Αναζήτηση με Αντιπαλότητα

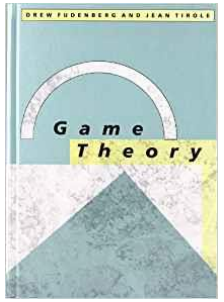
Μανόλης Κουμπαράκης

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 την 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 προυπόθεση 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 εναλλάσονται, 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.
  - PLAYER(*s*): Ορίζει ποιος παίκτης παίζει σε κάθε κατάσταση.
  - ACTIONS(*s*): Επιστρέφει το σύνολο των νόμιμων κινήσεων για κάθε κατάσταση.
  - RESULT(*s,a*): Ορίζει την κατάσταση που είναι το αποτέλεσμα κάθε κίνησης.
  - TERMINAL-TEST(*s*): Ό έλεγχος τερματισμού που είναι αληθής για τις καταστάσεις στις οποίες το παιχνίδι τελειώνει (**τερματικές καταστάσεις**) αλλιώς είναι ψευδής.
  - 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 $\frac{1}{2}$.

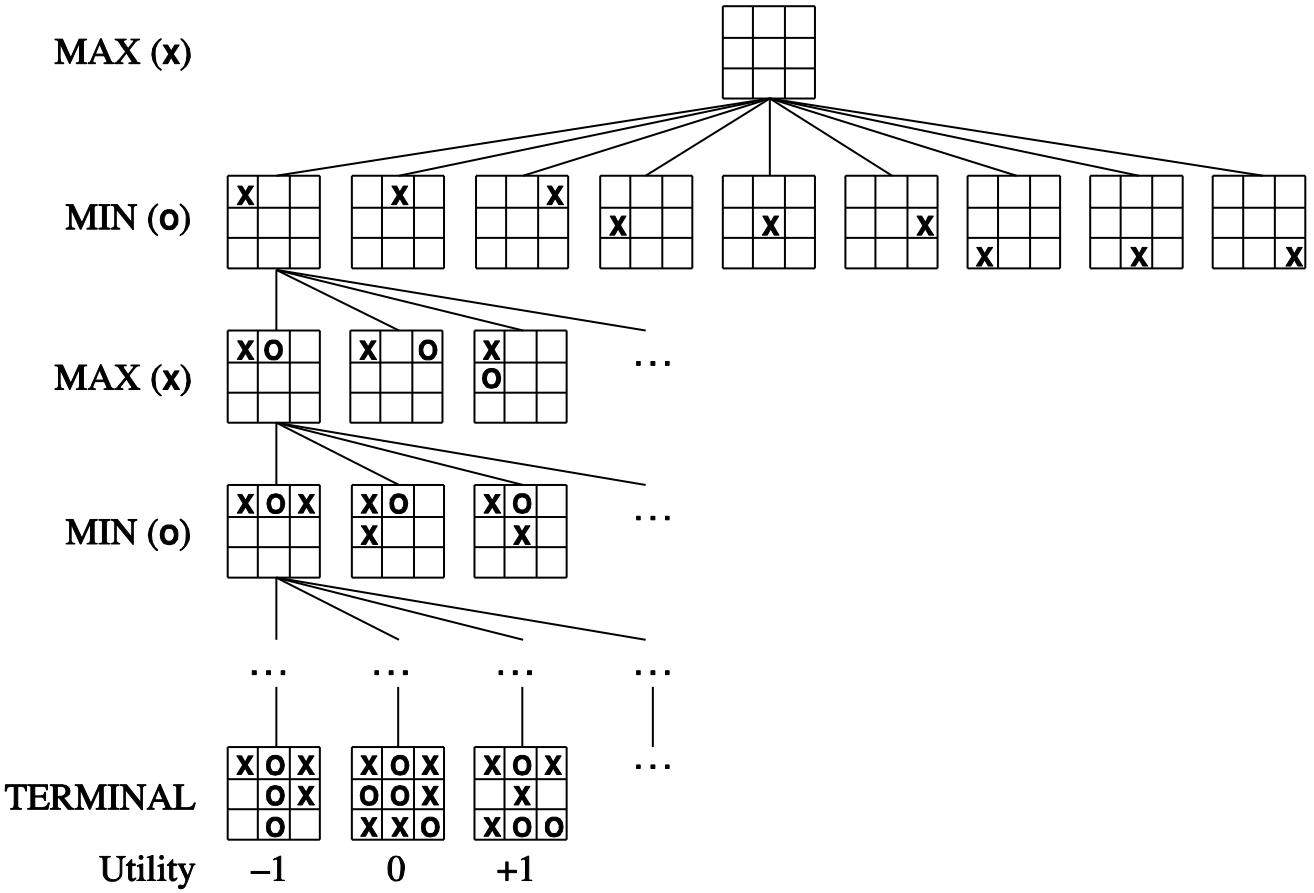
- Στο τάβλι, η συνάρτηση χρησιμότητας κυμαίνεται από 0 έως $+192$.

- 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 $\frac{1}{2} + \frac{1}{2}$.

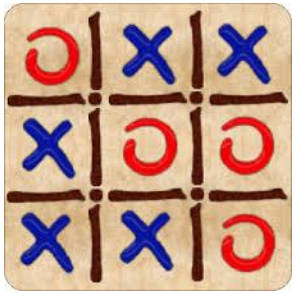
Δένδρο Παιχνιδιού

- 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

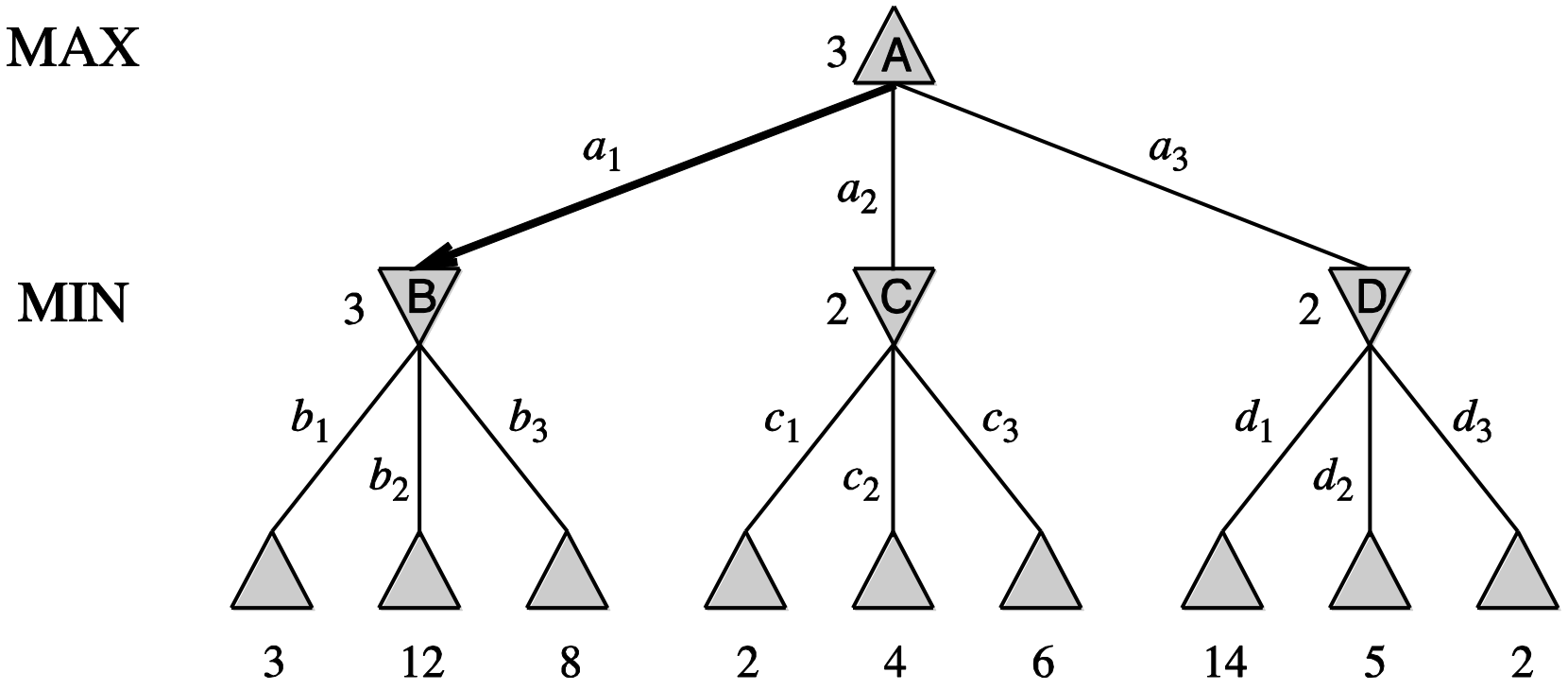


- Η τρίλιζα προχωράει με τον παίκτη MAX να βάζει ένα X σε ένα τετράγωνο και τον MIN να βάζει ένα O, μέχρι να φτάσουμε στους κόμβους φύλλα που αντιστοιχούν σε τερματικές καταστάσεις όπου ένας παίκτης έχει τρία σύμβολα στη σειρά ή όλα τα τετράγωνα είναι γεμάτα.
- **Ο αριθμός σε κάθε φύλλο δείχνει την τιμή 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).**
- Ένα δένδρο παιχνιδιού για την τρίλιζα είναι σχετικά μικρό: λιγότερες από $9! = 362880$ τερματικές καταστάσεις.
- Για το **σκάκι**, υπάρχουν πάνω από $\mathbf{10^{40}}$ **κόμβοι**. 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



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 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** (άλλος ένας λόγος για τον οποίο παίρνουν οι παίκτες το όνομα τους).

Τιμές Minimax

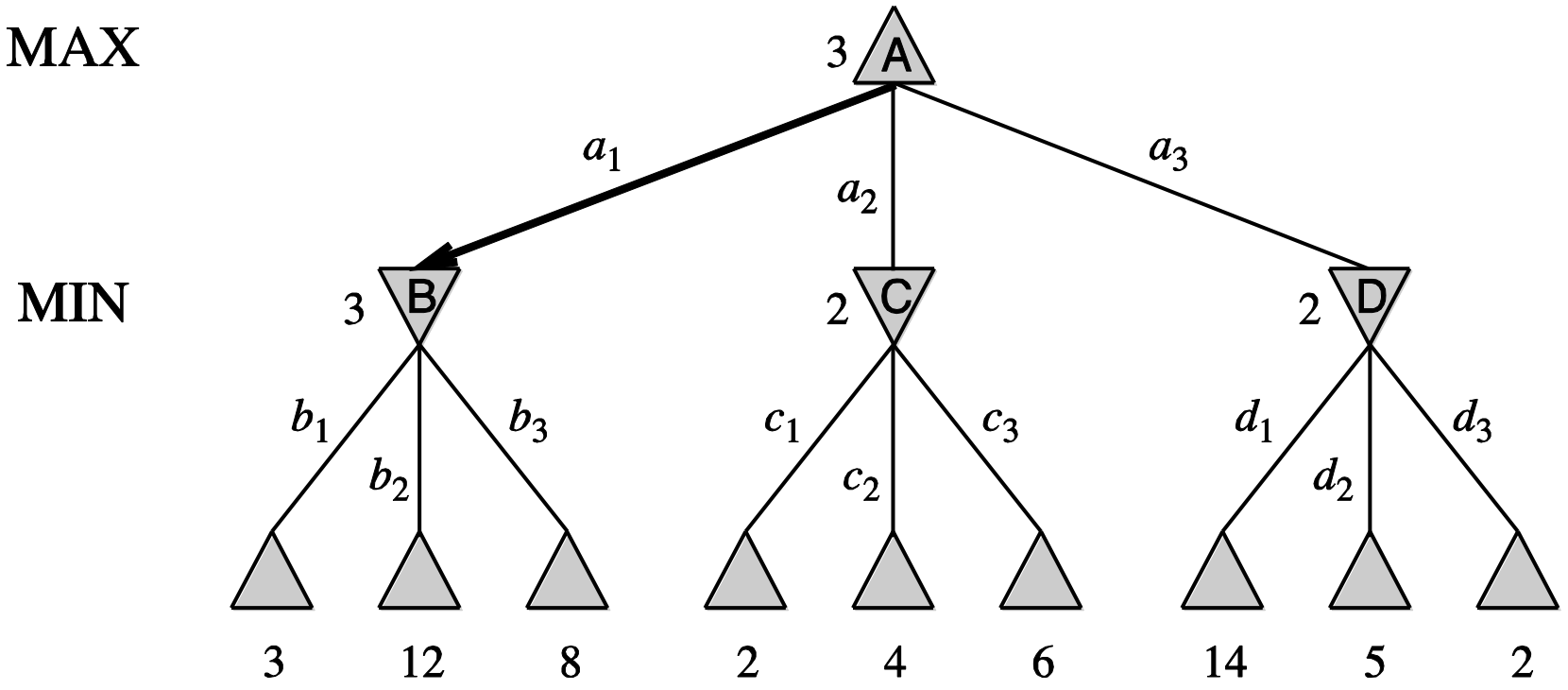
- Έτσι έχουμε την ακόλουθη ισότητα:

$$\text{MINIMAX}(s) =$$

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

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

Παράδειγμα



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 $\arg\max_{a \in \text{ACTIONS}(state)}$ MIN-VALUE(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**

    $v \leftarrow$MAX(v,MIN-VALUE(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**

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

  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 η αναδρομή.

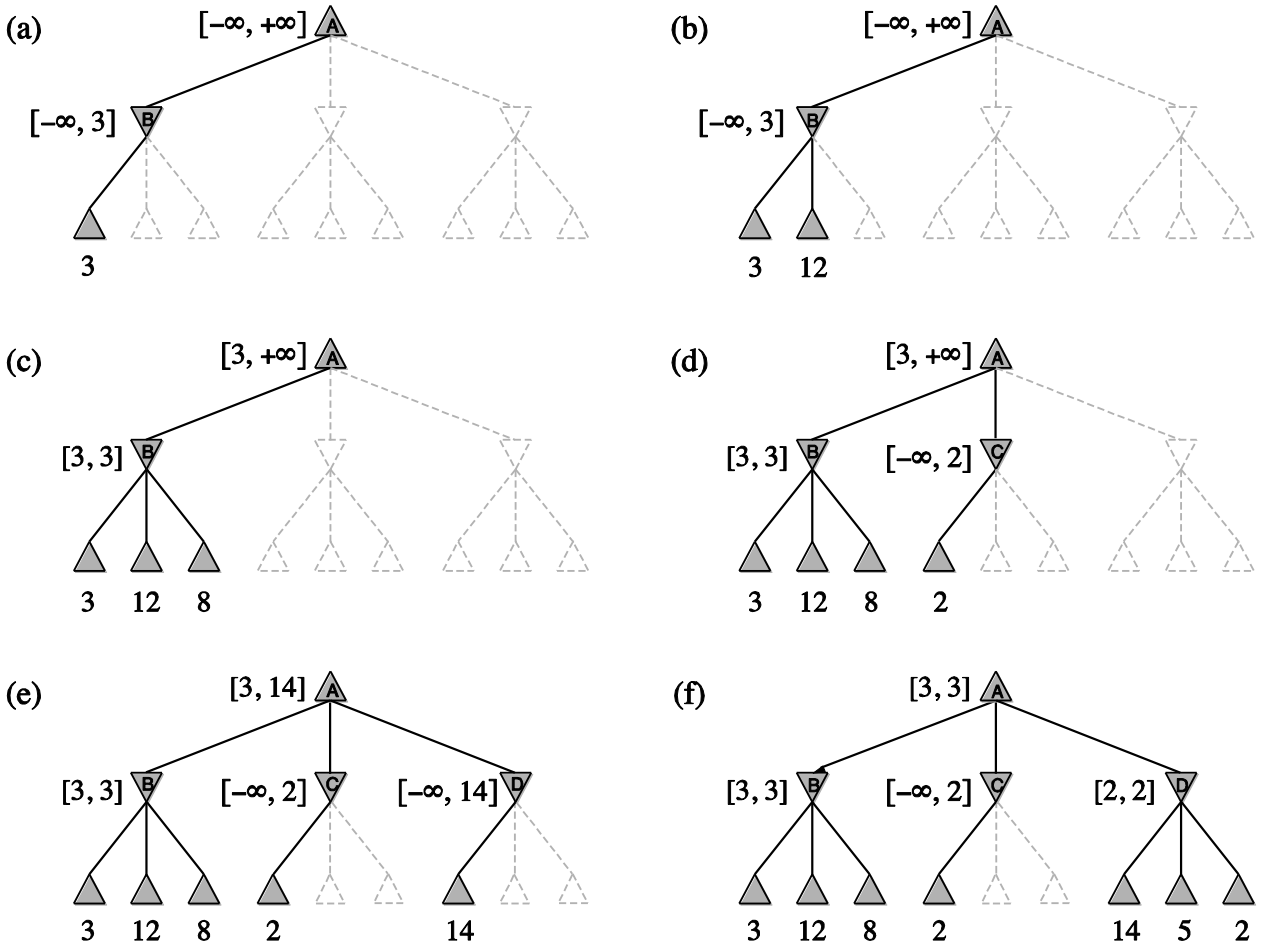
Υπολογιστική Πολυπλοκότητα

- 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


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3 12
(c)
[3, +∞]
A
[3, 3]
B
3 12 8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3 12 8 2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3 12 8 2 14
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3 12 8 2 14 5 2

Σχόλια

- Το πρώτο φύλλο κάτω από τον κόμβο Β έχει τιμή 3. Επομένως ο Β, που είναι κόμβος MIN, έχει τιμή **το πολύ 3**.
- Εξετάζοντας το δεύτερο και το τρίτο φύλλο κάτω από τον Β, 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

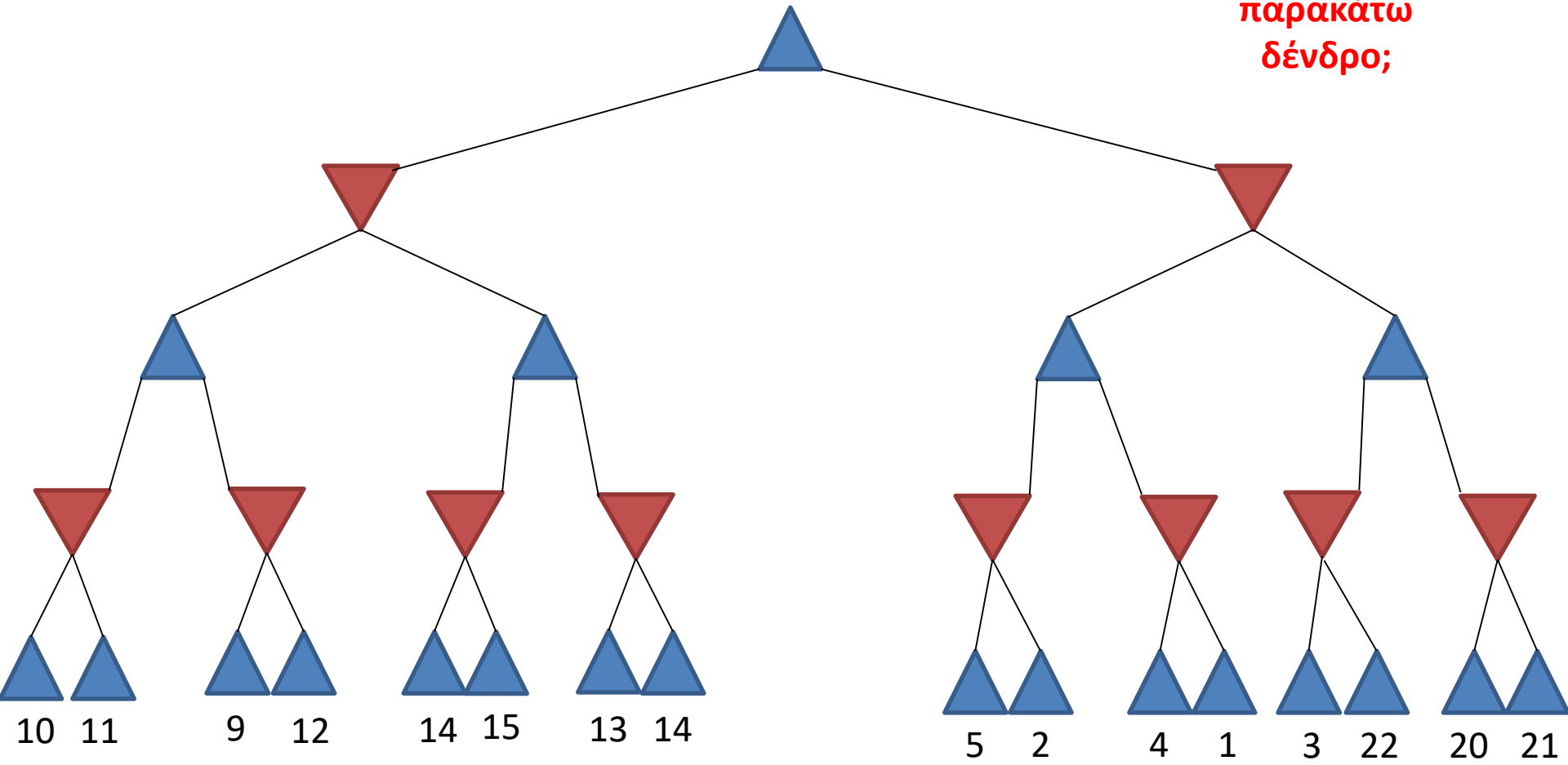
- Το πρώτο φύλλο κάτω από τον κόμβο C έχει τιμή 2. Επομένως ο C, που είναι κόμβος MIN, έχει τιμή **το πολύ 2**.
- Όμως ο Β έχει τιμή 3 άρα ο MAX στη ρίζα δεν θα διάλεγε ποτέ τον C. Επομένως, δεν έχει νόημα να εξετάσουμε τα άλλα παιδιά του C. Αυτό είναι ένα **παράδειγμα κλαδέματος άλφα-βήτα.**

Σχόλια

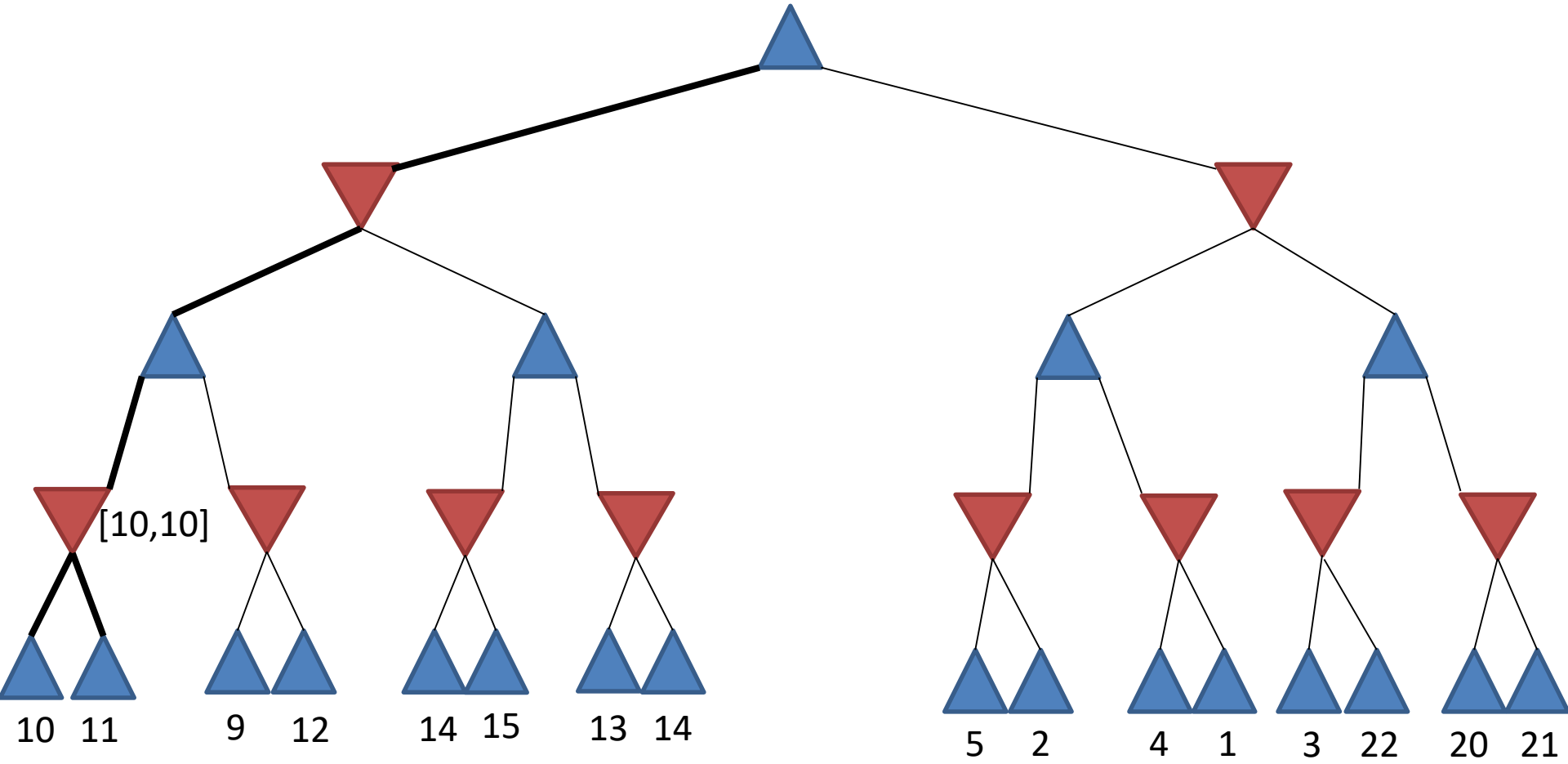
- Το πρώτο φύλλο κάτω από τον κόμβο D έχει την τιμή 14, γι αυτό η τιμή του D είναι το πολύ 14.
- Η τιμή αυτή είναι μεγαλύτερη από την καλύτερη εναλλακτική επιλογή του MAX (δηλαδή την τιμή 3) και άρα πρέπει να συνεχίζουμε να εξετάζουμε τα φύλλα κάτω από τον D.
- Εξετάζοντας τα άλλα δύο φύλλα βρίσκουμε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

Παράδειγμα

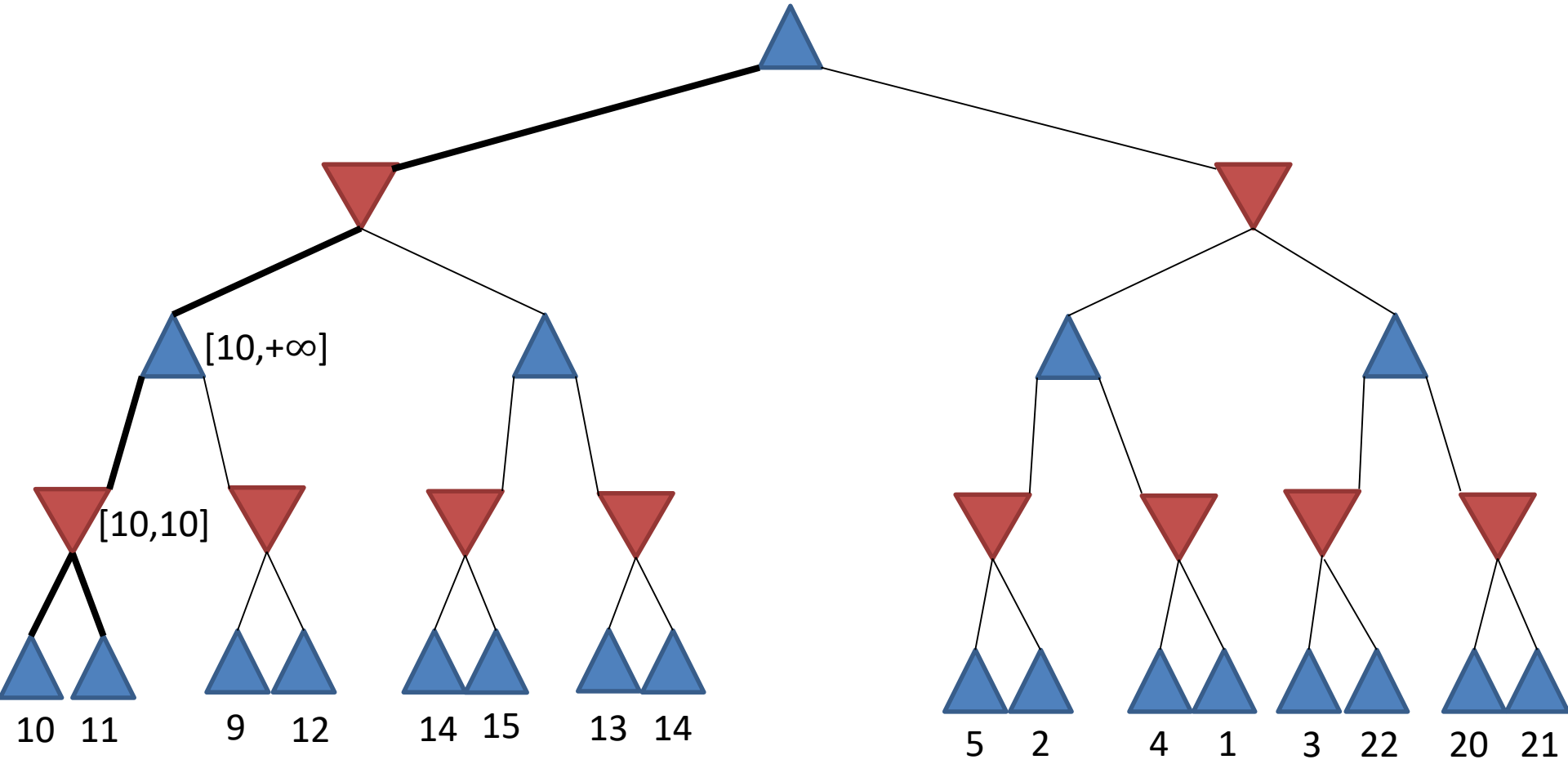
Ποιο είναι το αποτέλεσμα του κλαδέματος άλφα-βήτα στο παρακάτω δένδρο;



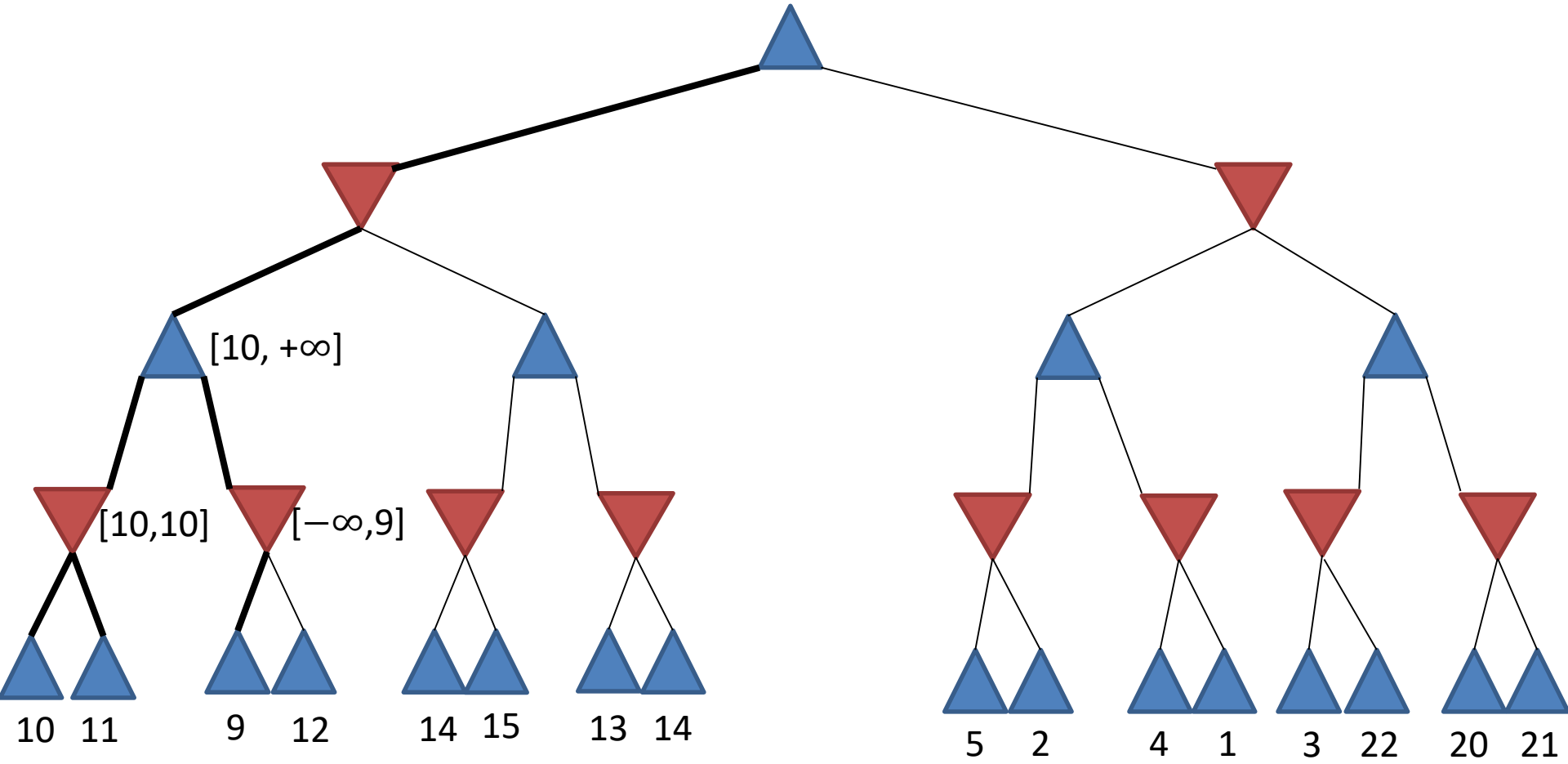
Παράδειγμα



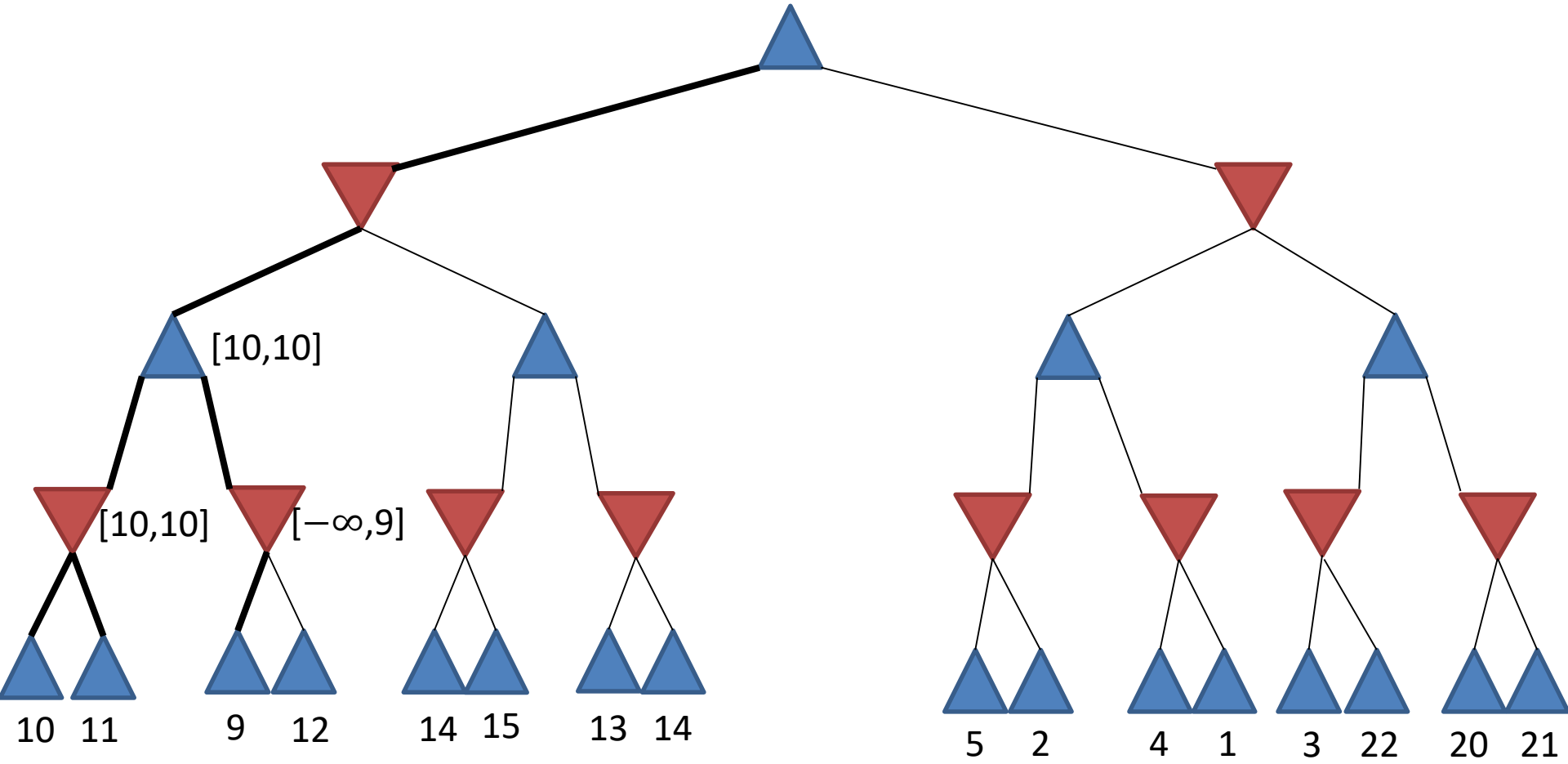
Παράδειγμα



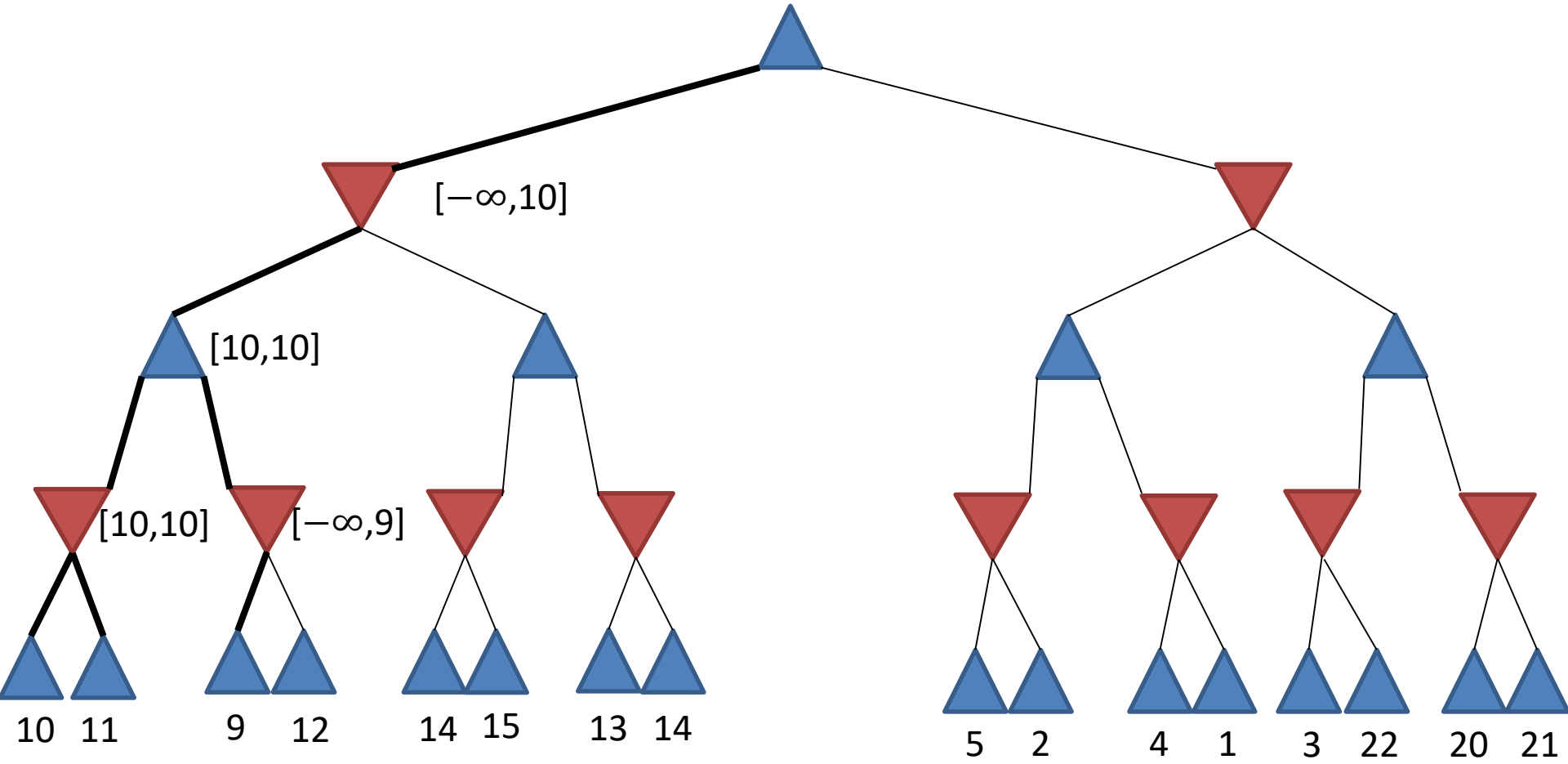
Παράδειγμα



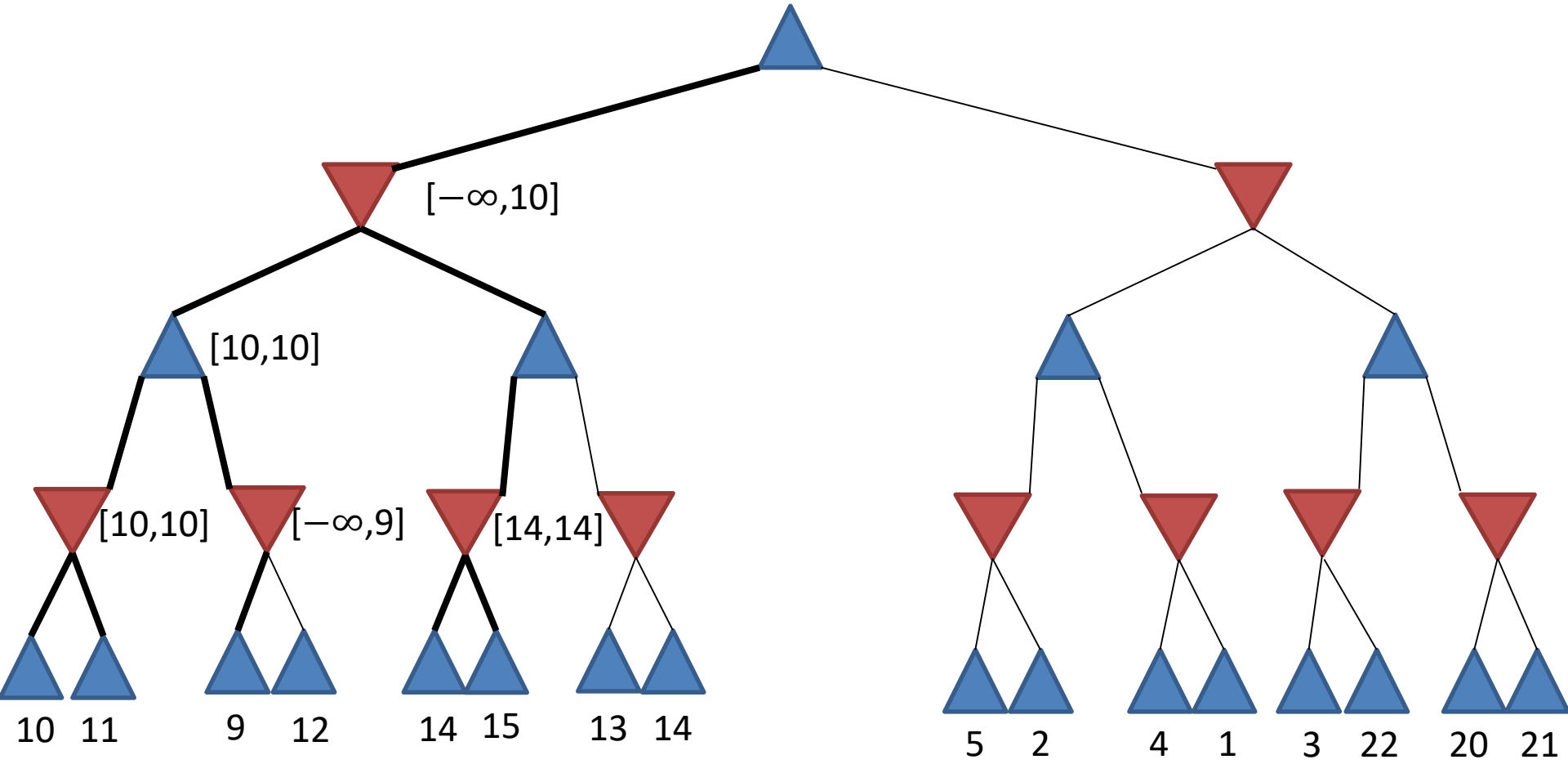
Παράδειγμα



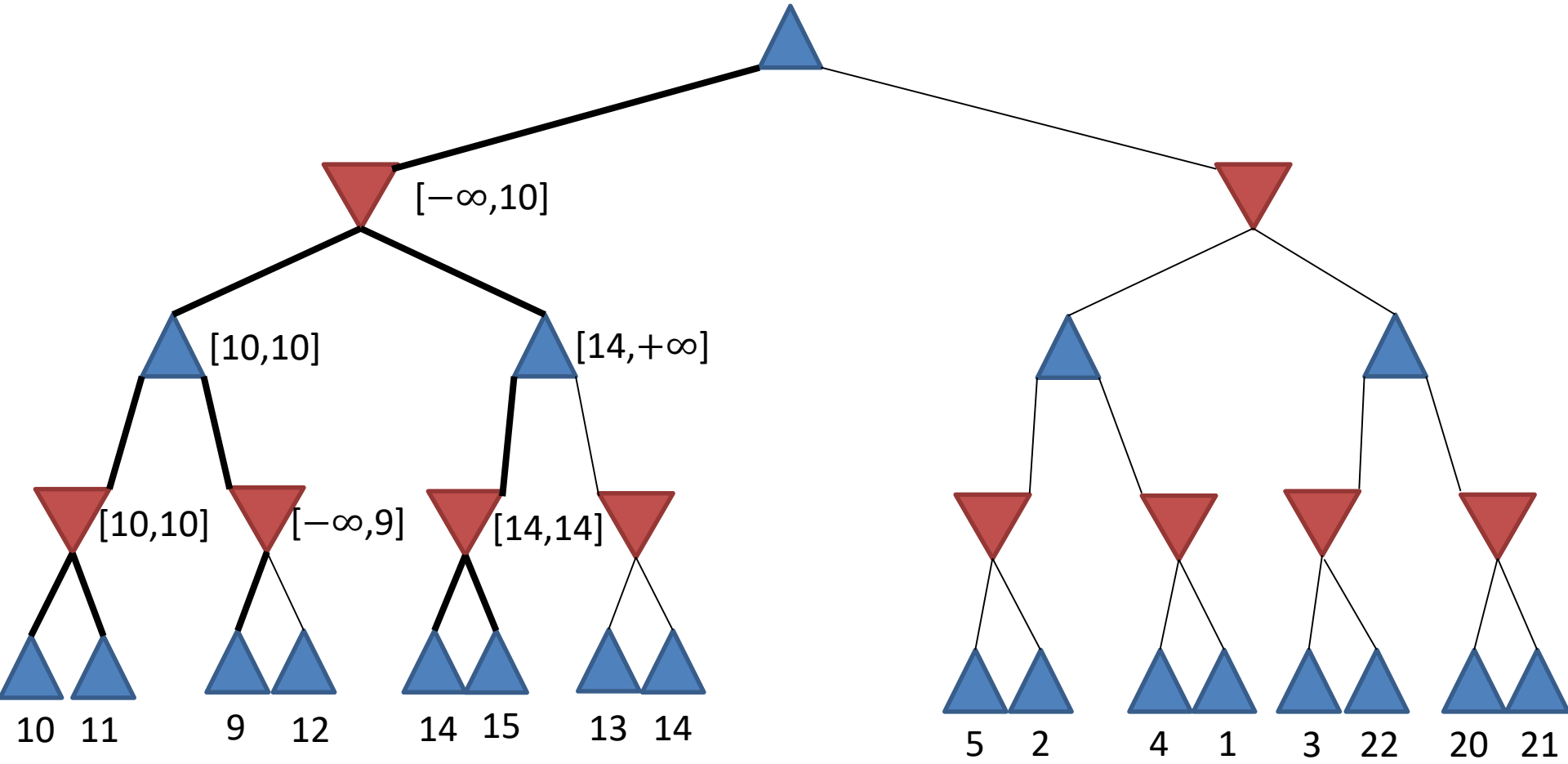
Παράδειγμα



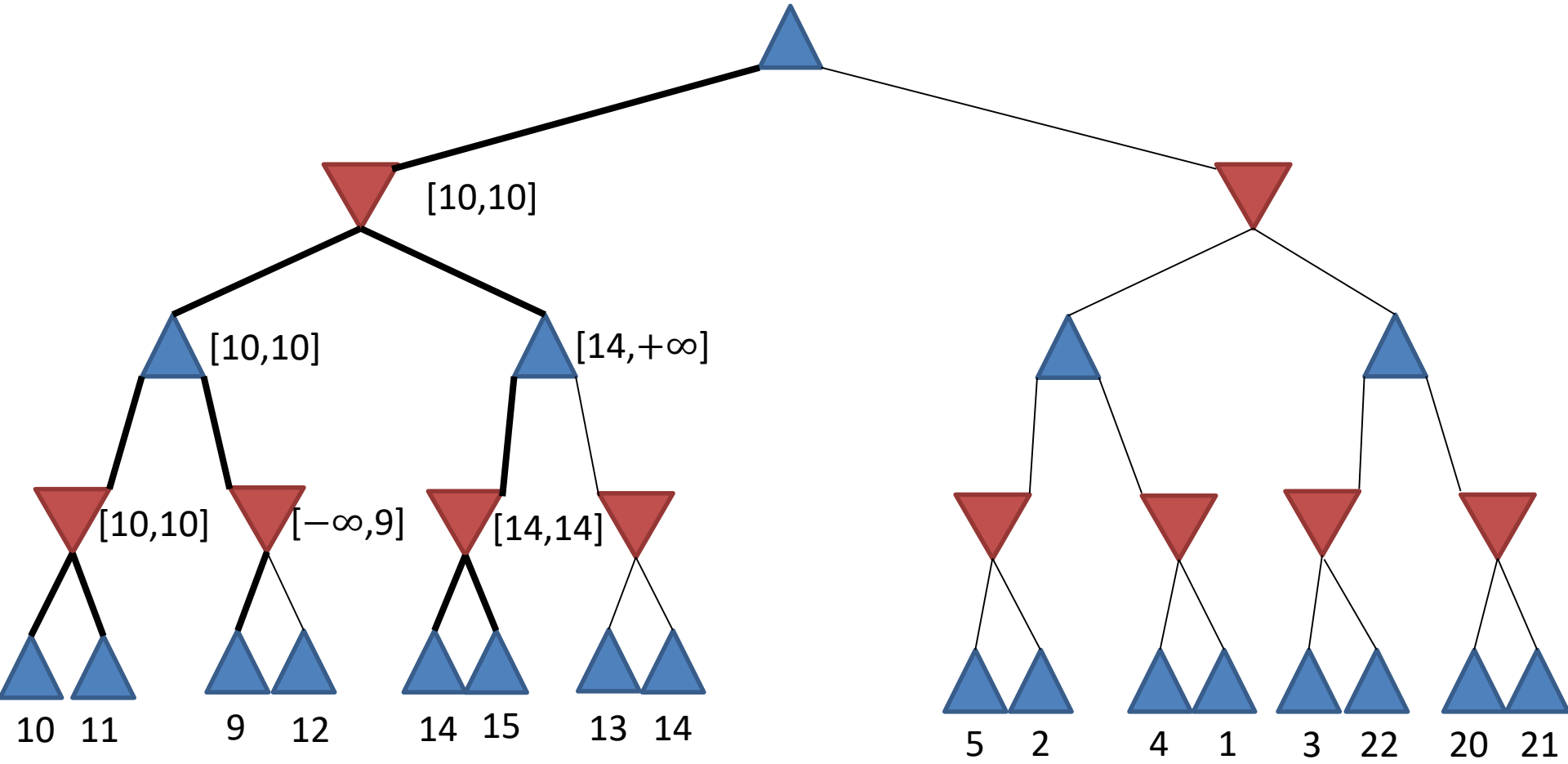
Παράδειγμα



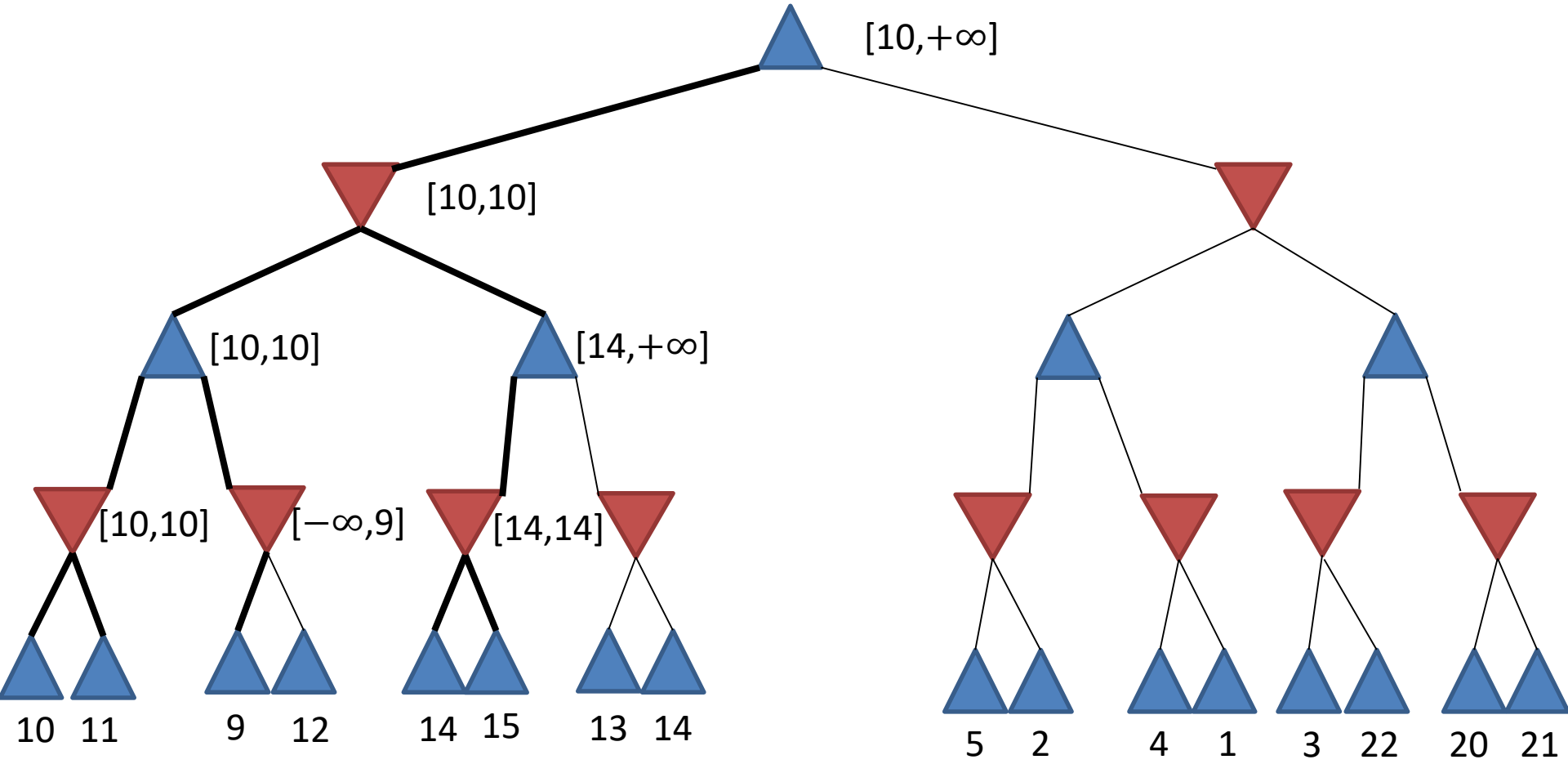
Παράδειγμα



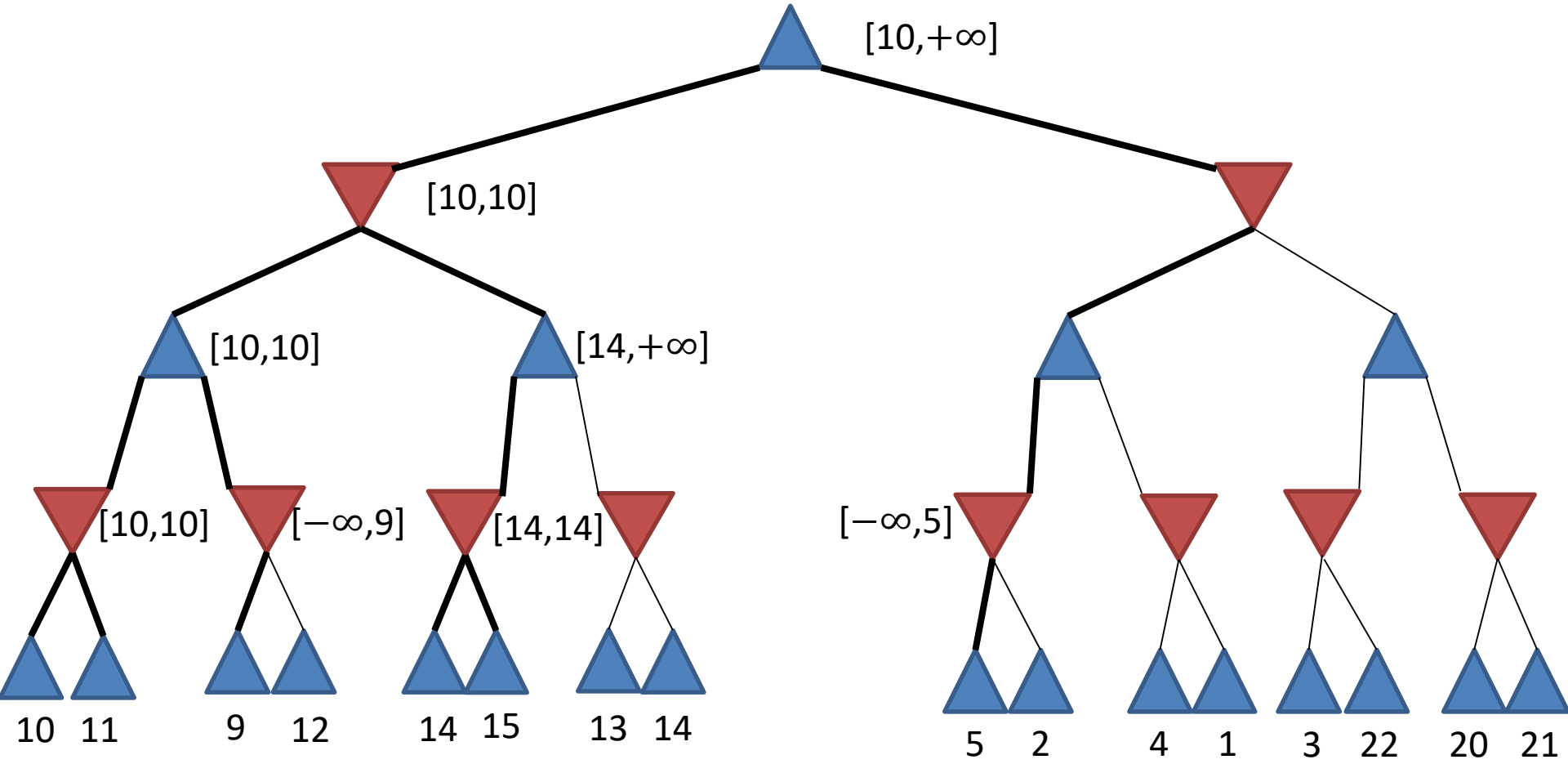
Παράδειγμα



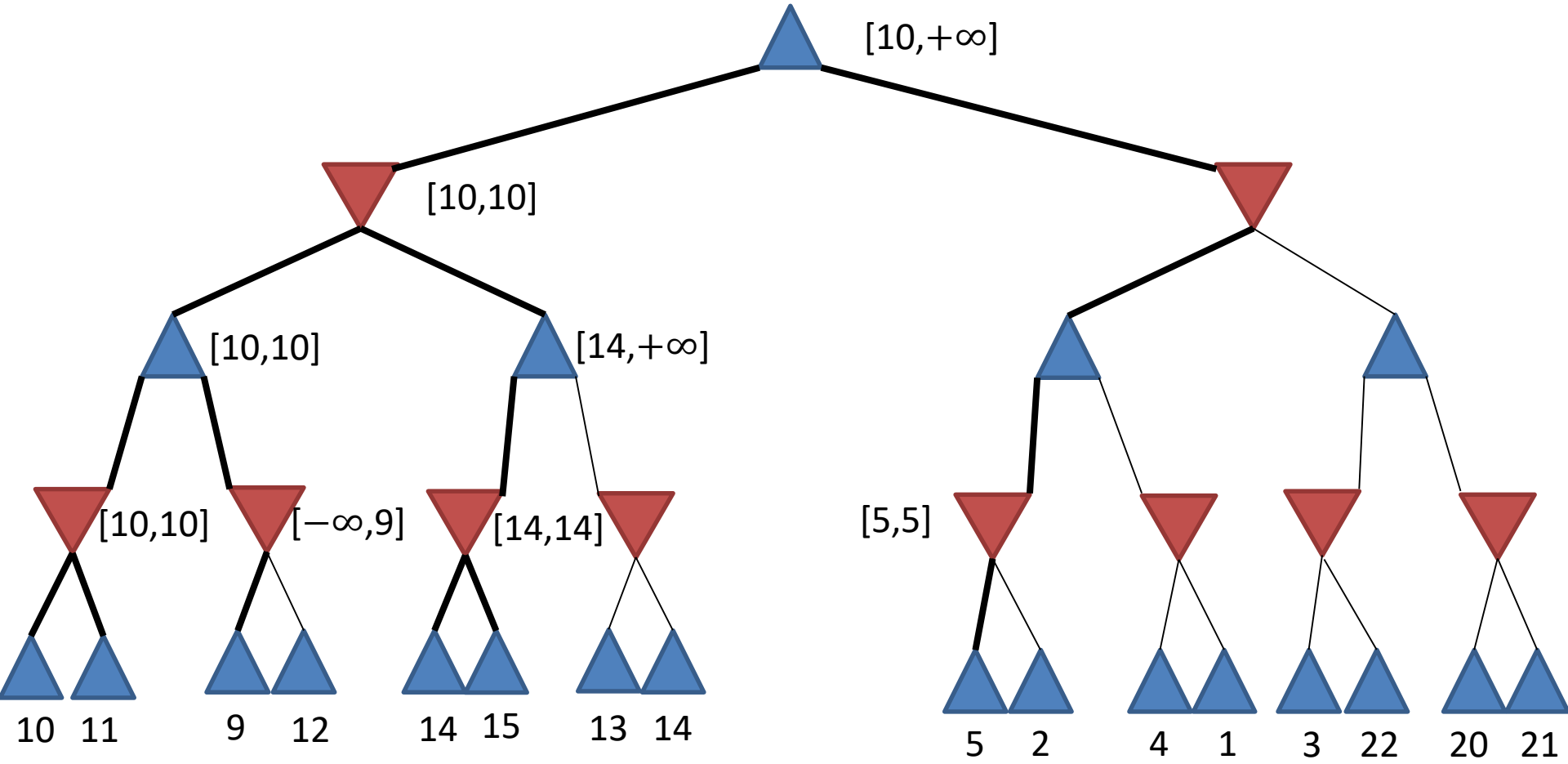
Παράδειγμα



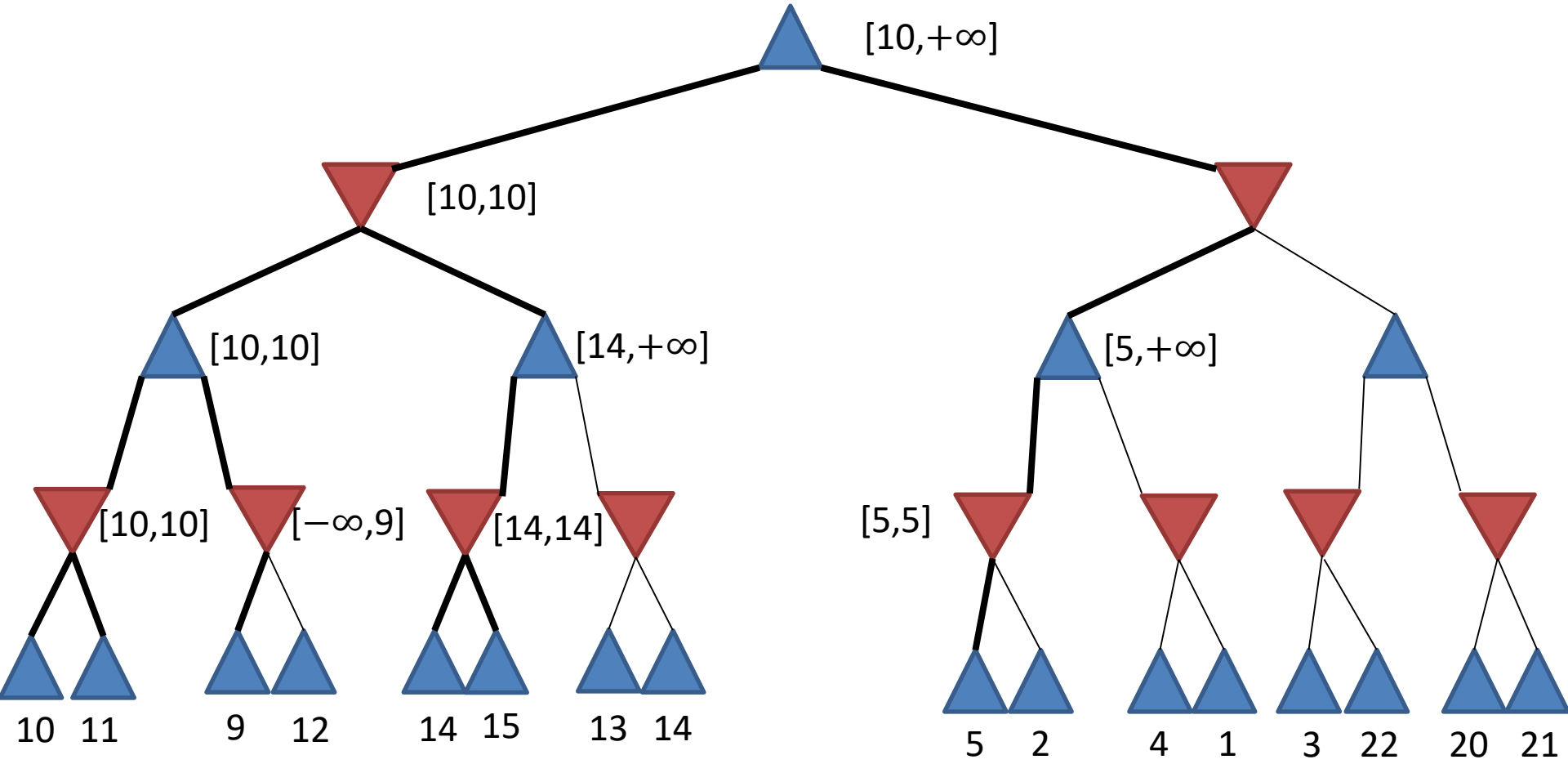
Παράδειγμα



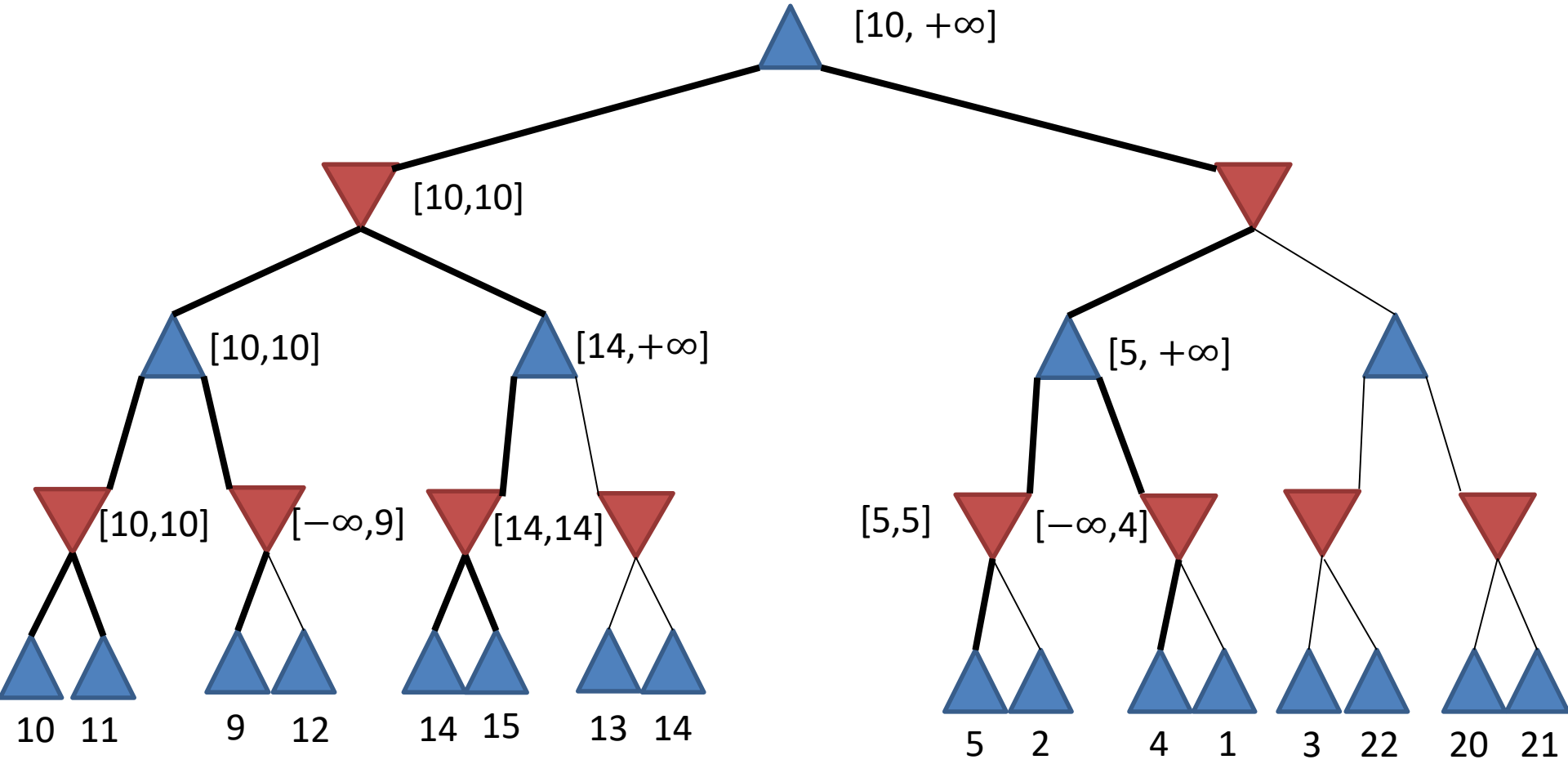
Παράδειγμα



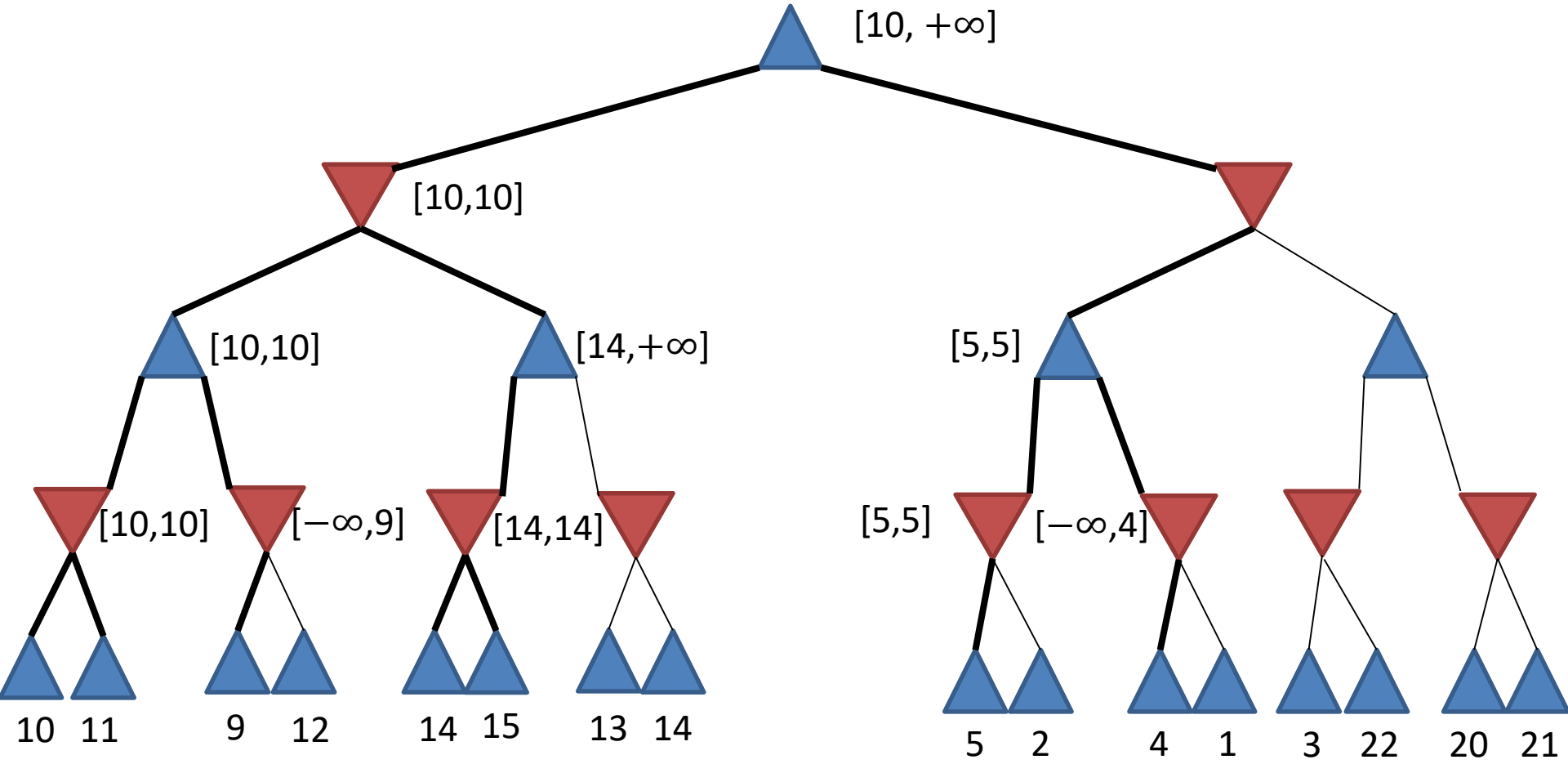
Παράδειγμα


[10,+∞]
[10,10]
[10,10]
[14,+∞]
[5,+∞]
[10,10]
[−∞,9]
[14,14]
[5,5]
10 11 9 12 14 15 13 14
5 2 4 1 3 22 20 21

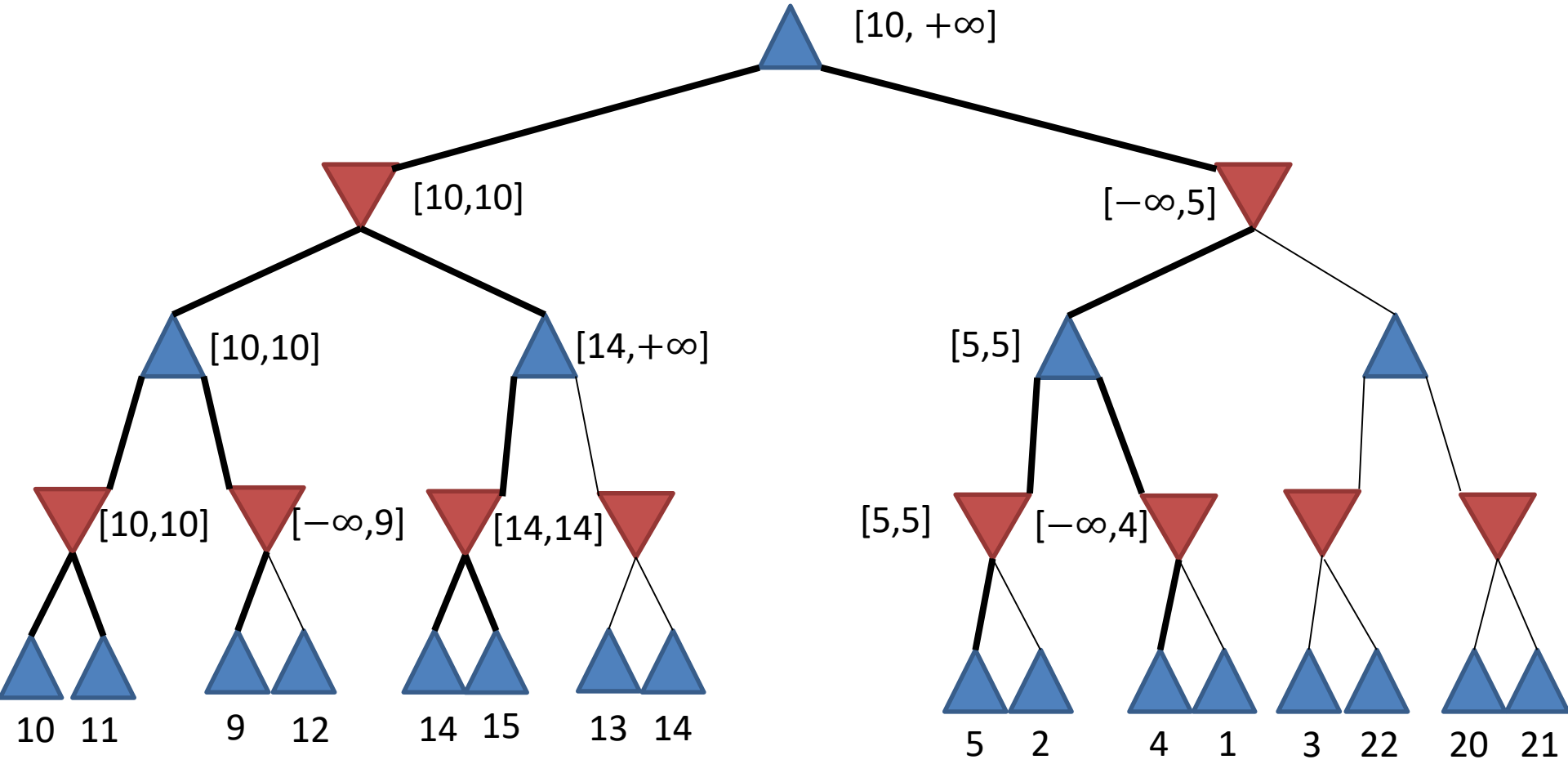
Παράδειγμα


[10, +∞]
[10,10]
[10,10]
[14,+∞]
[5, +∞]
[10,10]
[−∞,9]
[14,14]
[5,5]
[−∞,4]
10 11 9 12 14 15 13 14
5 2 4 1 3 22 20 21

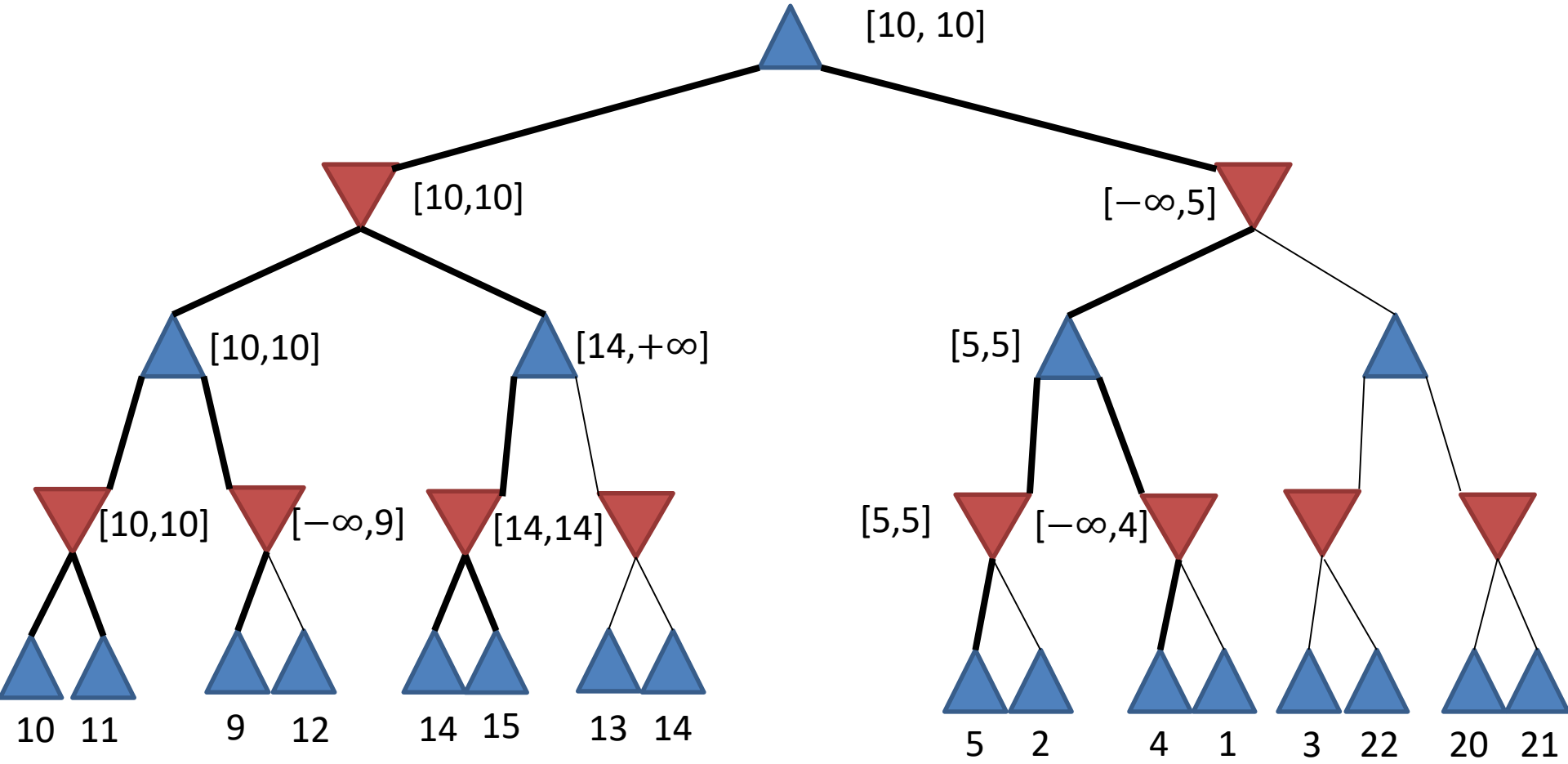
Παράδειγμα


[10, +∞]
[10,10]
[10,10]
[14,+∞]
[5,5]
[10,10]
[−∞,9]
[14,14]
[5,5]
[−∞,4]
10 11 9 12 14 15 13 14
5 2 4 1 3 22 20 21

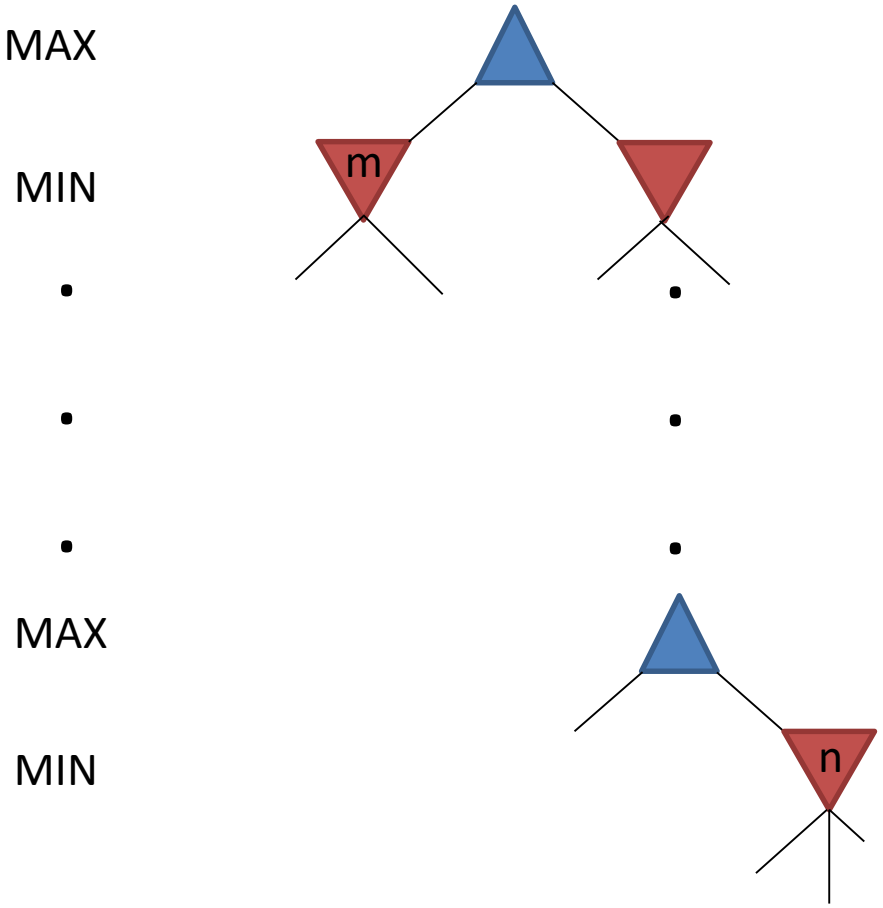
Παράδειγμα


[10, +∞]
[10,10]
[−∞,5]
[10,10]
[14,+∞]
[5,5]
[10,10]
[−∞,9]
[14,14]
[5,5]
[−∞,4]
10 11 9 12 14 15 13 14
5 2 4 1 3 22 20 21

Παράδειγμα


[10, 10]
[10,10]
[−∞,5]
[10,10]
[14,+∞]
[5,5]
[10,10]
[−∞,9]
[14,14]
[5,5]
[−∞,4]
10 11 9 12 14 15 13 14
5 2 4 1 3 22 20 21

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
  $v \leftarrow$MAX-VALUE($state, -\infty, +\infty$)
  return the $action$ in ACTIONS($state$) with value v

function MAX-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**
    $v \leftarrow$MAX(v,MIN-VALUE(RESULT($state, a$),α, β))
    if $v \geq \beta$ **then return** v
    $\alpha \leftarrow$MAX(α, 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**

    $v \leftarrow$MIN(v,MAX-VALUE(RESULT($state, a$),α, β))

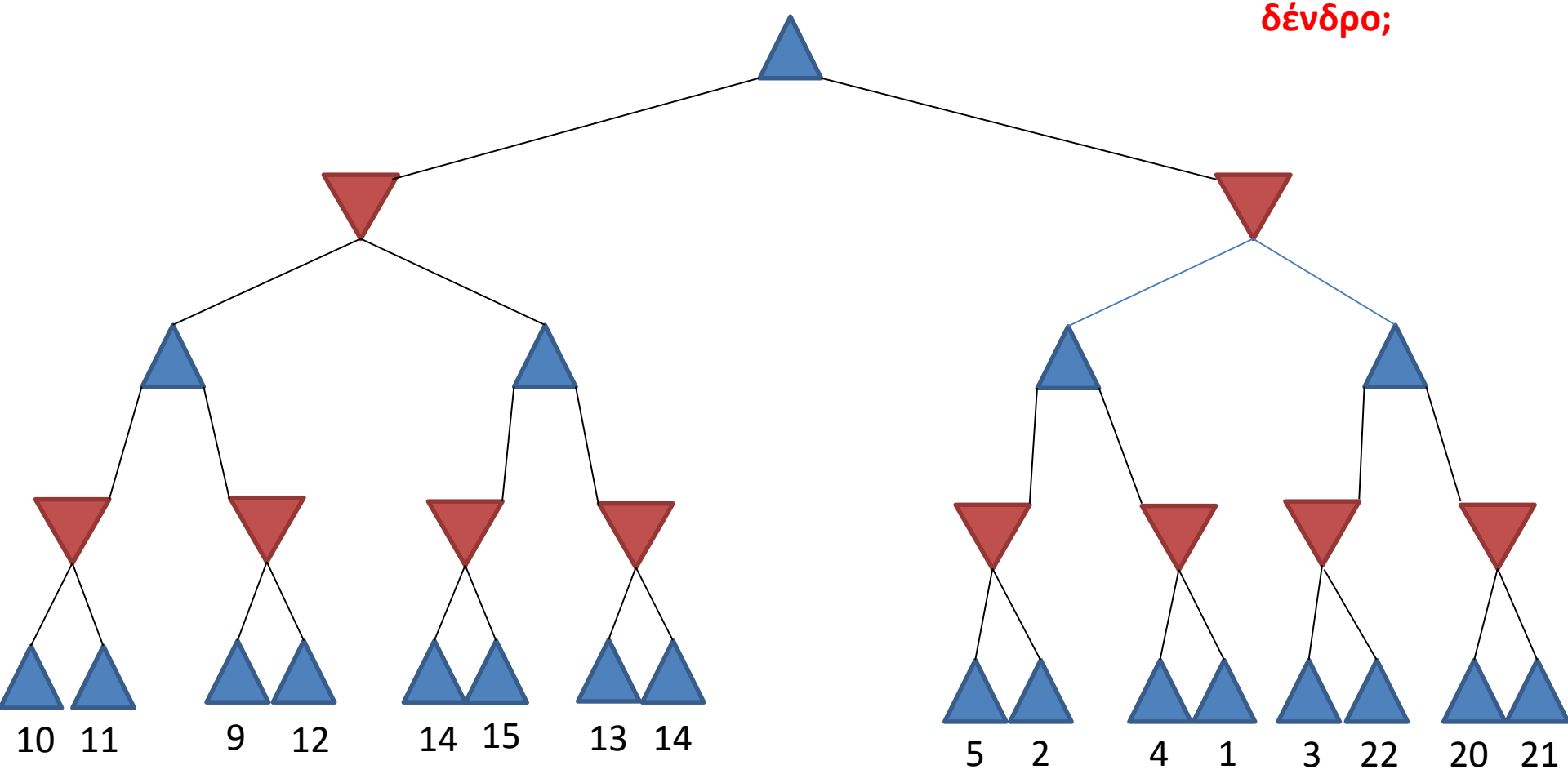
    if $v \leq \alpha$ **then return** v

    $\beta \leftarrow$MIN(β, v)

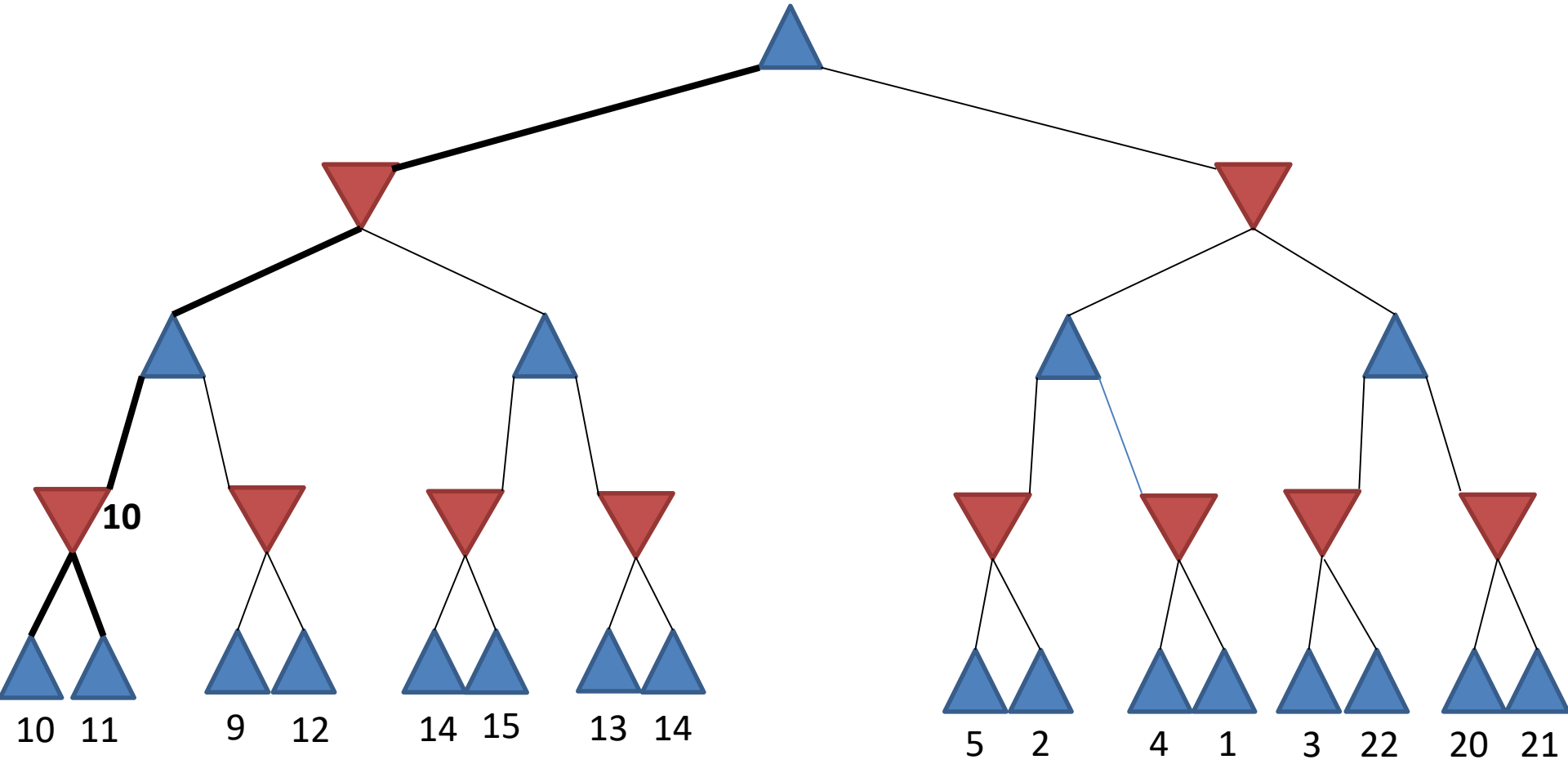
  return v

Παράδειγμα

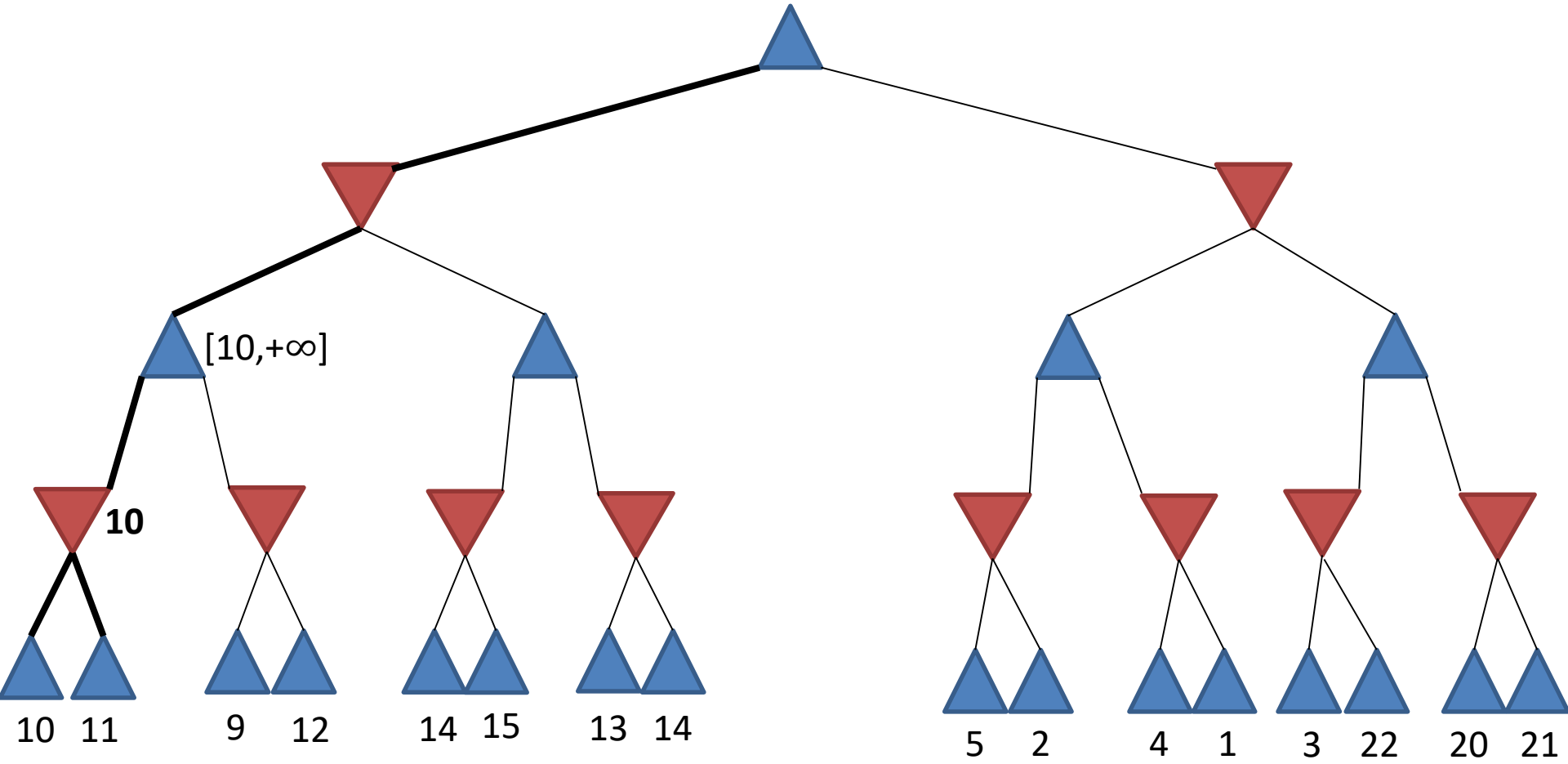
Ποιο είναι το αποτέλεσμα του αλγόριθμου στο παρακάτω δένδρο;



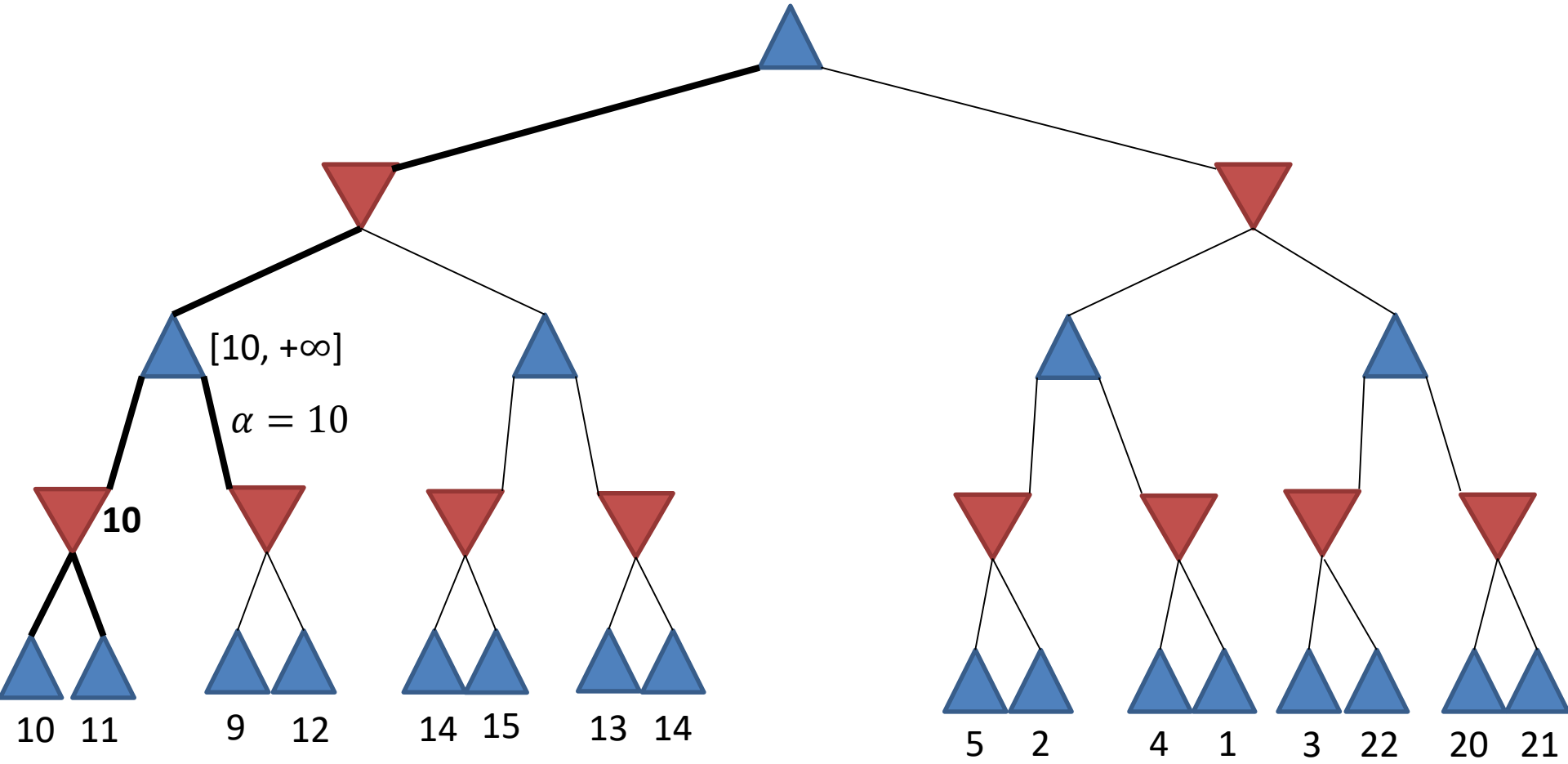
Παράδειγμα



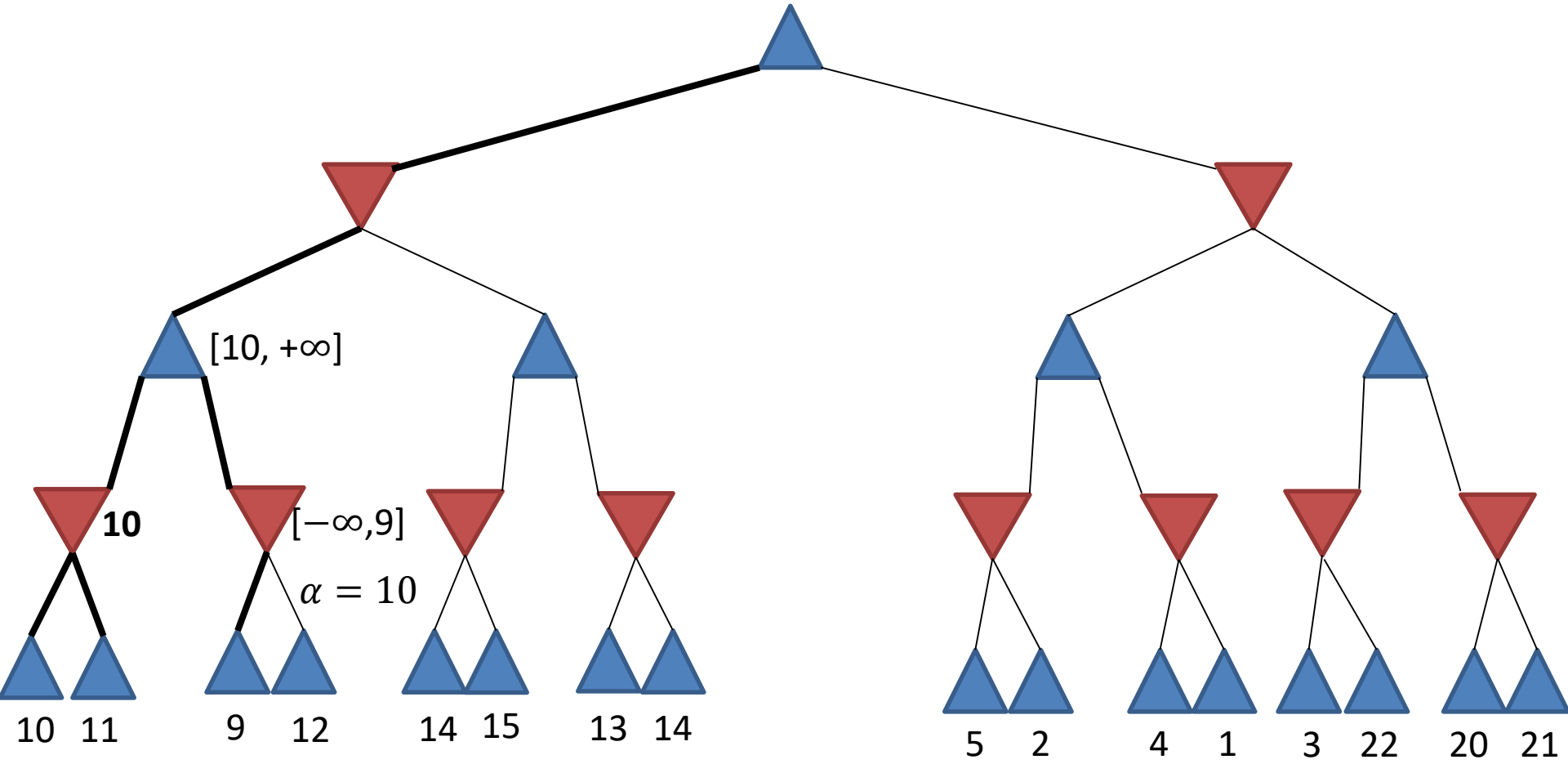
Παράδειγμα



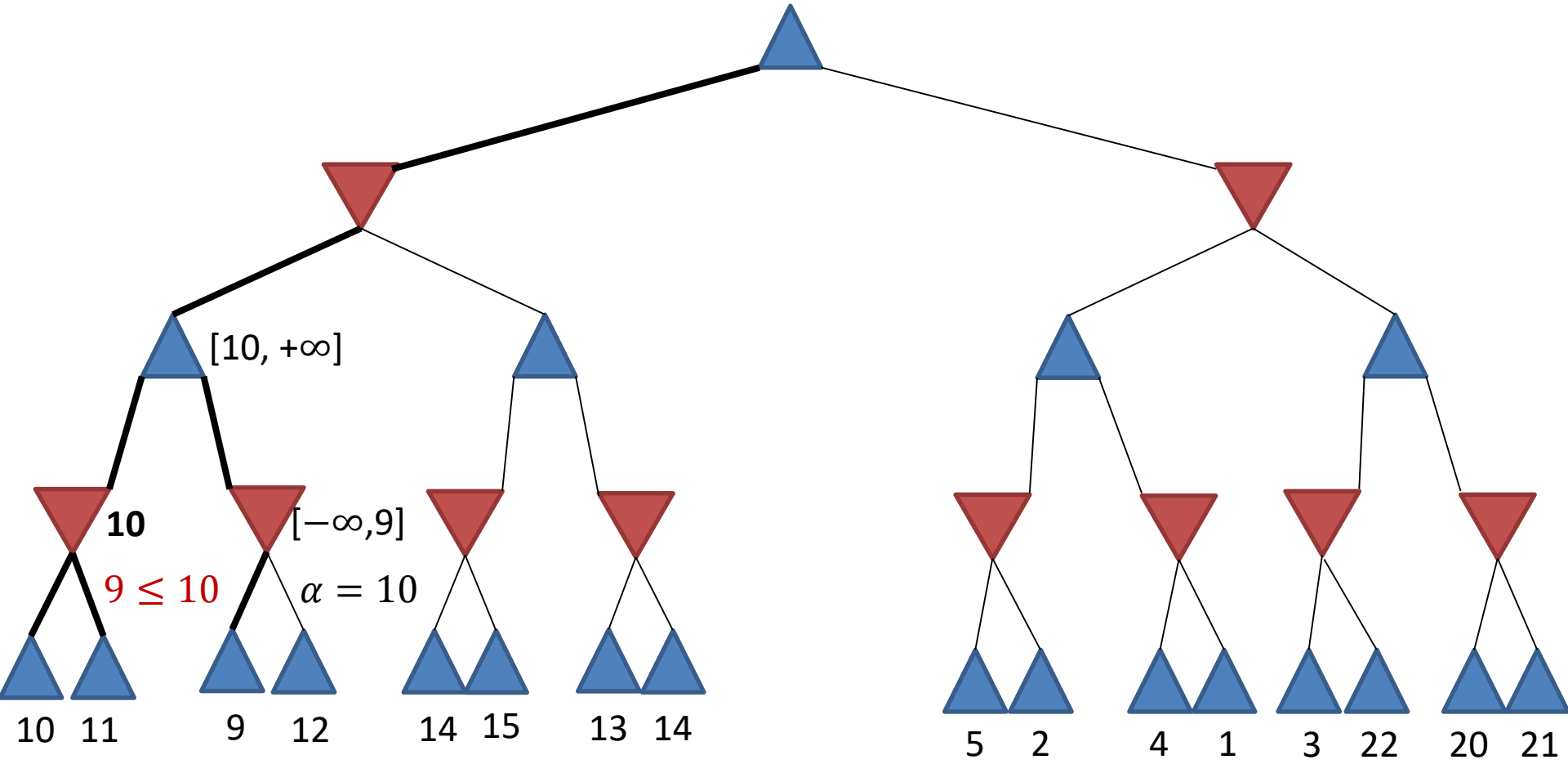
Παράδειγμα



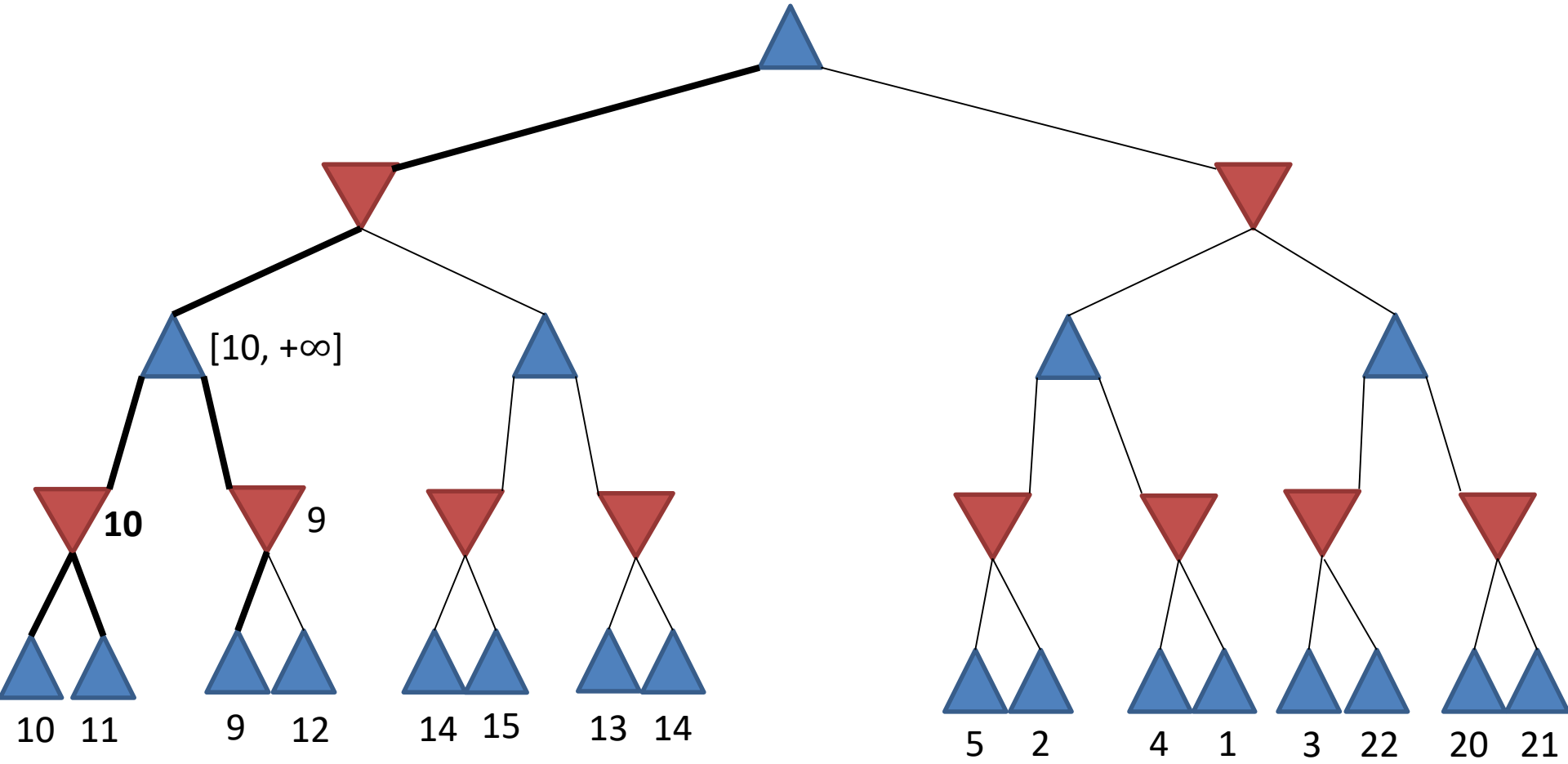
Παράδειγμα



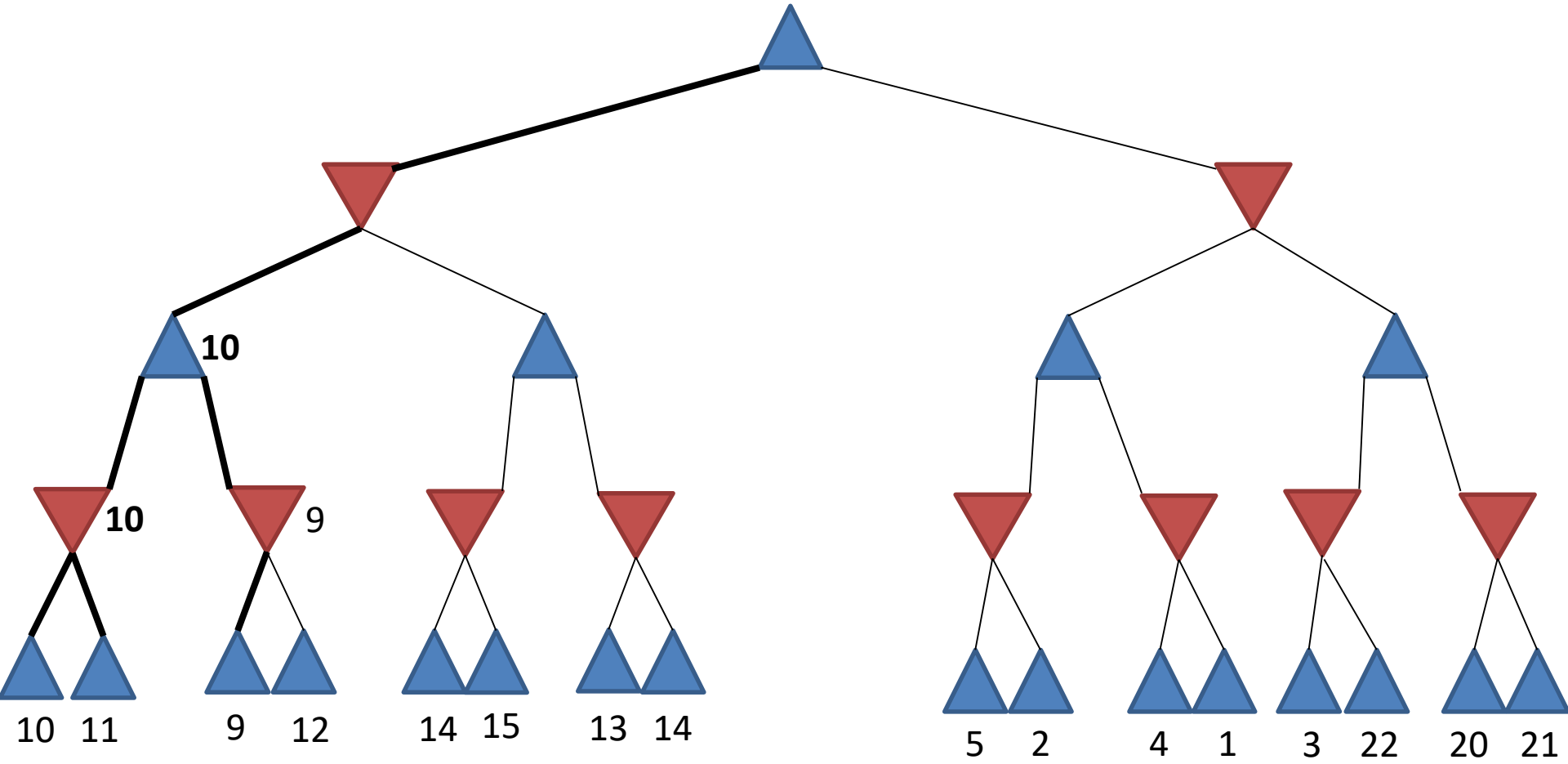
Παράδειγμα



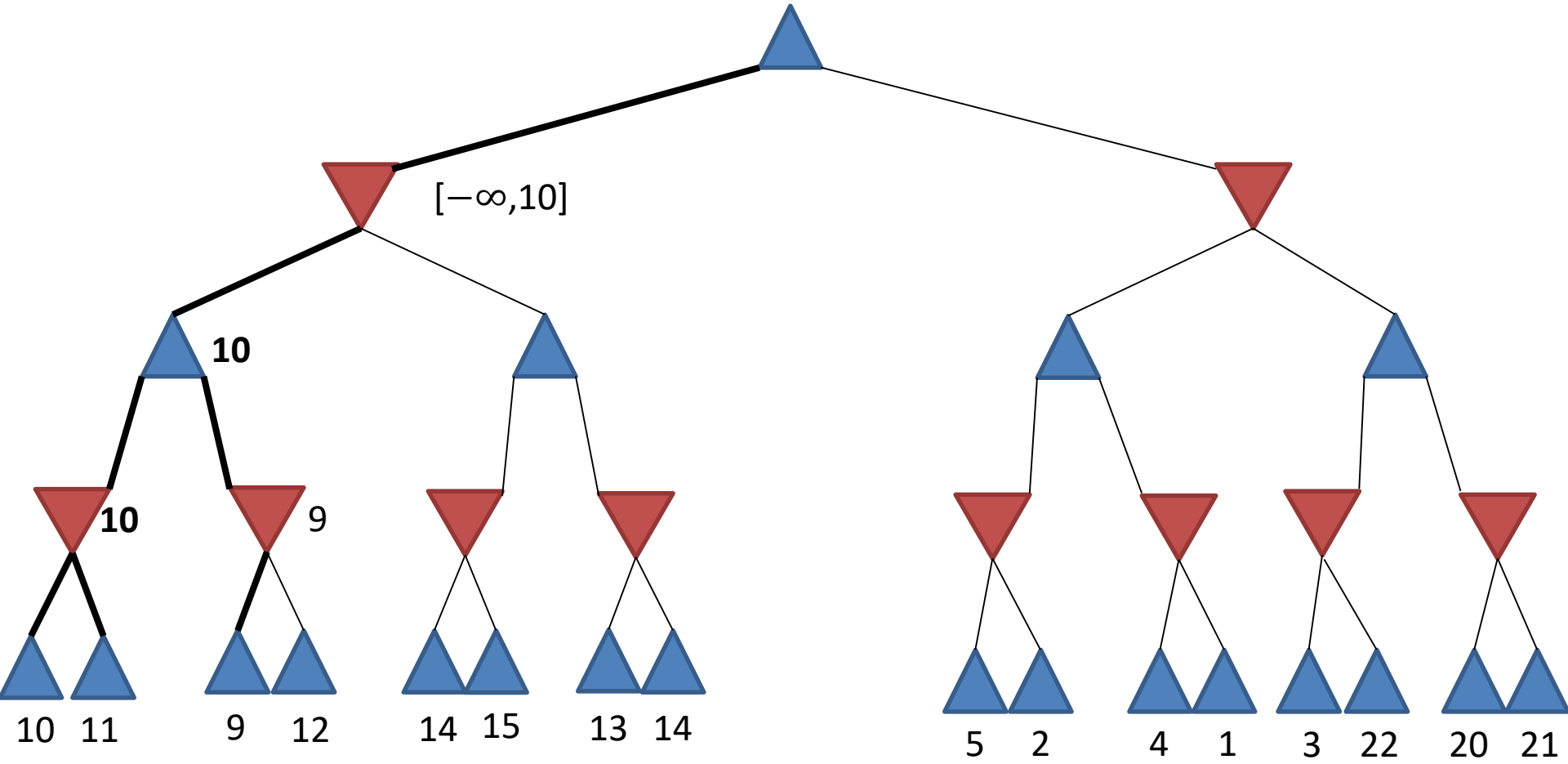
Παράδειγμα



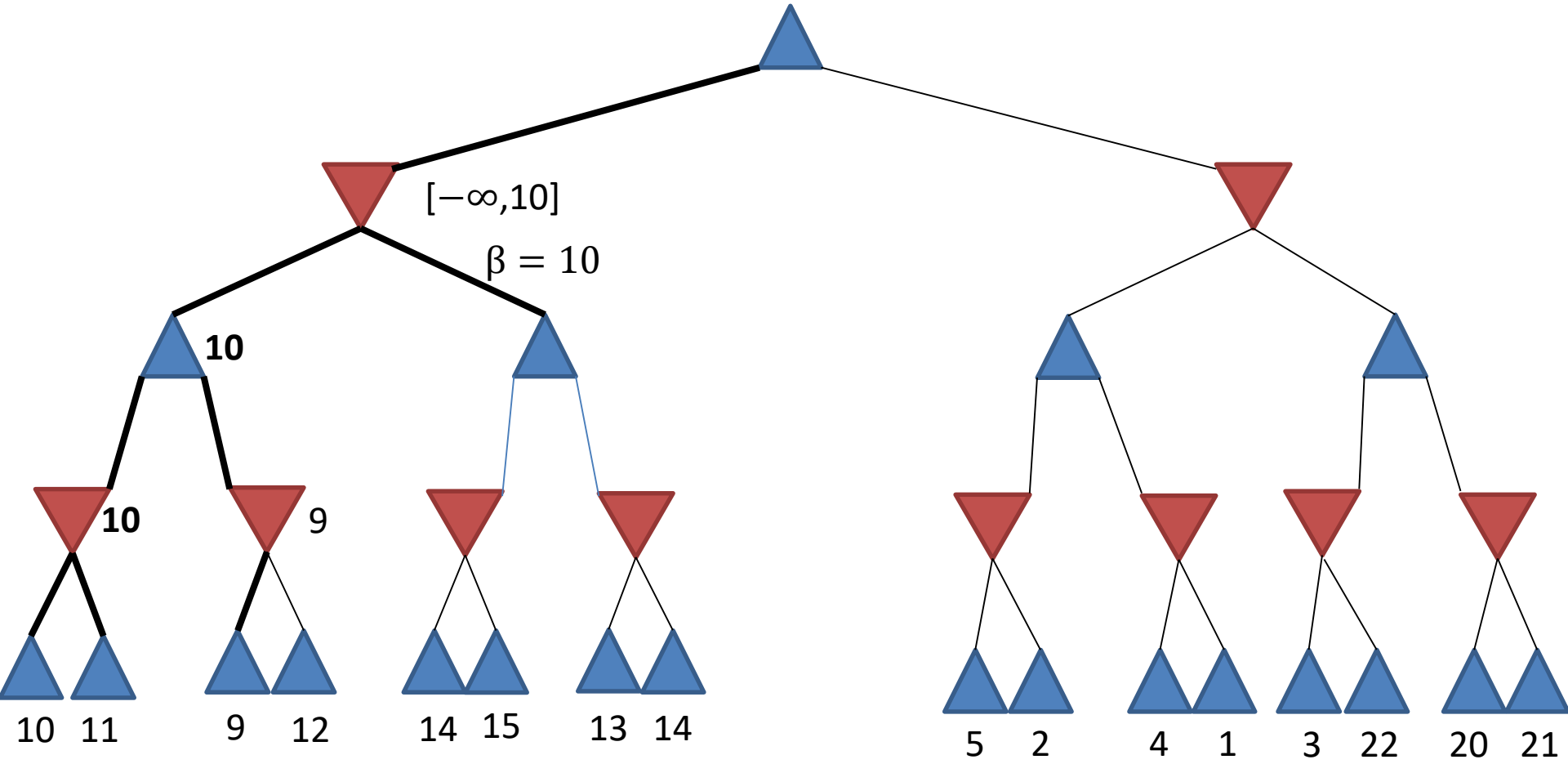
Παράδειγμα



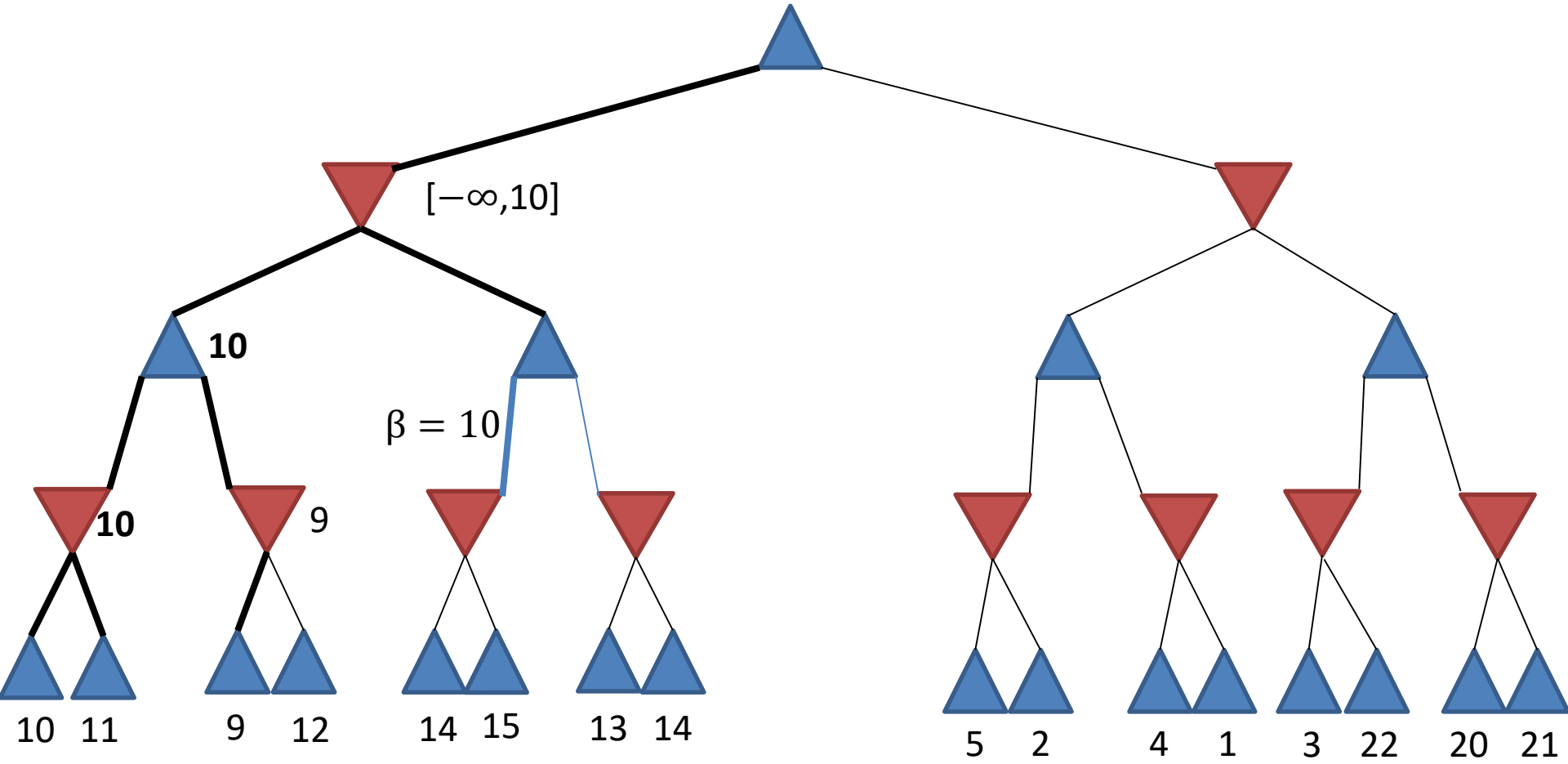
Παράδειγμα



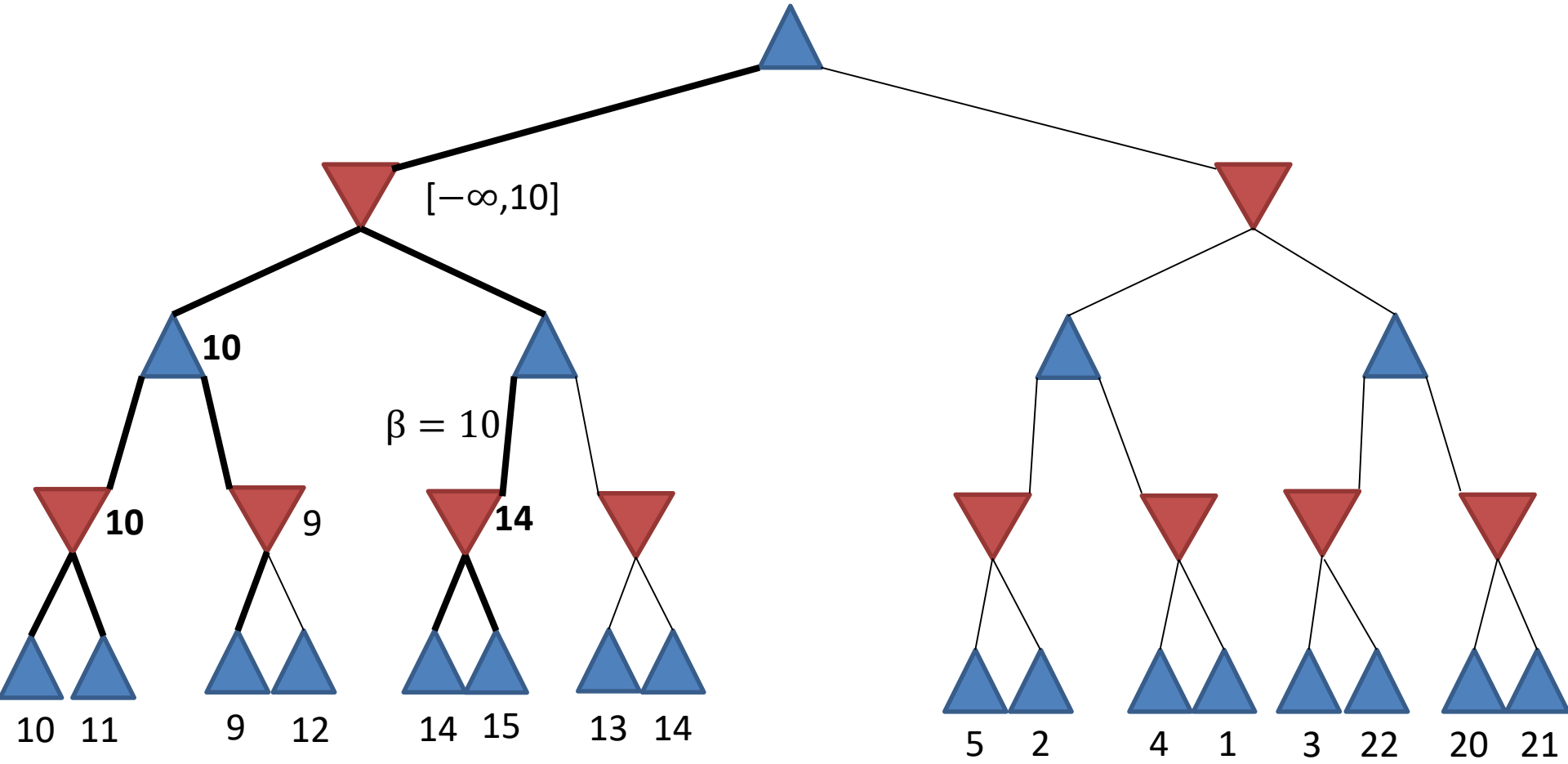
Παράδειγμα



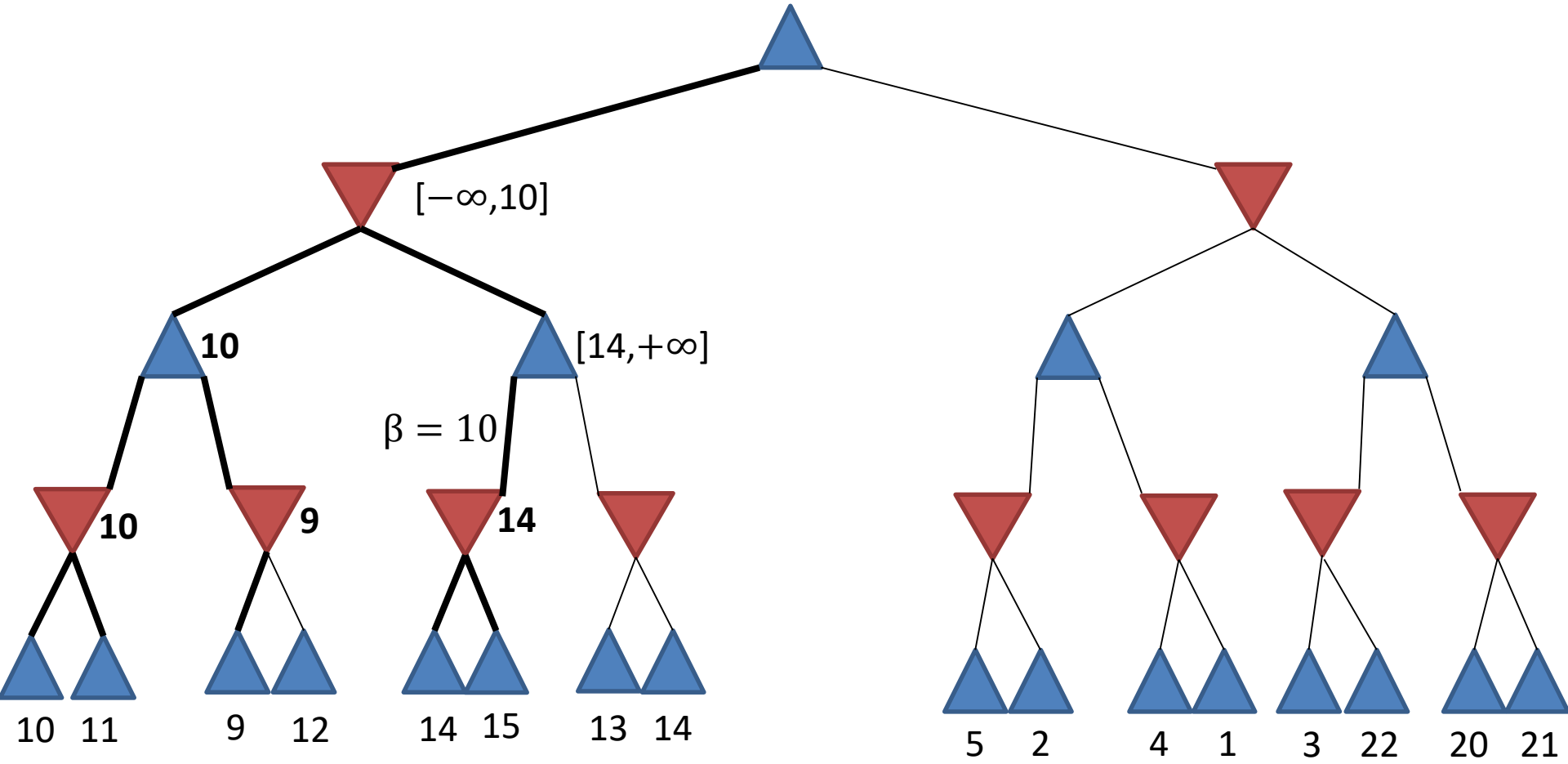
Παράδειγμα



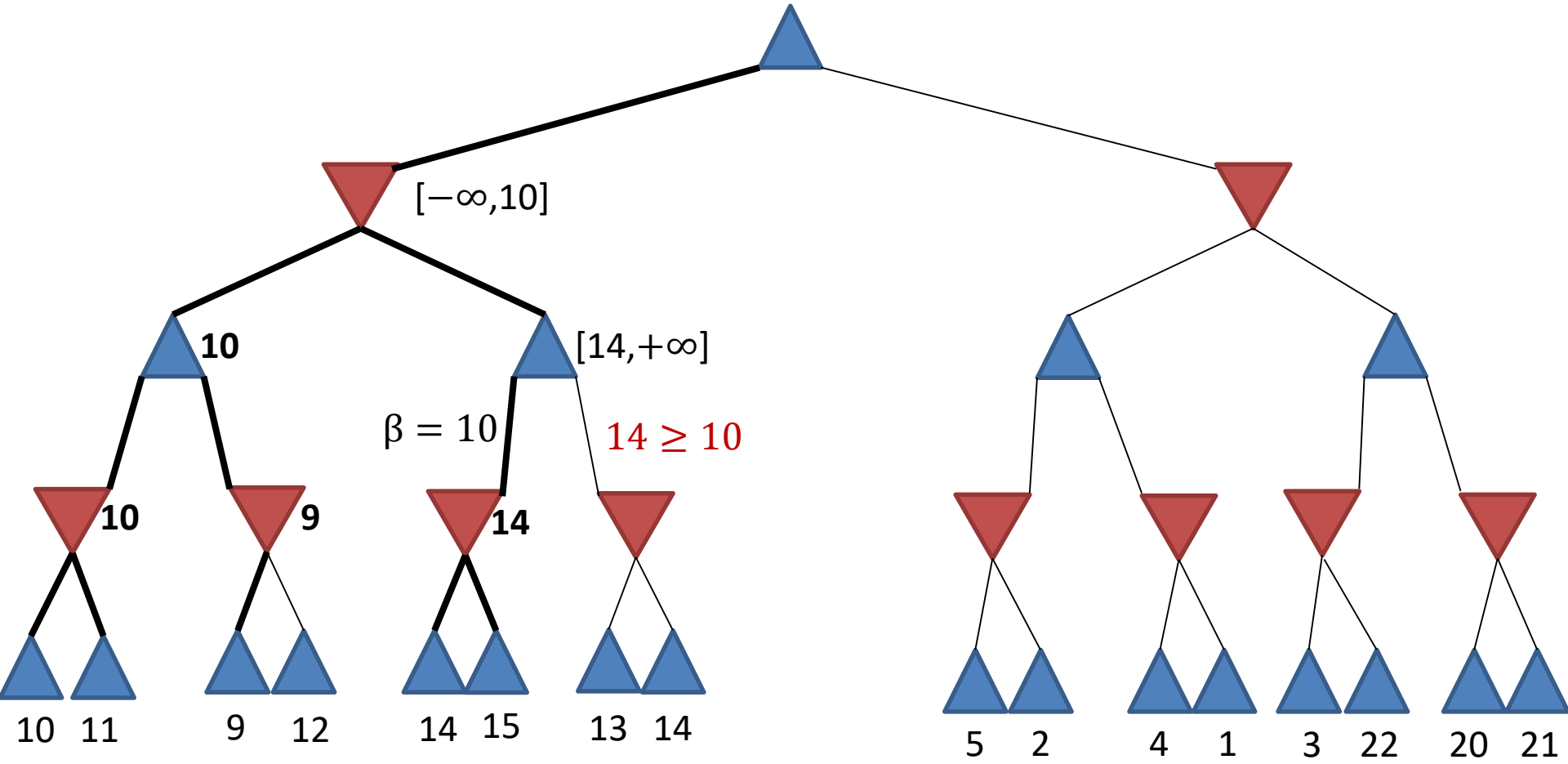
Παράδειγμα



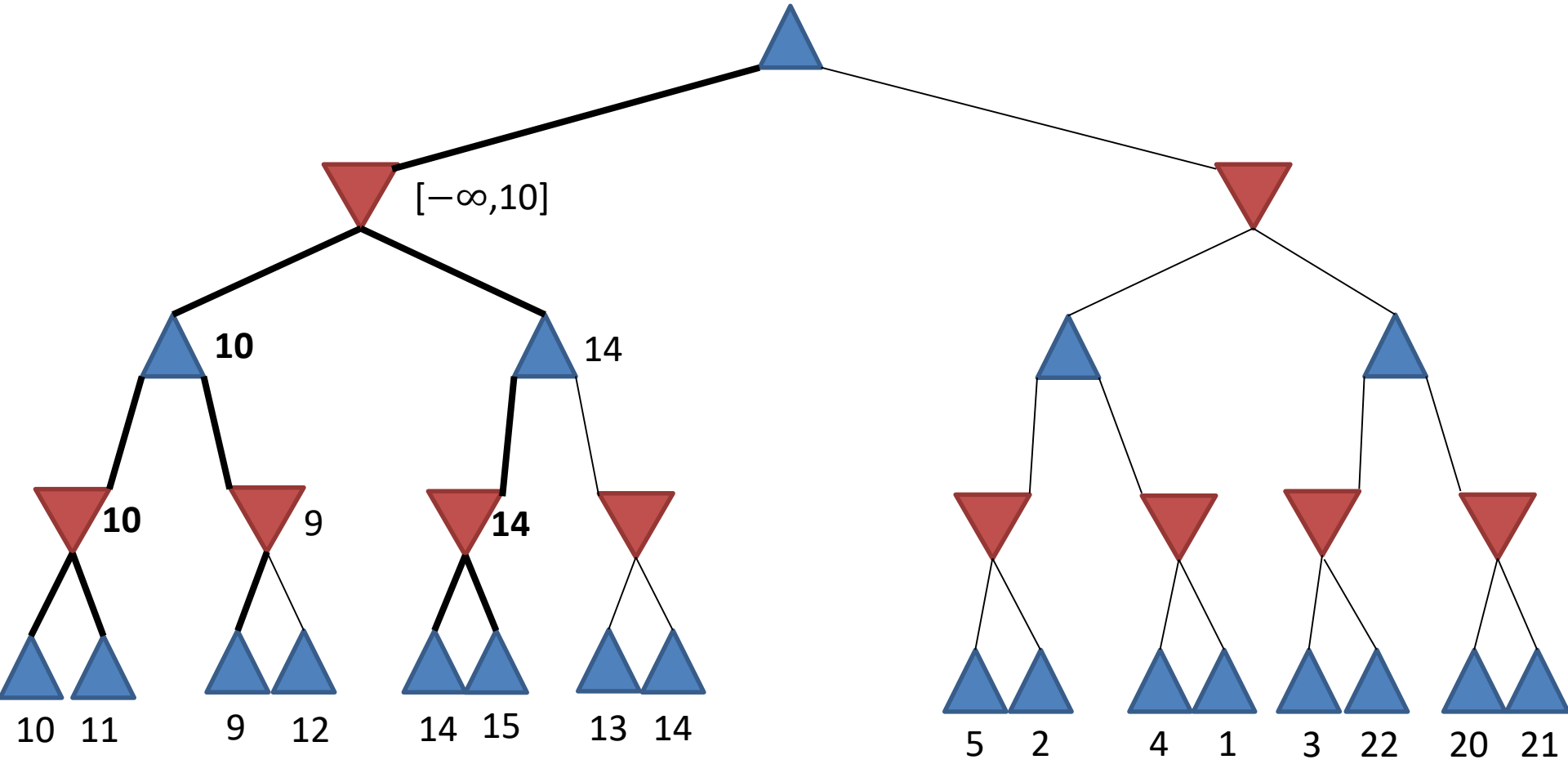
Παράδειγμα



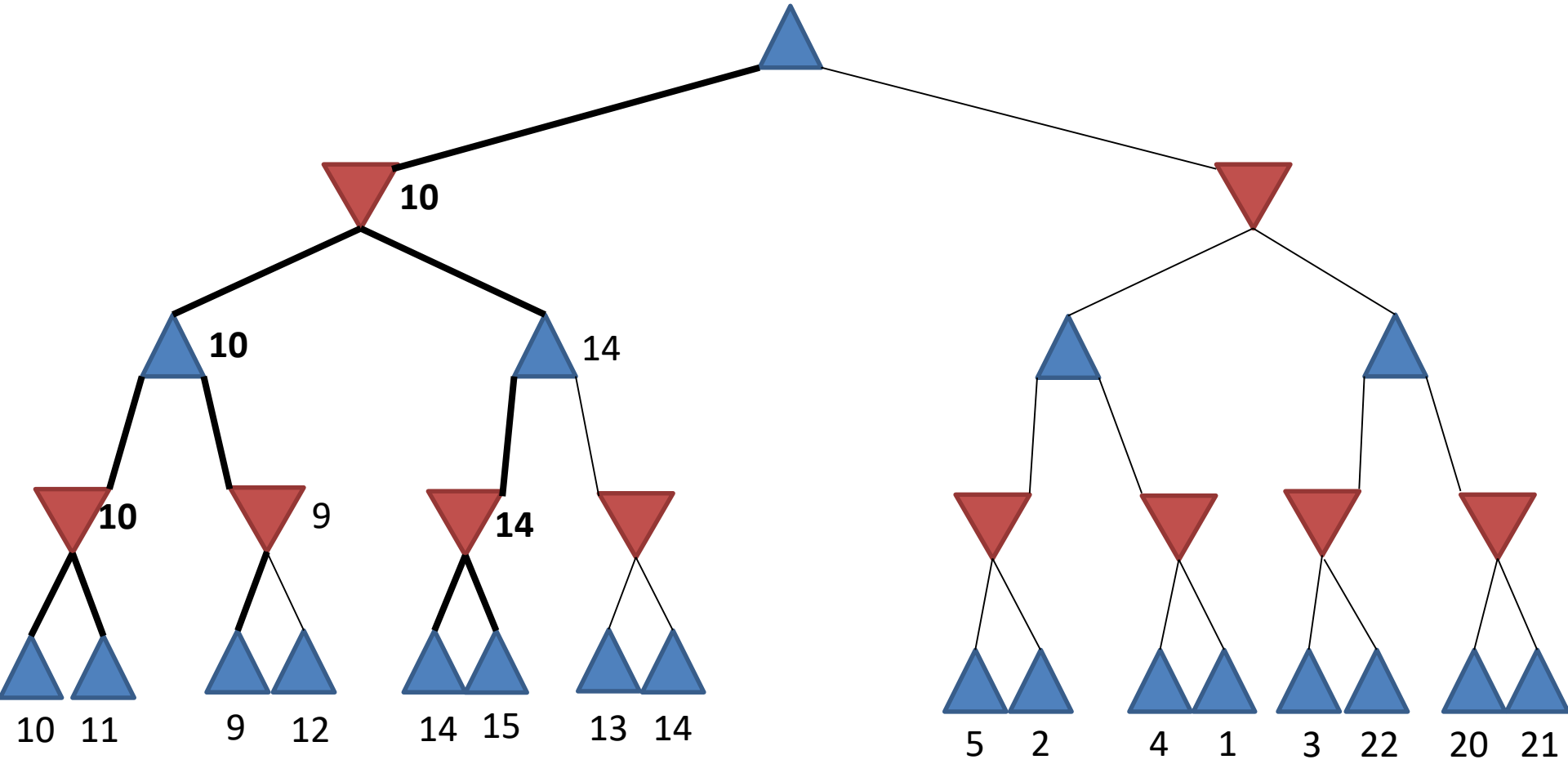
Παράδειγμα



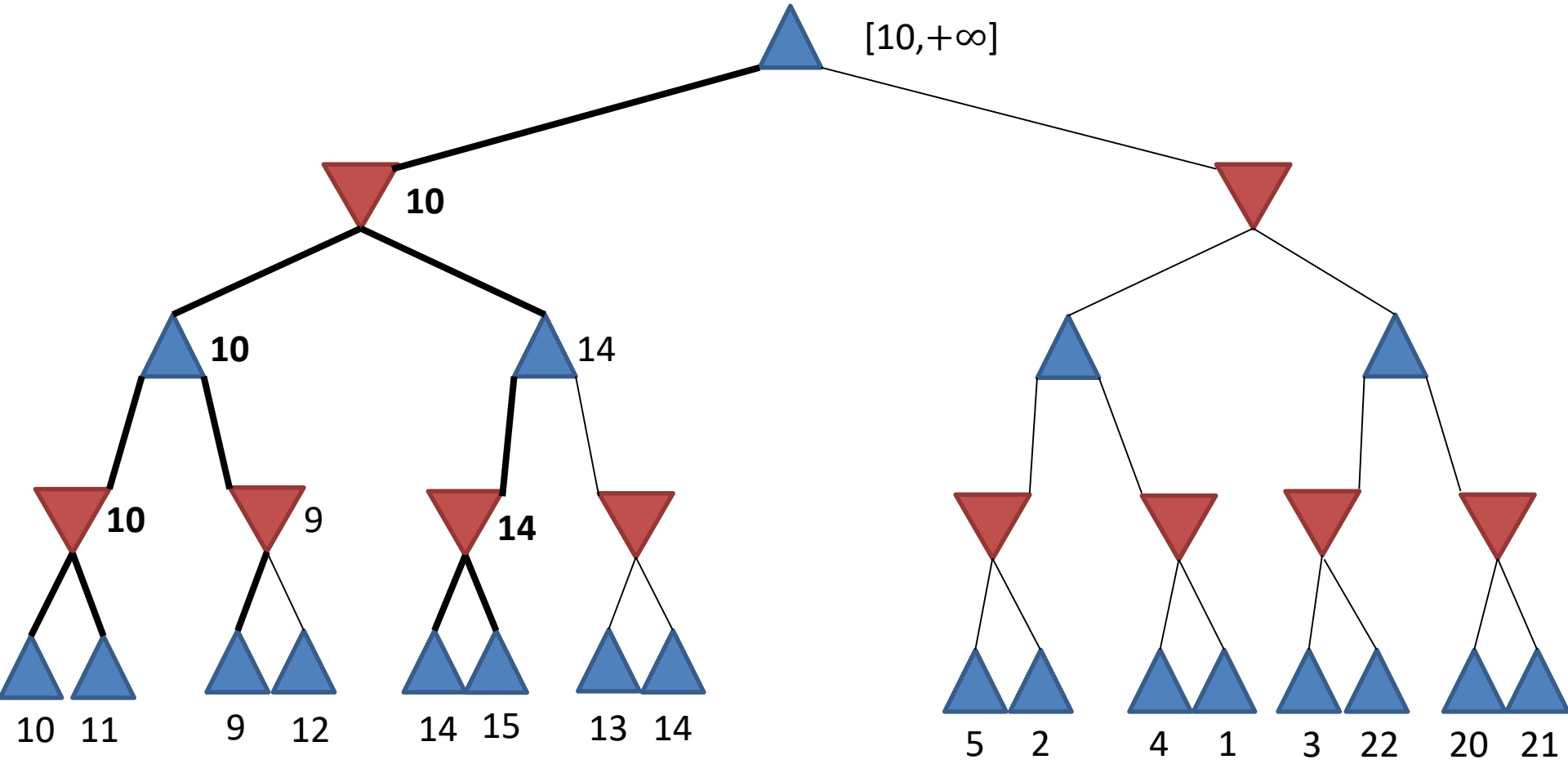
Παράδειγμα



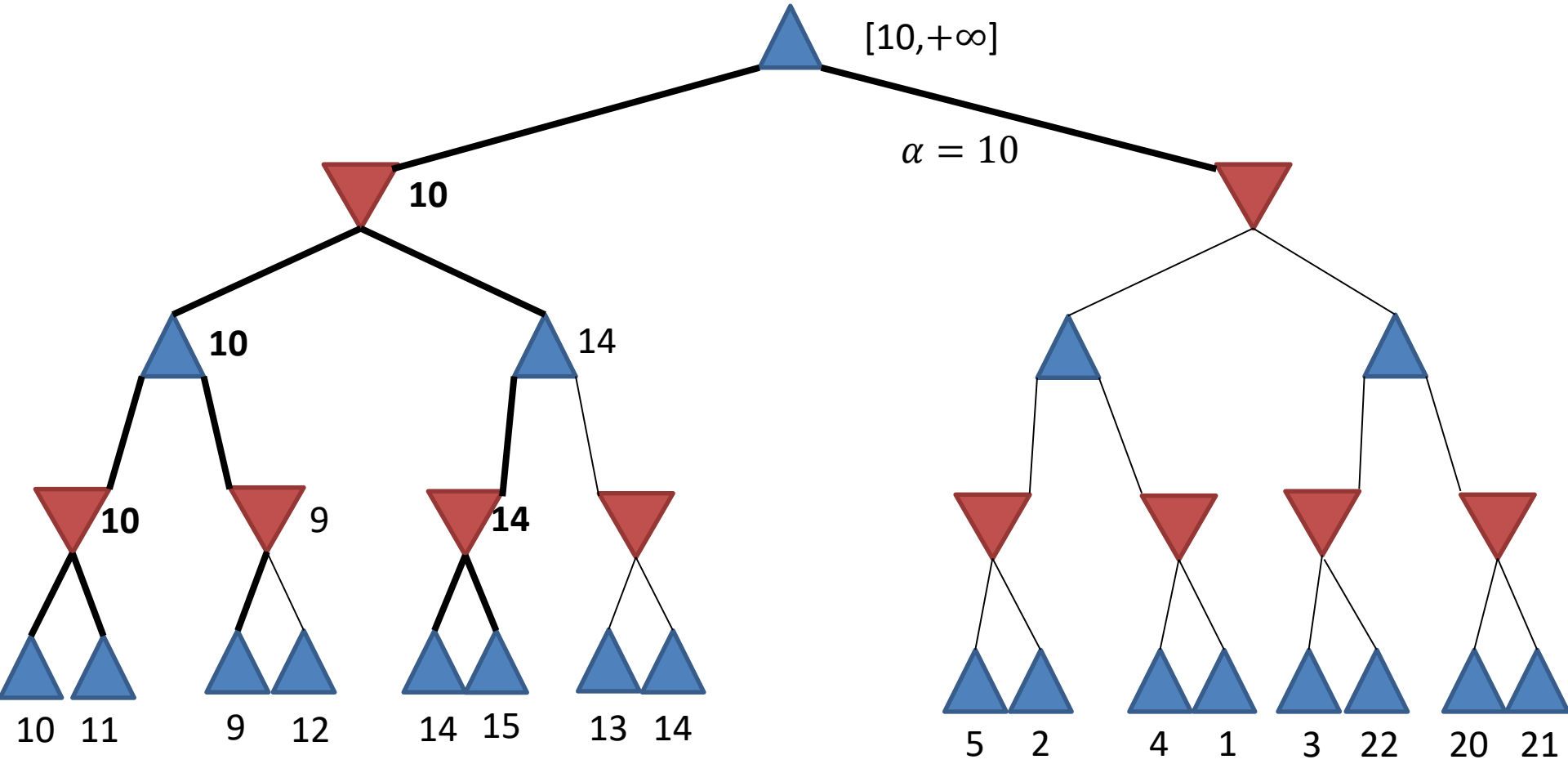
Παράδειγμα



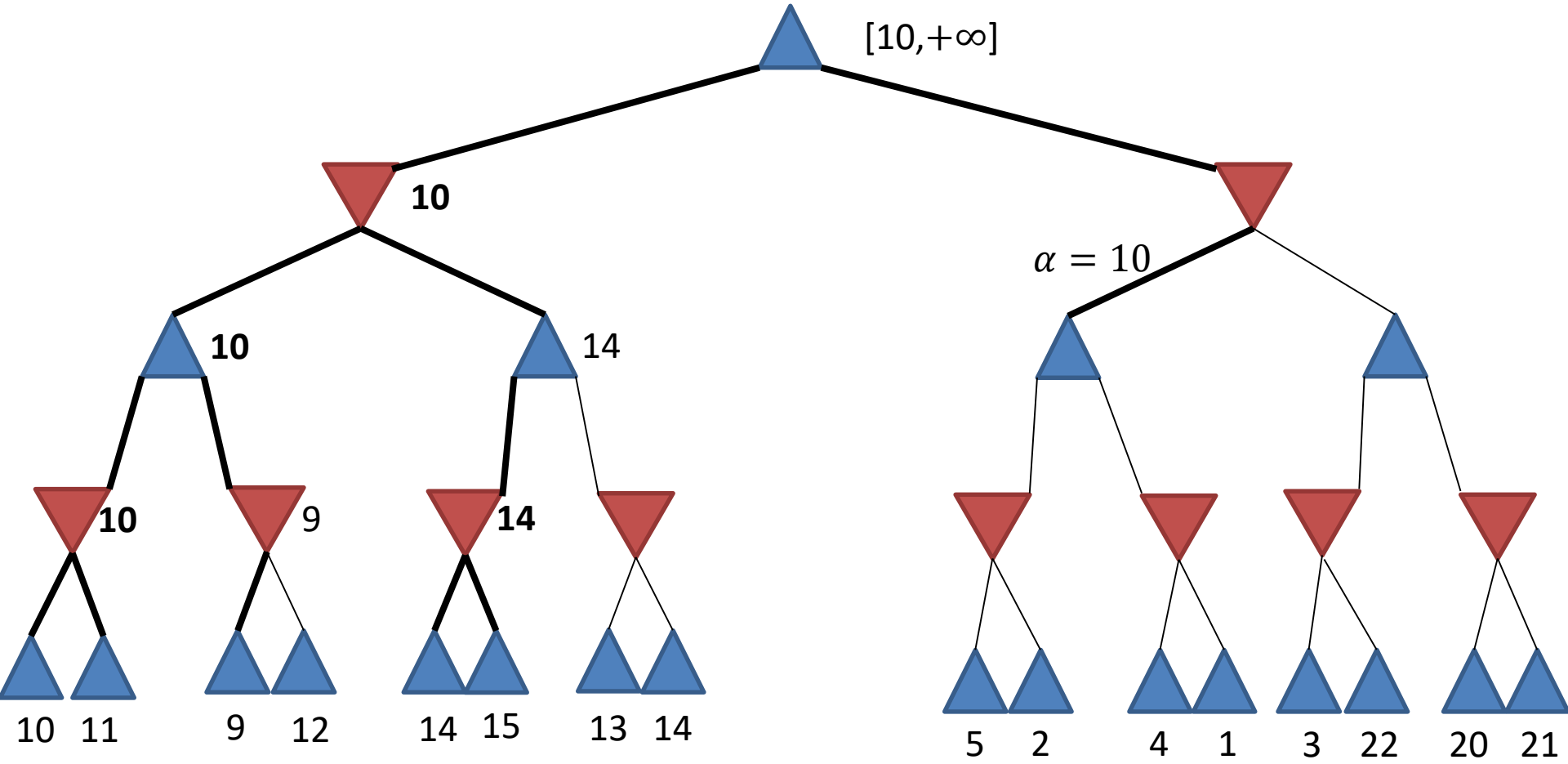
Παράδειγμα



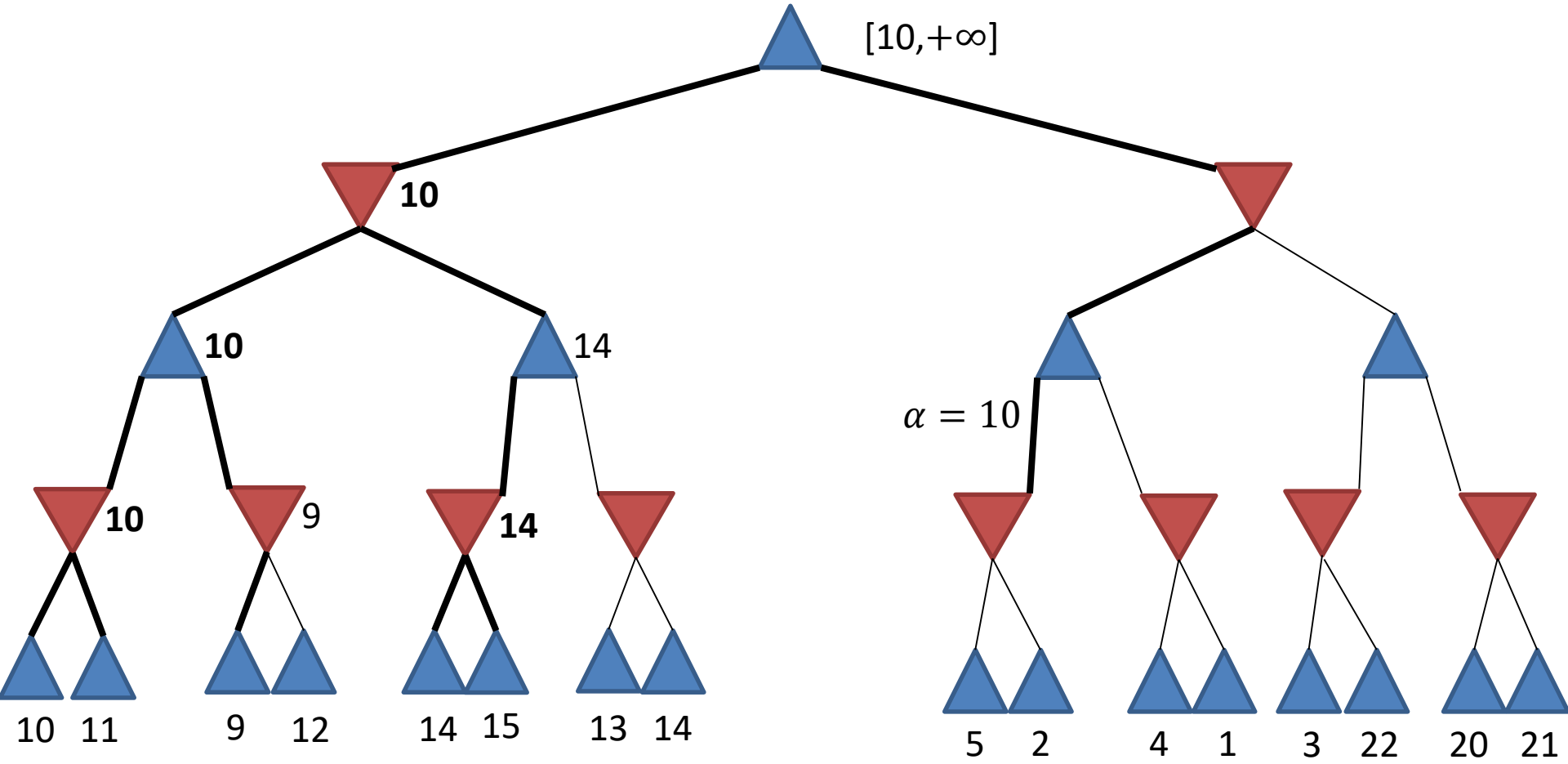
Παράδειγμα



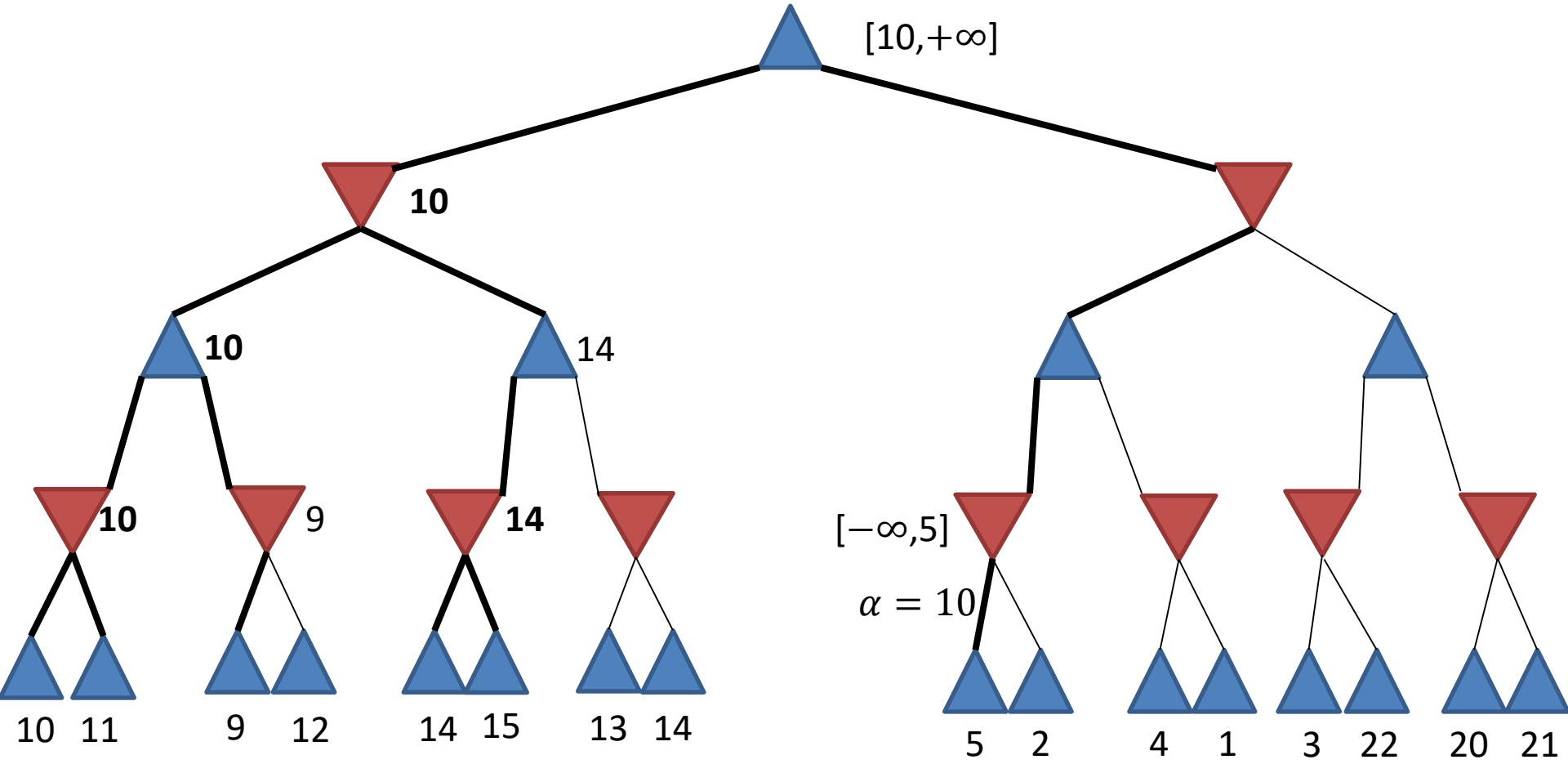
Παράδειγμα



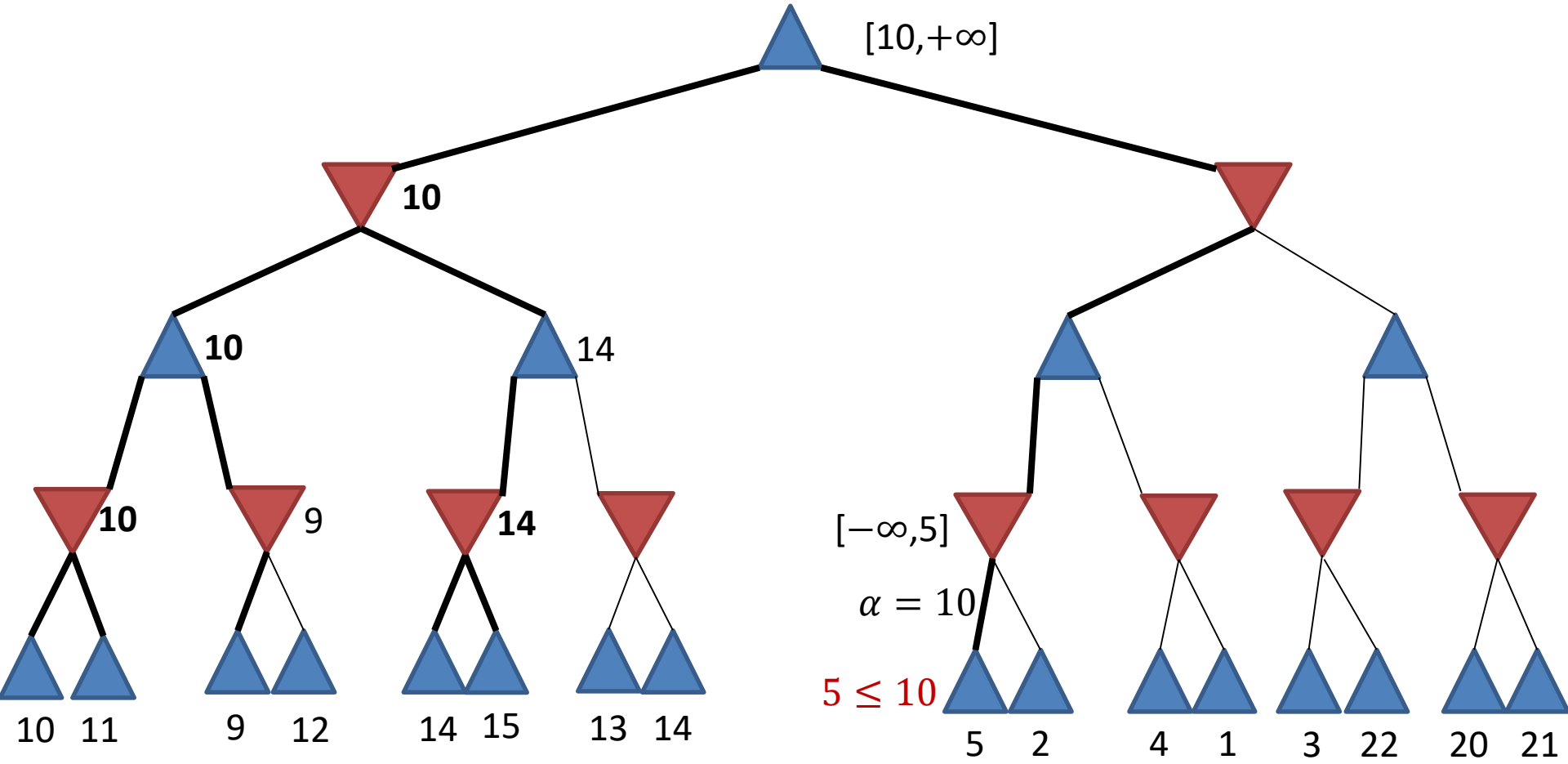
Παράδειγμα



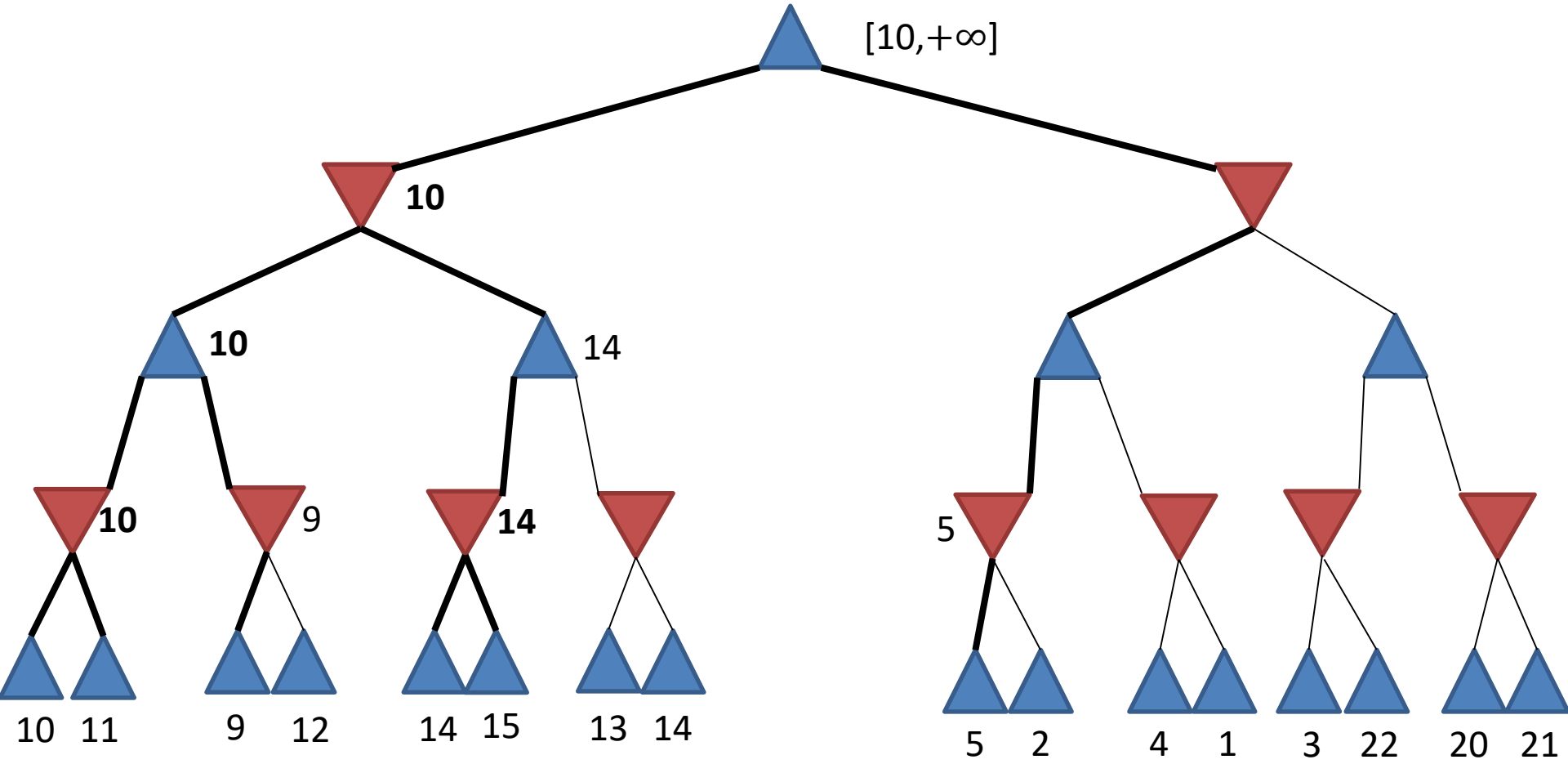
Παράδειγμα



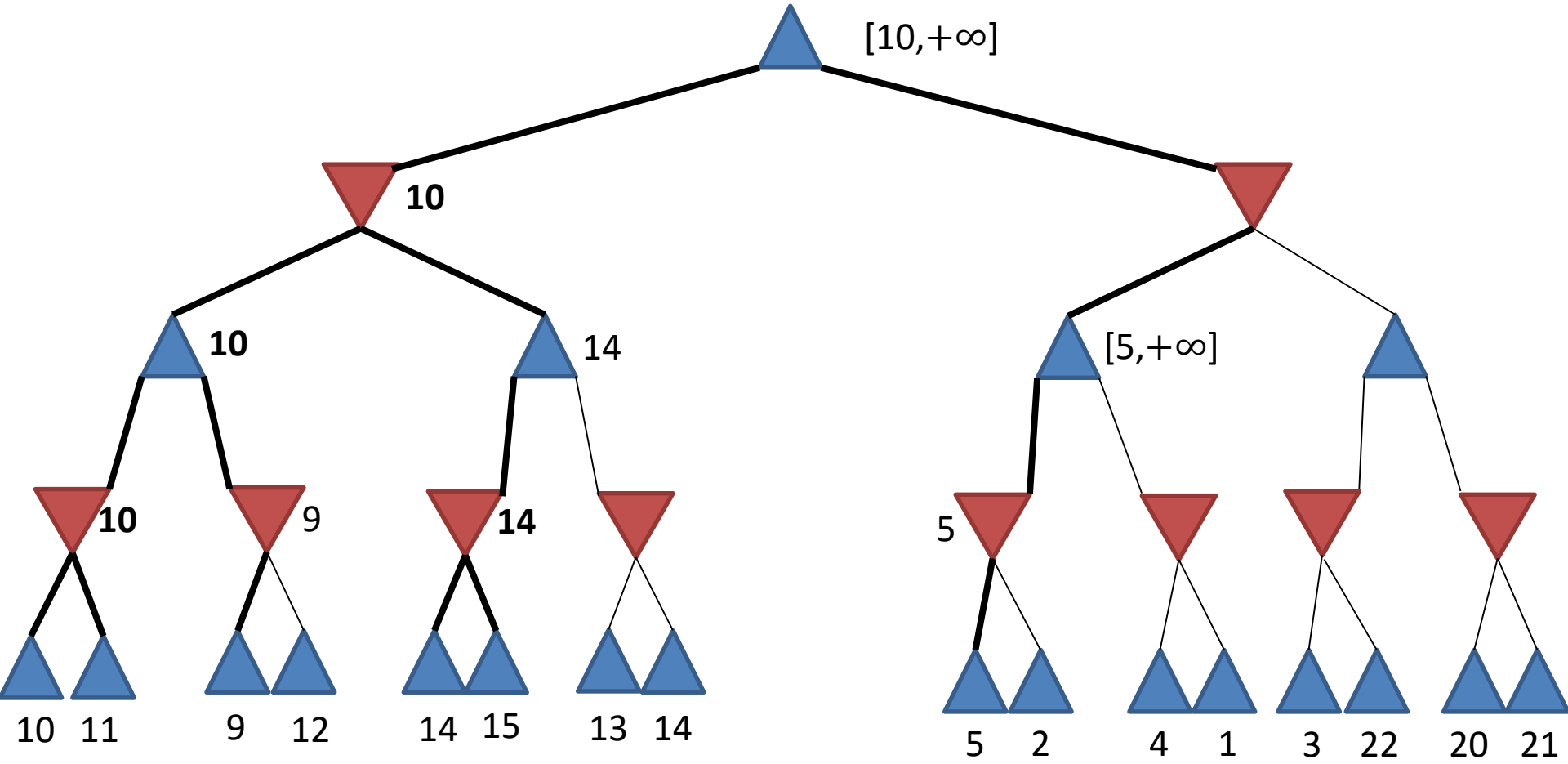
Παράδειγμα



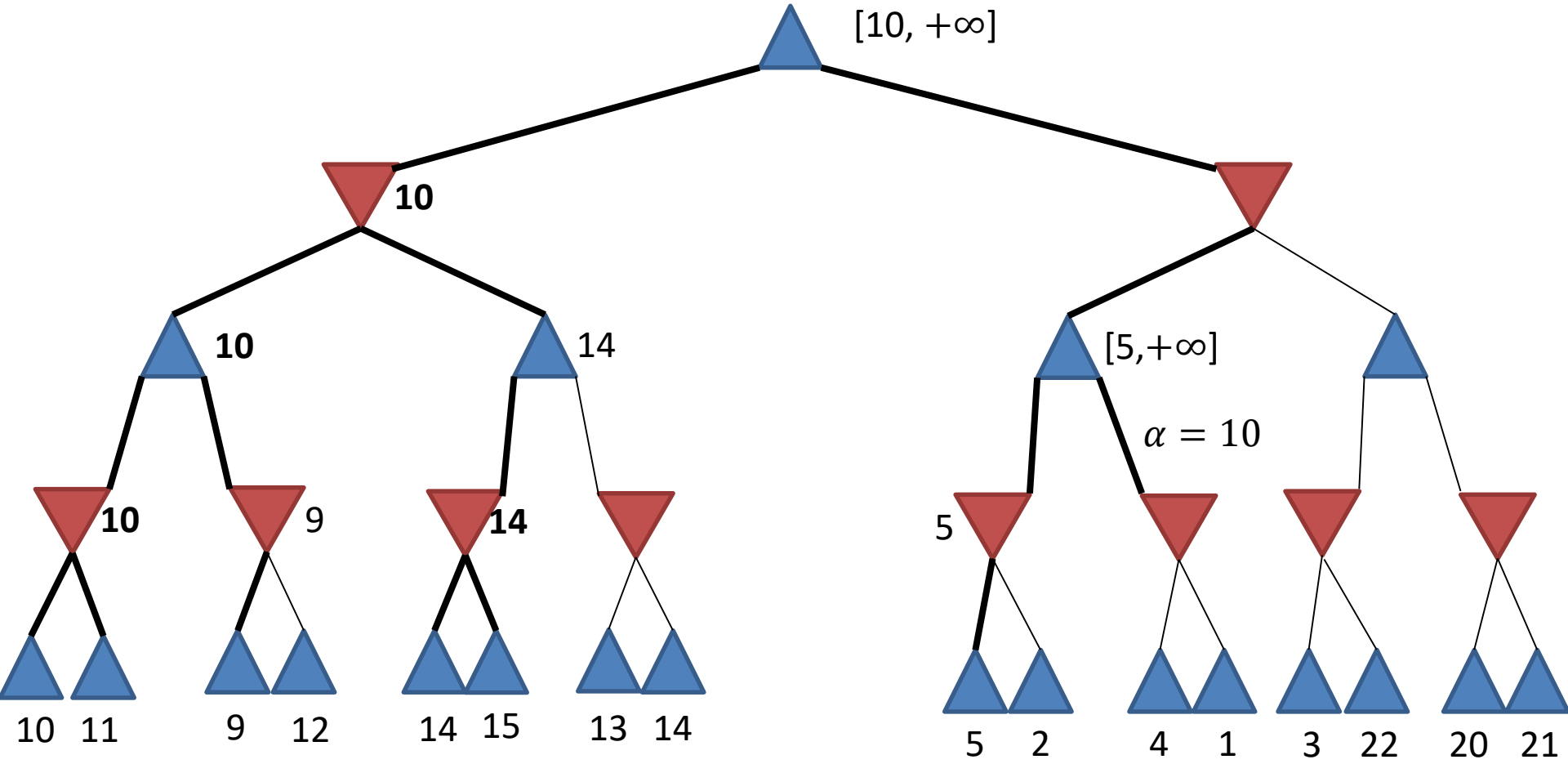
Παράδειγμα



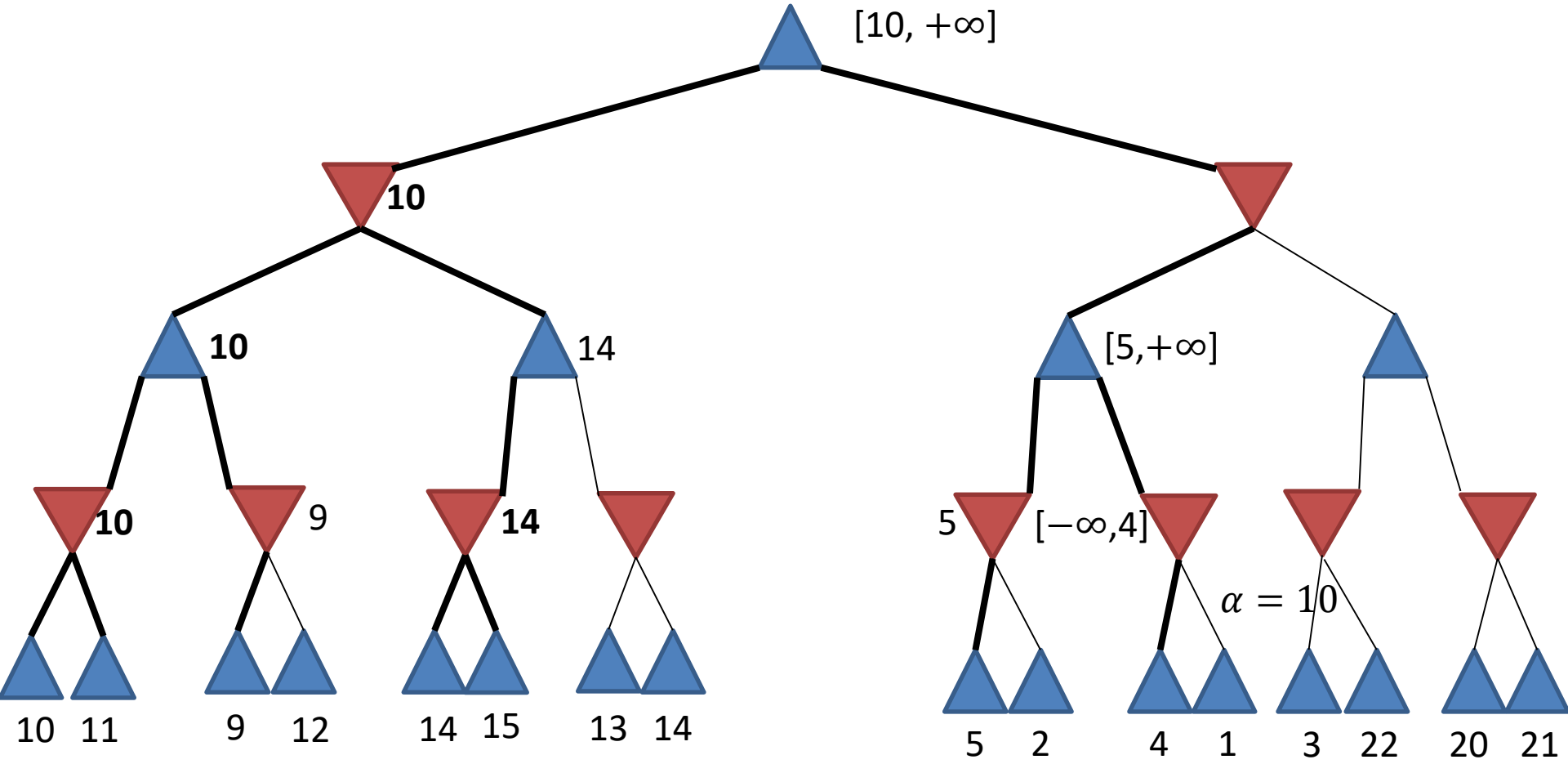
Παράδειγμα



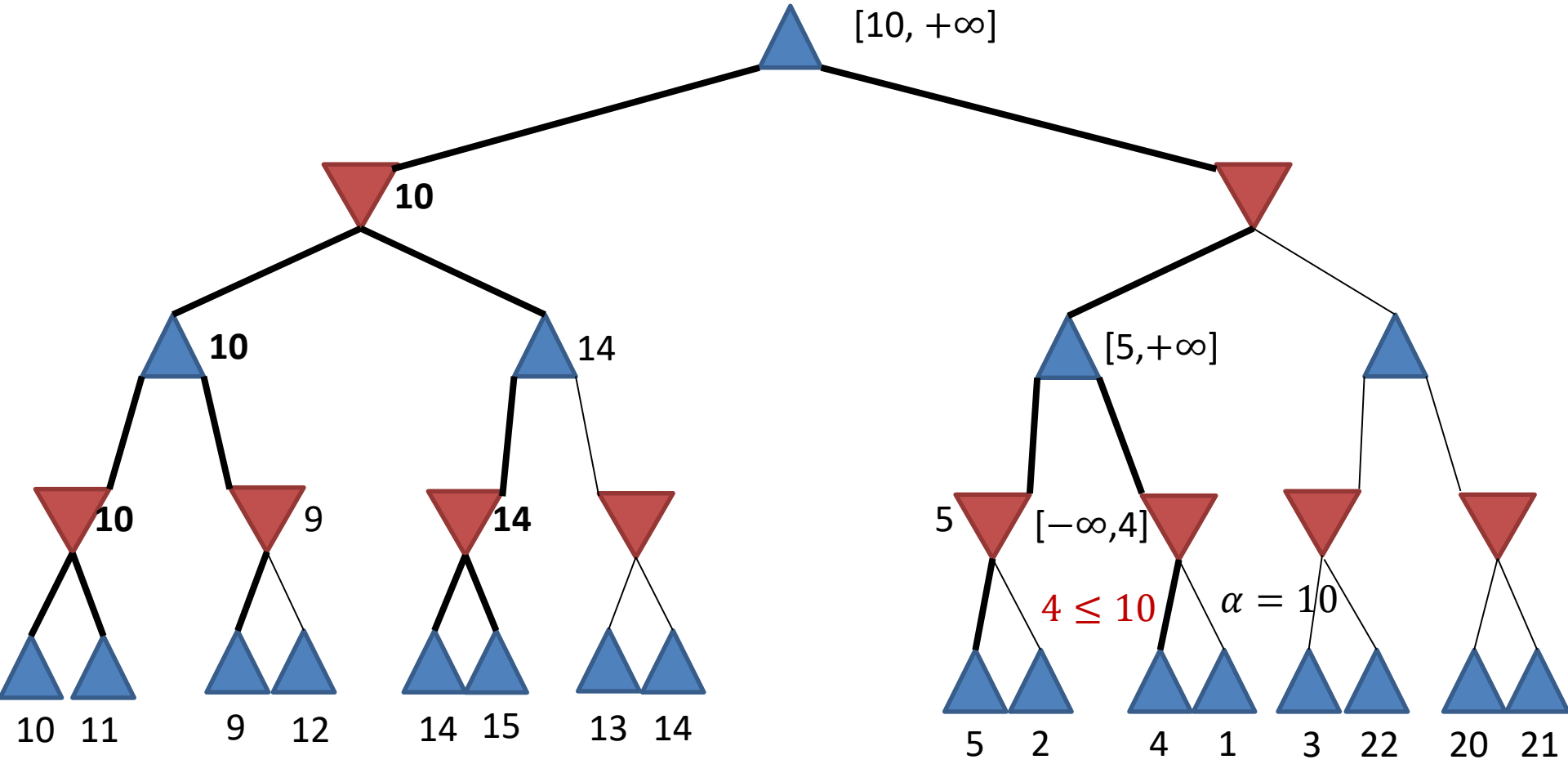
Παράδειγμα



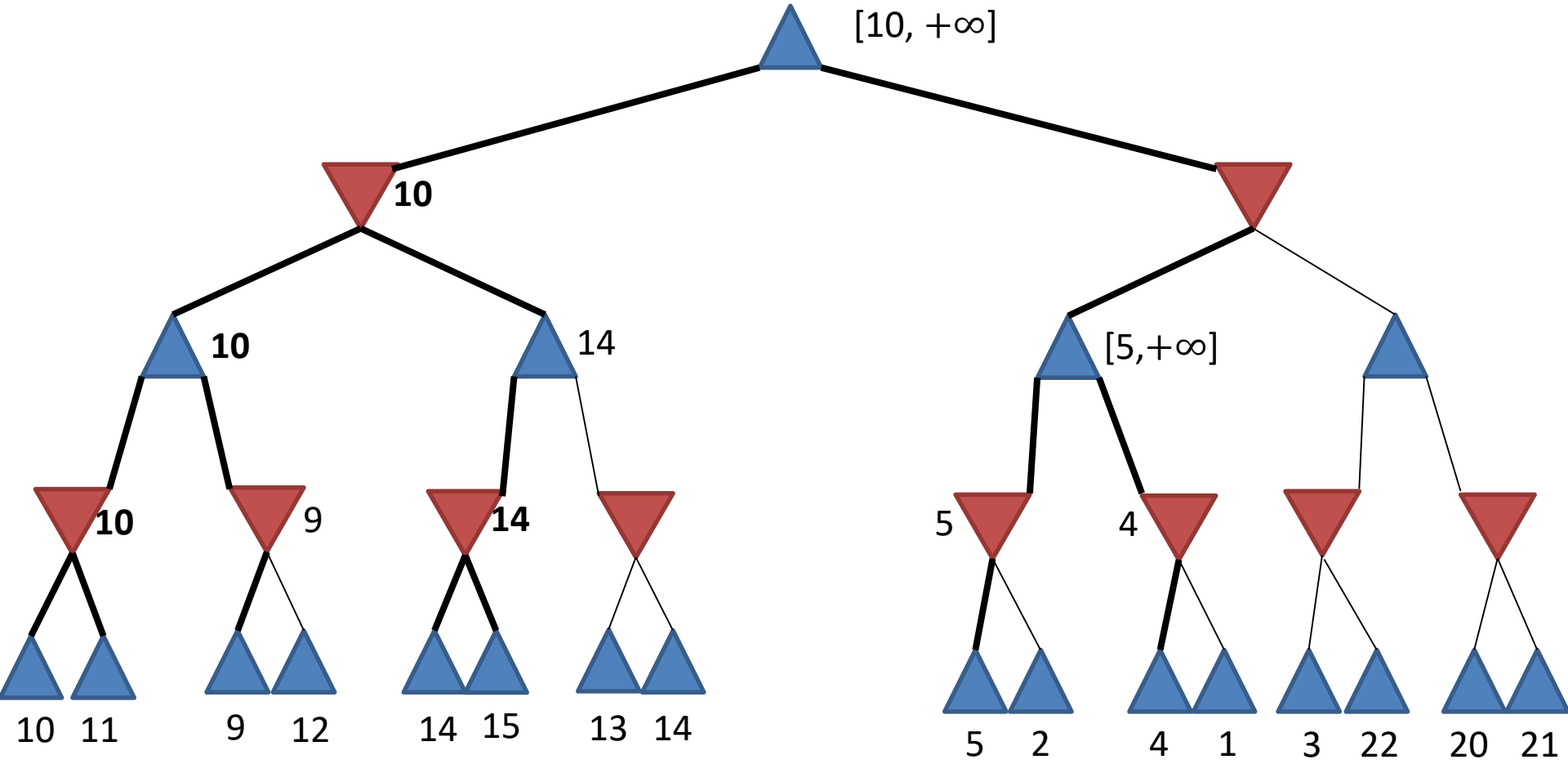
Παράδειγμα



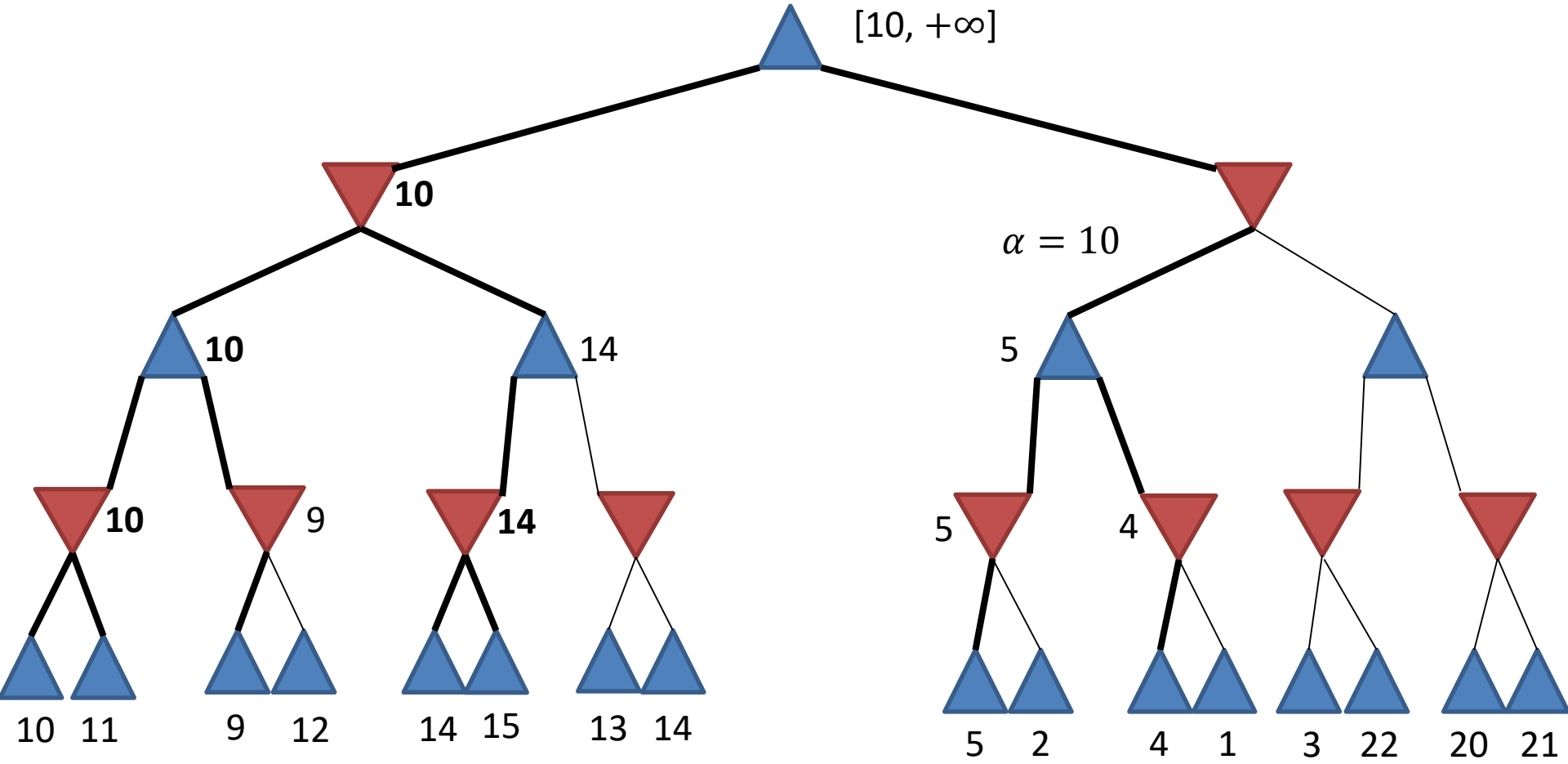
Παράδειγμα



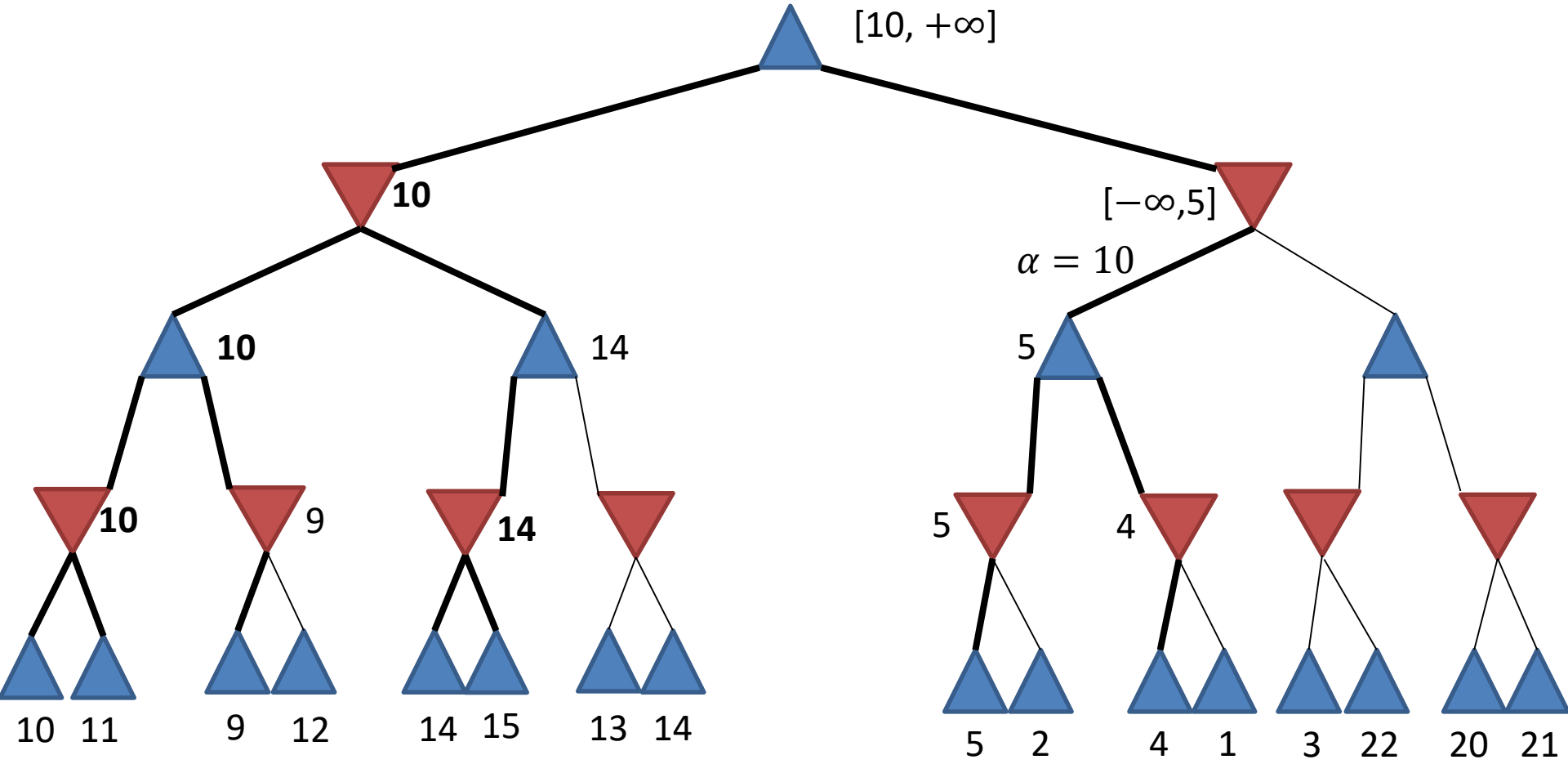
Παράδειγμα



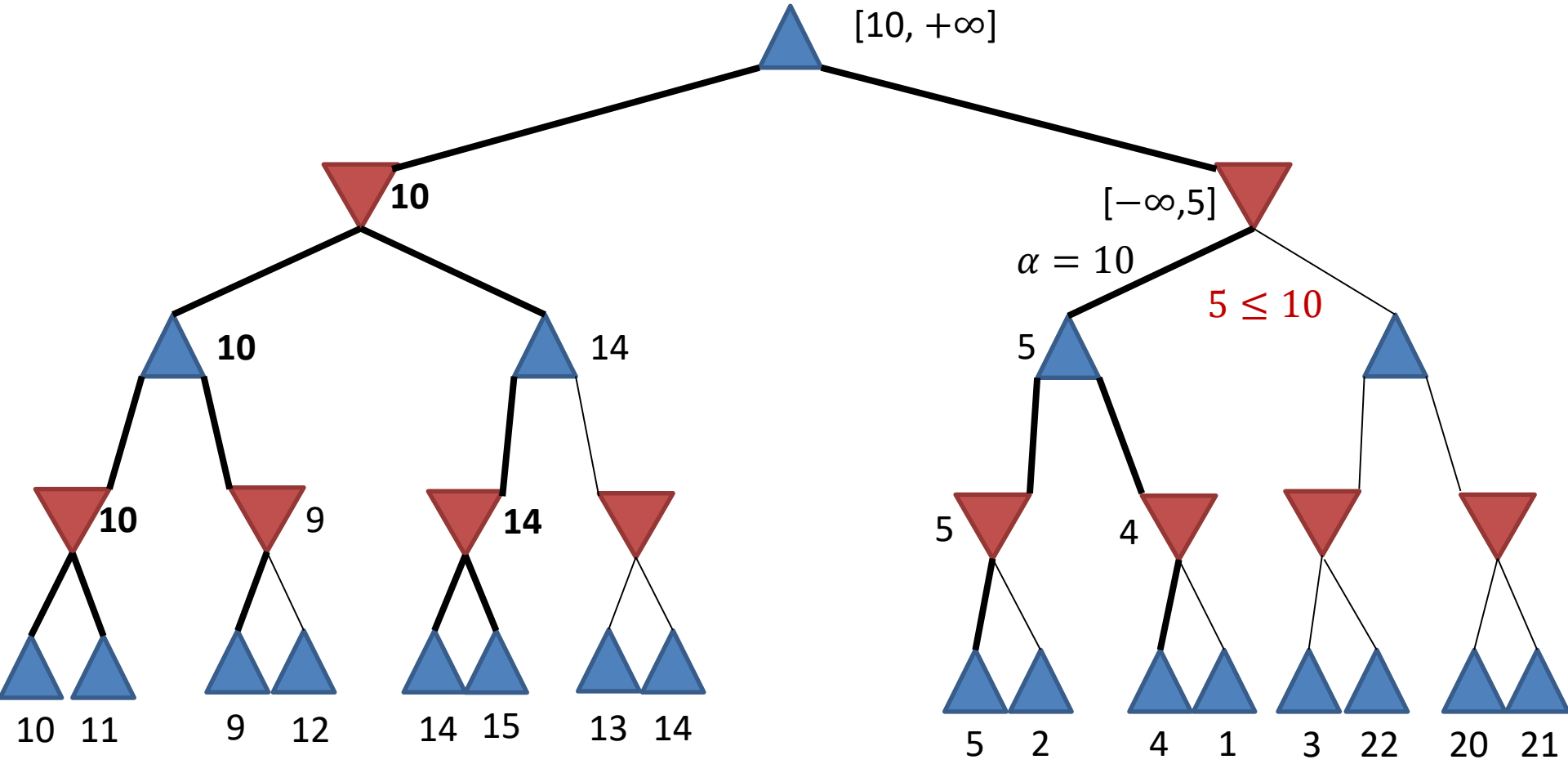
Παράδειγμα



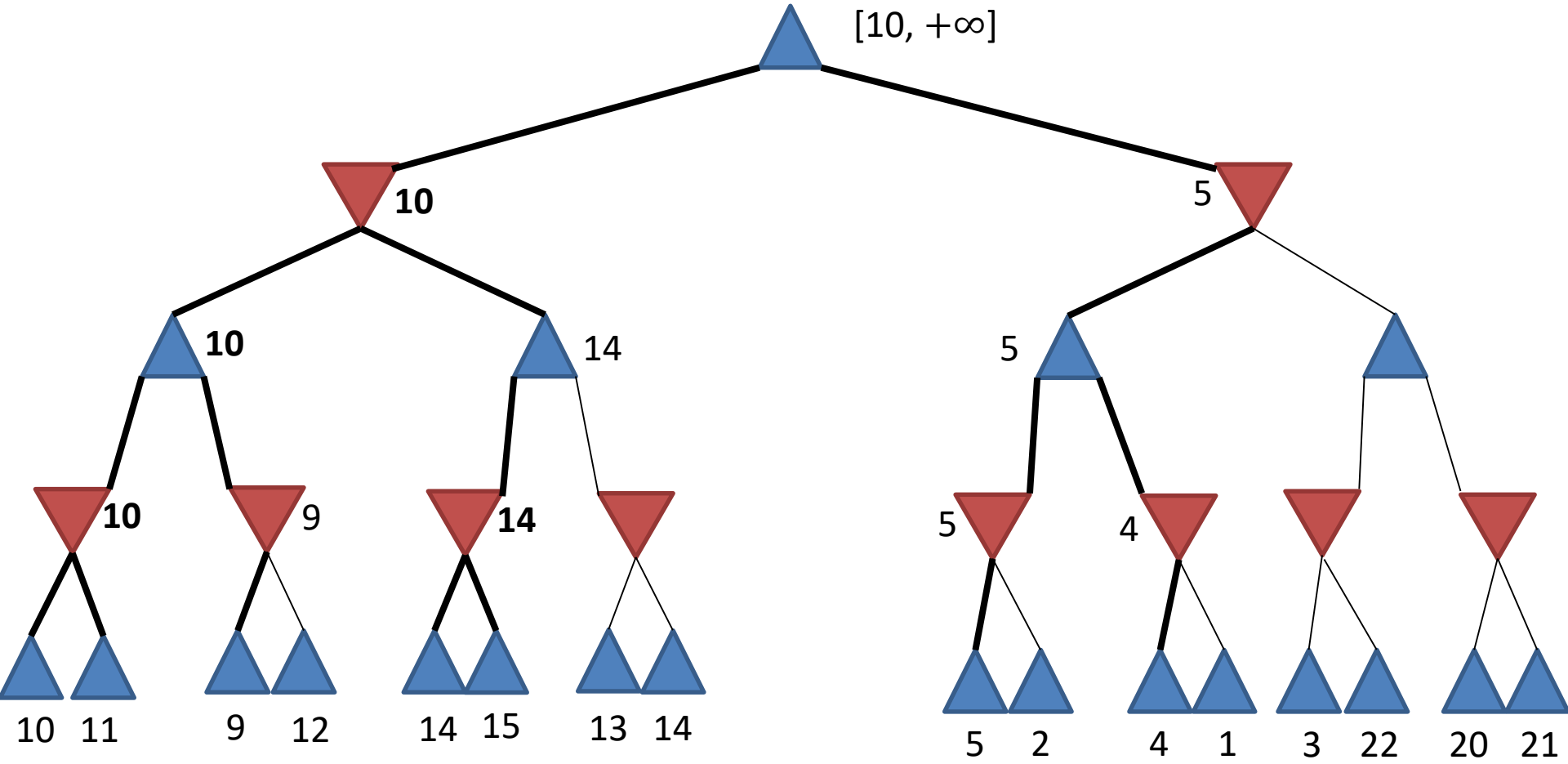
Παράδειγμα



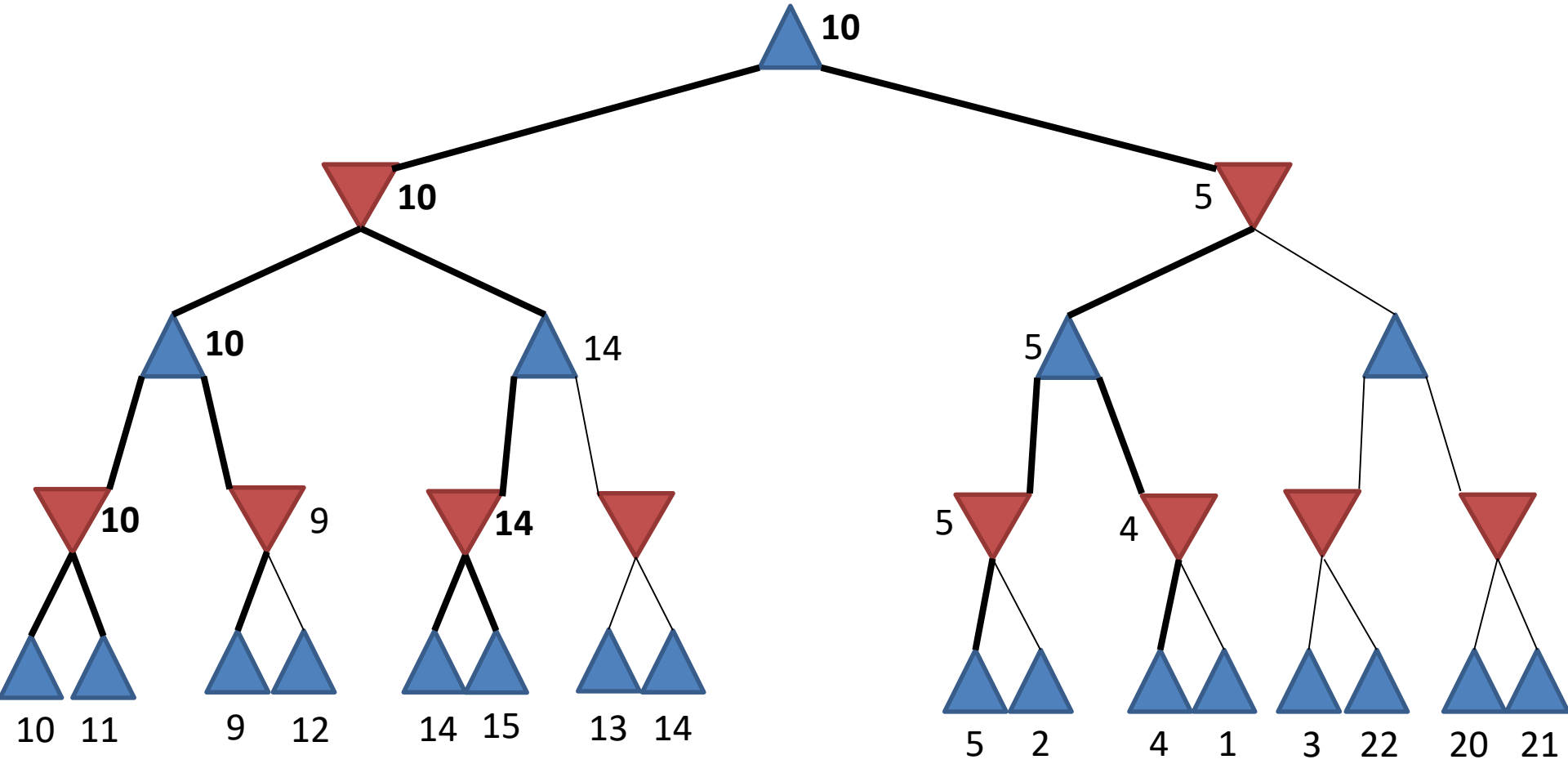
Παράδειγμα



Παράδειγμα



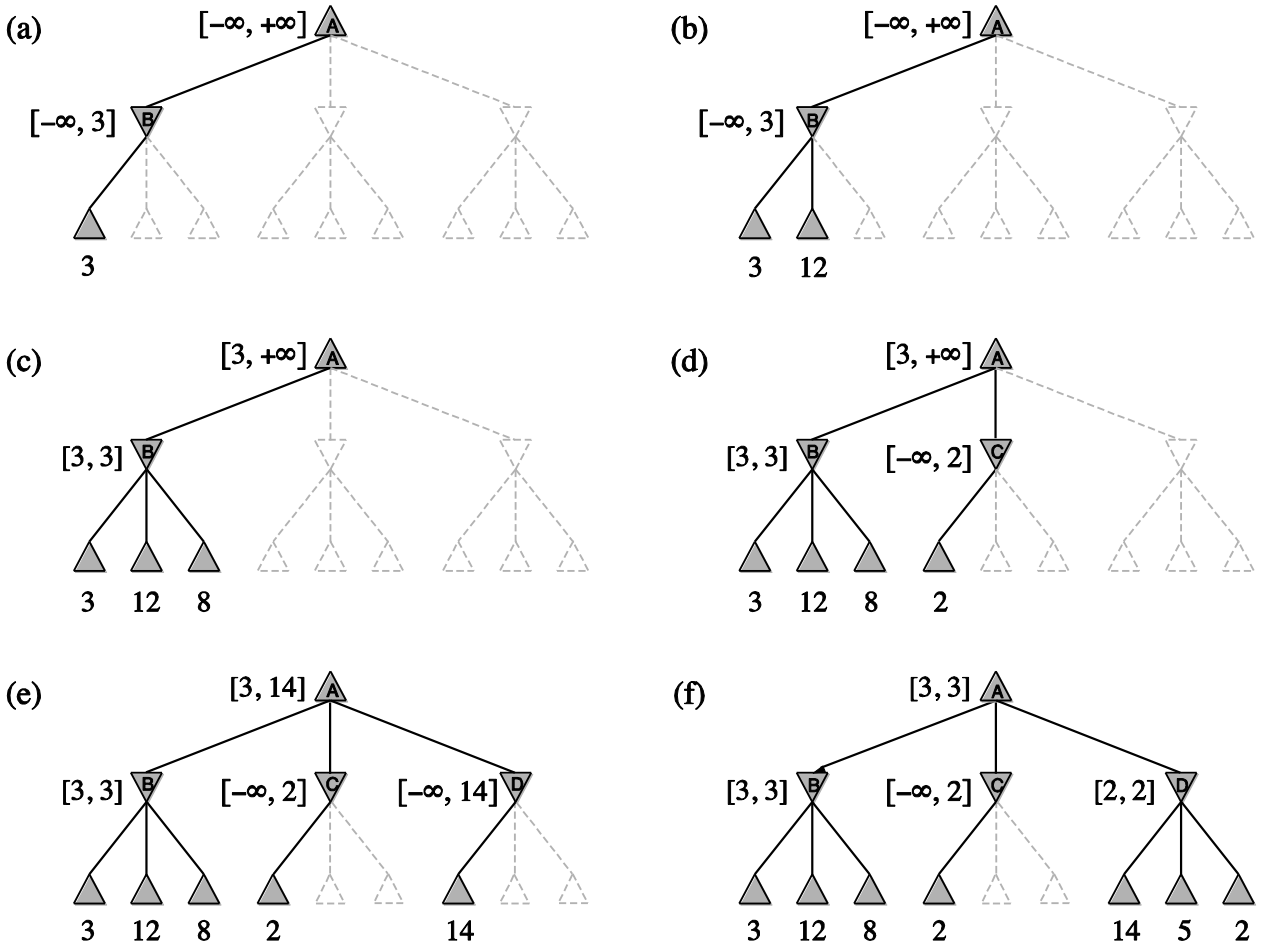
Παράδειγμα



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.

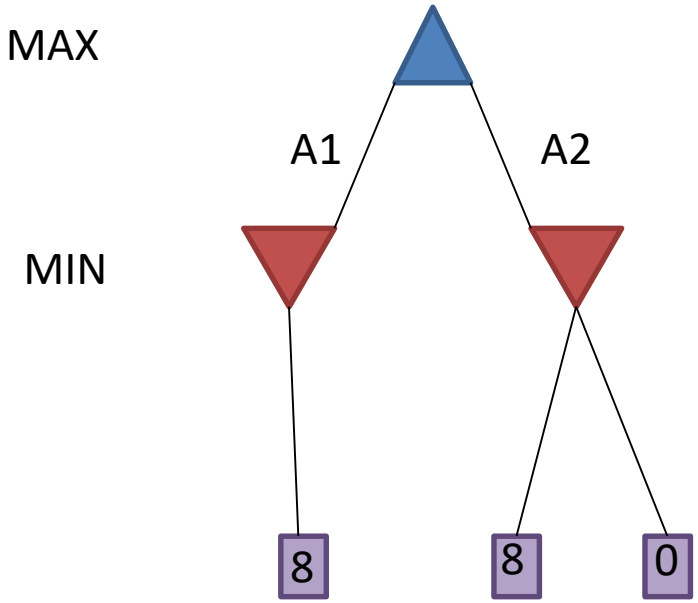
Παράδειγμα


(a)
[−∞, +∞]
A
[−∞, 3]
B
3
(b)
[−∞, +∞]
A
[−∞, 3]
B
3 12
(c)
[3, +∞]
A
[3, 3]
B
3 12 8
(d)
[3, +∞]
A
[3, 3]
B
[−∞, 2]
C
3 12 8 2
(e)
[3, 14]
A
[3, 3]
B
[−∞, 2]
C
[−∞, 14]
D
3 12 8 2 14
(f)
[3, 3]
A
[3, 3]
B
[−∞, 2]
C
[2, 2]
D
3 12 8 2 14 5 2

Παρατηρήσεις

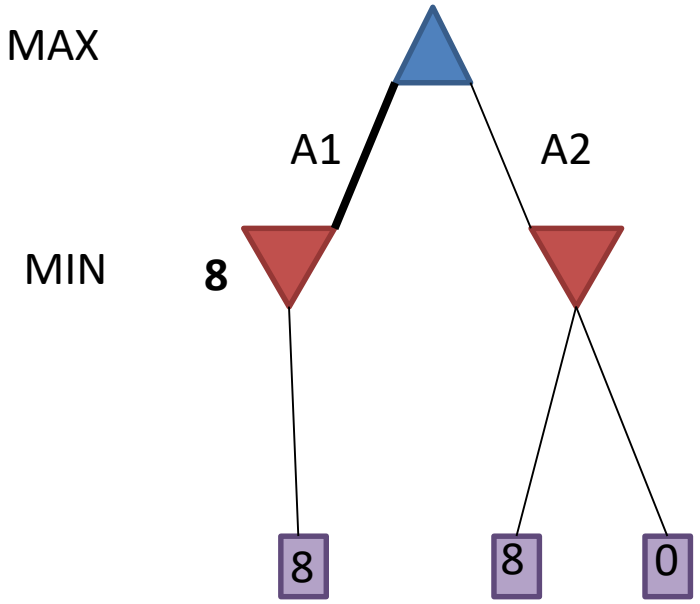
- Το κλάδεμα άλφα-βήτα δεν έχει **καμία επίπτωση** στην minimax τιμή της ρίζας η οποία υπολογίζεται σωστά.
- Οι τιμές στους ενδιάμεσους κόμβους μπορεί να είναι **λάθος**! (δηλαδή, να μην είναι οι minimax τιμές).
- Πρέπει να είμαστε προσεκτικοί αν θέλουμε να υπολογίσουμε την minimax απόφαση στη ρίζα (δηλαδή,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

Παράδειγμα

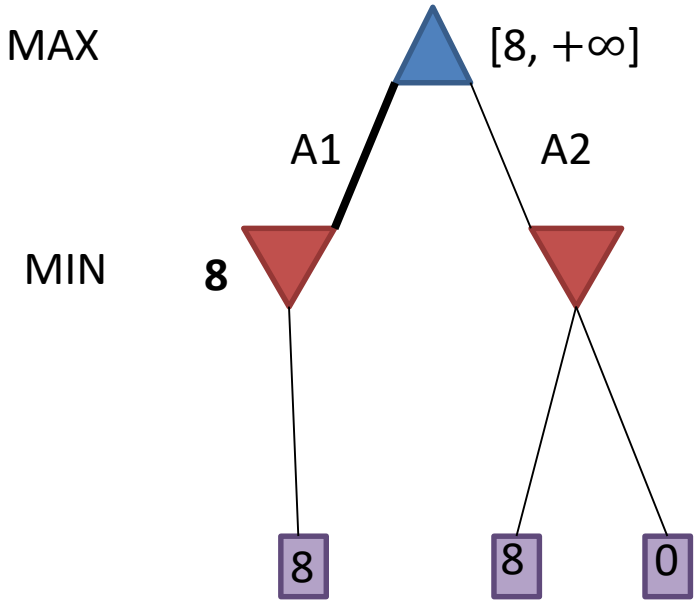


Ποια είναι η κίνηση που πρέπει να επιλέξει ο MAX;

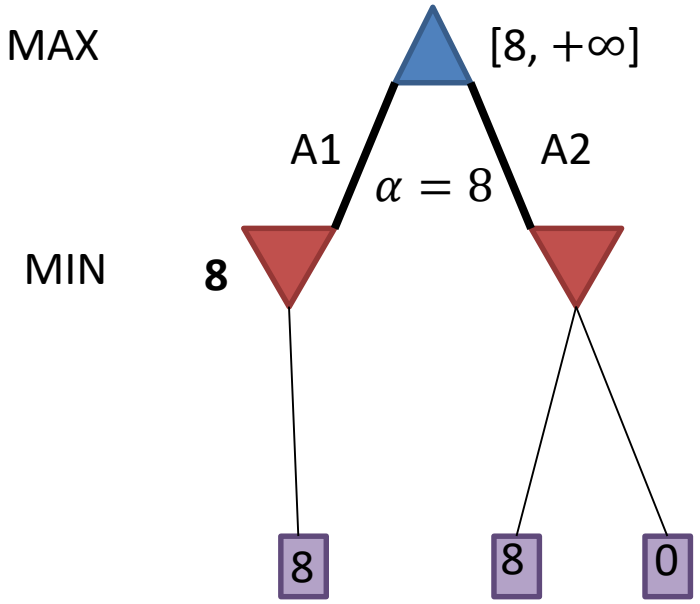
Εκτέλεση του ALPHA-BETA-SEARCH



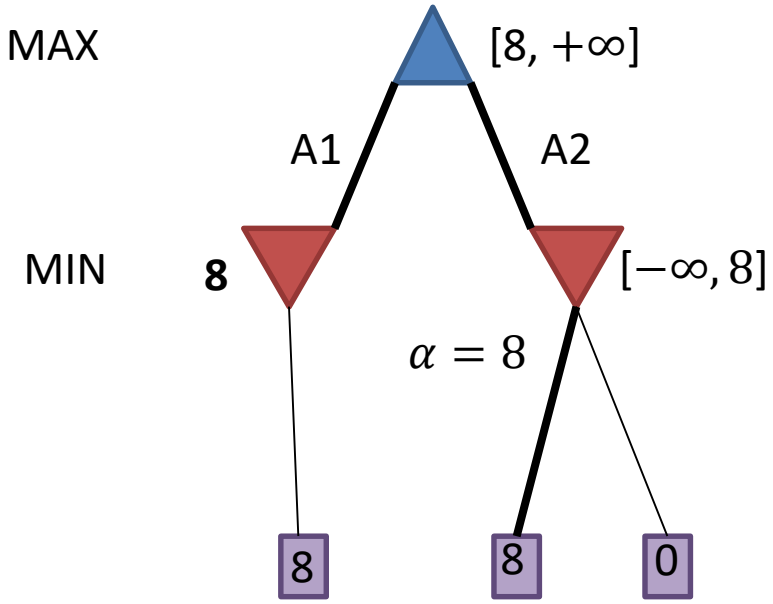
Εκτέλεση του ALPHA-BETA-SEARCH



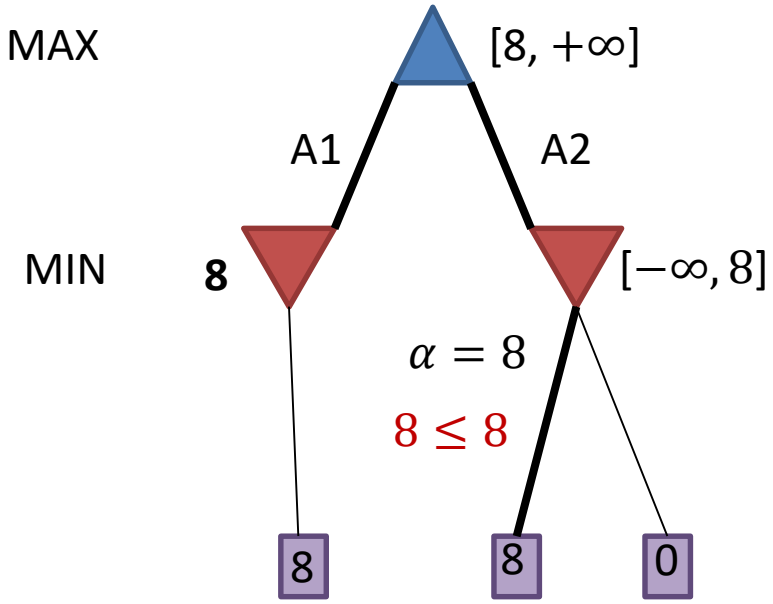
Εκτέλεση του ALPHA-BETA-SEARCH



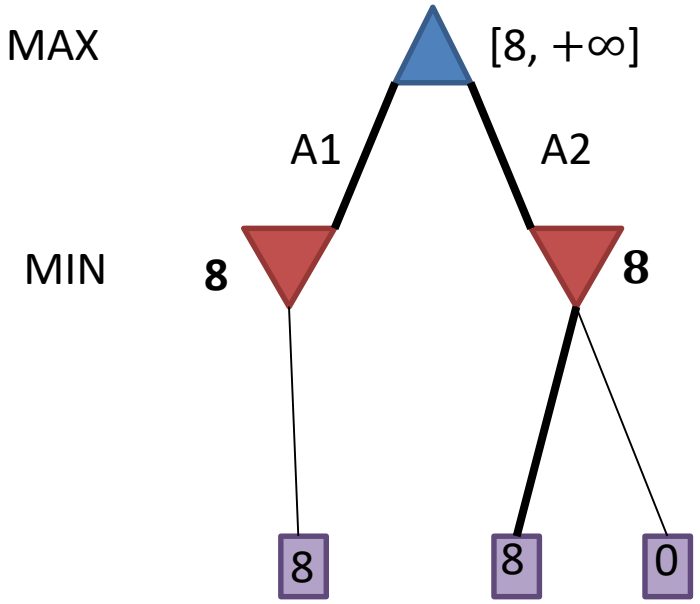
Εκτέλεση του Alpha-Beta-Search



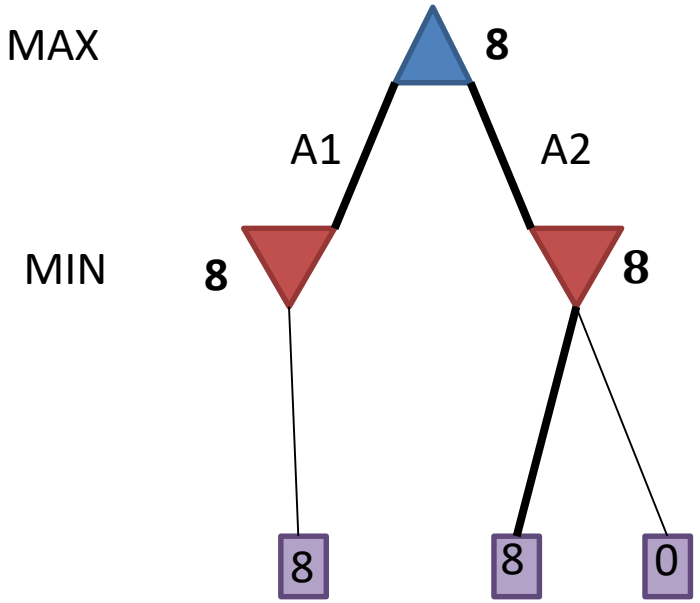
Εκτέλεση του ALPHA-BETA-SEARCH



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 $\boldsymbol{v > \beta}$ και $\boldsymbol{v < \alpha}$).

Υπολογιστική Πολυπλοκότητα

- Αν οι διάδοχες καταστάσεις **εξετάζονται με την καλύτερη δυνατή σειρά**, ο αλγόριθμος αναζήτησης άλφα-βήτα μπορεί να εξετάσει μόνο $\boldsymbol{O(b^{\frac{m}{2}})}$ κόμβους αντί για $O(b^m)$ που εξετάζει ο minimax.
- Αυτό σημαίνει ότι ο **δραστικός παράγοντας διακλάδωσης** γίνεται $\sqrt{\boldsymbol{b}}$ 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 χρησιμοποίηση 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:

$$\text{H} - \textsc{Minimax}(s, d) =$$

$$= \begin{cases} \textsc{Eval}(s) \text{ if } \textsc{Cutoff} - \textsc{Test}(s, d) \\ \max_{a \in \textsc{Actions}(s)} \text{H} - \textsc{Minimax}(\textsc{Result}(s, a), d + 1) \text{ if } \textsc{Player}(s) = \text{MAX} \\ \min_{a \in \textsc{Actions}(s)} \text{H} - \textsc{Minimax}(\textsc{Result}(s, a), d + 1) \text{ if } \textsc{Player}(s) = \text{MIN} \end{cases}$$

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 $(0{,}72 \times (+1)) + (0{,}20 \times 0) + \left(0{,}08 \times \frac{1}{2}\right) = 0.76$. 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 πιόνι 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:

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

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

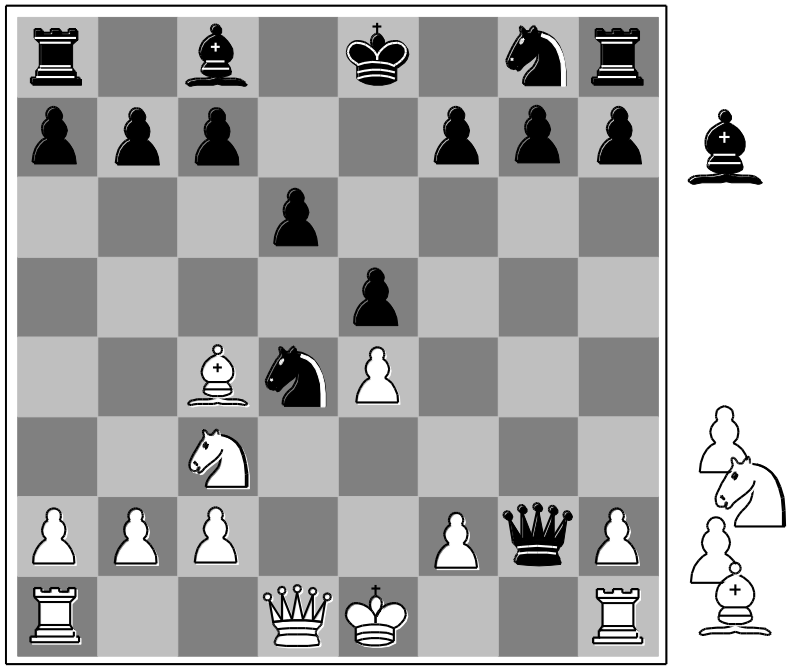
- Μπορούμε να τροποποιήσουμε τον αλγόριθμο ALPHA-BETA-SEARCH ώστε να καλεί τη συνάρτηση EVAL όταν πρέπει να αποκοπεί η αναζήτηση.
- Στον κώδικα που δώσαμε, αλλάζουμε τη γραμμή που περιέχει τον έλεγχο TERMINAL-TEST ως εξής:

  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.

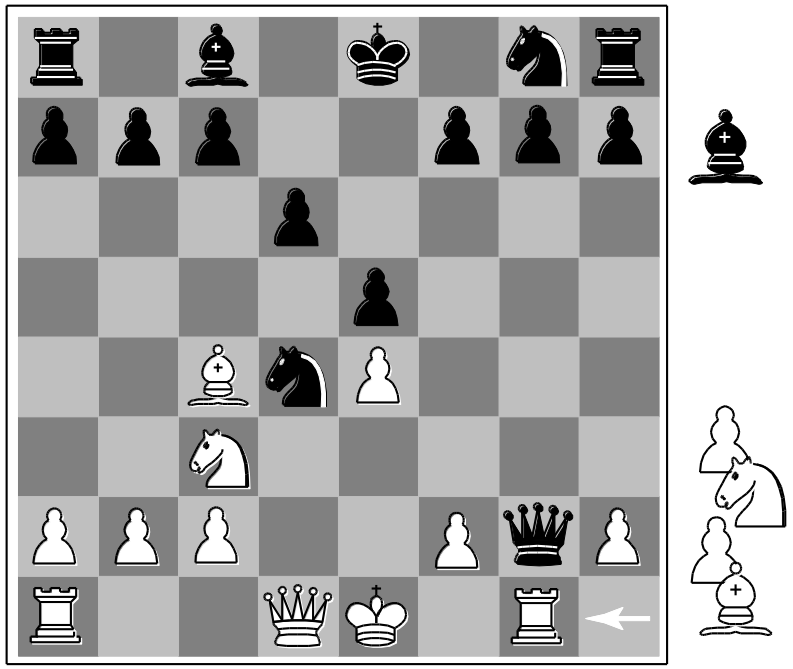
Αποκοπή Αναζήτησης

- 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.**

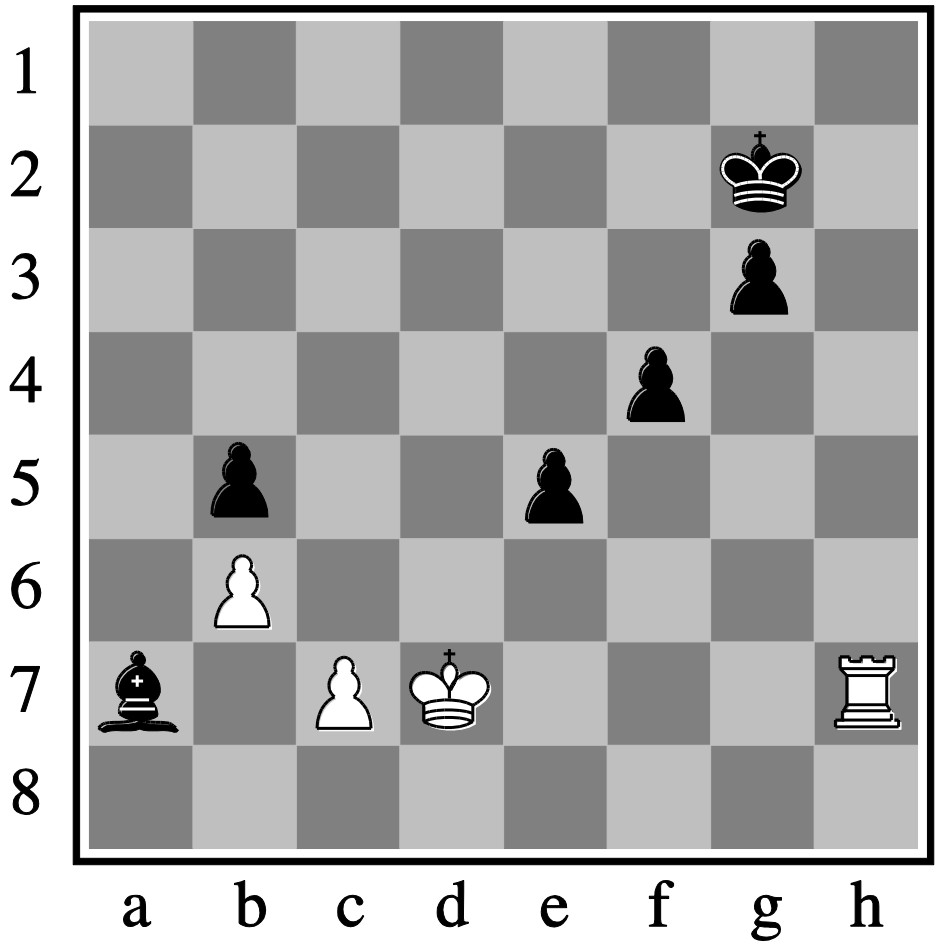
Αναζήτηση Ηρεμίας

- 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 πιόνι στη θέση e5. Αυτό θα οδηγήσει στην αιχμαλωσία του πιονιού από τον άσπρο βασιλιά.
- Ο Μαύρος μπορεί να προσπαθήσει το ίδιο πράγμα με το πιόνι 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 πιόνια, 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.**

Αναζήτηση Δένδρου Monte Carlo

- Υπάρχουν παιχνίδια στα οποία ο **παράγοντας διακλάδωσης είναι μεγάλος.** Για παράδειγμα, στο παιχνίδι στρατηγικής **Go** ξεκινάει από 361, το οποίο σημαίνει ότι η αναζήτηση άλφα-βήτα θα περιοριστεί σε 4 ή 5 στρώσεις.

- Για να αντιμετωπίσουν αυτό το θέμα, τα σύγχρονα προγράμματα Go έχουν εγκαταλείψει την αναζήτηση άλφα-βήτα και πλέον χρησιμοποιούν την τεχνική που λέγεται **αναζήτηση δένδρου Monte Carlo (Monte Carlo tree search).**

Προσομοιώσεις

- Η βασική ιδέα της αναζήτησης δένδρου Monte Carlo είναι ότι δεν χρησιμοποιείται κάποια συνάρτηση αξιολόγησης. Η **τιμή μιας κατάστασης εκτιμάται ως η μέση χρησιμότητα ενός πλήθους προσομοιώσεων (simulations, playouts ή rollouts)** ολοκληρωμένων παιχνιδιών που ξεκινούν από την κατάσταση αυτή.
- Σε μια προσομοίωση, επιλέγεται πρώτα η κίνηση για τον ένα παίκτη, μετά η κίνηση για τον άλλο, και αυτό συνεχίζεται μέχρι να φτάσουμε σε μια τερματική κατάσταση.
- Σε αυτό το σημείο, οι κανόνες του παιχνιδιού (και όχι οι επιρρεπείς σε λάθη ευρετικές μέθοδοι) καθορίζουν ποιος κερδίζει ή χάνει και με τι τιμή χρησιμότητας.
- Για παιχνίδια στα οποία τα μοναδικά αποτελέσματα είναι νίκη ή ήττα, **η μέση χρησιμότητα είναι ίδια με το ποσοστό νικών.**

Πολιτικές Παιξίματος

- Πως επιλέγουμε τις κινήσεις που θα κάνουμε κατά το παίξιμο;
- Για απλά παιχνίδια, μπορούμε να επιλέγουμε απλώς στην τύχη (υποθέτοντας ότι κάνουμε πολλές προσομοιώσεις).
- Για πολύπλοκα παιχνίδια, χρειαζόμαστε μια **πολιτική παιξίματος (playout policy) η οποία ενθαρρύνει τις καλές κινήσεις.**
- Για το Go και άλλα παιχνίδια, οι πολιτικές παιξίματος μαθαίνονται επιτυχώς από τα αντίστοιχα προγράμματα παίζοντας με τον εαυτό τους χρησιμοποιώντας **νευρωνικά δίκτυα.**
- Μερικές φορές χρησιμοποιούνται επίσης **ευρετικοί μηχανισμοί** συγκεκριμένοι για ένα παιχνίδι π.χ., «θεώρησε κινήσεις παρσίματος ενός κομματιού» για το σκάκι.

Αμιγής Αναζήτηση Monte Carlo

- Αφού θα έχουμε μια πολιτική παιξίματος, πρέπει να αποφασίσουμε **από ποιες θέσεις ξεκινάμε τις προσομοιώσεις** και **πόσες προσομοιώσεις κάνουμε από κάθε θέση**.
- Η απλούστερη απάντηση που ονομάζεται **αμιγής αναζήτηση Monte Carlo (pure Monte Carlo search)** είναι να κάνουμε *N* προσομοιώσεις ξεκινώντας από την τρέχουσα κατάσταση του παιχνιδιού, και να παρακολουθούμε ποιες από τις δυνατές κινήσεις από την τρέχουσα κατάσταση έχουν το υψηλότερο ποσοστό νικών.
- Για μερικά στοχαστικά παιχνίδια αυτή η στρατηγική συγκλίνει σε βέλτιστες αποφάσεις καθώς αυξάνεται το *N*, αλλά για τα περισσότερα παιχνίδια δεν επαρκεί – χρειαζόμαστε μια **πολιτική επιλογής (selection policy) που εστιάζει επιλεκτικά τους διαθέσιμους υπολογιστικούς πόρους στα σημαντικά τμήματα του δένδρου παιχνιδιού.**

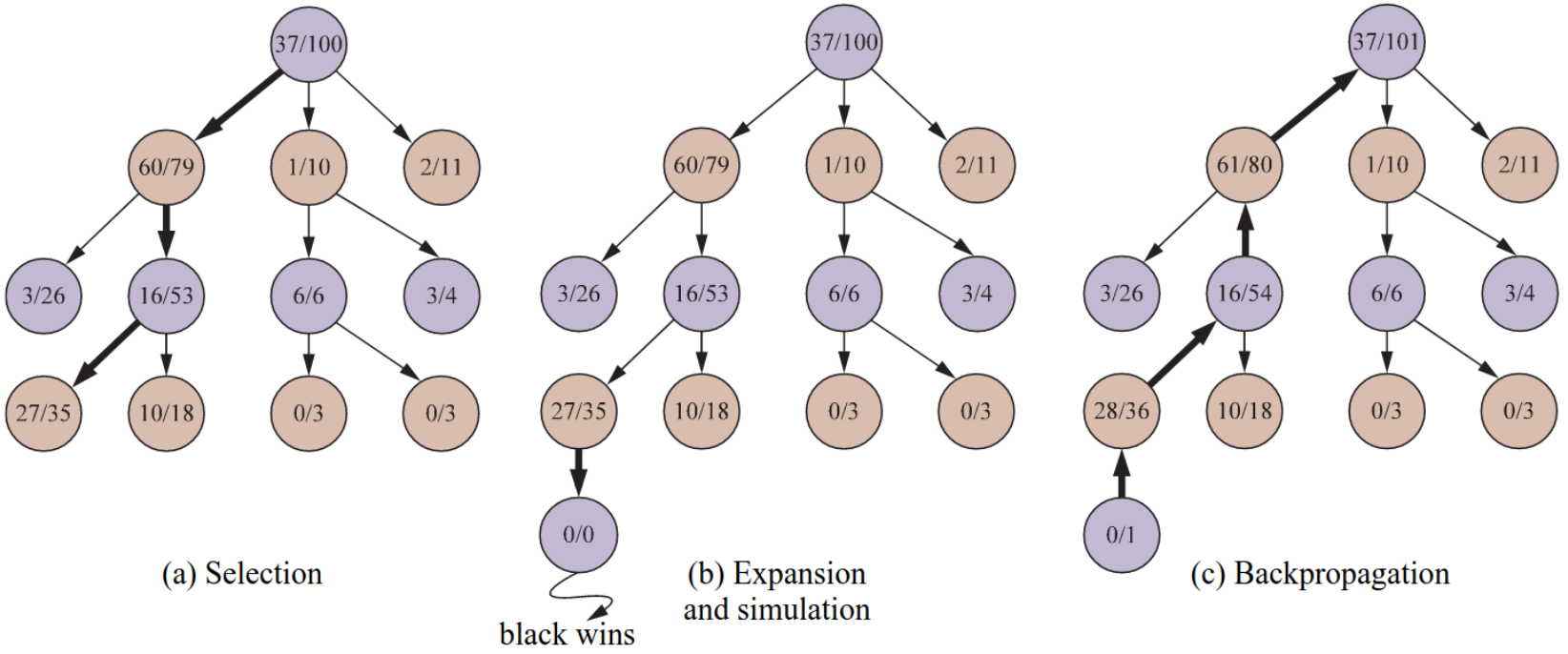
Πολιτική Επιλογής

- Η πολιτική επιλογής πρέπει να εξισορροπεί δύο παράγοντες: την **εξερεύνηση** των καταστάσεων για τις οποίες έχουν γίνει λίγες προσομοιώσεις, και την **εκμετάλλευση** των καταστάσεων που τα έχουν πάει καλά σε παλαιότερες προσομοιώσεις, ώστε να έχουμε μια ακριβή εκτίμηση της αξίας τους.
- Υπάρχει ένα **αντιστάθμισμα (tradeoff)** ανάμεσα στην εξερεύνηση και την εκμετάλλευση.

Αναζήτηση Δένδρου Monte Carlo – Ο Αλγόριθμος

- Η αναζήτηση δένδρου Monte Carlo επιτυγχάνει αυτή την εξισορρόπηση ως εξής. Σε κάθε κόμβο του δένδρου παιχνιδιού (που φυσικά αντιστοιχεί σε μια κατάσταση), δημιουργεί ένα **δένδρο αναζήτησης** και το επεκτείνει με επαναλήψεις των παρακάτω τεσσάρων βημάτων:
  - **Επιλογή**
  - **Επέκταση**
  - **Προσομοίωση**
  - **Οπισθοδιάδοση**

Παράδειγμα


37/100
60/79
1/10
2/11
3/26
16/53
6/6
3/4
27/35
10/18
0/3
0/3
(a) Selection
37/100
60/79
1/10
2/11
3/26
16/53
6/6
3/4
27/35
10/18
0/3
0/3
0/0
black wins
(b) Expansion
and simulation
37/101
61/80
1/10
2/11
3/26
16/54
6/6
3/4
28/36
10/18
0/3
0/3
0/1
(c) Backpropagation

Επιλογή

- **Ξεκινώντας από τη ρίζα του δένδρου αναζήτησης, επιλέγουμε μια κίνηση** (η οποία ενδείκνυται από την πολιτική επιλογής) που οδηγεί σε ένα διάδοχο κόμβο και **επαναλαμβάνουμε αυτή τη διαδικασία** διατρέχοντας προς τα κάτω το δένδρο μέχρι ένα φύλλο.
- Στο προηγούμενο παράδειγμα φαίνεται ένα δένδρο αναζήτησης με τη **ρίζα** να αναπαριστά μια κατάσταση όπου ο Λευκός έχει μόλις κάνει μια κίνηση και έχει κερδίσει 37/100 παιξίματα που έχουν γίνει μέχρι στιγμής.
- Το **παχύ βέλος** δείχνει την επιλογή κίνησης από τον Μαύρο που οδηγεί σε ένα κόμβο στο οποίο Μαύρος έχει κερδίσει 60/79 παιξίματα. Εδώ έχουμε ένα παράδειγμα **εκμετάλλευσης** επειδή αυτή η κίνηση έχει το καλύτερο ποσοστό νικών ανάμεσα στις τρεις διαθέσιμες. Θα μπορούσαμε όμως να κάνουμε **εξερεύνηση** και να επιλέξουμε την 2/11 με την ελπίδα να αποδειχτεί καλύτερη αφού αποκτήσουμε περισσότερες πληροφορίες για τον αντίστοιχο κόμβο.
- Συνεχίζουμε να κάνουμε επιλογή μέχρι να φτάσουμε στον **κόμβο-φύλλο** με την ένδειξη 27/35.

Επέκταση

- Επεκτείνουμε το δένδρο αναζήτησης δημιουργώντας ένα **κόμβο-παιδί** (αντίστοιχο με μια κίνηση) ο οποίος παίρνει την ετικέτα 0/0.
- Σε αυτό το βήμα, ορισμένες εκδόσεις του αλγόριθμου δημιουργούν **περισσότερους από ένα κόμβους-παιδιά.**

Προσομοίωση

- Πραγματοποιούμε ένα παίξιμο από το παιδί που μόλις δημιουργήθηκε επιλέγοντας κινήσεις για τους δύο παίκτες **σύμφωνα με την πολιτική παιξίματος.**
- Οι κινήσεις αυτές **δεν καταγράφονται** στο δένδρο αναζήτησης.
- Στο παραπάνω παράδειγμα, η προσομοίωση καταλήγει σε νίκη του Μαύρου.

Οπισθοδιάδοση

- Στο βήμα αυτό χρησιμοποιούμε το αποτέλεσμα της προσομοίωσης για να **ενημερώσουμε όλους τους κόμβους του δένδρου αναζήτησης προς τα πάνω μέχρι τη ρίζα.**
- Επειδή κέρδισε ο Μαύρος το παίξιμο, αυξάνεται ο αριθμός των νικών όσο και ο αριθμός των παιξιμάτων για τους κόμβους που αντιστοιχούν στον Μαύρο.
- Για τον Λευκό, αυξάνεται απλώς ο αριθμός των παιξιμάτων στους αντίστοιχους κόμβους.

Ο Αλγόριθμος

- Επαναλαμβάνουμε τα παραπάνω 4 βήματα είτε για ένα καθορισμένο αριθμό επαναλήψεων είτε μέχρι να λήξει ο διαθέσιμος χρόνος και μετά **επιστρέφουμε την κίνηση με τον μεγαλύτερο αριθμό παιξιμάτων.**
- **Παρατήρηση**: Δεν επιστρέφουμε τον κόμβο με τη μεγαλύτερη μέση χρησιμότητα! Η ιδέα είναι ότι ένας κόμβος με 65/100 νίκες είναι καλύτερος από ένα με 2/3 νίκες επειδή ο τελευταίος έχει μεγάλη αβεβαιότητα.
- Ο υπολογισμός της ποσότητας $UCB(n)$ που θα δούμε παρακάτω εξασφαλίζει ότι **ο κόμβος με τα περισσότερα παιξίματα είναι σχεδόν πάντα ο κόμβος με το μεγαλύτερο ποσοστό νικών** αφού η διαδικασία επιλογής ευνοεί το ποσοστό νικών όλο και περισσότερο καθώς αυξάνεται το πλήθος των παιξιμάτων.

Ο Αλγόριθμος

```
function MONTE-CARLO-TREE-SEARCH(state) returns an action
    tree ← NODE(state)
    while IS-TIME-REMAINING() do
        leaf ← SELECT(tree)
        child ← EXPAND(leaf)
        result ← SIMULATE(child)
        BACK-PROPAGATE(result, child)
    return the move in ACTIONS(state) whose node has highest number of playouts
```

Πολιτικές Επιλογής

- Μια πολύ αποτελεσματική πολιτική επιλογής λέγεται **«άνω όρια εμπιστοσύνης εφαρμοσμένα σε δένδρα» (upper confidence bounds applied to trees – UCT).**
- Με την πολιτική αυτή κάθε δυνατή κίνηση ταξινομείται με βάση ένα άνω όριο εμπιστοσύνης που συμβολίζεται με $UCB1$ και δίνεται από τον παρακάτω τύπο:

$$UCB1(n) = \frac{U(n)}{N(n)} + C \times \sqrt{\frac{\log N(Parent(n))}{N(n)}}$$

- Στον τύπο αυτό n είναι ένας **κόμβος**, $U(n)$ είναι η **συνολική χρησιμότητα** όλων των παιξιμάτων που έγιναν μέσω του κόμβου n, $N(n)$ είναι ο **αριθμός των παιξιμάτων** που έγιναν μέσω του κόμβου n, και $Parent(n)$ είναι ο **γονέας** του κόμβου n στο δένδρο.
- Το κλάσμα $\frac{U(n)}{N(n)}$ είναι ο **όρος εκμετάλλευσης** δηλαδή η μέση χρησιμότητα του κόμβου n.
- Ο όρος με την τετραγωνική ρίζα είναι ο **όρος εξερεύνησης**. Έχει παρονομαστή το πλήθος $N(n)$, το οποίο σημαίνει ότι ο όρος θα είναι μεγάλος για κόμβους που έχουν εξερευνηθεί μόνο λίγες φορές.
- Ο αριθμητής είναι ο λογάριθμος του πλήθους των φορών που θα εξερευνήσουμε τον κόμβο-γονέα του n. Αυτό σημαίνει ότι αν επιλέξουμε τον κόμβο n για κάποιο μη μηδενικό ποσοστό του χρόνου, ο όρος εξερεύνησης μηδενίζεται καθώς αυξάνεται το πλήθος, και τελικά τα παιξίματα δίνονται στον κόμβο με τη μεγαλύτερη μέση χρησιμότητα.
- Το C είναι μια σταθερά που **εξισορροπεί την εκμετάλλευση και την εξερεύνηση**. Υπάρχει ένα θεωρητικό αποτέλεσμα που λέει ότι το C πρέπει να είναι $\sqrt{2}$ όμως στην πράξη πρέπει να πειραματιστούμε με διάφορες τιμές και να διαλέξουμε την καλύτερη.

- **Σημείωση**: στο παραπάνω παράδειγμα έχουμε εφαρμόσει UCT και έχουν ολοκληρωθεί 100 επαναλήψεις.

Χρονική Πολυπλοκότητα

- Ο **χρόνος υπολογισμού ενός παιξίματος** είναι **γραμμικός** και όχι εκθετικός **ως προς το βάθος του δένδρου παιχνιδιού**, επειδή σε κάθε σημείο επιλογής γίνεται μόνο μια κίνηση. Αυτό μας δίνει άφθονο χρόνο για πολλά παιξίματα.

- **Παράδειγμα**: Υποθέστε ένα παιχνίδι με παράγοντα διακλάδωσης 32, όπου το μέσο παιχνίδι έχει διάρκεια 100 στρώσεων. Αν έχουμε αρκετή υπολογιστική ισχύ ώστε να μπορούμε να εξετάσουμε ένα δισεκατομμύριο καταστάσεις παιχνιδιού πριν χρειαστεί να κάνουμε μια κίνηση τότε:
  - Ο αλγόριθμος minimax μπορεί να πραγματοποιήσει αναζήτηση σε βάθος 6 στρώσεων.
  - Η αναζήτηση άλφα-βήτα με τέλεια διάταξη κινήσεων μπορεί να εκτελέσει αναζήτηση σε βάθος 12 στρώσεων.
  - Η αναζήτηση δένδρου Monte Carlo μπορεί να κάνει 10 εκατομμύρια παιξίματα.

- Ποια προσέγγιση είναι καλύτερη; Αυτό εξαρτάται από την ακρίβεια της συνάρτησης αποτίμησης σε σχέση με τις πολιτικές επιλογής και παιξίματος.

Σύγκριση

- Είναι κοινή πεποίθηση ότι η αναζήτηση δένδρου Monte Carlo υπερτερεί της αναζήτησης άλφα-βήτα όταν ο παράγοντας διακλάδωσης είναι μεγάλος ή όταν είναι δύσκολο να οριστεί μια καλή συνάρτηση αξιολόγησης.
- Επειδή η αναζήτηση Monte Carlo βασίζεται σε πολλά παιξίματα, δεν είναι ευάλωτη σε σφάλματα επιλογής σε ένα κόμβο όπως η αναζήτηση άλφα-βήτα.

Συνδυασμός

- Είναι δυνατόν να συνδυάσουμε πτυχές των αναζητήσεων άλφα-βήτα και Monte Carlo.
- Για παράδειγμα, σε παιχνίδια που μπορεί να διαρκέσουν πολλές κινήσεις, θα μπορούσαμε να χρησιμοποιήσουμε **πρώιμο τερματισμό παιξίματος**, όπου διακόπτουμε ένα παίξιμο που χρειάζεται πάρα πολλές κινήσεις και το αξιολογούμε με μια ευρετική συνάρτηση ή το δηλώνουμε ως ισοπαλία.

Νέα Παιχνίδια

- Η αναζήτηση Monte Carlo μπορεί να εφαρμοστεί σε **καινούρια παιχνίδια**, για τα οποία δεν υπάρχει συσσωρευμένη πείρα για να οριστεί μια συνάρτηση αξιολόγησης.
- Καλές πολιτικές μπορούν να μαθαίνονται με χρήση νευρωνικών δικτύων τα οποία εκπαιδεύονται παίζοντας μόνα τους.

Μειονεκτήματα

- Η αναζήτηση Monte Carlo έχει ένα **μειονέκτημα** όταν είναι πιθανό μια μόνο κίνηση να αλλάξει την έκβαση του παιχνιδιού αλλά λόγω της στοχαστικής φύσης της αναζήτησης ενδέχεται να μην εξεταστεί αυτή η κίνηση.
- Έχει επίσης ένα **μειονέκτημα** όταν υπάρχουν καταστάσεις παιχνιδιού που είναι προφανώς νίκη για ένα από τους παίκτες (σύμφωνα με την ανθρώπινη γνώση και μια συνάρτηση αξιολόγησης) αλλά αυτό χρειάζεται πολλές κινήσεις για να επαληθευθεί.

Σκάκι

- Εδώ και καιρό θεωρείται ότι η αναζήτηση άλφα-βήτα ταιριάζει καλύτερα σε παιχνίδια όπως το σκάκι με σχετικά χαμηλό παράγοντα διακλάδωσης και καλές συναρτήσεις αξιολόγησης.
- Πρόσφατα αποτελέσματα όμως έχουν δείξει ότι **η αναζήτηση Monte Carlo είναι αποτελεσματική τόσο στο σκάκι όσο και σε άλλα παιχνίδια.**

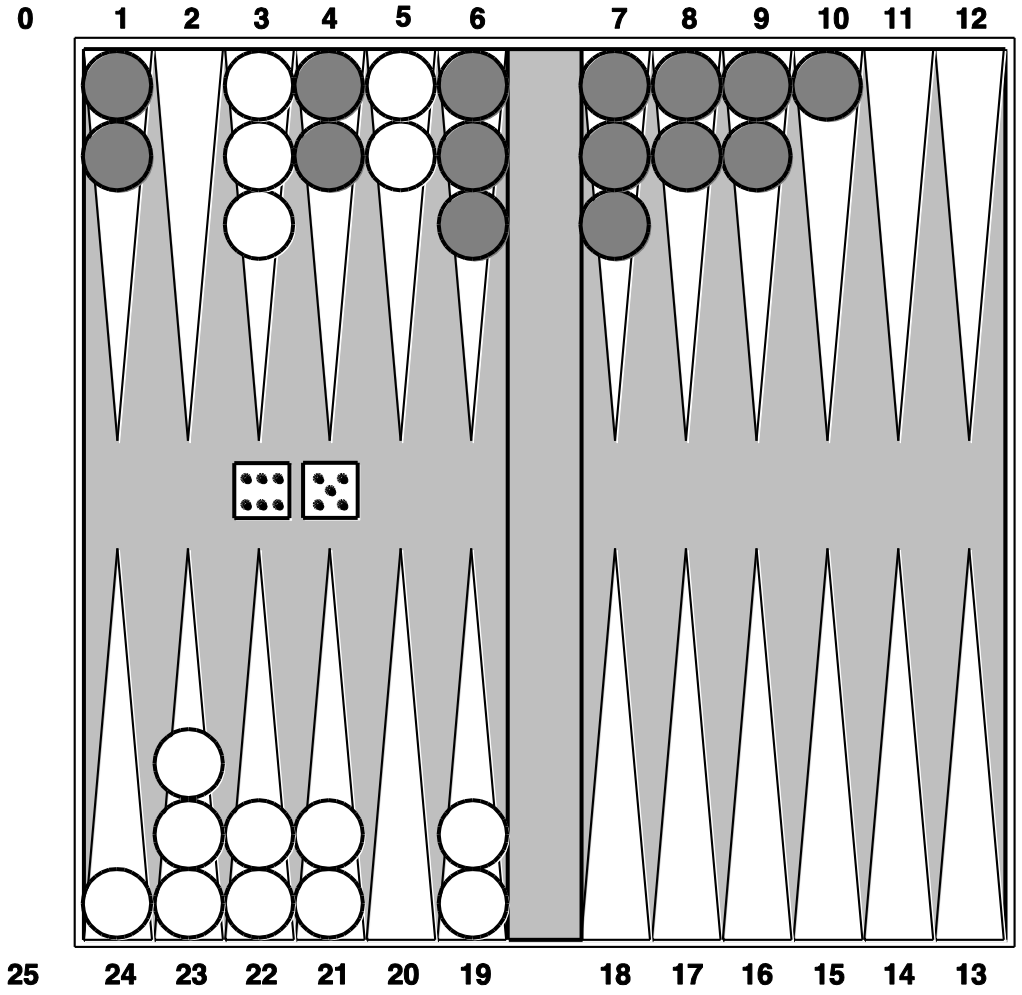
Ενισχυτική Μάθηση

- Η γενική ιδέα της προσομοίωσης μελλοντικών κινήσεων, της παρατήρησης του αποτελέσματος, και της χρήσης του αποτελέσματος για τον προσδιορισμό των κινήσεων που είναι καλές είναι ένα είδος **ενισχυτικής μάθησης (reinforcement learning).**

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



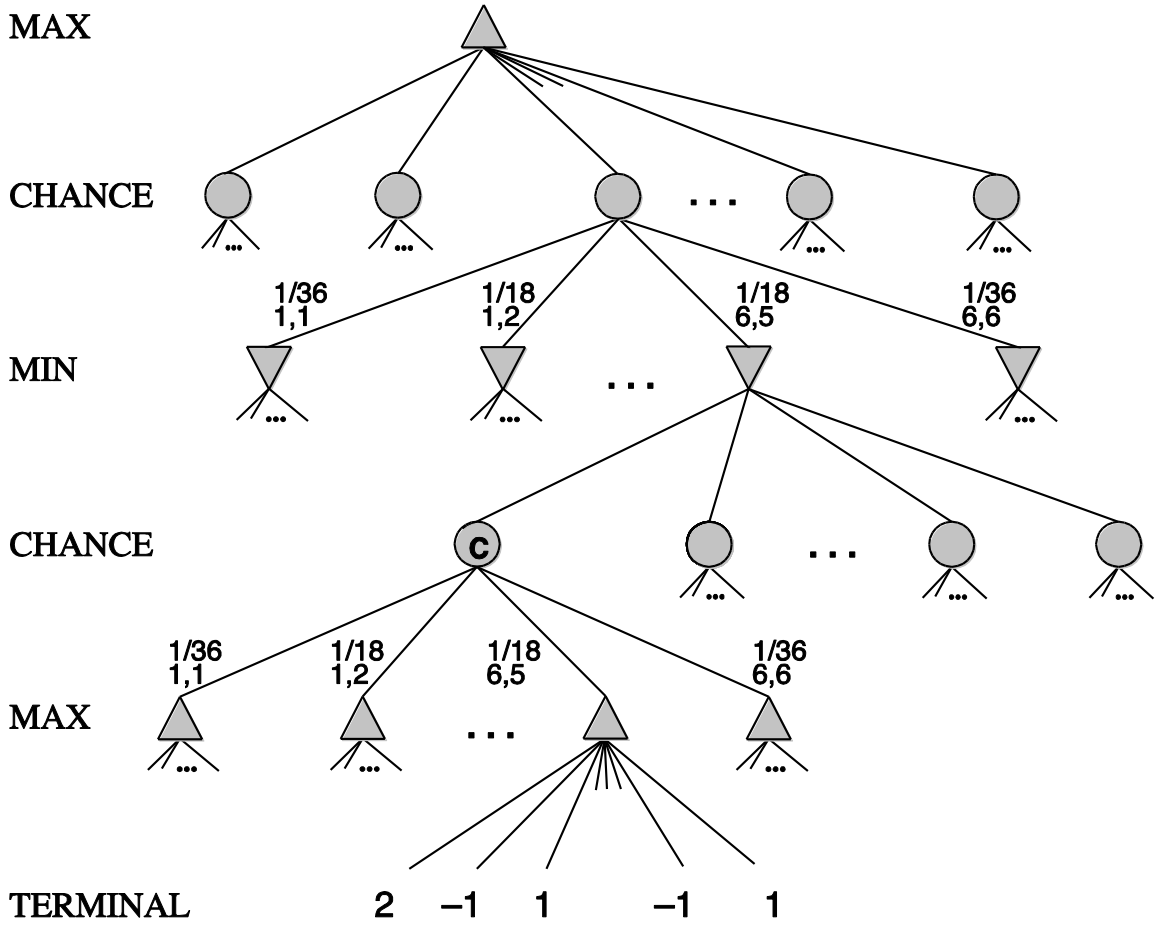
Παράδειγμα

- 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 $\frac{1}{1^{36}}$. 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:

$$\textsc{Expectiminimax}(s) =$$

$$\begin{cases} \textsc{Utility}(s) \text{ if } \textsc{Terminal}-\textsc{Test}(s) \\ \max_a \textsc{Expectiminimax}\big(\textsc{Result}(s,a)\big) \text{ if } \textsc{Player}(s) = \text{MAX} \\ \min_a \textsc{Expectiminimax}\big(\textsc{Result}(s,a)\big) \text{ if } \textsc{Player}(s) = \text{MIN} \\ \sum_r P(r)\textsc{Expectiminimax}\big(\textsc{Result}(s,r)\big) \text{ if } \textsc{Player}(s) = \text{CHANCE} \end{cases}$$

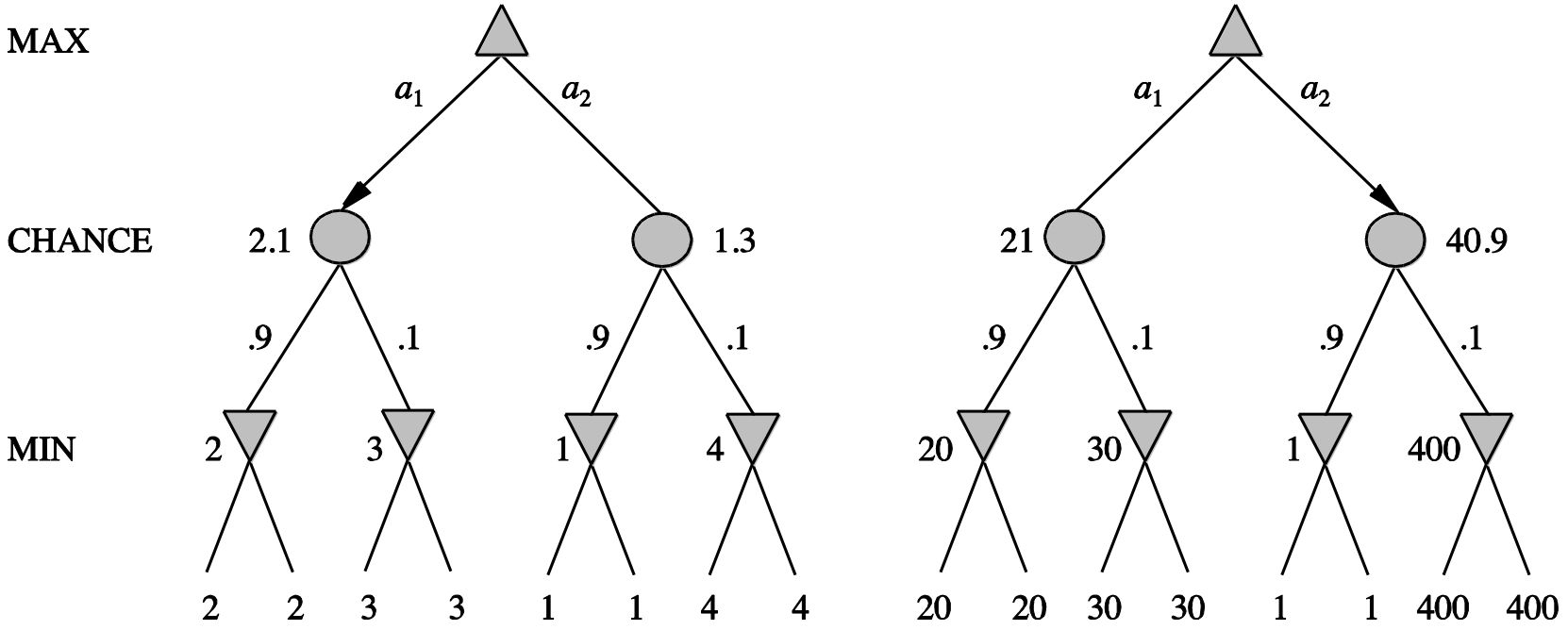
όπου το r συμβολίζει το αποτέλεσμα μια ζαριάς (ή ενός άλλου τυχαίου γεγονότος), $P(r)$ είναι η πιθανότητα του r και $\textsc{Result}(s,r)$ είναι η ίδια κατάσταση όπως η s με την επιπλέον συνθήκη ότι το αποτέλεσμα της ζαριάς είναι r.

- 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


MAX
CHANCE
MIN
a_1
a_2
2.1
1.3
.9
.1
2
3
1
4
2 2 3 3 1 1 4 4
21
40.9
20
30
400
20 20 30 30 1 1 400 400

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 $\boldsymbol{O(b^m n^m)}$ 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.

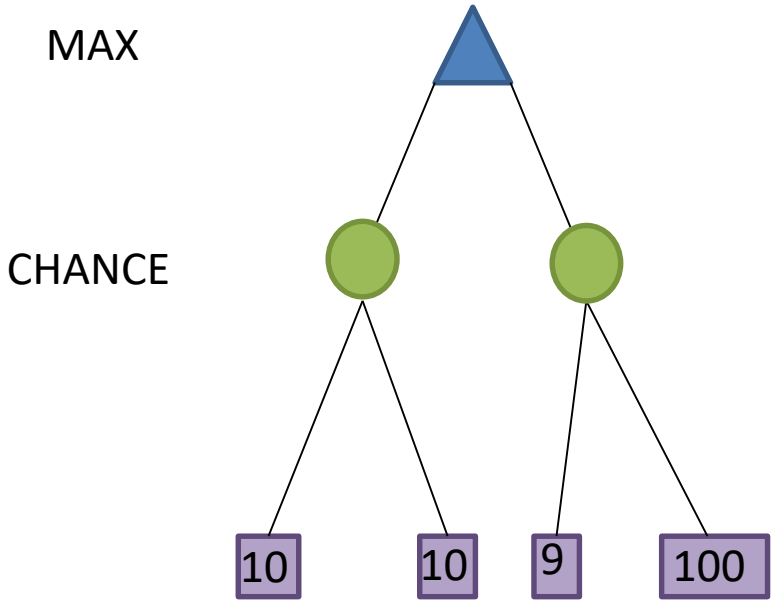
Άλλες Τεχνικές

- 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 **αναζήτηση δένδρου Monte Carlo** για να αποτιμήσουμε μια κατάσταση.
- Για το τάβλι, η τεχνική δουλεύει ως εξής. Από μια δοσμένη κατάσταση, ο αλγόριθμος παίζει χιλιάδες παιχνίδια με τον εαυτό του χρησιμοποιώντας τυχαίες ζαριές. Για το τάβλι, έχει δειχτεί 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 pacman με φαντάσματα τα οποία κινούνται τυχαία. Ο pacman είναι ο παίκτης MAX και τα φαντάσματα είναι κόμβοι τύχης.

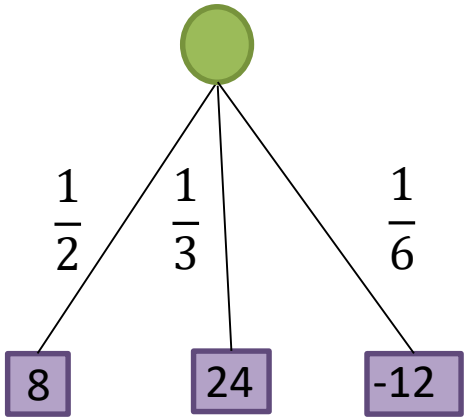
Παράδειγμα



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:

$$\textsc{Expectimax}(s) =$$

$$\begin{cases} \textsc{Utility}(s) \text{ if } \textsc{Terminal}-\textsc{Test}(s) \\ \max_a \textsc{Expectimax}\big(\textsc{Result}(s,a)\big) \text{ if } \textsc{Player}(s) = \text{MAX} \\ \sum_r P(r)\textsc{Expectimax}(\textsc{Result}(s,r)) \text{ if } \textsc{Player}(s) = \text{CHANCE} \end{cases}$$

όπου το r συμβολίζει το αποτέλεσμα μια ζαριάς (ή ενός άλλου τυχαίου γεγονότος), $P(r)$ είναι η πιθανότητα του r και $\textsc{Result}(s,r)$ 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.

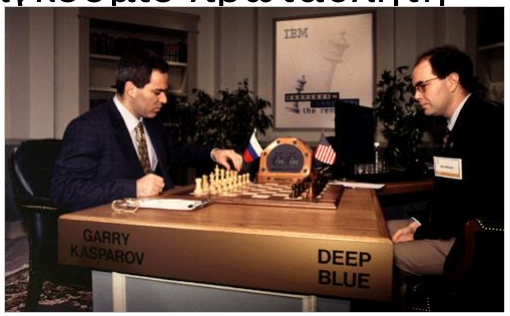
Προγράμματα Παιχνιδιών

- Το πρώτο πρόγραμμα παιχνιδιού ήταν για το παιχνίδι ντάμα (Stratchey, 1952).

- Το παιχνίδι **έχει λυθεί πλήρως** από τον Schaeffer και την ομάδα του (2007): το παιχνίδι λήγει ισόπαλο ανάμεσα σε δύο παίκτες που παίζουν τέλεια.

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 (π.χ., STOCKFISH, KOMODO, HOUDINI) μπορούν να τρέχουν σε συνηθισμένους υπολογιστές και έχουν ξεφύγει σε ικανότητες από τους παίκτες-ανθρώπους.**

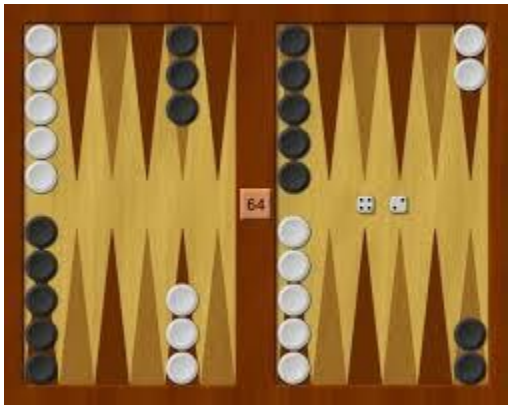
Προηγμένα Προγράμματα Παιχνιδιών

- Ντάμα.
  - Το πρόγραμμα CHINOOK (Schaeffer και συνεργάτες) 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 πρόσφατες 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.



Προηγμένα Προγράμματα Παιχνιδιών

- 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 οπτική αναγνώριση προτύπων, ενισχυτική μάθηση, βαθιά νευρωνικά δίκτυα και αναζήτηση δένδρου Monte Carlo.

Προηγμένα Προγράμματα Παιχνιδιών

- Το 2018 το πρόγραμμα AlphaZero της εταιρίας Google Deep Mind ξεπέρασε το AlphaGo και νίκησε και άλλα κορυφαία προγράμματα για σκάκι και shogi. Οι βασικές τεχνικές στο πρόγραμμα αυτό είναι αναζήτηση δένδρου Monte Carlo, βαθιά νευρωνικά δίκτυα και ενισχυτική μάθηση.

Προηγμένα Προγράμματα Παιχνιδιών

- Το 2019 παρουσιάστηκε το πρόγραμμα MuZero της εταιρίας Google Deep Mind που βασίζεται σε παρόμοιες ιδέες με το AlphaZero αλλά μαθαίνει ακόμα και τους κανόνες του παιχνιδιού παίζοντας παιχνίδια με τον εαυτό του!
- Το πρόγραμμα επιτυγχάνει κορυφαία αποτελέσματα για τα παιχνίδια Pacman, σκάκι, Go και 75 Atari games.

Προηγμένα Προγράμματα Παιχνιδιών

- Μπριτζ
  - Το πρόγραμμα GIB ήρθε 12ο (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.

Προηγμένα Προγράμματα Παιχνιδιών

- Μπορείτε να βρείτε περισσότερες πληροφορίες για προηγμένα προγράμματα παιχνιδιών καθώς και σχετική βιβλιογραφία στις βιβλιογραφικές και ιστορικές σημειώσεις του κεφαλαίου 5 του βιβλίου AIMA.

Μελέτη

- Βιβλίο AIMA
  - Κεφάλαιο 5.