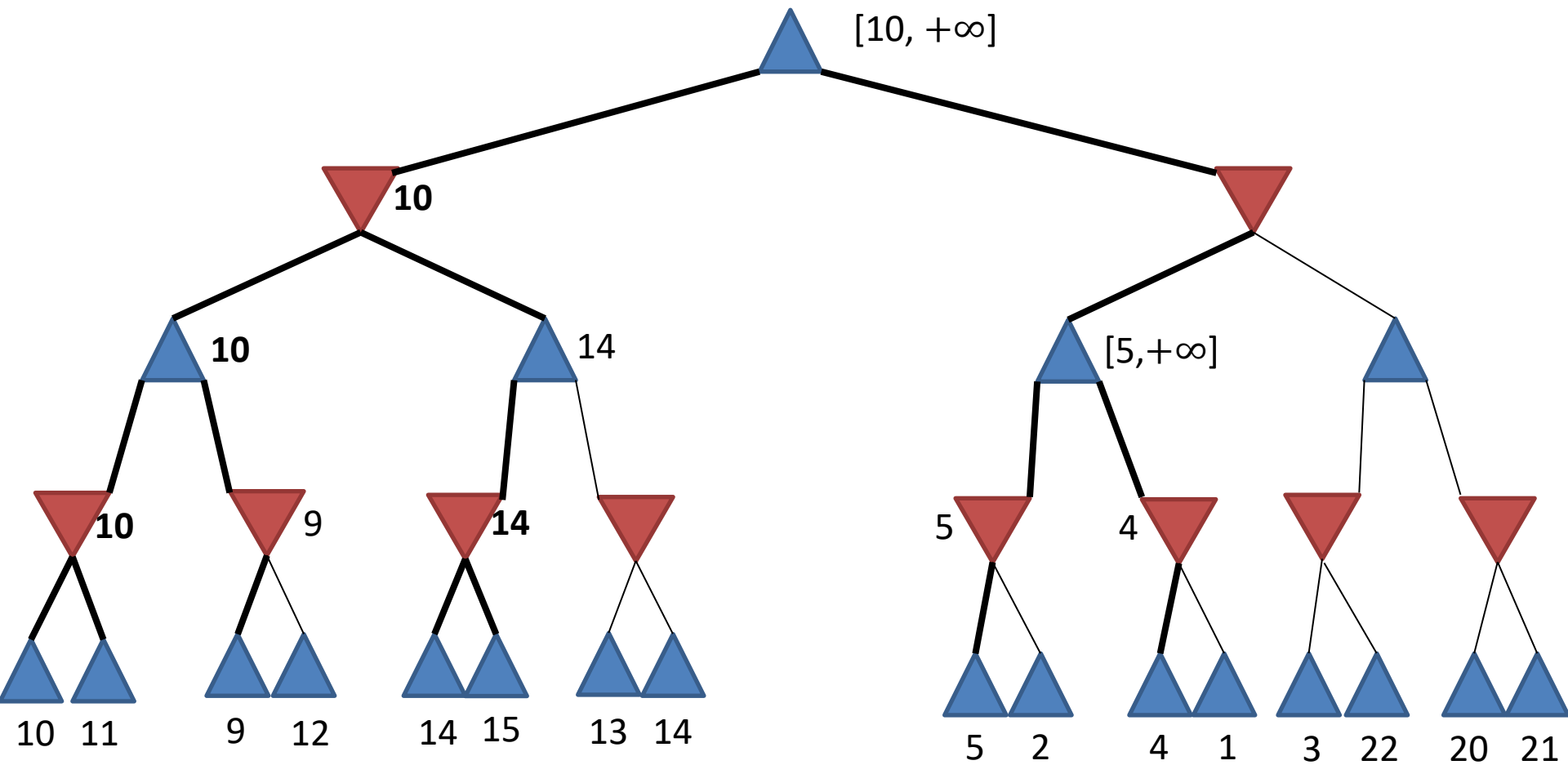
Παράδειγμα



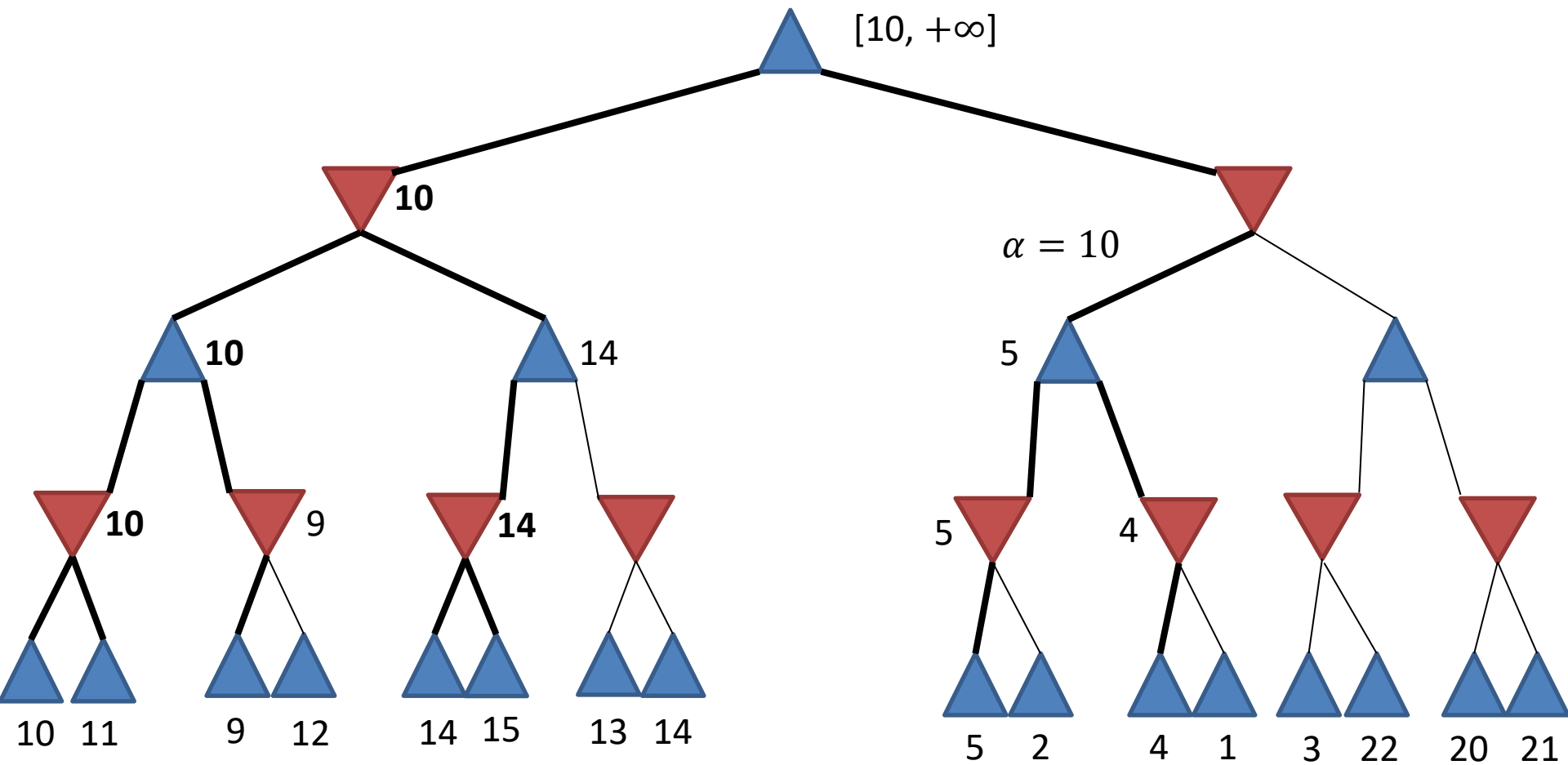
Παράδειγμα



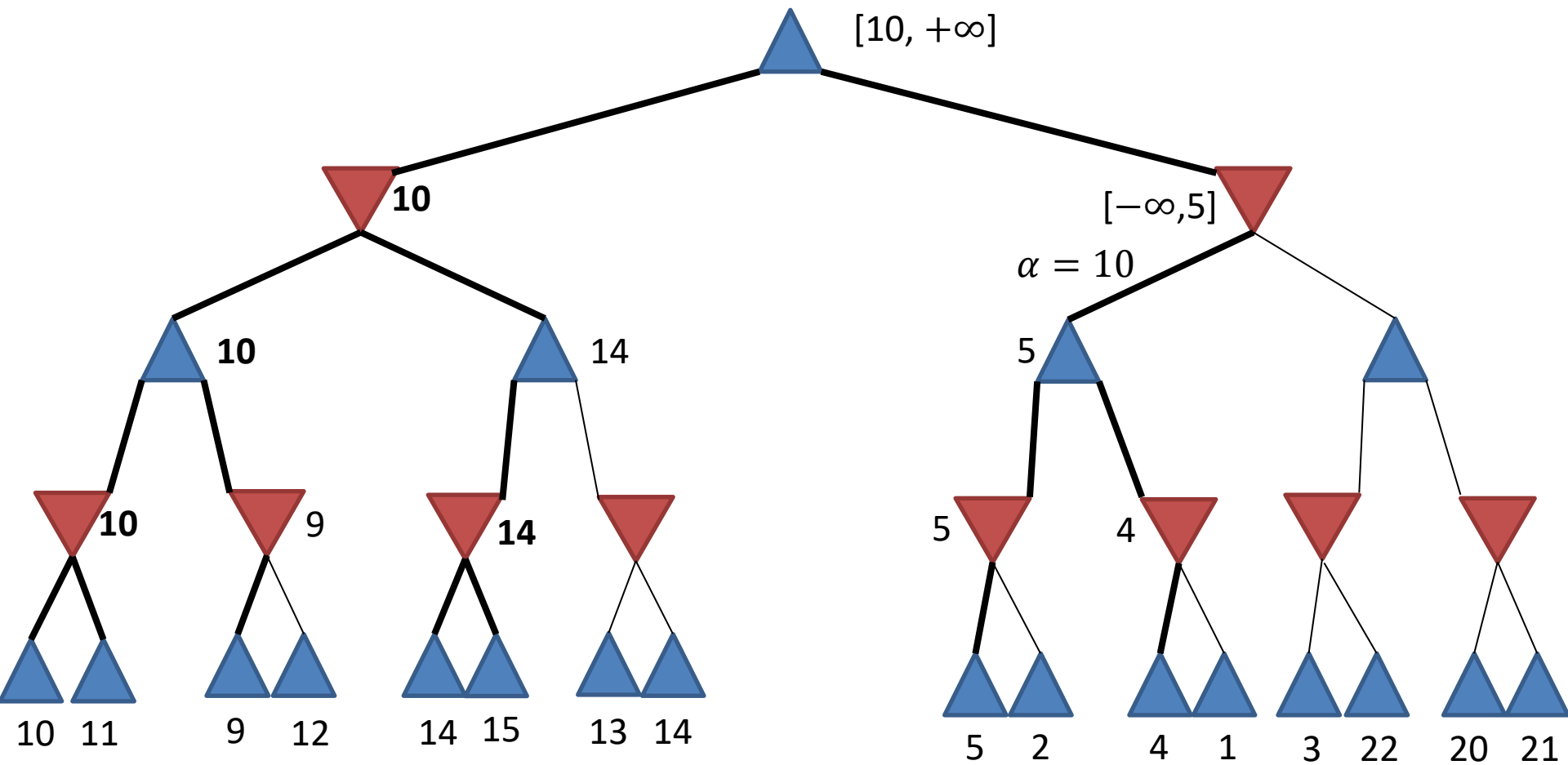
Παράδειγμα



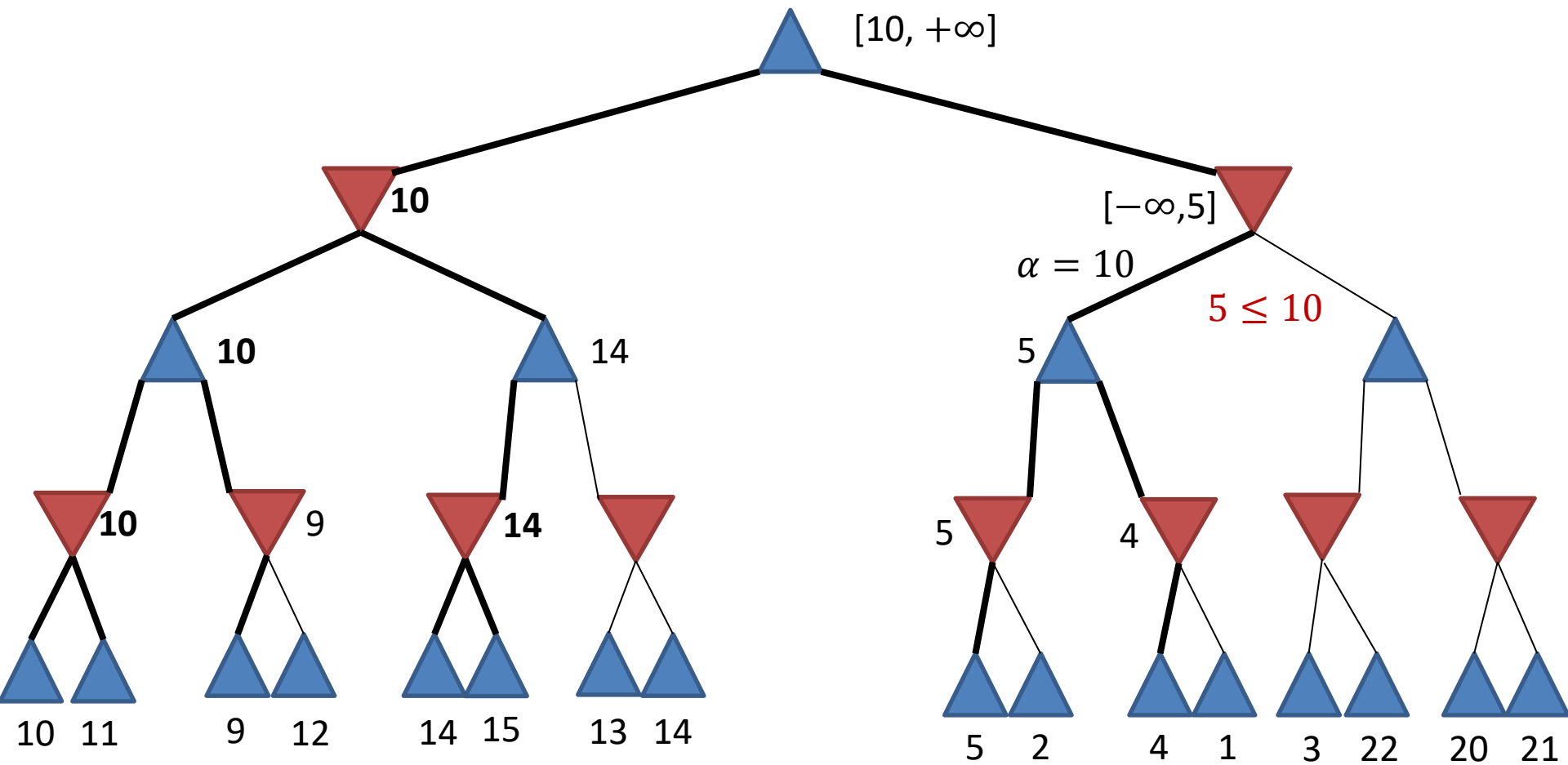
Παράδειγμα



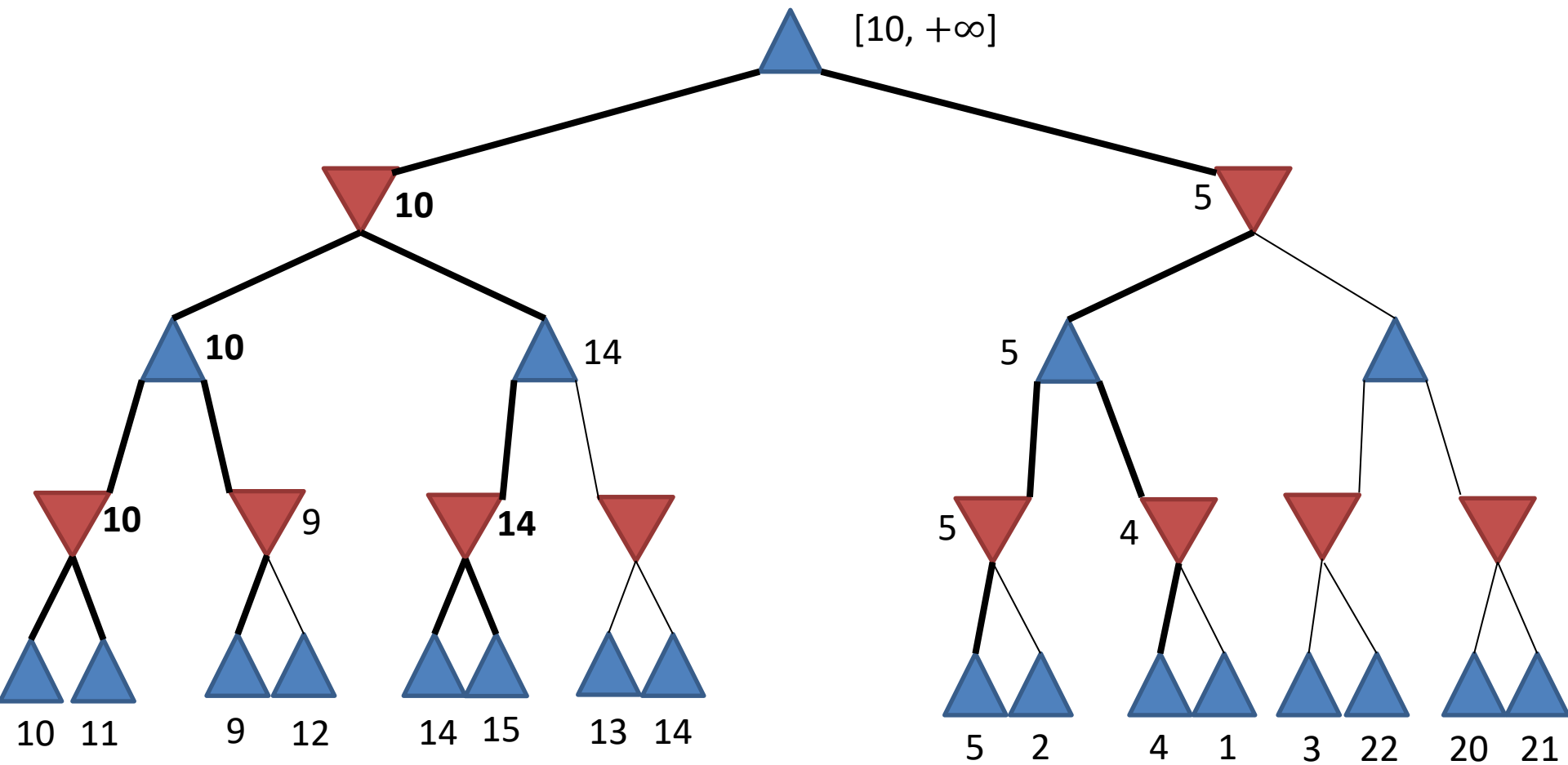
Παράδειγμα



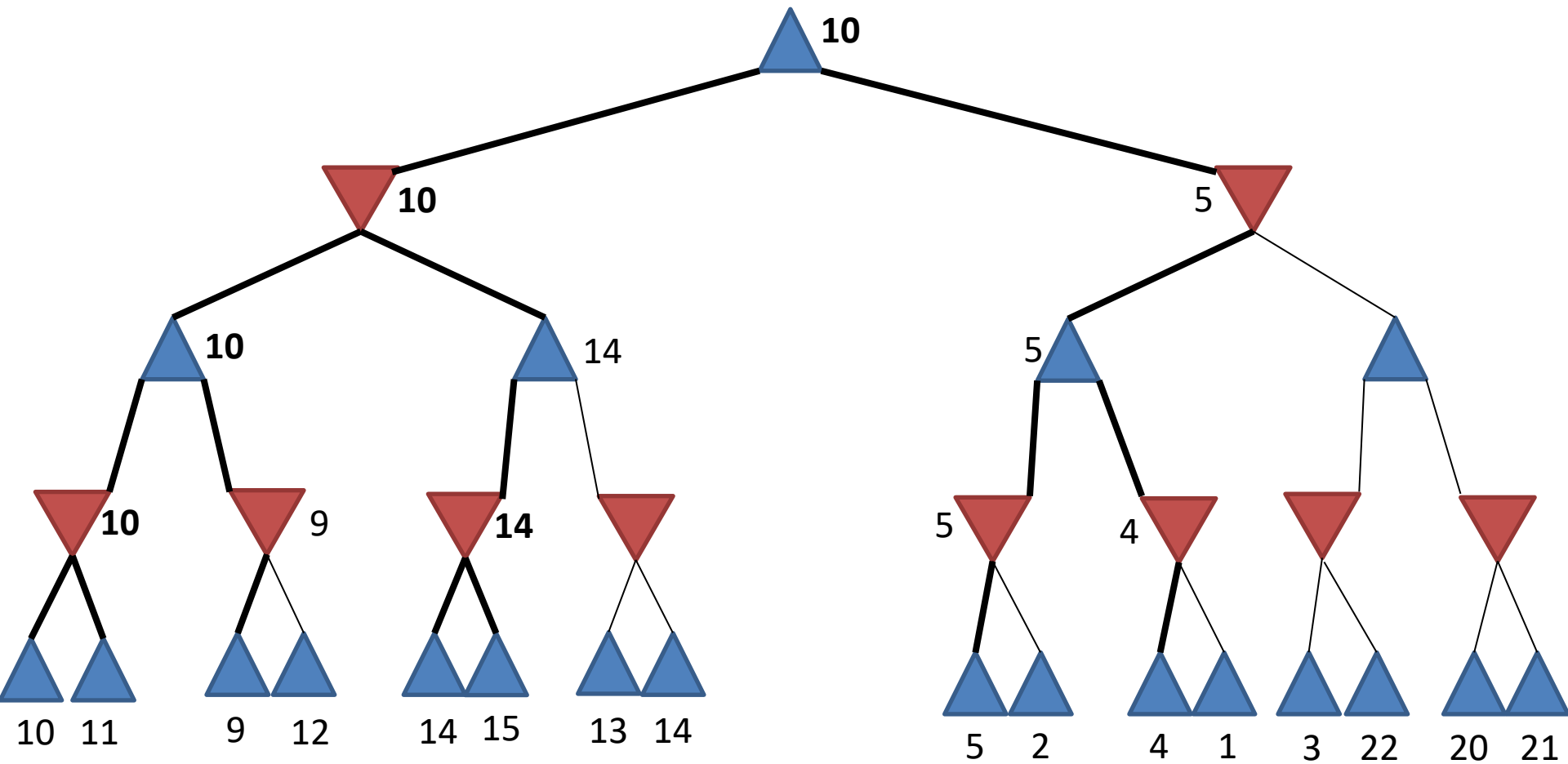
Παράδειγμα



Παράδειγμα



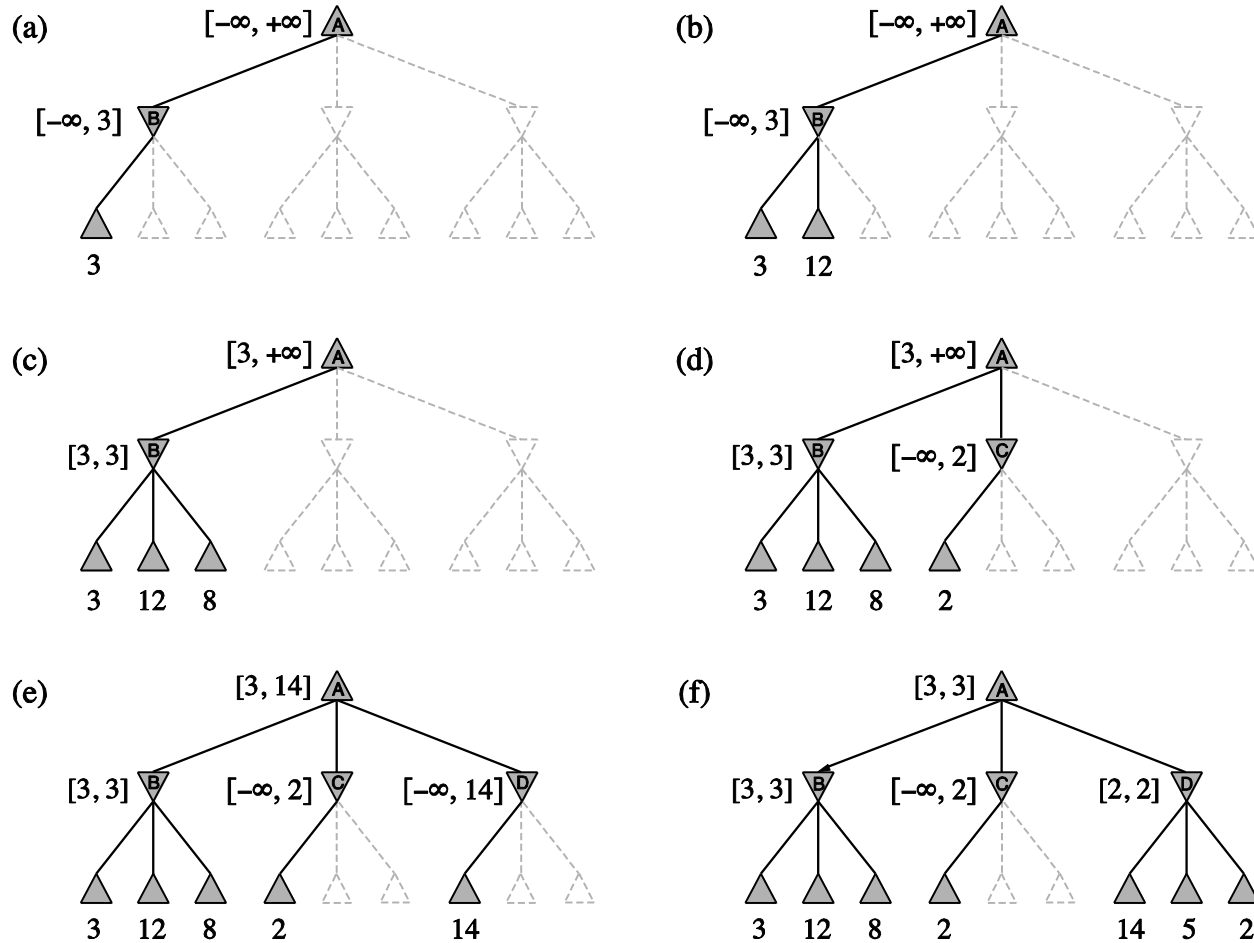
Παράδειγμα



Η Σειρά των Κινήσεων

- Η αποτελεσματικότητα του κλαδέματος άλφα-βήτα εξαρτάται από την **σειρά** με την οποία εξετάζονται οι καταστάσεις.
- Στο πρώτο παράδειγμα που παρουσιάσαμε, αν το φύλλο με τιμή 2 κάτω από τον κόμβο D εξετάζεται πρώτο, τότε μπορούμε να κλαδέψουμε τα άλλα δύο φύλλα.

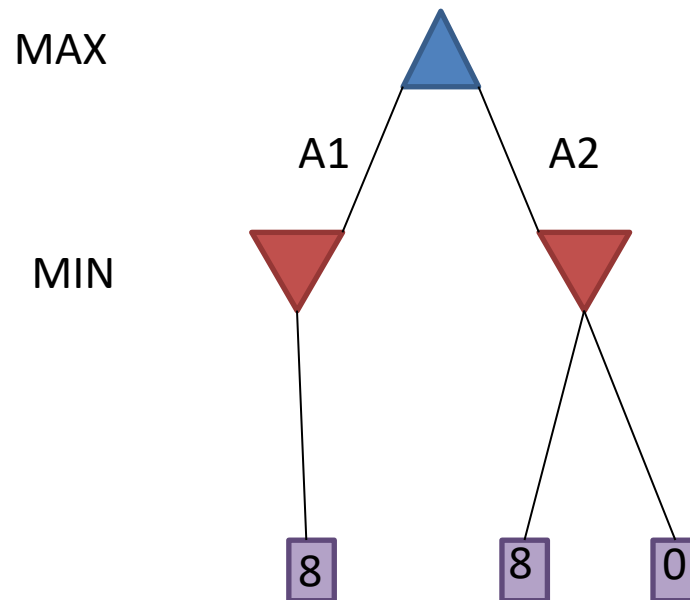
Παράδειγμα



Παρατηρήσεις

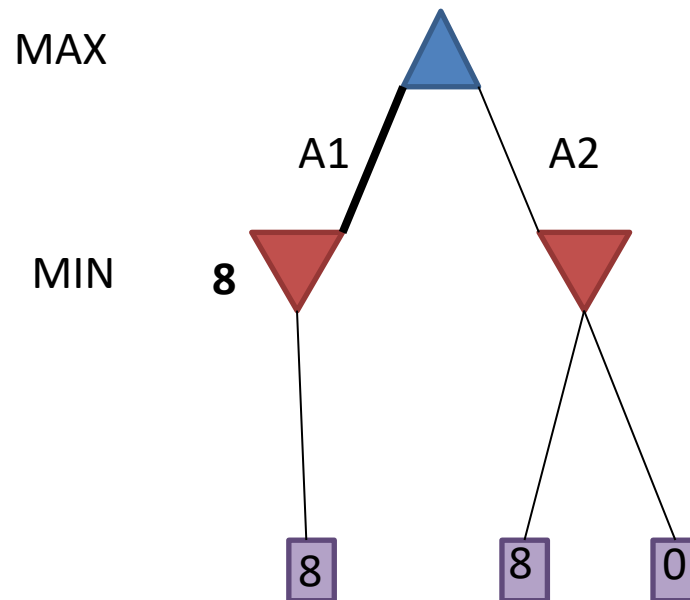
- Το κλάδεμα άλφα-βήτα δεν έχει **καμία επίπτωση** στην minimax τιμή της ρίζας η οποία υπολογίζεται σωστά.
- Οι τιμές στους ενδιάμεσους κόμβους μπορεί να είναι **λάθος!** (δηλαδή, να μην είναι οι minimax τιμές).
- Πρέπει να είμαστε προσεκτικοί αν θέλουμε να υπολογίσουμε την minimax απόφαση στη ρίζα (δηλαδή, την βέλτιστη κίνηση του MAX).

Παράδειγμα

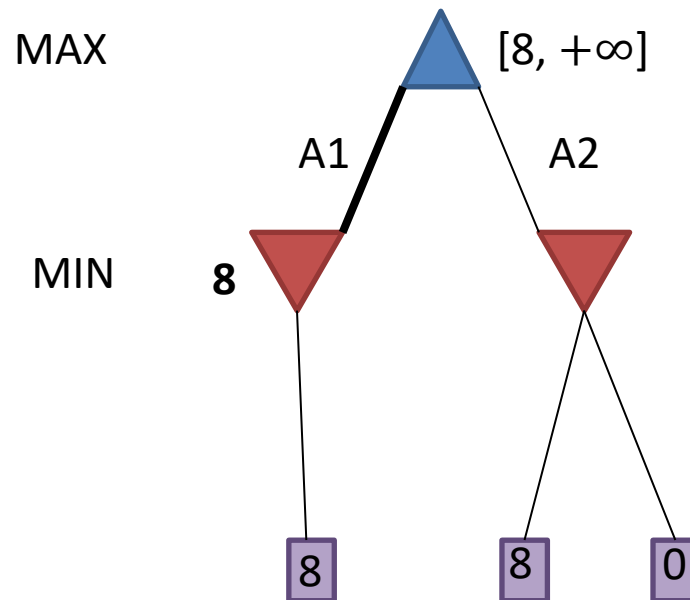


Ποια είναι η κίνηση που πρέπει να επιλέξει ο MAX;

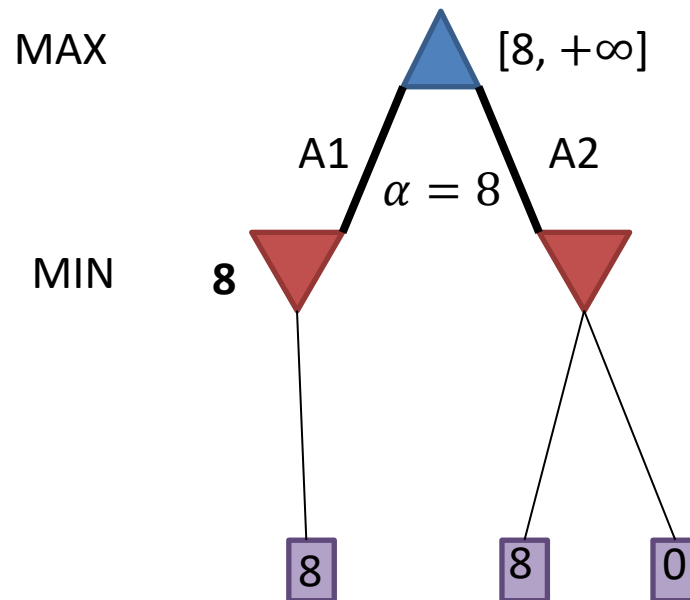
Εκτέλεση του ALPHA-BETA-SEARCH



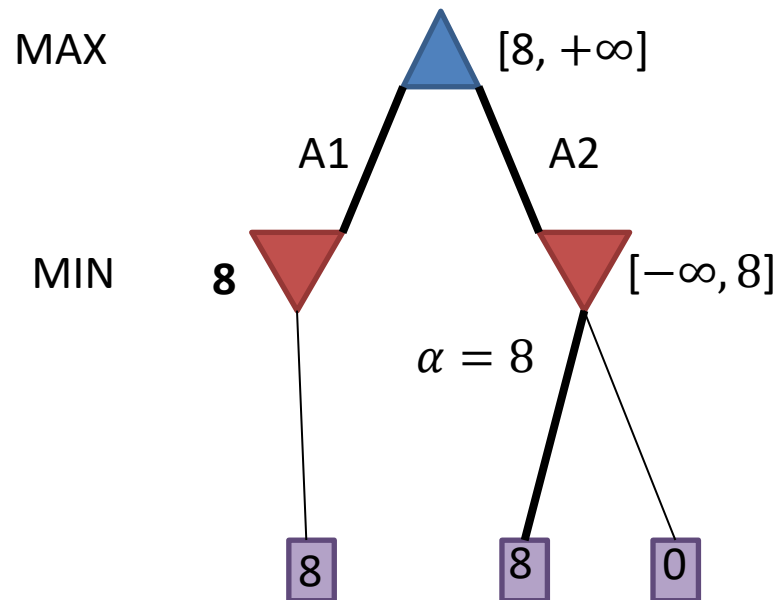
Εκτέλεση του ALPHA-BETA-SEARCH



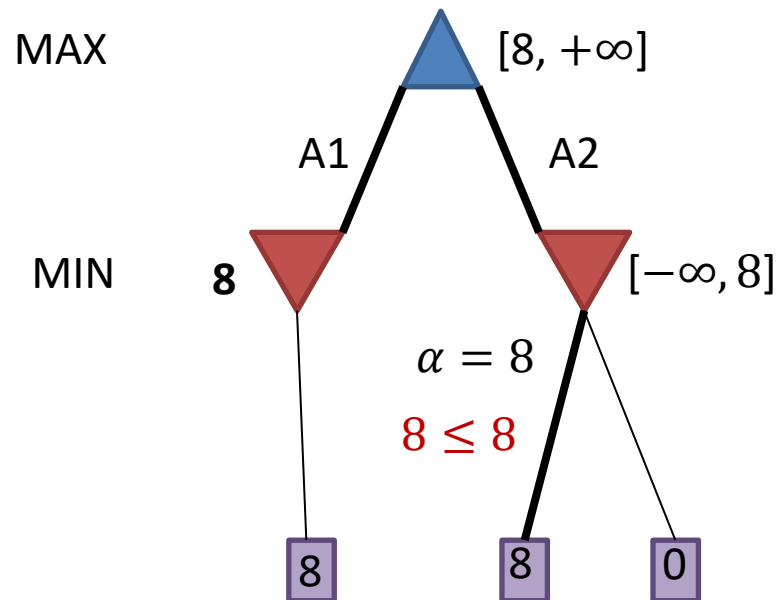
Εκτέλεση του ALPHA-BETA-SEARCH



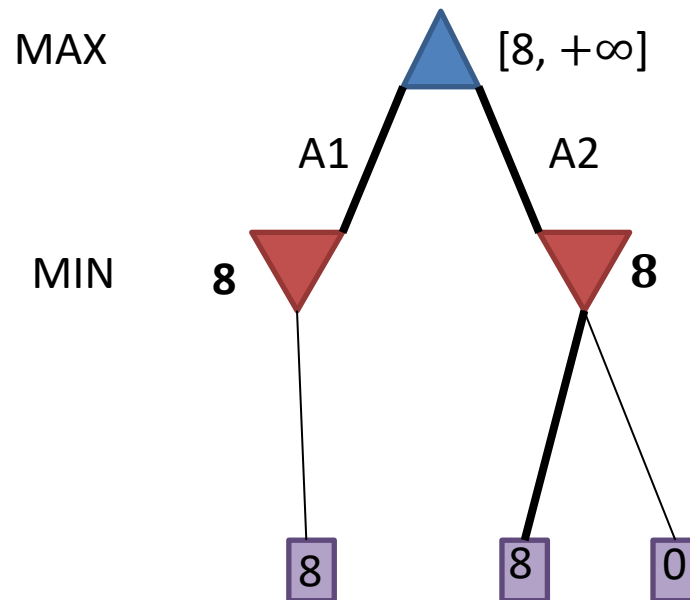
Εκτέλεση του ALPHA-BETA-SEARCH



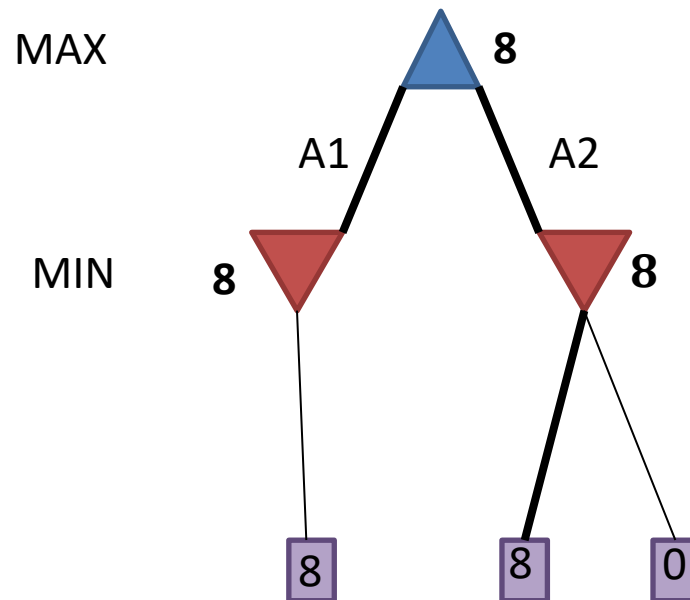
Εκτέλεση του ALPHA-BETA-SEARCH



Εκτέλεση του ALPHA-BETA-SEARCH



Εκτέλεση του ALPHA-BETA-SEARCH



Συζήτηση

- Αν τρέξουμε τον αλγόριθμο ALPHA-BETA-SEARCH, τότε οι τιμές που επιστρέφονται από τους MIN κόμβους είναι 8 και 8. Άρα, ο MAX μπορεί να επιλέξει οποιαδήποτε από τις A1 και A2. Όμως **μόνο η A1** είναι σωστή minimax απόφαση!
- Το πρόβλημα αυτό μπορεί να λυθεί αν οι εντολές κλαδέματος στις συναρτήσεις MAX-VALUE και MIN-VALUE χρησιμοποιούν **γνήσιες ανισότητες** (δηλαδή $v > \beta$ και $v < \alpha$).

Υπολογιστική Πολυπλοκότητα

- Αν οι διάδοχες καταστάσεις **εξετάζονται με την καλύτερη δυνατή σειρά**, ο αλγόριθμος αναζήτησης άλφα-βήτα μπορεί να εξετάσει μόνο $O(b^{\frac{m}{2}})$ κόμβους αντί για $O(b^m)$ που εξετάζει ο minimax.
- Αυτό σημαίνει ότι ο **δραστικός παράγοντας διακλάδωσης** γίνεται \sqrt{b} αντί για b . Για παράδειγμα, για το σκάκι 6 αντί 35.
- Με άλλα λόγια, ο αλγόριθμος αναζήτησης άλφα-βήτα μπορεί να επιλύσει ένα δένδρο με βάθος διπλάσιο του minimax στον ίδιο χρόνο.
- Αν οι διάδοχες καταστάσεις εξετάζονται με τυχαία σειρά, ο συνολικός αριθμός κόμβων που χρειάζεται να εξεταστούν είναι περίπου $O(b^{\frac{3m}{4}})$.
- Στο σκάκι, μια αρκετά απλή διάταξη των κινήσεων (πρώτα τα παρσίματα, μετά οι απειλές, μετά κινήσεις προς τα εμπρός, μετά κινήσεις προς τα πίσω) μας φέρνει πολύ κοντά στο φράγμα της καλύτερης διάταξης $O(b^{\frac{m}{2}})$.



Δυναμική Διάταξη Κινήσεων

- Αν δοκιμάσουμε πρώτα τις κινήσεις που βρήκαμε να είναι καλύτερες στο παρελθόν, τότε μπορούμε να έλθουμε πολύ κοντά στο θεωρητικό όριο $O(b^{\frac{m}{2}})$.
- Η καλύτερη κίνηση μπορεί να είναι η προηγούμενη κίνηση που κάναμε (συχνά οι ίδιες απειλές παραμένουν) ή μπορεί να βρεθεί με εξερεύνηση που ξεκινάει από την τωρινή κίνηση.
- Ένας τέτοιος τρόπος εξερεύνησης είναι να κάνουμε **αναζήτηση με επαναληπτική εκβάθυνση**.
- Οι καλύτερες κινήσεις συχνά λέγονται **φονικές (killer moves)** και η χρησιμοποίησή τους ονομάζεται ο **ευρετικός κανόνας των φονικών κινήσεων**.

Επαναλαμβανόμενες Καταστάσεις

- Σε πολλά παιχνίδια, συχνά εμφανίζονται **επαναλαμβανόμενες καταστάσεις** εξαιτίας της ύπαρξης αντιμεταθέσεων της ακολουθίας κινήσεων που μας φέρνουν στην ίδια κατάσταση (αυτές οι αντιμεταθέσεις ονομάζονται **transpositions**).
- Μπορούμε να χρησιμοποιήσουμε ένα **πίνακα κατακερματισμού (hash table)** για να αποθηκεύσουμε τις καταστάσεις που συναντούμε και τις τιμές χρησιμότητας τους ώστε να μην χρειάζεται να τις επαναλάβουμε. Ο πίνακας αυτός ονομάζεται **transposition table**.

Ατελείς Αποφάσεις σε Πραγματικό Χρόνο

- Ο αλγόριθμος minimax παράγει όλο το χώρο αναζήτησης του παιχνιδιού ενώ ο αλγόριθμος άλφα-βήτα μας επιτρέπει να κλαδέψουμε ένα μεγάλο μέρος του.
- Όμως ακόμα και ο αλγόριθμος άλφα-βήτα πρέπει να ψάξει σε όλη τη διαδρομή μέχρι τις τερματικές καταστάσεις, για ένα μέρος του χώρου αναζήτησης.
- Για πολλά παιχνίδια, η αναζήτηση αυτή δεν είναι πρακτική επειδή **οι κινήσεις πρέπει να γίνονται σε λογικό χρονικό διάστημα** (π.χ., μερικά λεπτά).

Ατελείς Αποφάσεις σε Πραγματικό Χρόνο

- Ο Claude Shannon στο άρθρο του “Programming a Computer for Playing Chess” (1950) πρότεινε ότι τα προγράμματα πρέπει να εγκαταλείπουν την αναζήτηση νωρίτερα εφαρμόζοντας μια **συνάρτηση αξιολόγησης (evaluation function)** στις καταστάσεις αναζήτησης.
- Έτσι ο αλγόριθμος minimax ή άλφα-βήτα μετατρέπεται ως εξής: η συνάρτηση χρησιμότητας αντικαθίσταται από μια **συνάρτηση αξιολόγησης EVAL** η οποία δίνει μια εκτίμηση της χρησιμότητας της κατάστασης, και ο έλεγχος τερματισμού αντικαθίσταται από ένα **έλεγχο αποκοπής (cut-off test)** ο οποίος αποφασίζει πότε θα εφαρμόζεται η EVAL.

Ευρετικός Αλγόριθμος Minimax

- Έτσι ο **ευρετικός αλγόριθμος minimax** για κατάσταση s και μέγιστο βάθος d ορίζεται από την ακόλουθη εξίσωση:

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

Συναρτήσεις Αξιολόγησης

- Μια **συνάρτηση αξιολόγησης** επιστρέφει μια εκτίμηση της αναμενόμενης χρησιμότητας μιας κατάστασης όπως ακριβώς οι ευρετικές συναρτήσεις επιστρέφουν μια εκτίμηση της απόστασης από το στόχο.
- Στην πράξη, μπορούμε να θεωρήσουμε ότι οι παίκτες ενός παιχνιδιού (π.χ. σκάκι) χρησιμοποιούν τέτοιες συναρτήσεις αξιολόγησης ώστε να αξιολογήσουν τη χρησιμότητα μιας κατάστασης του παιχνιδιού επειδή οι άνθρωποι είναι πολύ λιγότερο ικανοί στην εξαντλητική αναζήτηση από τα προγράμματα υπολογιστών.



Πως Σχεδιάζουμε Συναρτήσεις Αξιολόγησης;

- Μια συνάρτηση αξιολόγησης θα πρέπει να έχει τα παρακάτω χαρακτηριστικά:
 - Να διατάσσει τις τερματικές καταστάσεις με τον ίδιο τρόπο που τις διατάσσει η πραγματική συνάρτηση χρησιμότητας.
 - Να μην χρειάζεται υπερβολικά πολύ χρόνο για να υπολογιστεί.
 - Για μια μη τερματική κατάσταση, να σχετίζεται στενά με τις πραγματικές πιθανότητες νίκης του παίκτη ξεκινώντας από αυτή την κατάσταση.

Συναρτήσεις Αξιολόγησης

- Οι περισσότερες συναρτήσεις αξιολόγησης λειτουργούν υπολογίζοντας διάφορα **χαρακτηριστικά (features)** της κατάστασης, για παράδειγμα στο σκάκι τον αριθμό των άσπρων πιονιών, μαύρων πιονιών, άσπρων βασιλισσών, μαύρων βασιλισσών κλπ.
- Τα χαρακτηριστικά αυτά θεωρούμενα μαζί ορίζουν διάφορες **κατηγορίες ή κλάσεις ισοδυναμίας καταστάσεων**. Οι καταστάσεις της κάθε κατηγορίας έχουν την ίδια τιμή για όλα τα χαρακτηριστικά.
- Οποιαδήποτε δεδομένη κατηγορία θα περιέχει μερικές καταστάσεις που οδηγούν σε νίκη, μερικές καταστάσεις που οδηγούν σε ισοπαλία και μερικές καταστάσεις που οδηγούν σε ήττα.
- Η συνάρτηση αξιολόγησης δεν μπορεί να γνωρίζει ποιες καταστάσεις οδηγούν στο κάθε αποτέλεσμα, μπορεί όμως να επιστρέφει μια απλή τιμή που αντανακλά την αναλογία των καταστάσεων που οδηγούν στο κάθε αποτέλεσμα.
- Για παράδειγμα, έστω ότι η πείρα υποδεικνύει ότι το 72% των καταστάσεων που συναντάμε σε μια κατηγορία οδηγούν σε νίκη (χρησιμότητα +1), το 20% σε ήττα (χρησιμότητα 0) και το 8% σε ισοπαλία (χρησιμότητα $\frac{1}{2}$). Τότε μια λογική επιλογή για την αξιολόγηση των καταστάσεων της κατηγορίας είναι ο σταθμισμένος μέσος όρος $(0,72 \times (+1)) + (0,20 \times 0) + (0,08 \times \frac{1}{2}) = 0.76$. Αυτή είναι η **αναμενόμενη τιμή χρησιμότητας**.
- Στην πράξη, αυτό το είδος ανάλυσης απαιτεί πολλές κατηγορίες και επομένως πολλή μεγάλη εμπειρία για να εκτιμηθούν όλες οι πιθανότητες νίκης.



Συναρτήσεις Αξιολόγησης

- Αντί γι' αυτό, οι περισσότερες συναρτήσεις αξιολόγησης υπολογίζουν **ξεχωριστές αριθμητικές συνεισφορές για κάθε χαρακτηριστικό** και μετά τις συνδυάζουν για να βρουν μια εκτίμηση της κατάστασης.
- Για παράδειγμα, τα εισαγωγικά βιβλία στο σκάκι δίνουν μια προσεγγιστική αξία υλικού για το κάθε κομμάτι: το κάθε πiónι αξίζει 1, ένας αξιωματικός αξίζει 3, ένας πύργος 5 και η βασίλισσα 9.
- Οπότε η συνάρτηση αξιολόγησης μπορεί να εκφραστεί με μια **σταθμισμένη γραμμική συνάρτηση** της μορφής:

$$\text{EVAL}(s) = w_1 f_1(s) + w_2 f_2(s) + \dots + w_n f_n(s) = \sum_{i=1}^n w_i f_i(s)$$

όπου κάθε w_i είναι βάρος και κάθε f_i χαρακτηριστικό της κατάστασης.

- Για το σκάκι, τα f_i μπορούν να είναι οι αριθμοί κομματιών κάθε είδους που υπάρχουν στη σκακιέρα και τα w_i οι αξίες των κομματιών.



Συναρτήσεις Αξιολόγησης

- Η παραπάνω γραμμική συνάρτηση περιέχει την παραδοχή ότι η συνεισφορά κάθε χαρακτηριστικού είναι ανεξάρτητη από τις τιμές των άλλων χαρακτηριστικών.
- Αυτό προφανώς δεν ισχύει, άρα στην πράξη μπορεί να χρειαζόμαστε **μη γραμμικούς συνδυασμούς** των χαρακτηριστικών.
- Για παράδειγμα, στο σκάκι ένα ζευγάρι αξιωματικών μπορεί να αξίζει λίγο περισσότερο από το διπλάσιο της αξίας ενός αξιωματικού, και ένας αξιωματικός να αξίζει περισσότερο στην τελική φάση του παιχνιδιού από ότι στην αρχή.



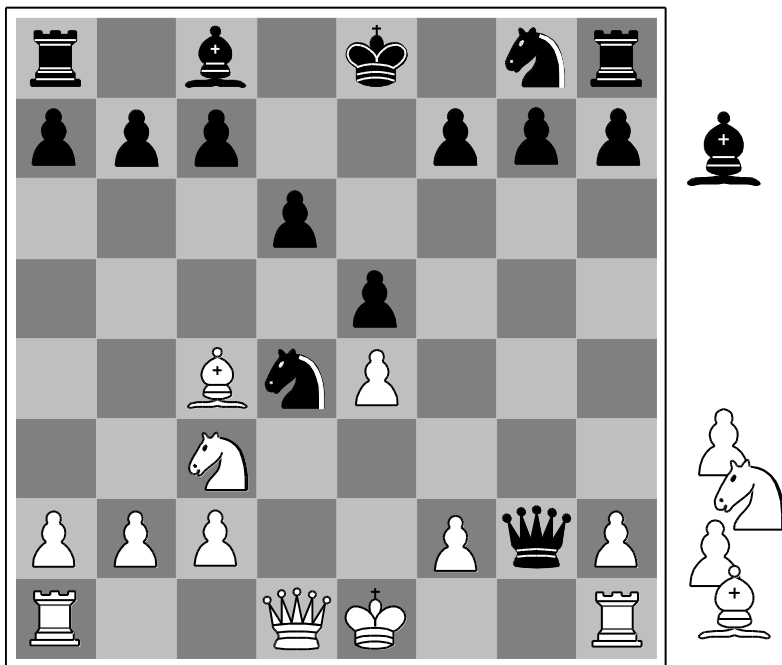
Αποκοπή Αναζήτησης

- Μπορούμε να τροποποιήσουμε τον αλγόριθμο ALPHA-BETA-SEARCH ώστε να καλεί τη συνάρτηση EVAL όταν πρέπει να αποκοπεί η αναζήτηση.
- Στον κώδικα που δώσαμε, αλλάζουμε τη γραμμή που περιέχει τον έλεγχο TERMINAL-TEST ως εξής:
 If CUTOFF-TEST(*state*, *depth*) **then return** EVAL(*state*)
- Πρέπει επίσης να φροντίσουμε να αυξάνεται το τρέχον βάθος μετά από κάθε αναδρομική κλήση.
- Μπορούμε να ορίσουμε ένα όριο βάθους d και να κάνουμε την CUTOFF-TEST(*state*, *depth*) να επιστρέφει *true* όταν $depth \geq d$. Το d μπορεί να επιλεγεί ώστε η κίνηση να επιλέγεται στον καθορισμένο χρόνο.
- Η CUTOFF-TEST πρέπει να επιστρέφει *true* και για τις τερματικές καταστάσεις (όπως η TERMINAL-TEST).
- Εδώ μπορούμε επίσης να εφαρμόσουμε **αναζήτηση με επαναληπτική εκβάθυνση**. Όταν εξαντλείται ο χρόνος, το πρόγραμμα επιστρέφει την κίνηση που επιλέχθηκε από τη βαθύτερη αναζήτηση που έχει ολοκληρωθεί.

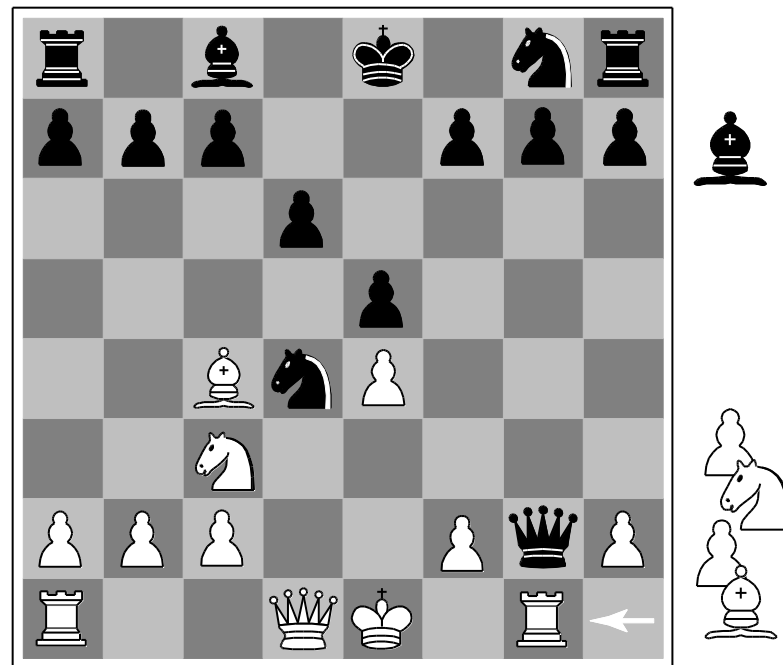
Αποκοπή Αναζήτησης

- Οι μέθοδοι που μόλις περιγράψαμε μπορεί να οδηγήσουν σε λάθη, λόγω της προσεγγιστικής φύσης της συνάρτησης αξιολόγησης.
- **Παράδειγμα:** Θεωρούμε την απλή συνάρτηση αξιολόγησης για το σκάκι που προτείναμε νωρίτερα και η οποία βασίζεται στο υλικό πλεονέκτημα.
- Έστω ότι το πρόγραμμα κάνει αναζήτηση μέχρι το όριο βάθους και φτάνει στη κατάσταση που φαίνεται στην επόμενη διαφάνεια (σκακιέρα a) όπου ο Μαύρος υπερέχει κατά ένα ίππο και δύο πιόνια.

Παράδειγμα



(a) White to move



(b) White to move

Παράδειγμα

- Το πρόγραμμα θα αξιολογούσε την κατάσταση αυτή και θα εκτιμούσε ότι θα οδηγούσε σε νίκη του Μαύρου.
- Με την επόμενη κίνηση του όμως (σκακιέρα b), ο Λευκός παίρνει τη βασίλισσα του Μαύρου χωρίς κανένα αντάλλαγμα.
- Επομένως η κατάσταση αυτή θα οδηγήσει σε νίκη του Λευκού όμως το πρόγραμμα δεν μπορεί να το γνωρίζει αυτό **αν δεν ερευνήσει μία στρώση πιο μπροστά.**

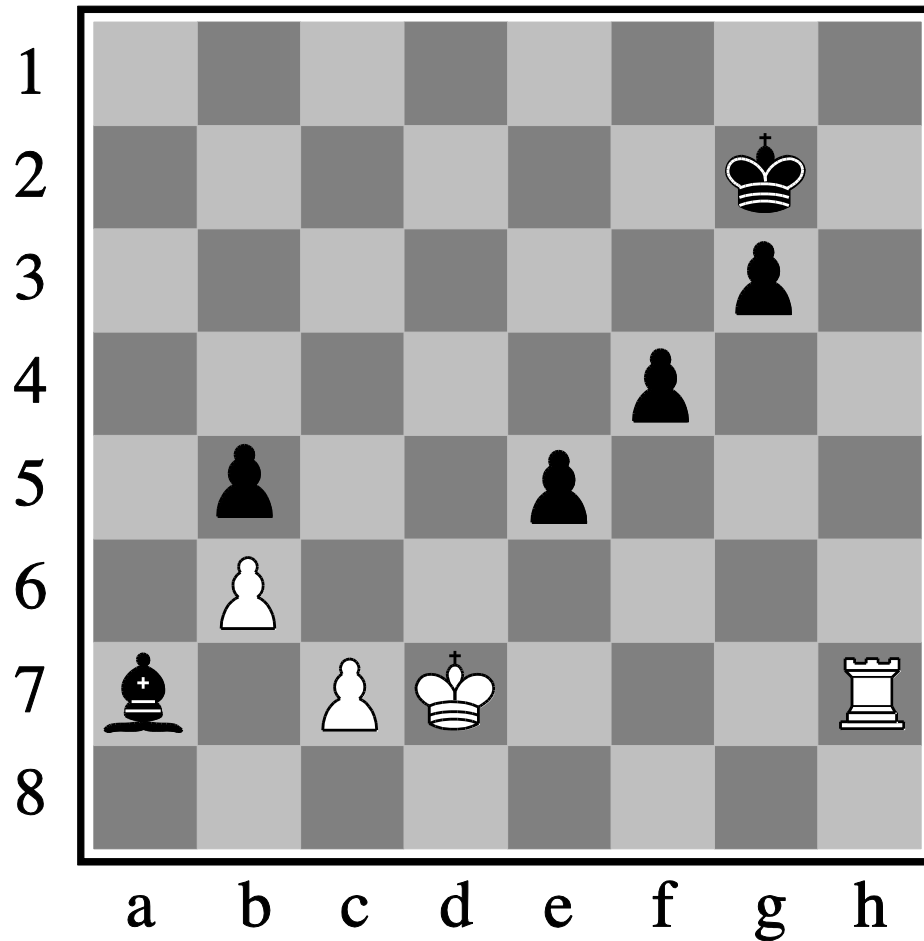
Αναζήτηση Ηρεμίας

- Χρειαζόμαστε ένα καλύτερο τρόπο για να αξιολογούμε καταστάσεις.
- Η συνάρτηση αξιολόγησης θα πρέπει να εφαρμόζεται μόνο σε καταστάσεις που είναι **ήρεμες** δηλαδή καταστάσεις στις οποίες η αξιολόγηση δεν θα αλλάξει δραματικά στο κοντινό μέλλον.
- Οι καταστάσεις που δεν είναι ήρεμες θα πρέπει να επεκτείνονται μέχρι να φτάσουμε σε ήρεμες καταστάσεις. Αυτή η πρόσθετη αναζήτηση, ονομάζεται **αναζήτηση ηρεμίας**.

Η Επίδραση του Ορίζοντα

- Ένα άλλο σχετικό πρόβλημα είναι η **επίδραση του ορίζοντα (horizon effect)**.
- Το πρόβλημα αυτό παρουσιάζεται όταν το πρόγραμμα έχει να αντιμετωπίσει μια κίνηση του αντιπάλου που προκαλεί σοβαρή ζημιά και είναι αναπόφευκτη, αλλά μπορεί να αποφευχθεί προσωρινά με τακτικές καθυστέρησης του αντιπάλου.

Παράδειγμα



Παράδειγμα

- Αν εξετάσουμε την σκακιστική παρτίδα της προηγούμενης διαφάνειας θα δούμε ότι δεν υπάρχει τρόπος να ξεφύγει ο μαύρος αξιωματικός.
- Ο άσπρος πύργος μπορεί να τον αιχμαλωτίσει μετακινούμενος στη θέση h1, μετά στην a1, και μετά στην a7 δηλαδή μετά από βάθος 6 στρώσεις.
- Αλλά ο Μαύρος έχει μια ακολουθία κινήσεων που σπρώχνει την αιχμαλωσία του αξιωματικού «πέρα από τον ορίζοντα».

Παράδειγμα

- Υποθέστε ότι ο Μαύρος εκτελεί αναζήτηση μέχρι βάθους 8 στρώσεις.
- Οι περισσότερες κινήσεις του Μαύρου θα οδηγήσουν στην τελική αιχμαλωσία του αξιωματικού και άρα θα θεωρηθούν κακές κινήσεις.
- Όμως ο Μαύρος μπορεί να προσπαθήσει να τσεκάρει τον άσπρο βασιλιά με το πiónι στη θέση e5. Αυτό θα οδηγήσει στην αιχμαλωσία του πιονιού από τον άσπρο βασιλιά.
- Ο Μαύρος μπορεί να προσπαθήσει το ίδιο πράγμα με το πiónι στη θέση f4 οπότε θα προκύψει άλλη μια αιχμαλωσία του πιονιού του.
- Αυτές οι κινήσεις χρειάζονται 4 στρώσεις, άρα οι υπόλοιπες 4 στρώσεις δεν φτάνουν για να αιχμαλωτιστεί ο μαύρος αξιωματικός.
- Ο Μαύρος λοιπόν θα νομίζει ότι αυτή η ακολουθία κινήσεων έσωσε τον αξιωματικό με αντίτιμο δύο πiónια, ενώ αυτό που πραγματικά έγινε είναι η απώθηση της αιχμαλωσίας του αξιωματικού πέρα από τον ορίζοντα που μπορεί να δει ο Μαύρος.

Πρώιμο Κλάδεμα

- Το **πρώιμο κλάδεμα (forward pruning)** είναι η τεχνική με την οποία κάποιες κινήσεις σε ένα δοσμένο κόμβο κλαδεύονται αμέσως χωρίς να τις εξετάσουμε.
- Ένας τρόπος για να υλοποιήσουμε το πρώιμο κλάδεμα είναι να κάνουμε **ακτινική αναζήτηση (beam search)** δηλαδή, σε κάθε στρώση, να θεωρούμε μόνο μια «ακτίνα» αποτελούμενη από τις n καλύτερες κινήσεις σύμφωνα με την συνάρτηση αποτίμησης και να αγνοούμε τις υπόλοιπες.
- Δυστυχώς, αυτή η μέθοδος είναι επικίνδυνη γιατί δεν έχουμε καμία βεβαιότητα ότι δεν θα κλαδέψουμε την καλύτερη κίνηση.
- Στη βιβλιογραφία υπάρχουν προγράμματα όπου αυτή η τεχνική έχει χρησιμοποιηθεί επιτυχώς με την χρήση στατιστικών που προέρχονται από προηγούμενη εμπειρία ώστε να μειωθεί η πιθανότητα να κλαδευτεί η καλύτερη κίνηση (π.χ., το πρόγραμμα LOGISTELLO).



Συζήτηση

- Συνδυάζοντας όλες τις τεχνικές που έχουμε συζητήσει μέχρι τώρα μπορούμε να υλοποιήσουμε ένα πρόγραμμα που παίζει σκάκι (ή άλλα παιχνίδια).
- Ας υποθέσουμε ότι έχουμε υλοποιήσει μια συνάρτηση αποτίμησης για το σκάκι, ένα λογικό έλεγχο αποκοπής με αναζήτηση ηρεμίας και ένα μεγάλο πίνακα αντιμεταθέσεων.
- Αν η υλοποίηση μας μπορεί να παράγει και να αποτιμά περίπου ένα εκατομμύριο κόμβους το δευτερόλεπτο σε ένα πολύ ισχυρό PC, τότε μπορούμε να ψάχνουμε σε δέντρα παιχνιδιού με περίπου 200 εκατομμύρια κόμβους κάτω από τυπικούς ελέγχους χρόνου (τρία λεπτά για κάθε κίνηση).
- Ο παράγοντας διακλάδωσης για το σκάκι είναι περίπου 35 κατά μέσο όρο και 35^5 είναι 50 εκατομμύρια. Άρα αν χρησιμοποιούσαμε αναζήτηση minimax θα μπορούσαμε να βλέπουμε μπροστά μόνο περίπου 5 στρώσεις.
- Ένα τέτοιο πρόγραμμα θα έχανε εύκολα από ένα μεσαίο παίκτη σκακιού, οποίος μπορεί να σχεδιάζει μπροστά 6 ή 8 κινήσεις.
- Με αναζήτηση άλφα-βήτα φτάνουμε στις 10 στρώσεις και έχουμε ένα πρόγραμμα που παίζει σκάκι σε **επίπεδο ειδικού**.
- Για να φτάσουμε σε **επίπεδο «γκραν μαίτρ»** θα χρειαστούμε επιπλέον τεχνικές κλαδέματος, μια πολύ καλή συνάρτηση αποτίμησης και μια μεγάλη βάση δεδομένων με βέλτιστες κινήσεις ανοιγμάτων και φινάλε.



Χρήση Πινάκων με Κινήσεις

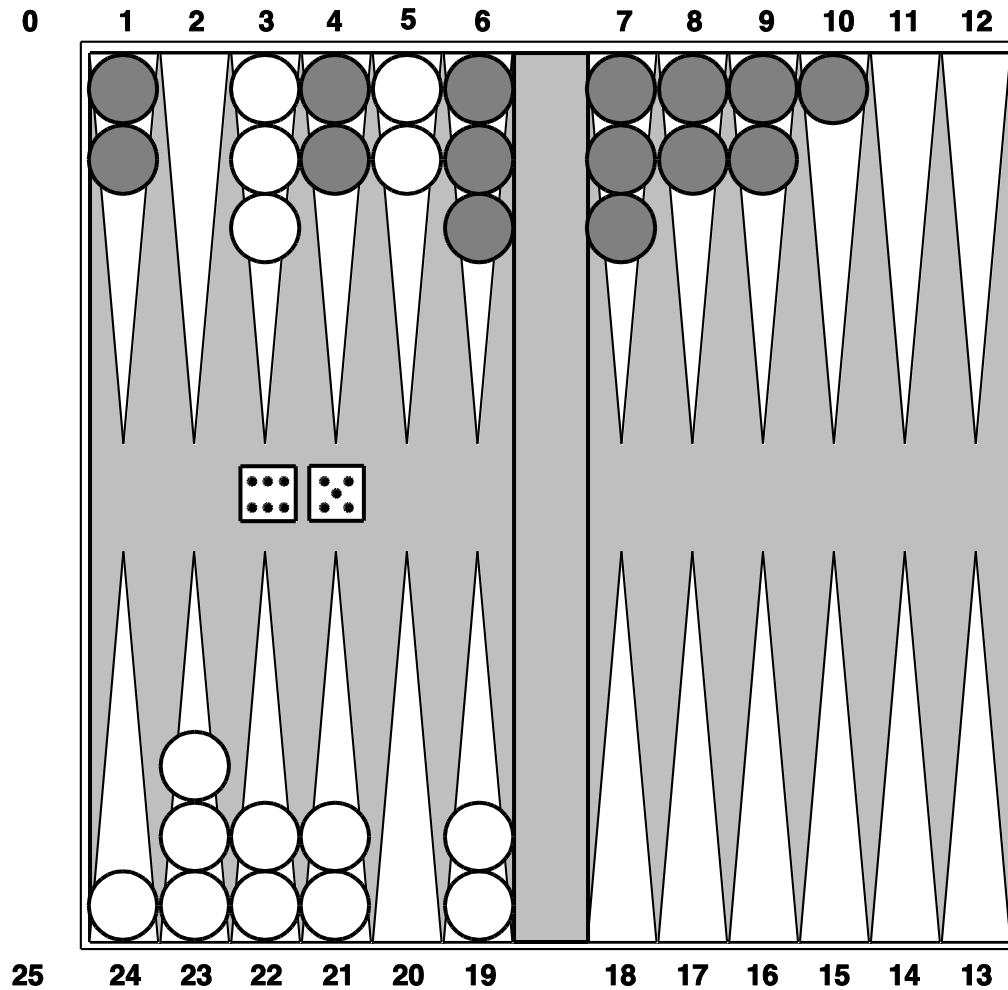
- Κατά την αρχή και το τέλος ενός παιχνιδιού, τα περισσότερα προγράμματα παιχνιδιών χρησιμοποιούν **πίνακες** που περιέχουν ενδεδειγμένες κινήσεις αντί να κάνουν αναζήτηση.
- Οι πίνακες αυτοί προέρχονται από την εμπειρία ειδικών όπως αυτή δίδεται, για παράδειγμα, σε εξειδικευμένα βιβλία για το κάθε παιχνίδι.
- Τα προγράμματα βέβαια μπορούν επίσης να κρατούν στατιστικές σχετικά με το ποιες αρχικές κινήσεις οδήγησαν σε νίκη.
- Στη βιβλιογραφία υπάρχουν εργασίες που έχουν επιλύσει πλήρως όλα τα δυνατά φινάλε παιχνιδιών σκακιού που υπάρχουν με το πολύ 5 κομμάτια και μερικά με 6 κομμάτια. Ένα πρόγραμμα για το σκάκι μπορεί να χρησιμοποιήσει τους αντίστοιχους πίνακες για το φινάλε ενός παιχνιδιού αντί να κάνει αναζήτηση.
- Αν μπορούσαμε να επιλύσουμε πλήρως φινάλε με 32 κομμάτια αντί για 6, τότε ο Άσπρος θα ήξερε με την πρώτη κίνηση του παιχνιδιού αν αυτή θα οδηγήσει σε νίκη, ισοπαλία ή ήττα. Αυτό δεν έχει επιτευχθεί μέχρι σήμερα για το σκάκι, **έχει όμως επιτευχθεί για την ντάμα όπως θα δούμε στη συνέχεια.**

Στοχαστικά Παιχνίδια

- Πολλά παιχνίδια περιλαμβάνουν ένα **στοιχείο τύχης** π.χ., τη ρίψη ζαριών.
- Τα παιχνίδια αυτά λέγονται **στοχαστικά**.
- Αυτά τα παιχνίδια μας φέρνουν πιο κοντά στον πραγματικό κόσμο όπου η ύπαρξη εξωγενών γεγονότων μας οδηγούν σε απρόβλεπτες καταστάσεις.
- Το τάβλι είναι ένα αντιπροσωπευτικό παιχνίδι που συνδυάζει τύχη και ικανότητα.



Παράδειγμα



Παράδειγμα

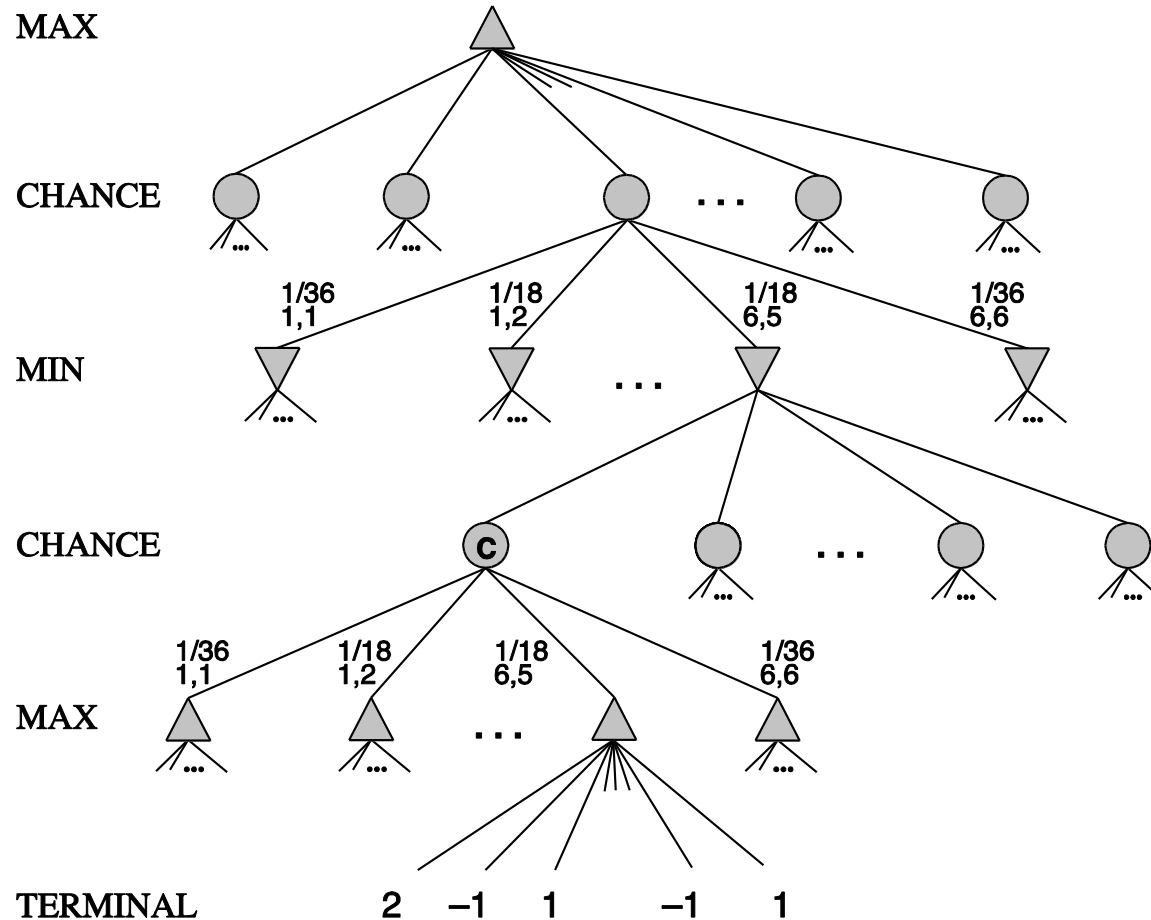
- Στην προηγούμενη εικόνα, ο παίκτης με τα λευκά πούλια έχει φέρει 6-5 και πρέπει να επιλέξει μεταξύ τεσσάρων νόμιμων κινήσεων: (5-10, 5-11), (5-11, 19-24), (5-10, 10-16) και (5-11, 11-16).
- Ο παραπάνω συμβολισμός $x-y$ για τις κινήσεις σημαίνει να μετακινήσει ένα πούλι από τη θέση x στη θέση y .

Δένδρα Παιχνιδιού για Στοχαστικά Παιχνίδια

- Τα δένδρα παιχνιδιού για στοχαστικά παιχνίδια περιλαμβάνουν ένα νέο τύπο κόμβου, τους **κόμβους τύχης (chance nodes)**.
- Οι κόμβοι τύχης συμβολίζονται με κύκλους.
- Για το τάβλι, τα κλαδιά που ξεκινούν από κάθε κόμβο τύχης αντιπροσωπεύουν τις δυνατές ζαριές και τις πιθανότητες τους.
- Υπάρχουν 36 δυνατοί τρόποι για να ριφθούν δύο ζάρια. Επειδή όμως η ζαριά 5-6 είναι ίδια με τη ζαριά 6-5, υπάρχουν **21 δυνατές ζαριές**. Η πιθανότητα κάθε «διπλής» ζαριάς (π.χ., 1-1) είναι $\frac{1}{36}$. Η πιθανότητα κάθε ζαριάς που δεν είναι διπλή είναι $\frac{1}{18}$.



Παράδειγμα Δένδρου Παιχνιδιού για το Τάβλι



Expectiminimax Τιμές

- Στα στοχαστικά παιχνίδια, οι καταστάσεις δεν έχουν οριστικές τιμές minimax. Μπορούμε όμως να υπολογίσουμε τις **αναμενόμενες τιμές minimax (expected minimax values)** ως το μέσο όρο των τιμών για όλα τα δυνατά αποτελέσματα των κόμβων τύχης.
- Έτσι οδηγούμαστε σε μια γενίκευση της τιμής minimax των αιτιοκρατικών παιχνιδιών σε μια **τιμή expectiminimax** για στοχαστικά παιχνίδια.

Expectiminimax Τιμές

- Η μαθηματική παράσταση για τις τιμές **expectiminimax** είναι:

EXPECTIMINIMAX(s) =

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

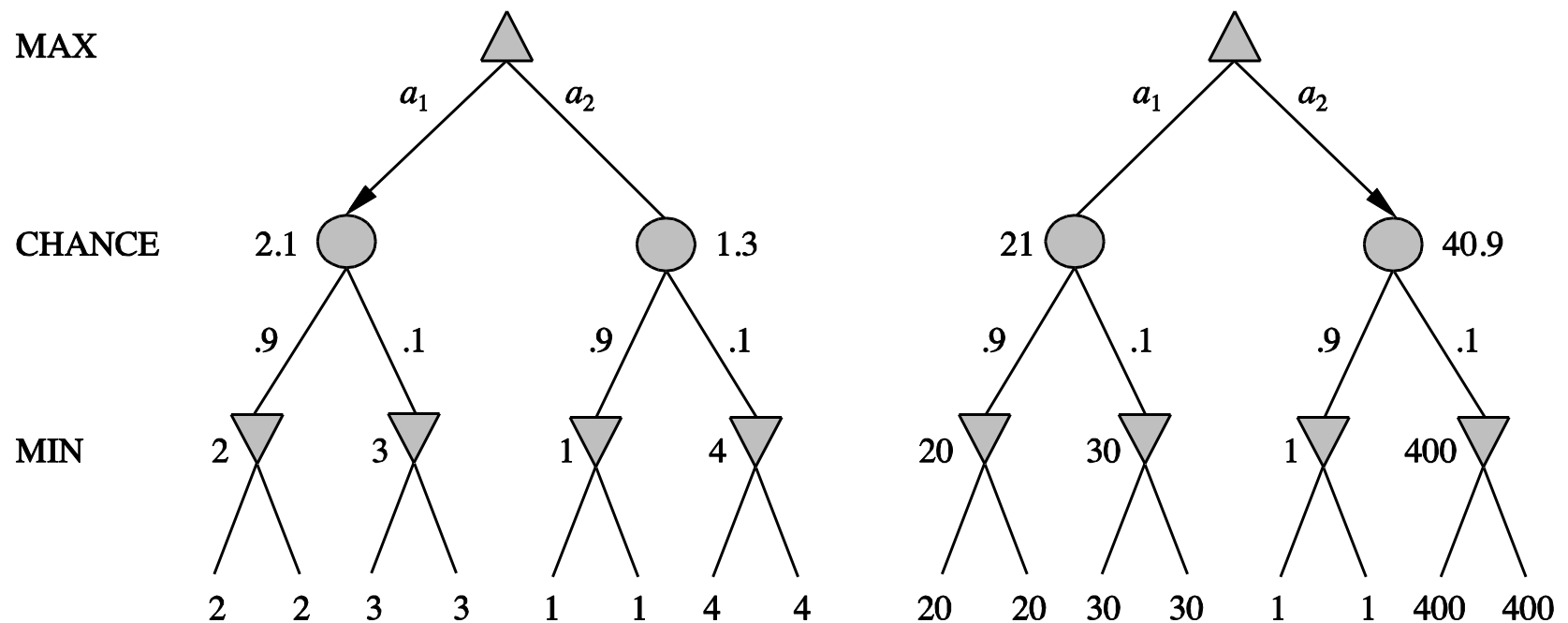
όπου το r συμβολίζει το αποτέλεσμα μια ζαριάς (ή ενός άλλου τυχαίου γεγονότος), $P(r)$ είναι η πιθανότητα του r και $\text{RESULT}(s, r)$ είναι η ίδια κατάσταση όπως η s με την επιπλέον συνθήκη ότι το αποτέλεσμα της ζαριάς είναι r .

- Ο αλγόριθμος που αντιστοιχεί στην παραπάνω παράσταση αφήνεται σαν άσκηση.

Συναρτήσεις Αξιολόγησης για Στοχαστικά Παιχνίδια

- Κάποιος μπορεί να σκεφτεί ότι οι συναρτήσεις αξιολόγησης για στοχαστικά παιχνίδια πρέπει να είναι ακριβώς όπως οι συναρτήσεις αξιολόγησης για αιτιοκρατικά παιχνίδια δηλαδή να δίνουν καλύτερους βαθμούς στις καλύτερες καταστάσεις.
- Όμως η ύπαρξη των κόμβων τύχης δημιουργεί κάποια προβλήματα όπως φαίνεται στο παρακάτω παράδειγμα.

Παράδειγμα



Παράδειγμα

- Με μια συνάρτηση αξιολόγησης που αναθέτει τιμές 1,2,3,4 στα φύλλα, η κίνηση α_1 είναι η καλύτερη.
- Με μια συνάρτηση αξιολόγησης που αναθέτει τιμές 1, 20, 30, 400 στα φύλλα, η κίνηση α_2 είναι η καλύτερη.
- Επομένως ή αλλαγή στην κλίμακα αξιολόγησης αλλάζει την κίνηση που επιλέγεται.
- Για να αποφύγουμε αυτό το πρόβλημα ευαισθησίας, η συνάρτηση αξιολόγησης θα πρέπει να είναι ένας **θετικός γραμμικός συνδυασμός** της πιθανότητας νίκης από μια κατάσταση (ή γενικότερα της αναμενόμενης χρησιμότητας της κατάστασης).
- Αυτή είναι μια γενικότερη ιδιότητα των καταστάσεων στις οποίες υπάρχει αβεβαιότητα και μελετάται σε βάθος στο κεφάλαιο 16 του βιβλίου (Θεωρία Αποφάσεων).

Πολυπλοκότητα της Expectiminimax

- Η πολυπλοκότητας της expectiminimax είναι $O(b^m n^m)$ όπου b είναι ο παράγοντας διακλάδωσης, m είναι το μέγιστο βάθος του δένδρου παιχνιδιού και n ο μέγιστος αριθμός αποτελεσμάτων που έχει ένας κόμβος τύχης.
- Στο τάβλι το n είναι 21 και το b περίπου 20 αλλά μπορεί να φτάσει και 4000 για ζαριές που είναι διπλές.
- Επομένως δεν είναι ρεαλιστικό να βλέπουμε περισσότερο από 3 στρώσεις μπροστά.

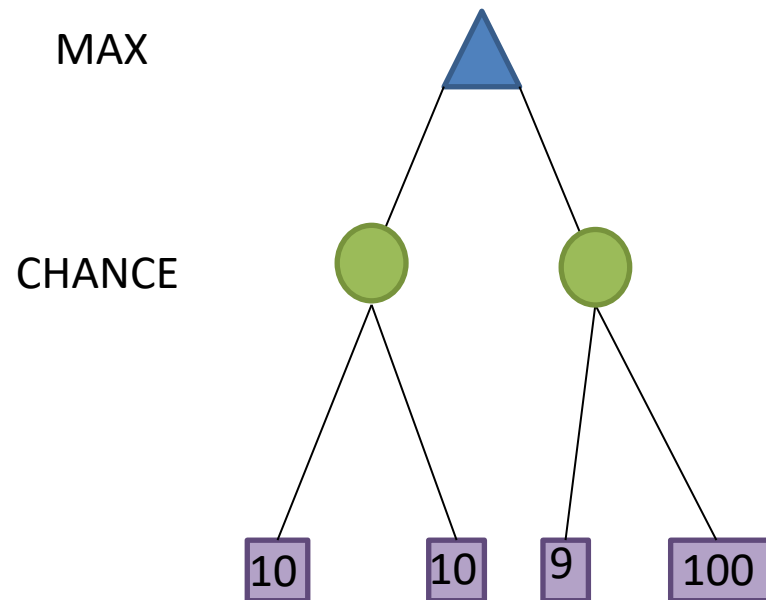
Άλλες Τεχνικές

- Μπορούμε να αναπτύξουμε μια μέθοδο όπως την αναζήτηση άλφα-βήτα για στοχαστικά παιχνίδια. Η μέθοδος θα επιτρέπει και το κλάδεμα κόμβων τύχης.
- Μια άλλη τεχνική είναι να χρησιμοποιήσουμε **προσομοίωση Monte Carlo** για να αποτιμήσουμε μια κατάσταση.
- Η τεχνική αυτή δουλεύει ως εξής. Από μια δοσμένη κατάσταση, ο αλγόριθμος παίζει χιλιάδες παιχνίδια με τον εαυτό του χρησιμοποιώντας τυχαίες ζαριές. Για το τάβλι, έχειδειχτεί ότι τότε το ποσοστό των νικών είναι μια καλή αξιολόγηση της δοσμένης κατάστασης.

Αναζήτηση Expectimax

- Μπορούμε να θεωρήσουμε δένδρα παιχνιδιού που περιέχουν μόνο κόμβους MAX και τύχης δηλαδή δεν περιέχουν κόμβους MIN. Αυτή είναι η περίπτωση της **αναζήτησης expectimax**.
- **Παράδειγμα:** Το δένδρο παιχνιδιού για το παιχνίδι racman με φαντάσματα τα οποία κινούνται τυχαία. Ο racman είναι ο παίκτης MAX και τα φαντάσματα είναι κόμβοι τύχης.

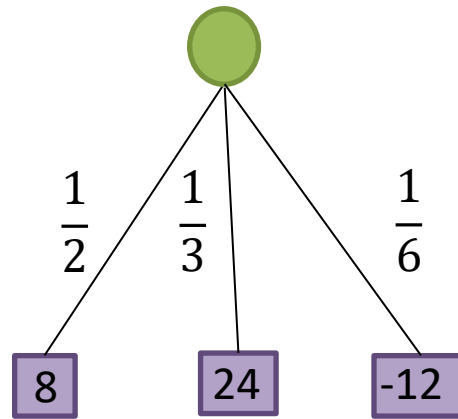
Παράδειγμα



Αναζήτηση Expectimax

- Όπως και στην περίπτωση της expectiminimax, οι τιμές των κόμβων τύχης είναι **αναμενόμενες τιμές (expected values)** και υπολογίζονται από το μέσο όρο των τιμών για όλα τα δυνατά αποτελέσματα των κόμβων τύχης.

Παράδειγμα



$$v = \left(\frac{1}{2}\right)(8) + \left(\frac{1}{3}\right)(24) + \left(\frac{1}{6}\right)(-12) = 10$$

Expectimax Τιμές

- Η μαθηματική παράσταση για τις τιμές **expectimax** είναι:


$EXPECTIMAX(s) =$

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

όπου το r συμβολίζει το αποτέλεσμα μια ζαριάς (ή ενός άλλου τυχαίου γεγονότος), $P(r)$ είναι η πιθανότητα του r και $RESULT(s, r)$ είναι η ίδια κατάσταση όπως η s με την επιπλέον συνθήκη ότι το αποτέλεσμα της ζαριάς είναι r .

- Ο αλγόριθμος που αντιστοιχεί στην παραπάνω παράσταση αφήνεται σαν άσκηση.

Προηγμένα Προγράμματα Παιχνιδιών

- Σκάκι
 - Το πρόγραμμα DEEP BLUE της IBM κέρδισε τον παγκόσμιο πρωταθλητή Gary Kasparov το 1997.
- 
- Το πρόγραμμα DEEP BLUE χρησιμοποιούσε ένα παράλληλο υπολογιστή, εξειδικευμένο hardware, αναζήτηση άλφα-βήτα, μια μεγάλη βάση ανοιγμάτων και μια μεγάλη βάση προηγούμενων παιχνιδιών από γκραν μαίτρ.
 - Από τότε μια σειρά από άλλα προγράμματα έχουν βελτιώσει τις επιδόσεις του DEEP BLUE.
 - Με βάση τις πιο πρόσφατες εξελίξεις, μπορούμε να πούμε ότι **τα κορυφαία σκακιστικά προγράμματα σήμερα μπορούν να τρέχουν σε συνηθισμένους υπολογιστές και έχουν ξεφύγει σε ικανότητες από τους παίκτες-ανθρώπους.**

Προηγμένα Προγράμματα Παιχνιδιών

- Ντάμα.
 - Το πρόγραμμα CHINOOK νίκησε τον παγκόσμιο πρωταθλητή το 1992.
 - Από το 2007 και μετά το CHINOOK μπορεί να παίζει τέλεια χρησιμοποιώντας αναζήτηση άλφα-βήτα συνδυασμένη με μια βάση 39 τρισεκατομμυρίων κινήσεων που τελειώνουν το παιχνίδι.



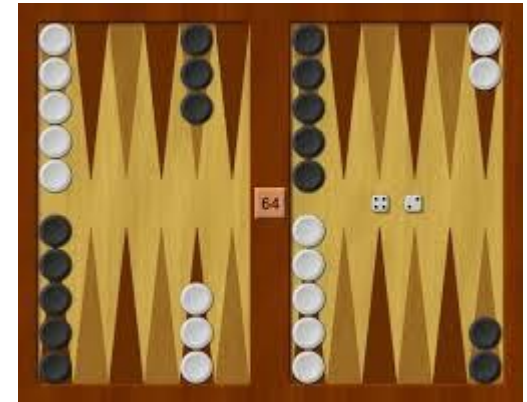
- Οθέλλο.
 - Το πρόγραμμα LOGISTELLO νίκησε τον παγκόσμιο πρωταθλητή το 1997.
 - Με βάση τις πιο πρόσφατε εξελίξεις, τα σημερινά κορυφαία προγράμματα για Οθέλλο είναι πολύ καλύτερα από ανθρώπους παίκτες.



Προηγμένα Προγράμματα Παιχνιδιών

- Τάβλι

- Το πρόγραμμα TD-GAMMON μπορεί να συναγωνιστεί κορυφαίους παίκτες από το 1992.
- Το βασικό χαρακτηριστικό του TD-GAMMON είναι η συνάρτηση αξιολόγησης του που βασίζεται σε τεχνικές ενισχυτικής μάθησης και νευρωνικών δικτύων.



- Go

- Μέχρι πρόσφατα η ανάπτυξη προγραμμάτων που παίζουν Go ήταν πολύ δύσκολη εξαιτίας του μεγάλου χώρου αναζήτησης και της δυσκολίας να σχεδιαστεί μια καλή συνάρτηση αξιολόγησης.
- Το πρόγραμμα ALPHAGO της εταιρίας Google Deep Mind νίκησε τον Ευρωπαίο πρωταθλητή το Μάρτιο του 2016.
- Το ALPHAGO βασίζεται σε βαθιά νευρωνικά δίκτυα.



Προηγμένα Προγράμματα Παιχνιδιών

- Μπριτζ
 - Το πρόγραμμα GIB ήρθε 12^ο (στους 35) στο παγκόσμιο πρωτάθλημα μπριτζ χρησιμοποιώντας τεχνικές γενίκευσης βασισμένης στις εξηγήσεις (explanation-based generalization). Επίσης, κέρδισε το παγκόσμιο πρωτάθλημα μπριτζ για υπολογιστές το 2000.
- Σκραμπλ
 - Το 2006 το πρόγραμμα QUACKLE νίκησε τον παγκόσμιο πρωταθλητή.



Μελέτη

- Βιβλίο ΑΙΜΑ
 - Κεφ. 6 στη 2^η έκδοση που έχει μεταφραστεί στα ελληνικά.
 - Κεφ. 5 στην 3^η έκδοση.

