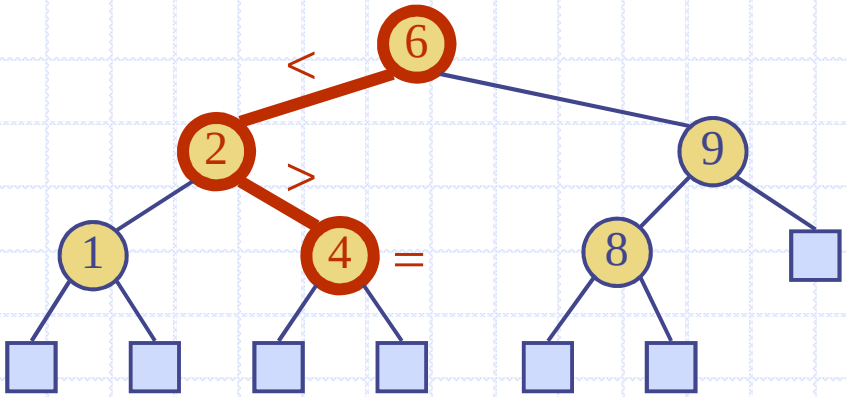
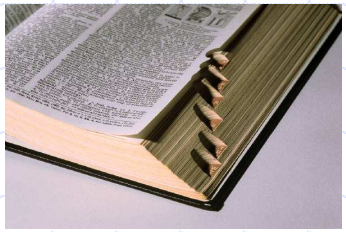
Δυαδικά Δένδρα Αναζήτησης

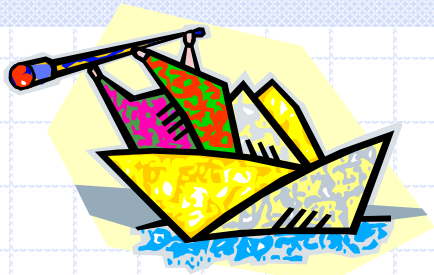


Ταξινομημένοι Χάρτες

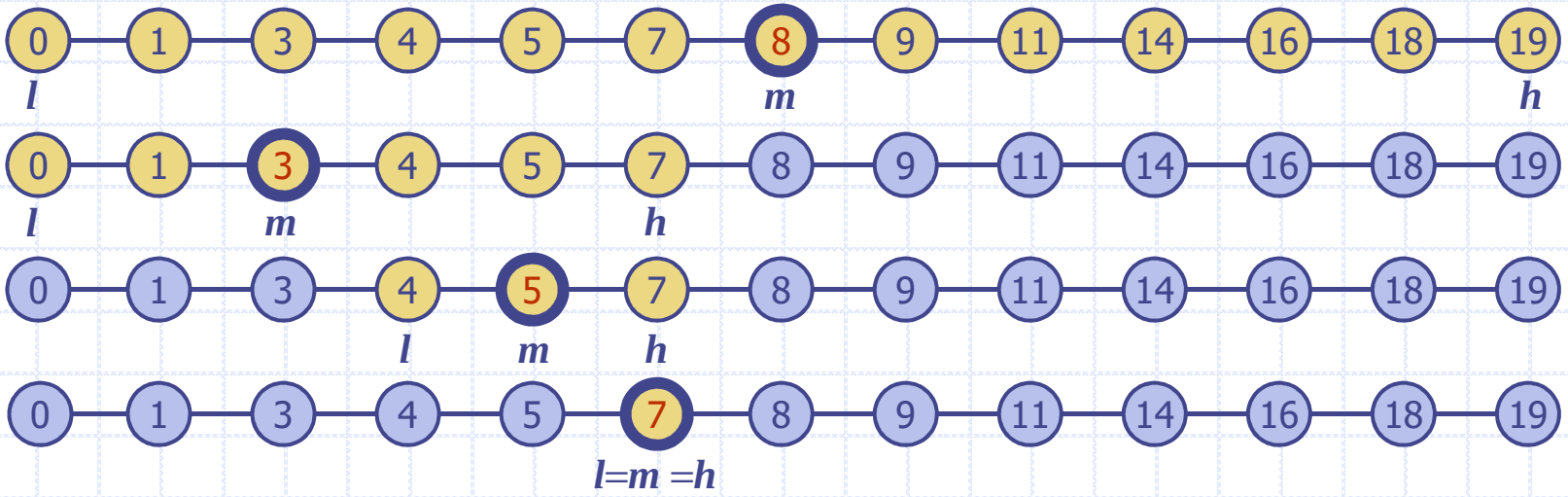


- Υποθέτουμε ότι τα κλειδιά προέρχονται από ολικη διάταξη.
- Νέες πράξεις:
  - firstEntry(): η καταχώρηση με την μικρότερη τιμή κλειδιού
  - lastEntry(): η καταχώρηση με τη μεγαλύτερη τιμή κλειδιού
  - floorEntry(k): η καταχώρηση με το μεγαλύτερο κλειδί $\leq k$
  - ceilingEntry(k): η καταχώρηση με το μικρότερο κλειδί $\geq k$
  - Οι πράξεις αυτές επιστρέφουν null αν ο χάρτης είναι κενός

Δυαδική Αναζήτηση



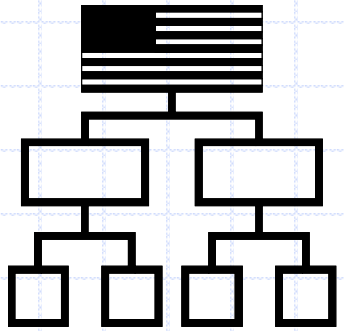
- Η δυαδική αναζήτηση μπορεί να εκτελέσει τις πράξεις get, floorEntry και ceilingEntry σε ένα ταξινομημένο χάρτη υλοποιημένο με πίνακα, ταξινομημένο με βάση το κλειδί
  - σε κάθε βήμα, το πλήθος των υποψήφιων κλειδιών υποδιπλασιάζεται
  - τερματίζει μετά από O(log n) βήματα
- Παράδειγμα: find(7)



Πίνακες Αναζήτησης

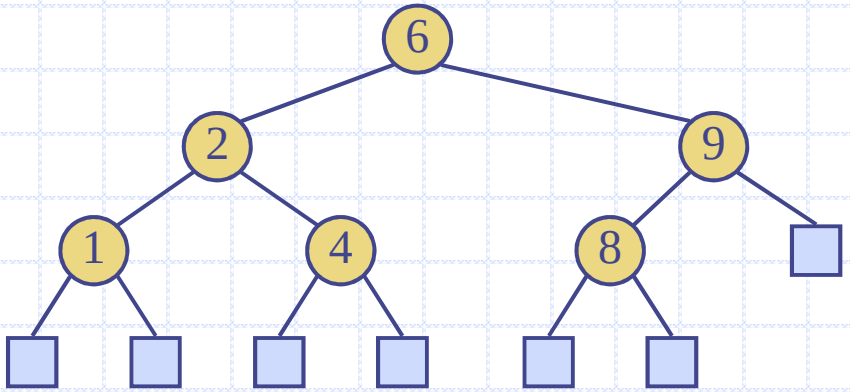
- Ένας πίνακας αναζήτησης είναι ένας ταξινομημένος χάρτης που υλοποιείται από ταξινομημένη ακολουθία
  - Αποθηκεύουμε τα στοιχεία σε μια ακολουθία που βασίζεται σε πίνακα, ταξινομημένο με βάση το κλειδί
  - Χρησιμοποιούμε έναν εξωτερικό τελεστή σύγκρισης για τα κλειδιά
- Απόδοση:
  - get, floorEntry και ceilingEntry απαιτούν $O(\log n)$ χρόνο, με δυαδική αναζήτηση
  - get απαιτεί $O(n)$ χρόνο αφού στη χειρότερη περίπτωση χρειάζεται ολίσθηση $n/2$ στοιχείων για τη δημιουργία χώρου για το νέο στοιχείο
  - remove απαιτεί $O(n)$ χρόνο αφού στη χειρότερη περίπτωση χρειάζεται ολίσθηση $n/2$ στοιχεία να ολισθήσουν μετά την απομάκρυνση
- Ο πίνακας αναζήτησης είναι αποτελεσματικός μόνο για λεξικά μικρού μεγέθους ή για λεξικά που οι πιο συχνές πράξεις είναι αναζητήσεις, ενώ οι εισαγωγές και οι διαγραφές είναι σπάνιες

Δυαδικά Δένδρα Αναζήτησης



- Ένα δυαδικό δένδρο αναζήτησης είναι ένα δυαδικό δένδρο που αποθηκεύει κλειδιά (ή καταχωρήσεις με κλειδί) στους εσωτερικούς κόμβους του και ικανοποιεί την παρακάτω ιδιότητα:
  - Έστω u, v, και w τρεις κόμβοι τέτοιοι ώστε ο u είναι στο αριστερό υποδένδρο του v και ο w είναι στο δεξιό υποδένδρο του v. Έχουμε $key(u) \leq key(v) \leq key(w)$
- Οι εξωτερικοί κόμβοι δεν αποθηκεύουν στοιχεία

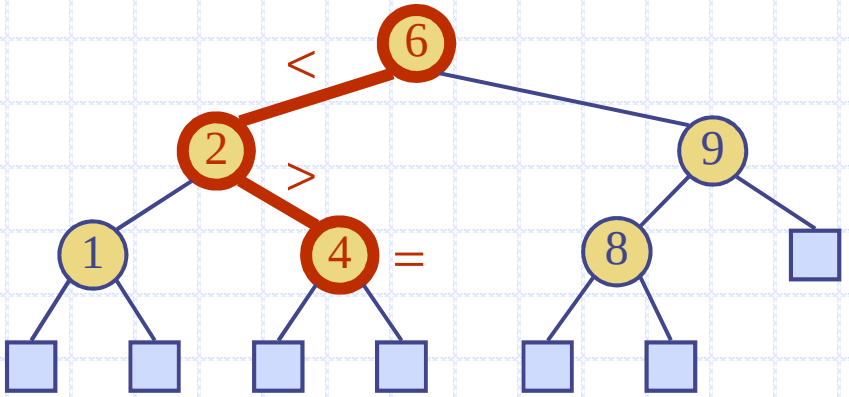
- Μια ενδοδιατεταγμένη σάρωση ενός δυαδικού δένδρου αναζήτησης επισκέπτεται τα κλειδιά σε αύξουσα σειρά



Αναζήτηση

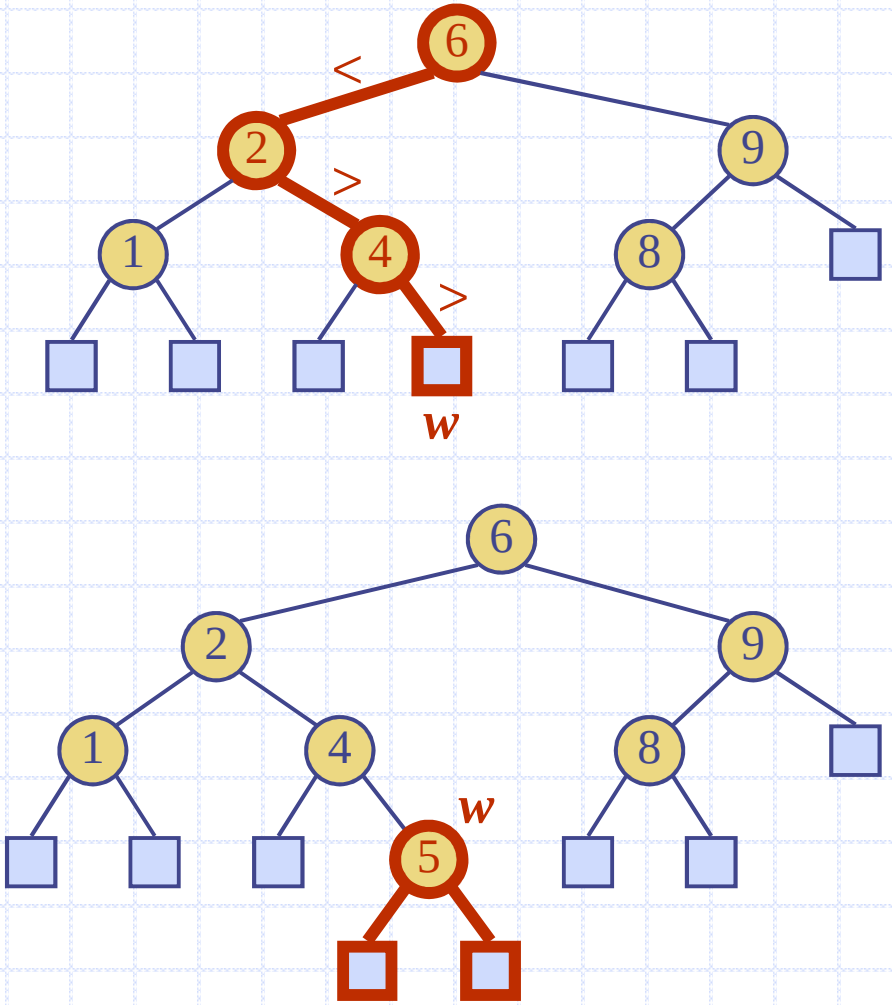
- Για αναζήτηση του κλειδιού k, ξεκινάμε από τη ρίζα και ακολουθούμε μια προς τα κάτω διαδρομή
- Ο επόμενος προς επίσκεψη κόμβος εξαρτάται από τη σύγκριση του k με το κλειδί του τρέχοντος κόμβου
- Αν φθάσουμε σε φύλλο, το κλειδί δεν υπάρχει
- Παράδειγμα: get(4):
  - Κλήση TreeSearch(4,root)
- Παρόμοιοι είναι και οι αλγόριθμοι floorEntry και ceilingEntry

```
Algorithm TreeSearch(k, v)
    if T.isExternal (v)
        return v
    if k < key(v)
        return TreeSearch(k, T.left(v))
    else if k = key(v)
        return v
    else { k > key(v) }
        return TreeSearch(k, T.right(v))
```



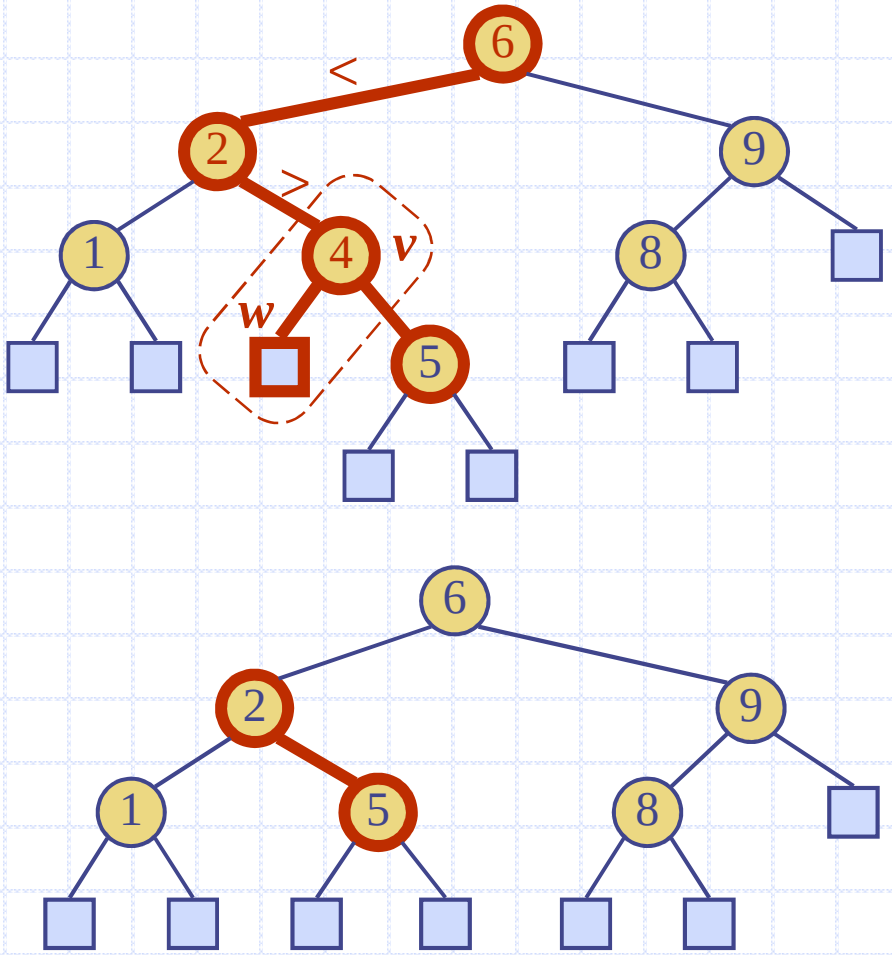
Εισαγωγή

- Για εκτέλεση της πράξης put(k, o), αναζητούμε το κλειδί k (με χρήση της TreeSearch)
- Έστω ότι το k δεν είναι ήδη στο δένδρο, και έστω w ο κόμβος που σταματά η αναζήτηση
- Εισάγουμε το k στον κόμβο w και επεκτείνουμε τον w σε ένα εσωτερικό κόμβο
- Παράδειγμα: εισαγωγή του 5



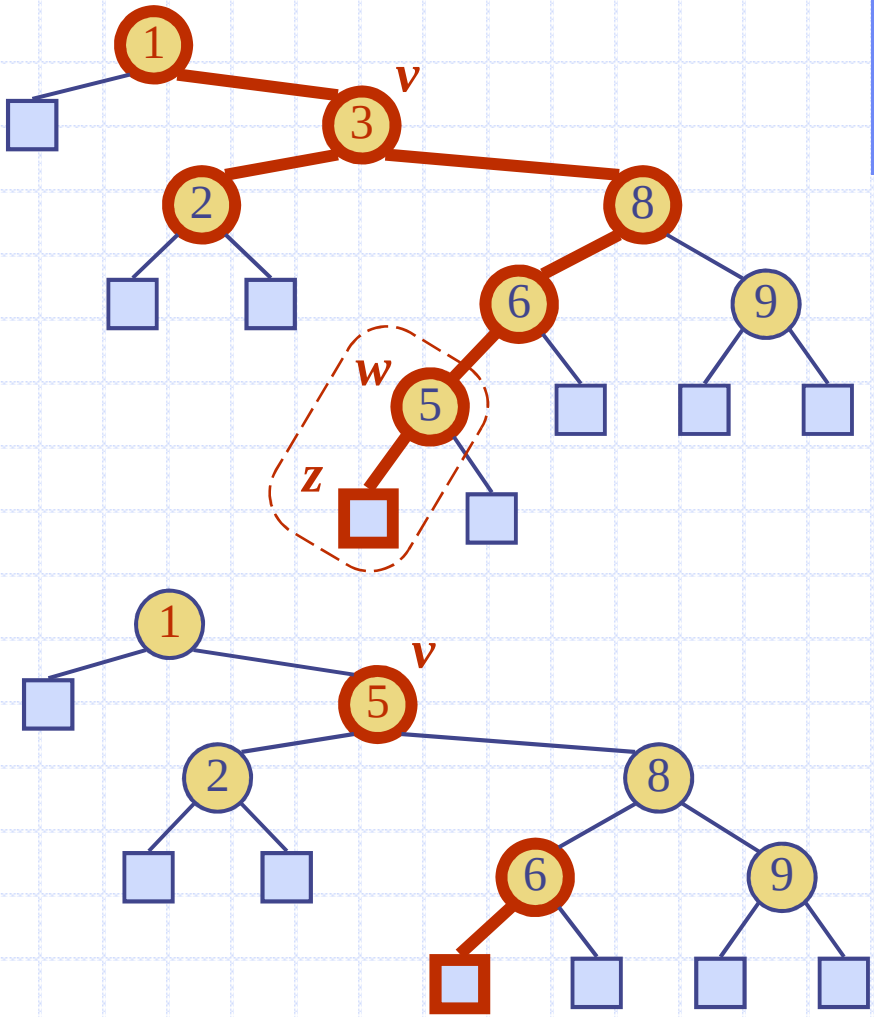
Διαγραφή

- Για εκτέλεση της remove(k), ψάχνουμε για το κλειδί k
- Ας υποθέσουμε ότι το k είναι στο δένδρο, και έστω ότι είναι καταχωρημένο στον κόμβο v
- Αν ο κόμβος v έχει ένα παιδί φύλλο w, διαγράφουμε από το δένδρο τον v και τον w με την πράξη removeExternal(w), που διαγράφει τον w και τον γονέα του
- Παράδειγμα: διαγραφή του 4



Διαγραφή (συν.)

- Εξετάζουμε την περίπτωση τα παιδιά του προς διαγραφή κόμβου v με κλειδί το k είναι και τα δύο εσωτερικά
  - βρίσκουμε τον εσωτερικό κόμβο w που ακολουθεί τον v σε μια ενδοδιατεταγμένη σάρωση
  - αντιγράφουμε το κλειδί του w στον κόμβο v
  - διαγράφουμε τον w και το αριστερό παιδί του z (που πρέπει να είναι φύλλο) με χρήση της πράξης removeExternal(z)
- Παράδειγμα: διαγραφή του 3



Απόδοση

- Έστω ένας διατεταγμένος χάρτης με n στοιχεία που υλοποιείται από ένα δυαδικό δένδρο ύψους h
  - ο απαιτούμενος χώρος είναι $O(n)$
  - οι μέθοδοι get, floorEntry, ceilingEntry, put και remove απαιτούν χρόνο $O(h)$
- Το ύψος h είναι $O(n)$ στη χειρότερη περίπτωση και $O(\log n)$ στην καλύτερη