Παροράματα στις σημειώσεις

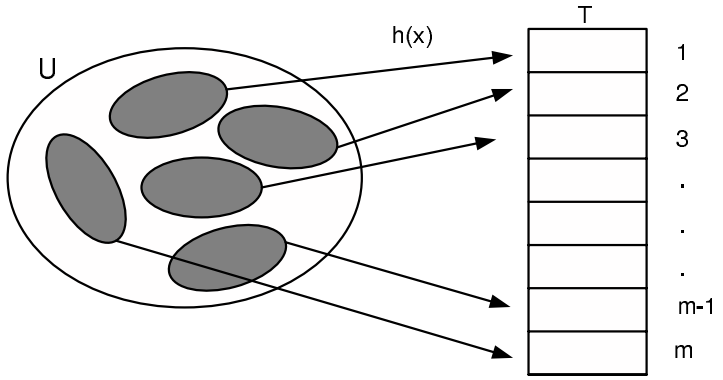
- σελ. 42 η σχέση 2.1 δίνει αποτέλεσμα $n(n+1)/2$
- σελ. 48 η σχέση 2.6 είναι

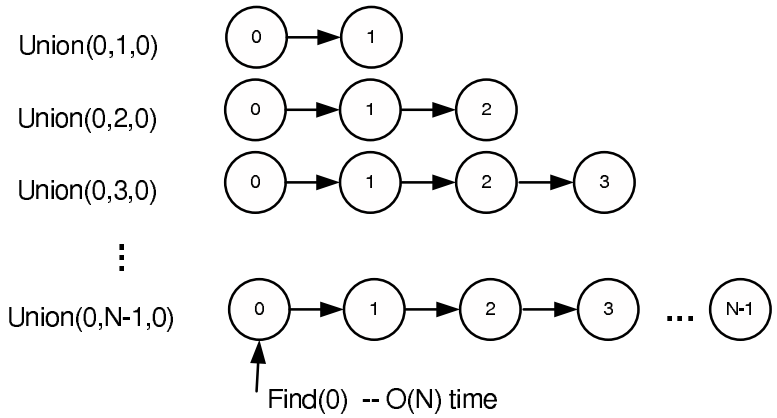
$$O\left(n\left(1+\sum_{k=0}^{\infty} k\cdot\frac{1}{2^k}\right)\right)$$

- στο Σχήμα 2.8 αντικαταστήστε το j με i.
- στο Σχήμα 2.9, στην "Εναλλαγή 8 με 1", είναι $i=5$
- σελ. 56, γραμμή 2, η σωστή σχέση είναι:

$$QS_{av}(n+1) = 2 + \frac{n+2}{n+1}\left(2+\frac{n+1}{n}\cdot QS_{av}(n-1)\right)$$

- σελ. 84, γραμμή -10, η σωστή διατύπωση είναι δηλαδή ότι ο δείκτης παρεμβολής είναι ίσος με την μέση τιμή της X.
- σελ. 133, Θεώρημα 5.38, η συνολική πολυπλοκότητα για n τυχαίες εισαγωγές/διαγραφές είναι $O(n)$ αντί $O(n\log n)$ που αναφέρεται.
- σελ. 174, το σχήμα 7.2 είναι:





Σχήμα 1: Σενάριο όπου η πράξη FIND στο στοιχείο που υποδεικνύεται με βέλος, χρειάζεται N βήματα

- στο τέλος της σελ. 231 λείπει το Σχήμα 10.2, το οποίο είναι το Σχήμα 1
- σελ. 232, γραμμή 4, η σωστή διατύπωση είναι: $...rank(y) = rank(x) - 1$. Όταν ο y έγινε γιος του x,...

Δομες Δεδομενων

Ταξινόμηση

Σαλτογιάννη Αθανασία

Ταξινόμηση

Ταξινόμηση

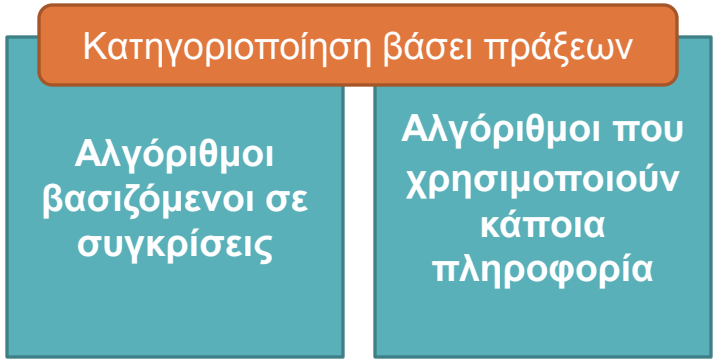
- Τι εννοούμε όταν λέμε ταξινόμηση;

Ταξινόμηση

- Τι εννοούμε όταν λέμε ταξινόμηση;
- Ποια είδη αλγορίθμων ταξινόμησης υπάρχουν;

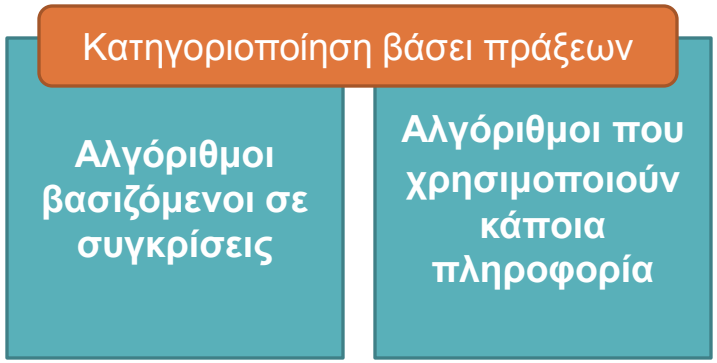
Ταξινόμηση

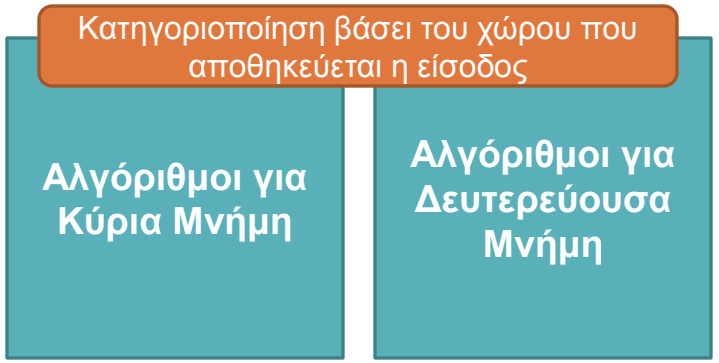
- Τι εννοούμε όταν λέμε ταξινόμηση;
- Ποια είδη αλγορίθμων ταξινόμησης υπάρχουν;



Ταξινόμηση

- Τι εννοούμε όταν λέμε ταξινόμηση;
- Ποια είδη αλγορίθμων ταξινόμησης υπάρχουν;





Ταξινόμηση

- Τι εννοούμε όταν λέμε ταξινόμηση;
- Ποια είδη αλγορίθμων ταξινόμησης υπάρχουν;

Τι είναι Κύρια Μνήμη;

Τι είναι Δευτερεύουσα Μνήμη;

Bubble Sort – Ταξινόμηση Φυσαλίδας

- **Έχω τον πίνακα $S = \{s_1, s_2, \dots, s_n\}$ προς ταξινόμηση**
  - Σαρώνω όλο τον πίνακα από την αρχή
  - Συγκρίνω κάθε φορά τα στοιχεία $S[i]$ και $S[i+1]$
  - Εναλλάσσω τα στοιχεία έτσι ώστε το μικρότερο να είναι αριστερά και το μεγαλύτερο δεξιά
  - Συνεχίζω μέχρι να σαρωθεί όλος ο πίνακας και το μεγαλύτερο στοιχείο φτάσει δεξιά
  - Επαναλαμβάνω την παραπάνω διαδικασία για τα μη ταξινομημένα στοιχεία που έμειναν

Bubble Sort – Ταξινόμηση Φυσαλίδας

- **Έχω τον πίνακα $S = \{s_1, s_2, \dots, s_n\}$ προς ταξινόμηση**
  - Σαρώνω όλο τον πίνακα από την αρχή
  - Συγκρίνω κάθε φορά τα στοιχεία $S[i]$ και $S[i+1]$
  - Εναλλάσσω τα στοιχεία έτσι ώστε το μικρότερο να είναι αριστερά και το μεγαλύτερο δεξιά
  - Συνεχίζω μέχρι να σαρωθεί όλος ο πίνακας και το μεγαλύτερο στοιχείο φτάσει δεξιά
  - Επαναλαμβάνω την παραπάνω διαδικασία για τα μη ταξινομημένα στοιχεία που έμειναν

- Αλγόριθμος για Κύρια Μνήμη
- Αλγόριθμος βασιζόμενος σε συγκρίσεις

Bubble Sort – Ταξινόμηση Φυσαλίδας

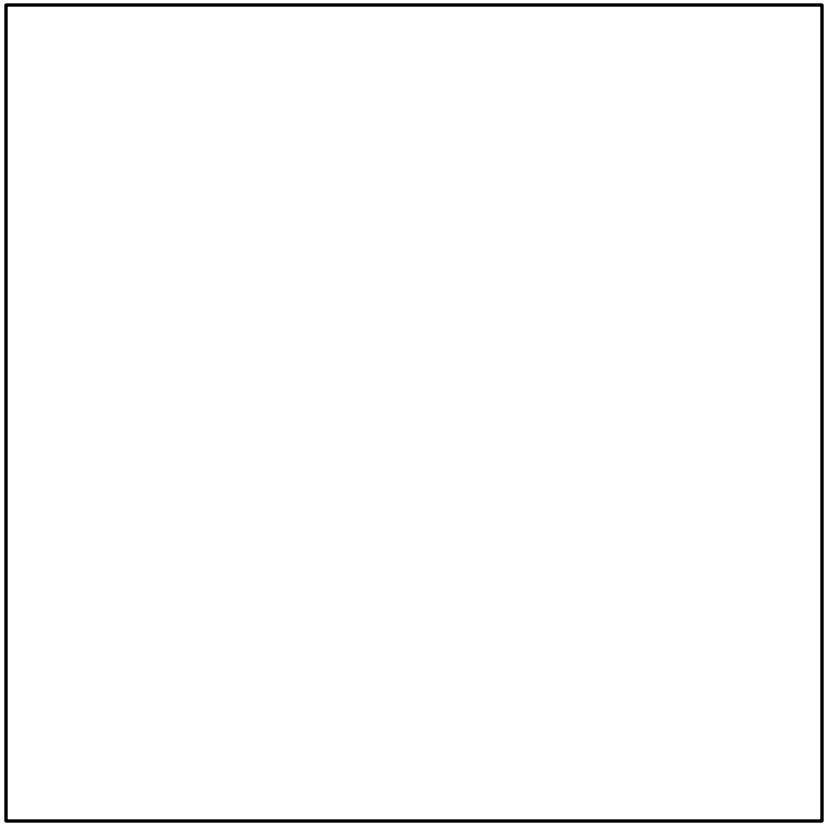
Παράδειγμα

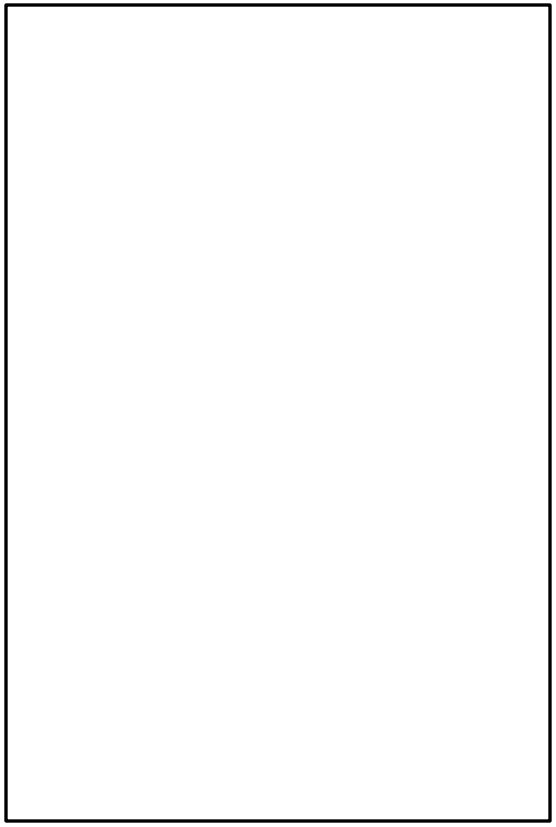
Έστω S = {15, 9, 6, 22, 10, 8, 4}

Bubble Sort – Ταξινόμηση Φυσαλίδας

Παράδειγμα

Έστω S = {15, 9, 6, 22, 10, 8, 4}





Bubble Sort – Ταξινόμηση Φυσαλίδας

Παράδειγμα

Έστω S = {15, 9, 6, 22, 10, 8, 4}

Συγκρίνω 15 και 9	**15**, **9**, 6, 22, 10, 8, 4

Bubble Sort – Ταξινόμηση Φυσαλίδας

Παράδειγμα

Έστω S = {15, 9, 6, 22, 10, 8, 4}

Συγκρίνω 15 και 9 → Εναλλάσσω	**15**, **9**, 6, 22, 10, 8, 4

Bubble Sort – Ταξινόμηση Φυσαλίδας

Παράδειγμα

Έστω S = {15, 9, 6, 22, 10, 8, 4}

Συγκρίνω 15 και 9 → Εναλλάσσω	**15**, **9**, 6, 22, 10, 8, 4 9, 15, 6, 22, 10, 8, 4

Bubble Sort – Ταξινόμηση Φυσαλίδας

Παράδειγμα

Έστω S = {15, 9, 6, 22, 10, 8, 4}

Συγκρίνω 15 και 9 → Εναλλάσσω	**15**, **9**, 6, 22, 10, 8, 4 9, 15, 6, 22, 10, 8, 4
Συγκρίνω 15 και 6	9, **15**, **6**, 22, 10, 8, 4

Bubble Sort – Ταξινόμηση Φυσαλίδας

Παράδειγμα

Έστω S = {15, 9, 6, 22, 10, 8, 4}

Συγκρίνω 15 και 9 → Εναλλάσσω	**15**, **9**, 6, 22, 10, 8, 4 9, 15, 6, 22, 10, 8, 4
Συγκρίνω 15 και 6 → Εναλλάσσω	9, **15**, **6**, 22, 10, 8, 4

Bubble Sort – Ταξινόμηση Φυσαλίδας

Παράδειγμα

Έστω S = {15, 9, 6, 22, 10, 8, 4}

Συγκρίνω 15 και 9 → Εναλλάσσω	**15**, **9**, 6, 22, 10, 8, 4 9, 15, 6, 22, 10, 8, 4
Συγκρίνω 15 και 6 → Εναλλάσσω	9, **15**, **6**, 22, 10, 8, 4 9, 6, 15, 22, 10, 8, 4

Bubble Sort – Ταξινόμηση Φυσαλίδας

Παράδειγμα

Έστω S = {15, 9, 6, 22, 10, 8, 4}

Συγκρίνω 15 και 9 → Εναλλάσσω	**15**, **9**, 6, 22, 10, 8, 4 9, 15, 6, 22, 10, 8, 4
Συγκρίνω 15 και 6 → Εναλλάσσω	9, **15**, **6**, 22, 10, 8, 4 9, 6, 15, 22, 10, 8, 4
Συγκρίνω 15 και 22 → Αφήνω	9, 6, **15**, **22**, 10, 8, 4
Συγκρίνω 22 και 10 → Εναλλάσσω	9, 6, 15, **22**, **10**, 8, 4
Συγκρίνω 22 και 8 → Εναλλάσσω	9, 6, 15, 10, **22**, **8**, 4
Συγκρίνω 22 και 4 → Εναλλάσσω	9, 6, 15, 10, 8, **22**, **4**
	9, 6, 15, 10, 8, 4, **22**

Bubble Sort – Ταξινόμηση Φυσαλίδας

Παράδειγμα

Έστω S = {15, 9, 6, 22, 10, 8, 4}

Συγκρίνω 15 και 9 → Εναλλάσσω	**15**, **9**, 6, 22, 10, 8, 4 9, 15, 6, 22, 10, 8, 4
Συγκρίνω 15 και 6 → Εναλλάσσω	9, **15**, **6**, 22, 10, 8, 4 9, 6, 15, 22, 10, 8, 4
Συγκρίνω 15 και 22 → Αφήνω	9, 6, **15**, **22**, 10, 8, 4
Συγκρίνω 22 και 10 → Εναλλάσσω	9, 6, 15, **22**, **10**, 8, 4
Συγκρίνω 22 και 8 → Εναλλάσσω	9, 6, 15, 10, **22**, **8**, 4
Συγκρίνω 22 και 4 → Εναλλάσσω	9, 6, 15, 10, 8, **22**, **4**
Επαναλαμβάνω τα βήματα για αυτό το κομμάτι	9, 6, 15, 10, 8, 4, **22**

Bubble Sort – Ταξινόμηση Φυσαλίδας

Παράδειγμα

Έστω S = {15, 9, 6, 22, 10, 8, 4}

Συγκρίνω 15 και 9 → Εναλλάσσω	**15**, **9**, 6, 22, 10, 8, 4 9, 15, 6, 22, 10, 8, 4
Συγκρίνω 15 και 6 → Εναλλάσσω	9, **15**, **6**, 22, 10, 8, 4 9, 6, 15, 22, 10, 8, 4
Συγκρίνω 15 και 22 → Αφήνω	9, 6, **15**, **22**, 10, 8, 4
Συγκρίνω 22 και 10 → Εναλλάσσω	9, 6, 15, **22**, **10**, 8, 4
Συγκρίνω 22 και 8 → Εναλλάσσω	9, 6, 15, 10, **22**, **8**, 4
Συγκρίνω 22 και 4 → Εναλλάσσω	9, 6, 15, 10, 8, **22**, **4**
Επαναλαμβάνω τα βήματα για αυτό το κομμάτι	9, 6, 15, 10, 8, 4, **22**

Bubble Sort – Ταξινόμηση Φυσαλίδας

- **Χειρότερη περίπτωση**
  - Αντίστροφα ταξινομημένος πίνακας
  - Στο πέρασμα i χρειάζονται (n-i) συγκρίσεις και ανταλλαγές
  - Συνολικό πλήθος συγκρίσεων και ανταλλαγών

$$\sum_{i=1}^{n-1} i = \frac{\mathrm{n}(\mathrm{n}-1)}{2} = \mathrm{O}(\mathrm{n}^2)$$

- **Καλύτερη περίπτωση**
  - Ταξινομημένος πίνακας
  - Κανένα στοιχείο δεν μετακινείται
  - Συνολικό πλήθος συγκρίσεων

$$\sum_{i=1}^{n-1} 1 = (\mathrm{n}-1) = \mathrm{O}(n)$$

Insertion Sort – Ταξινόμηση με Εισαγωγή

- **Έχω τον πίνακα $S = \{s_1, s_2, \dots, s_n\}$ προς ταξινόμηση**
  - Αρχικά όλα τα στοιχεία είναι μη ταξινομημένα
  - Ταξινομούμε τα στοιχεία ένα κάθε φορά
    - Παίρνουμε ένα στοιχεία από αυτά που δεν έχουν ταξινομηθεί (i≥2)
    - Βρίσκουμε την κατάλληλη θέση μεταξύ των ταξινομημένων
    - Μετακινούμε τα μεγαλύτερα στοιχεία μία θέση δεξιά
    - Εισάγουμε το στοιχείο στην θέση που απελευθερώθηκε
  - Επαναλαμβάνουμε μέχρι να ταξινομήσουμε όλα τα στοιχεία

Insertion Sort – Ταξινόμηση με Εισαγωγή

- **Έχω τον πίνακα $S = \{s_1, s_2, \dots, s_n\}$ προς ταξινόμηση**
  - Αρχικά όλα τα στοιχεία είναι μη ταξινομημένα
  - Ταξινομούμε τα στοιχεία ένα κάθε φορά
    - Παίρνουμε ένα στοιχεία από αυτά που δεν έχουν ταξινομηθεί (i≥2)
    - Βρίσκουμε την κατάλληλη θέση μεταξύ των ταξινομημένων
    - Μετακινούμε τα μεγαλύτερα στοιχεία μία θέση δεξιά
    - Εισάγουμε το στοιχείο στην θέση που απελευθερώθηκε
  - Επαναλαμβάνουμε μέχρι να ταξινομήσουμε όλα τα στοιχεία

- Αλγόριθμος για Κύρια Μνήμη
- Αλγόριθμος βασιζόμενος σε συγκρίσεις

Insertion Sort – Ταξινόμηση με Εισαγωγή

Παράδειγμα

Έστω S = {25, 57, 48, 37, 12, 92, 86, 33}

Insertion Sort – Ταξινόμηση με Εισαγωγή

Παράδειγμα

Έστω S = {25, 57, 48, 37, 12, 92, 86, 33}

	25	**57**	48	37	12	92	86	33
Βήμα 1	25	57	**48**	37	12	92	86	33
Βήμα 2	25	48	57	**37**	12	92	86	33
Βήμα 3	25	37	48	57	**12**	92	86	33
Βήμα 4	12	25	37	48	57	**92**	86	33
Βήμα 5	12	25	37	48	57	92	**86**	33
Βήμα 6	12	25	37	48	57	86	92	**33**
Βήμα 7	12	25	33	37	48	57	86	92

Heap Sort – Ταξινόμηση Σωρού

- **Έχω τον πίνακα $S = \{s_1, s_2, \dots, s_n\}$ προς ταξινόμηση**
  - **Φάση Δόμησης**
    - Μετατρέπω τον πίνακα S σε σωρό ξεκινώντας από την θέση 1
  - **Φάση Διαλογής**
    - Διαλέγω και απομακρύνω το μεγαλύτερο στοιχείο, το οποίο θα το φέρω στην ρίζα του σωρού. Επαναλαμβάνω για τα υπόλοιπα στοιχεία φέρνοντας στην ρίζα ξεκινώντας πάλι από την θέση 1. Θέλω πάντα να διατηρείται η ιδιότητα του Σωρού, δηλ. τιμή(πατέρας(v))≥ τιμή(v)

Heap Sort – Ταξινόμηση Σωρού

- **Έχω τον πίνακα $S = \{s_1, s_2, \dots, s_n\}$ προς ταξινόμηση**
  - **Φάση Δόμησης**
    - Μετατρέπω τον πίνακα S σε σωρό ξεκινώντας από την θέση 1
  - **Φάση Διαλογής**
    - Διαλέγω και απομακρύνω το μεγαλύτερο στοιχείο, το οποίο θα το φέρω στην ρίζα του σωρού. Επαναλαμβάνω για τα υπόλοιπα στοιχεία φέρνοντας στην ρίζα ξεκινώντας πάλι από την θέση 1. Θέλω πάντα να διατηρείται η ιδιότητα του Σωρού, δηλ. τιμή(πατέρας(v))≥ τιμή(v)
- Αλγόριθμος για Κύρια Μνήμη
- Αλγόριθμος βασιζόμενος σε συγκρίσεις

Heap Sort – Ταξινόμηση Σωρού

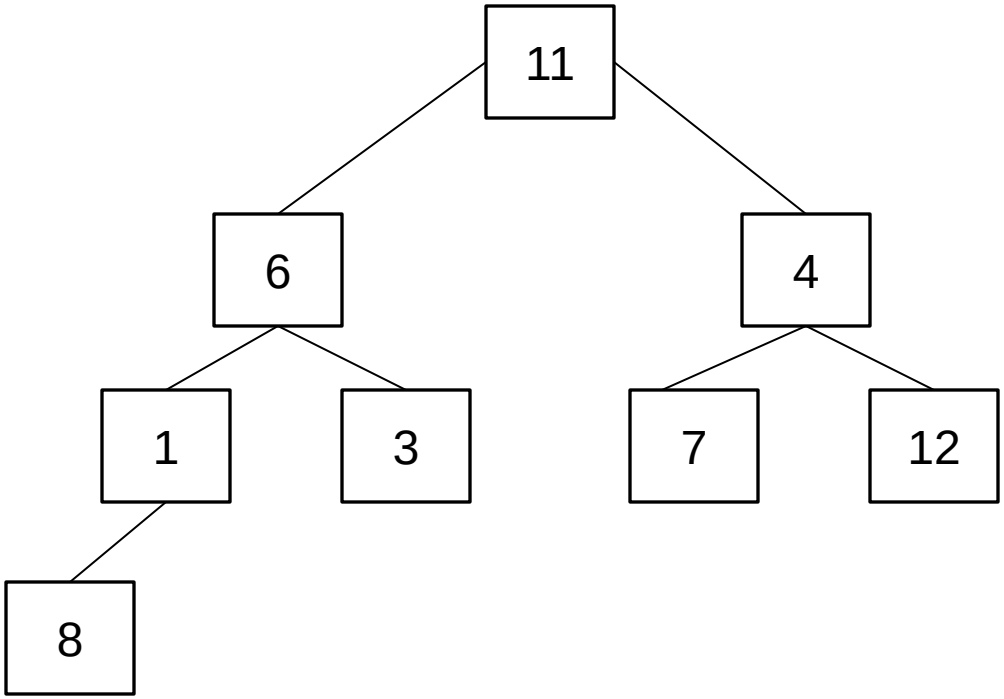
Παράδειγμα

Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}

Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

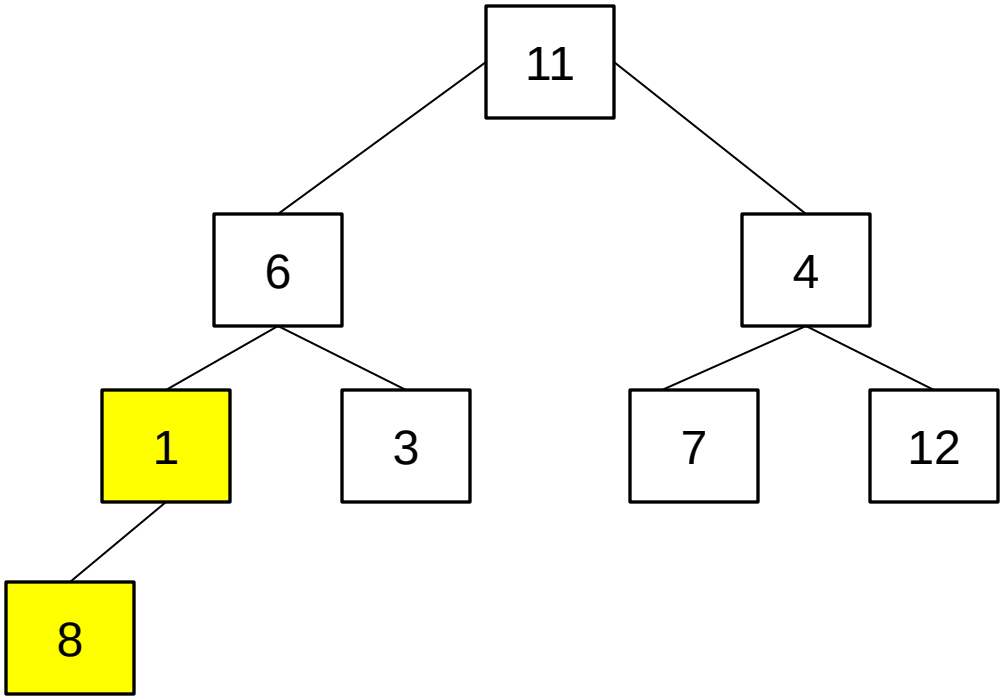
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

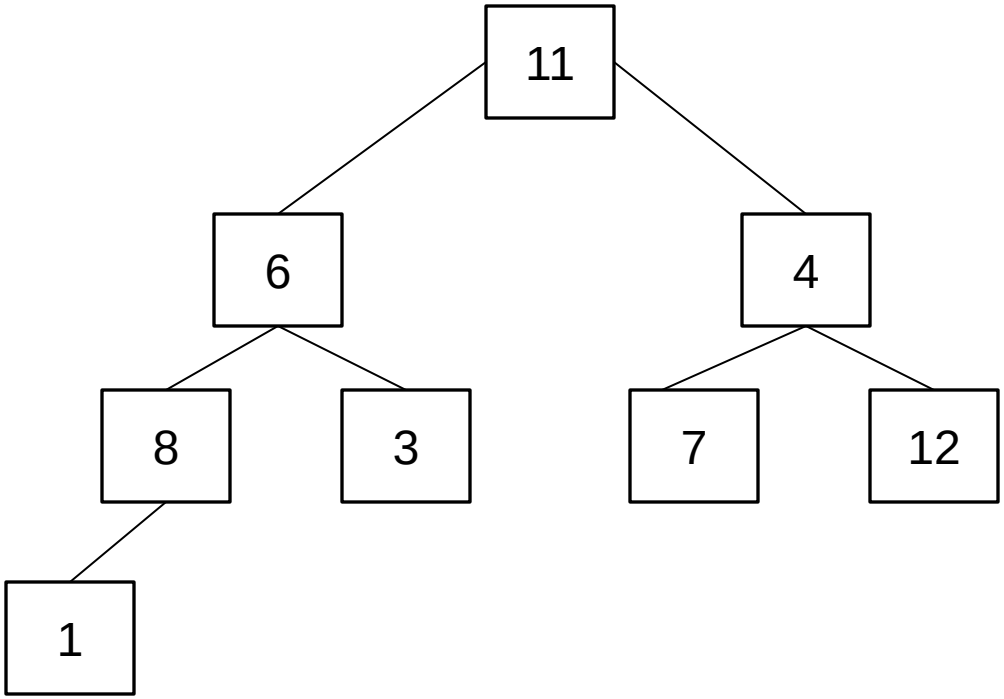
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

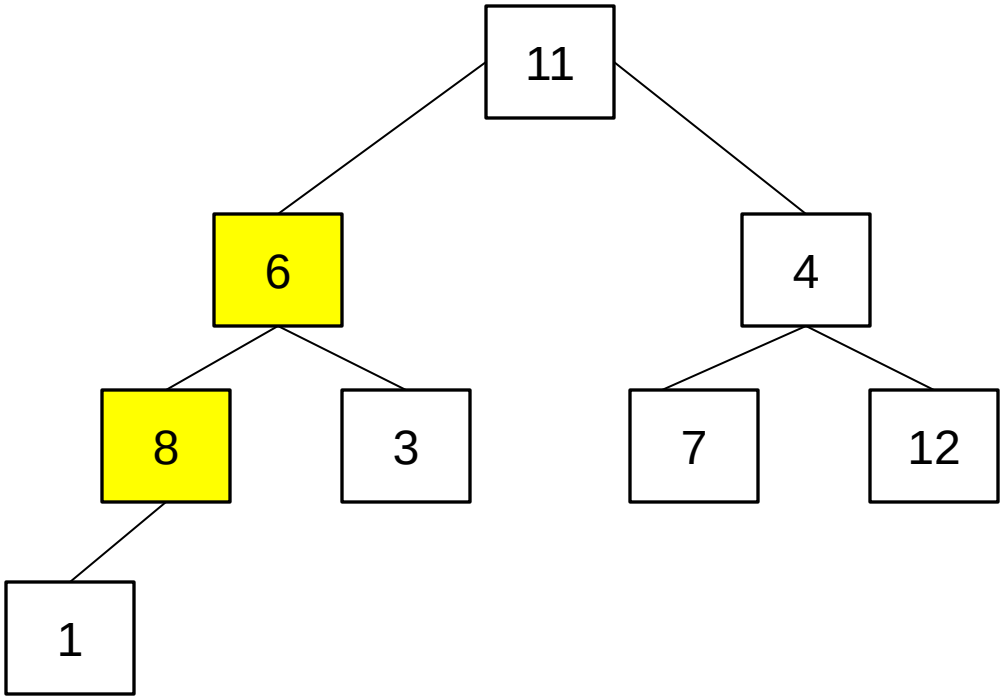
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

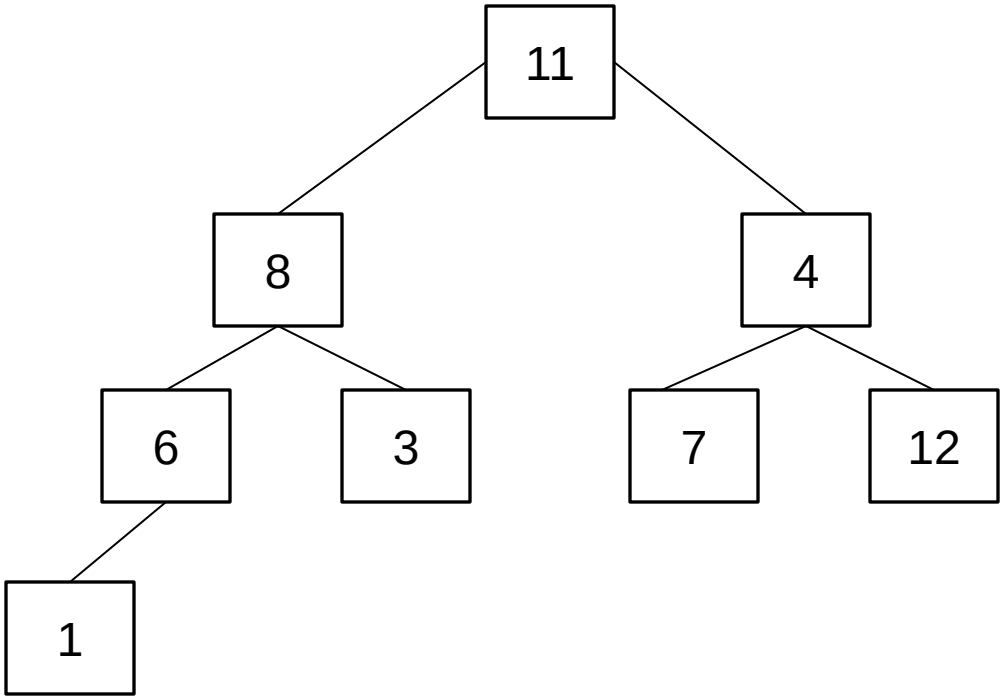
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

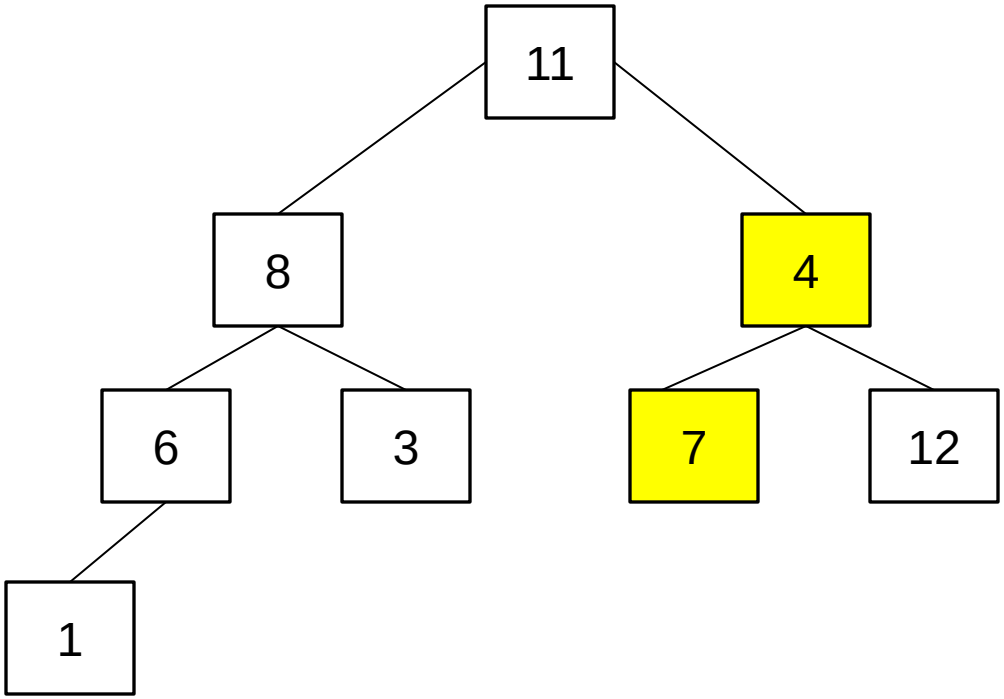
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

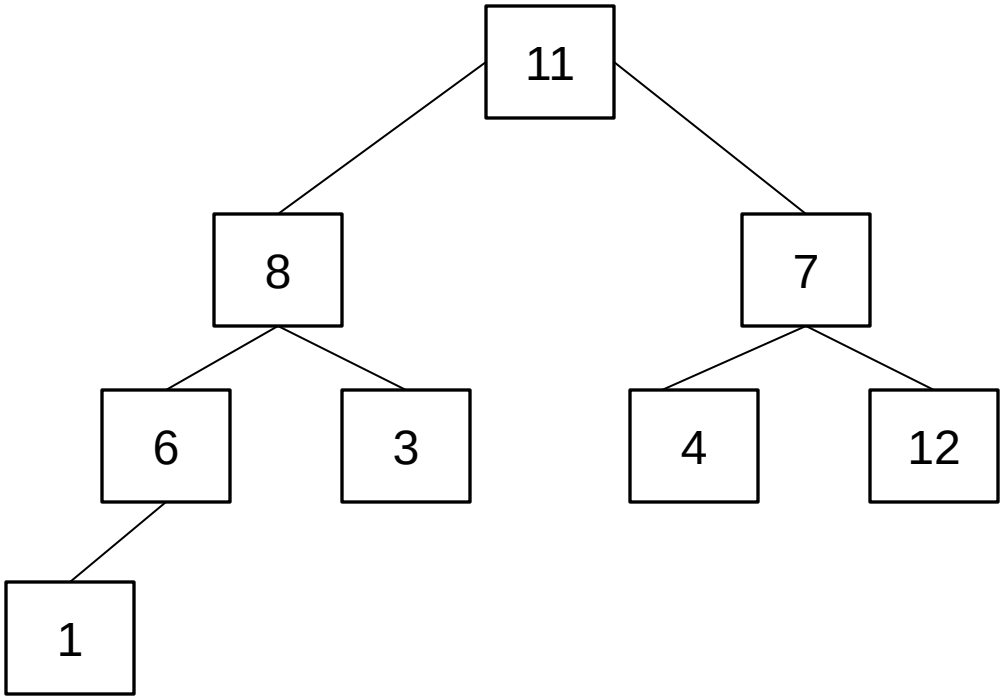
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

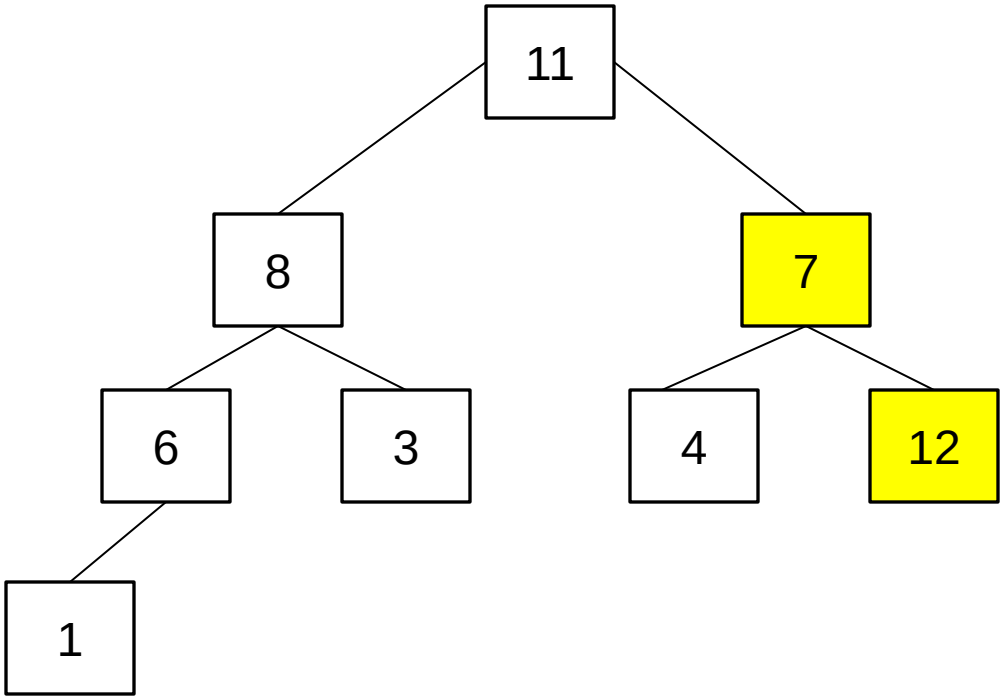
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

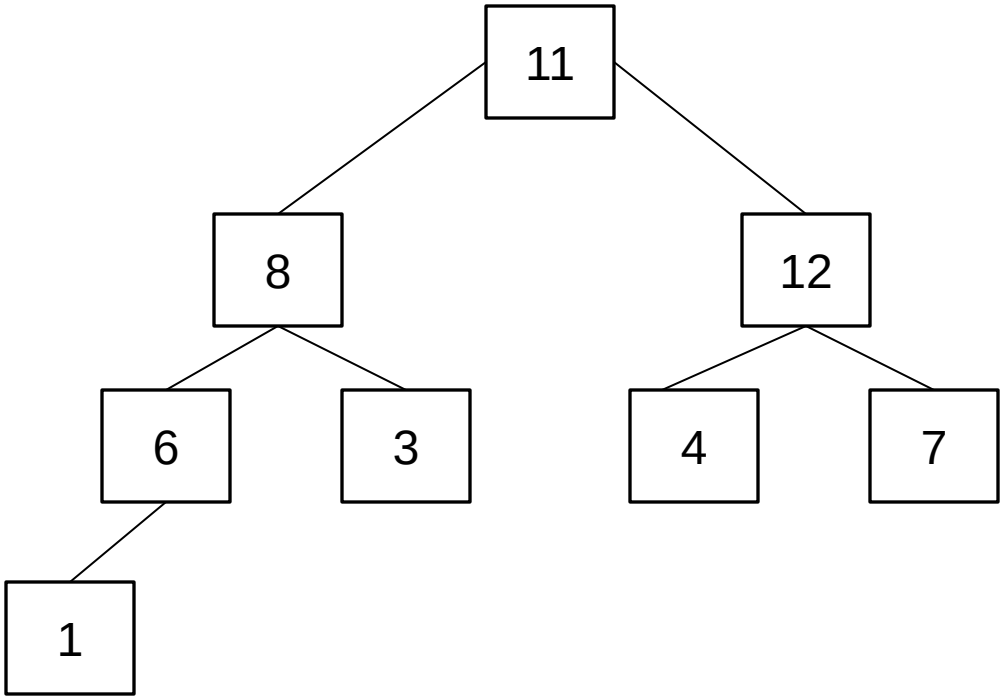
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

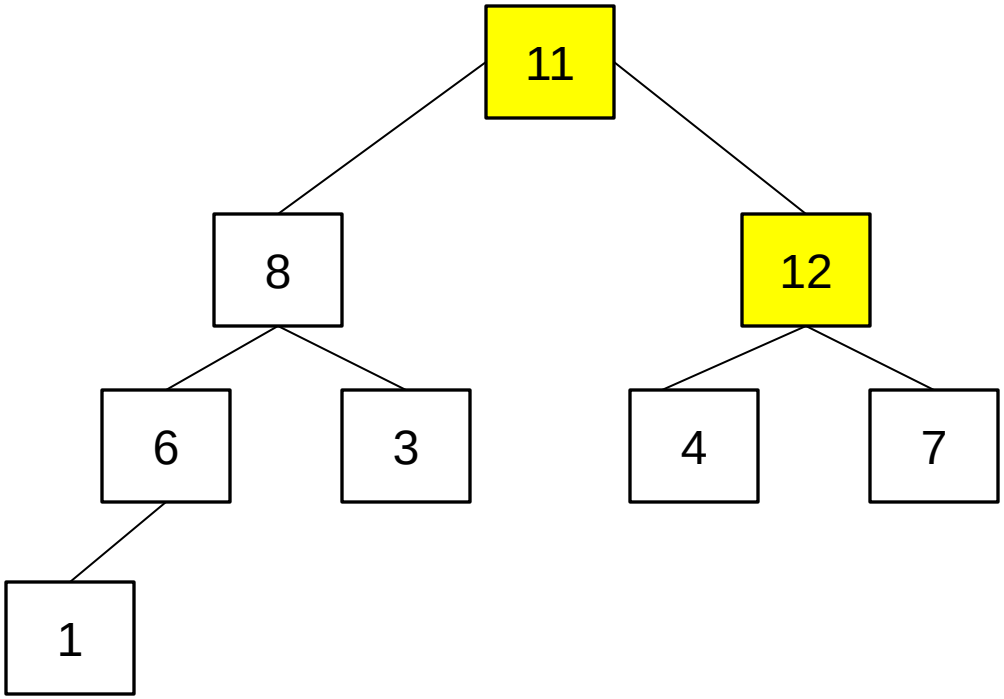
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

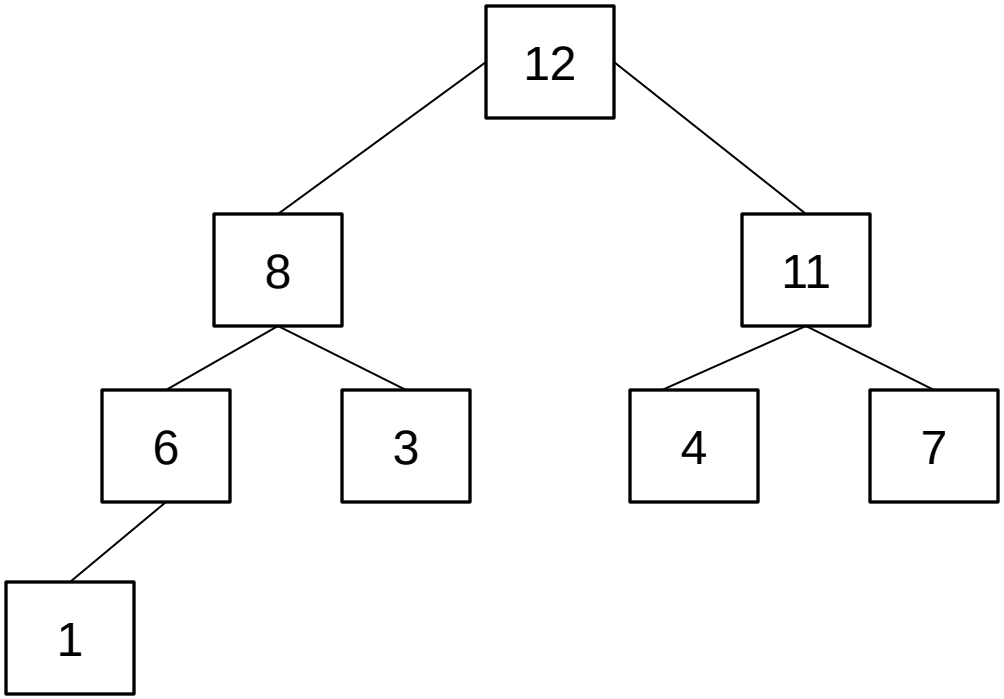
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

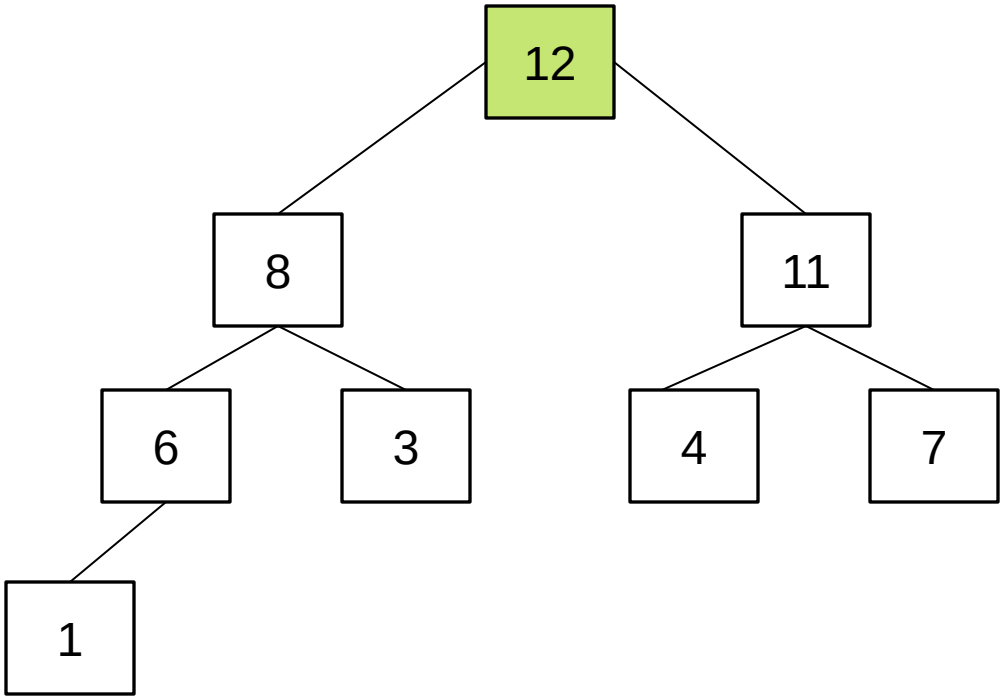
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

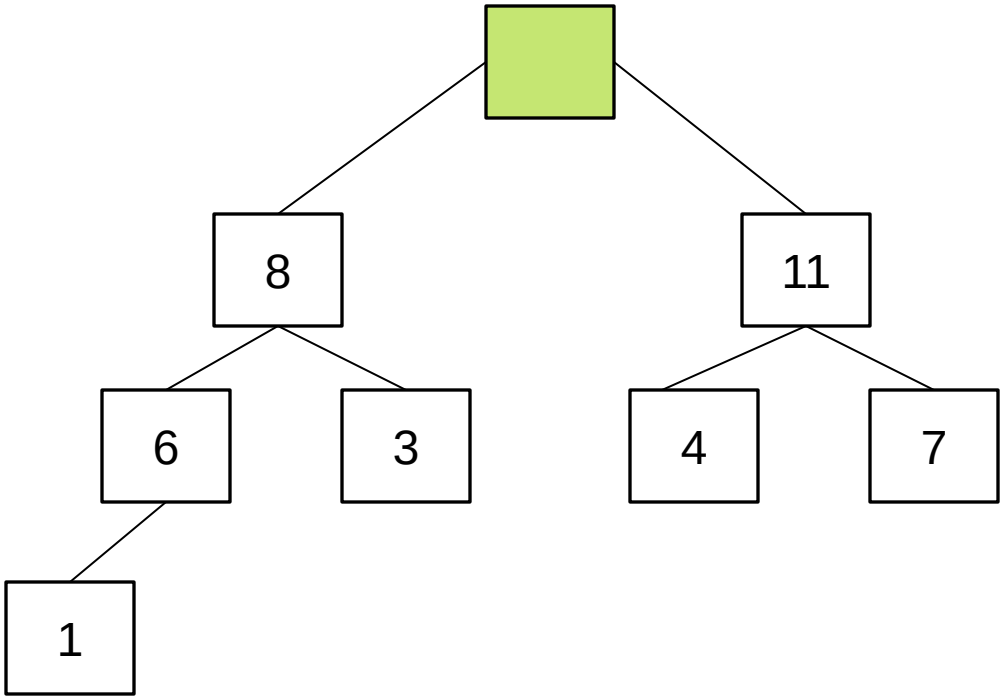
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

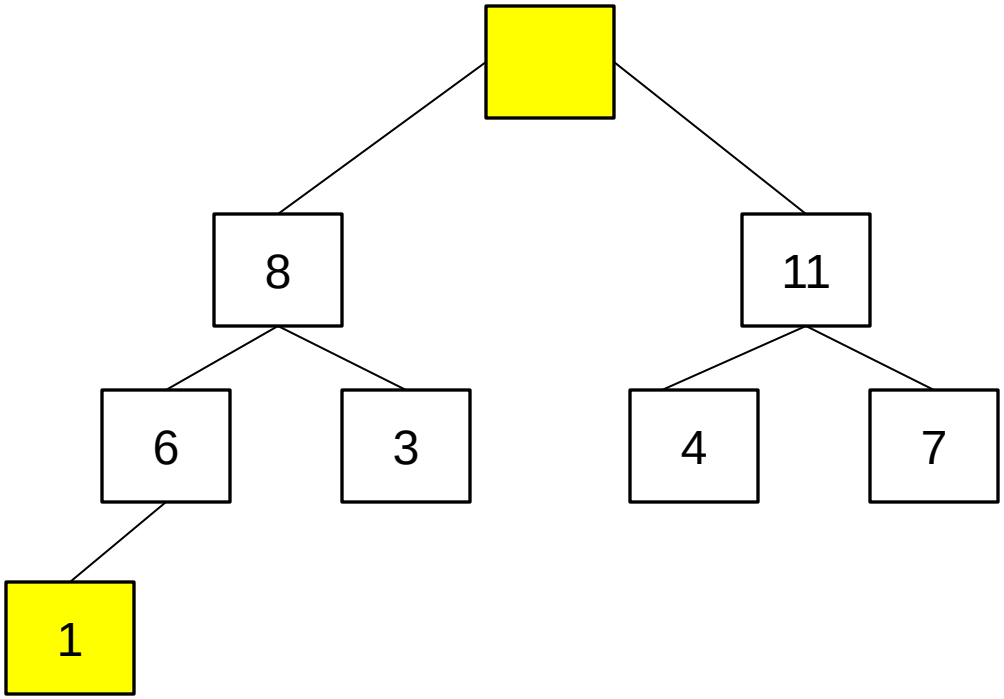
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

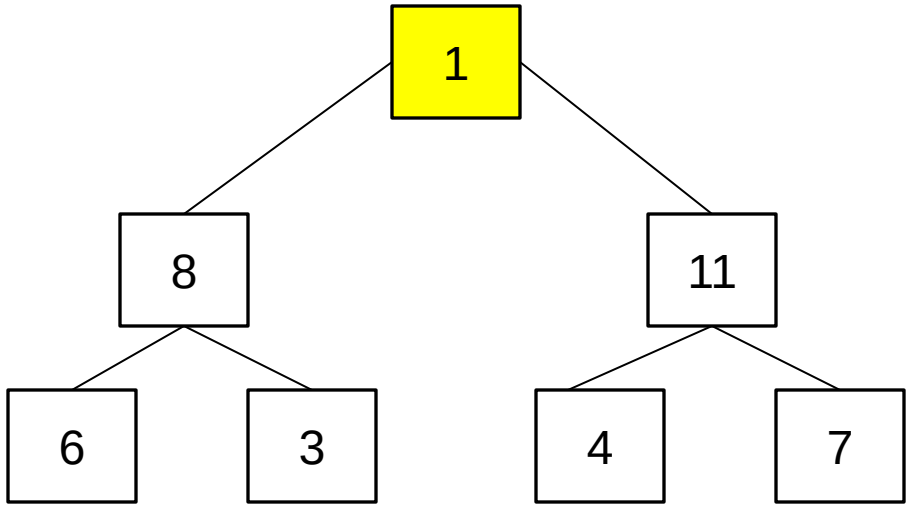
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

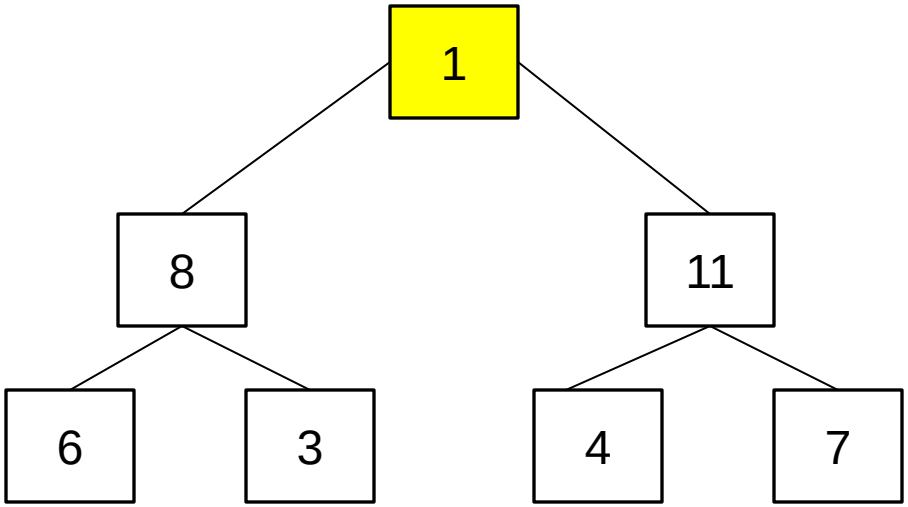
Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Heap Sort – Ταξινόμηση Σωρού

Παράδειγμα

Έστω S = {11, 6, 4, 1, 3, 7, 12, 8}



Με τον ίδιο τρόπο συνεχίζουμε για τα υπόλοιπα

Merge Sort – Ταξινόμηση με Συμβολή

- **Έχω τον πίνακα $S = \{s_1, s_2, \ldots, s_n\}$ προς ταξινόμηση**
  - Διαιρούμε τον πίνακα σε δύο περίπου ίσα μέρη
  - Ταξινομούμε αναδρομικά κάθε μέρος του πίνακα
  - Συγχωνεύουμε τα δύο μέρη

- Αλγόριθμος για Κύρια Μνήμη
- Αλγόριθμος βασιζόμενος σε συγκρίσεις

Merge Sort – Ταξινόμηση με Συμβολή

Παράδειγμα

Έστω S = {25, 57, 48, 37, 12, 92, 86, 33}

25	57	48	37	12	92	86	33

Merge Sort – Ταξινόμηση με Συμβολή

Παράδειγμα

Έστω S = {25, 57, 48, 37, 12, 92, 86, 33}

25	57	48	37

12	92	86	33

Merge Sort – Ταξινόμηση με Συμβολή

Παράδειγμα

Έστω S = {25, 57, 48, 37, 12, 92, 86, 33}

Merge Sort – Ταξινόμηση με Συμβολή

Παράδειγμα

Έστω S = {25, 57, 48, 37, 12, 92, 86, 33}

Merge Sort – Ταξινόμηση με Συμβολή

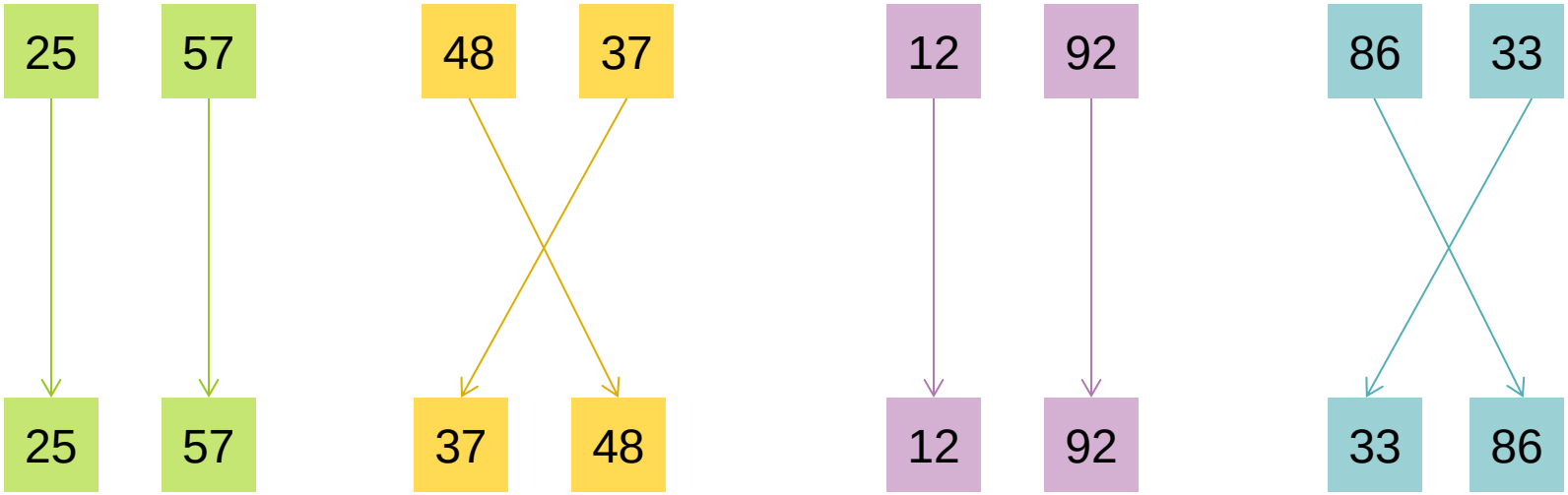
Παράδειγμα

Έστω S = {25, 57, 48, 37, 12, 92, 86, 33}

Merge Sort – Ταξινόμηση με Συμβολή

Παράδειγμα

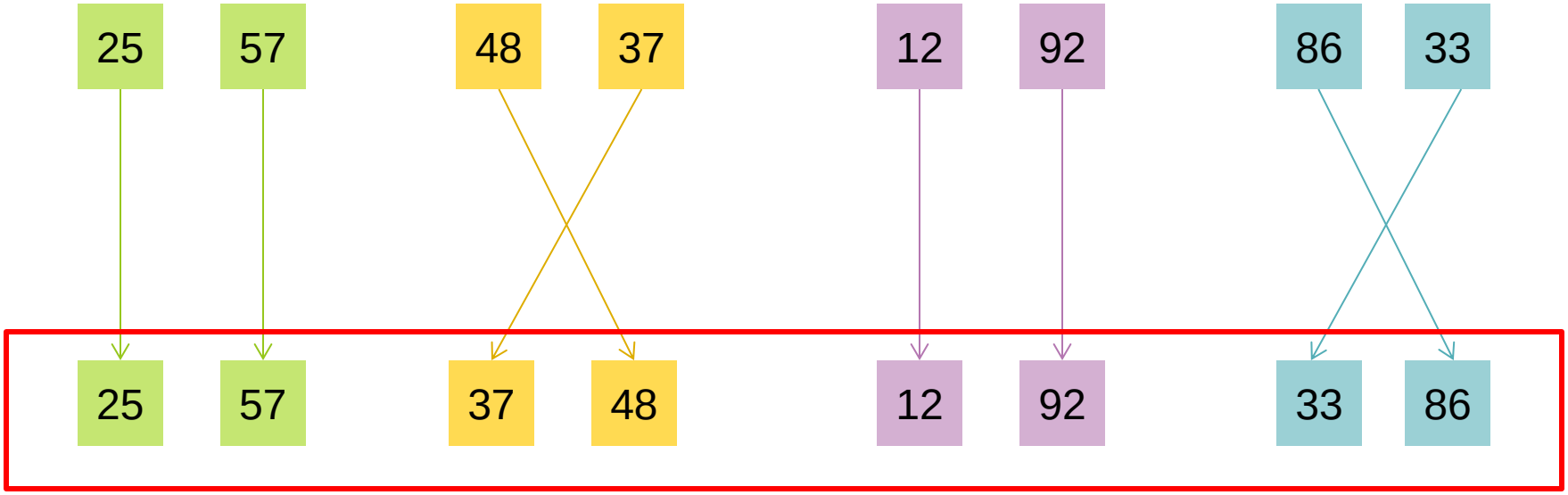
Έστω S = {25, 57, 48, 37, 12, 92, 86, 33}



Merge Sort – Ταξινόμηση με Συμβολή

Παράδειγμα

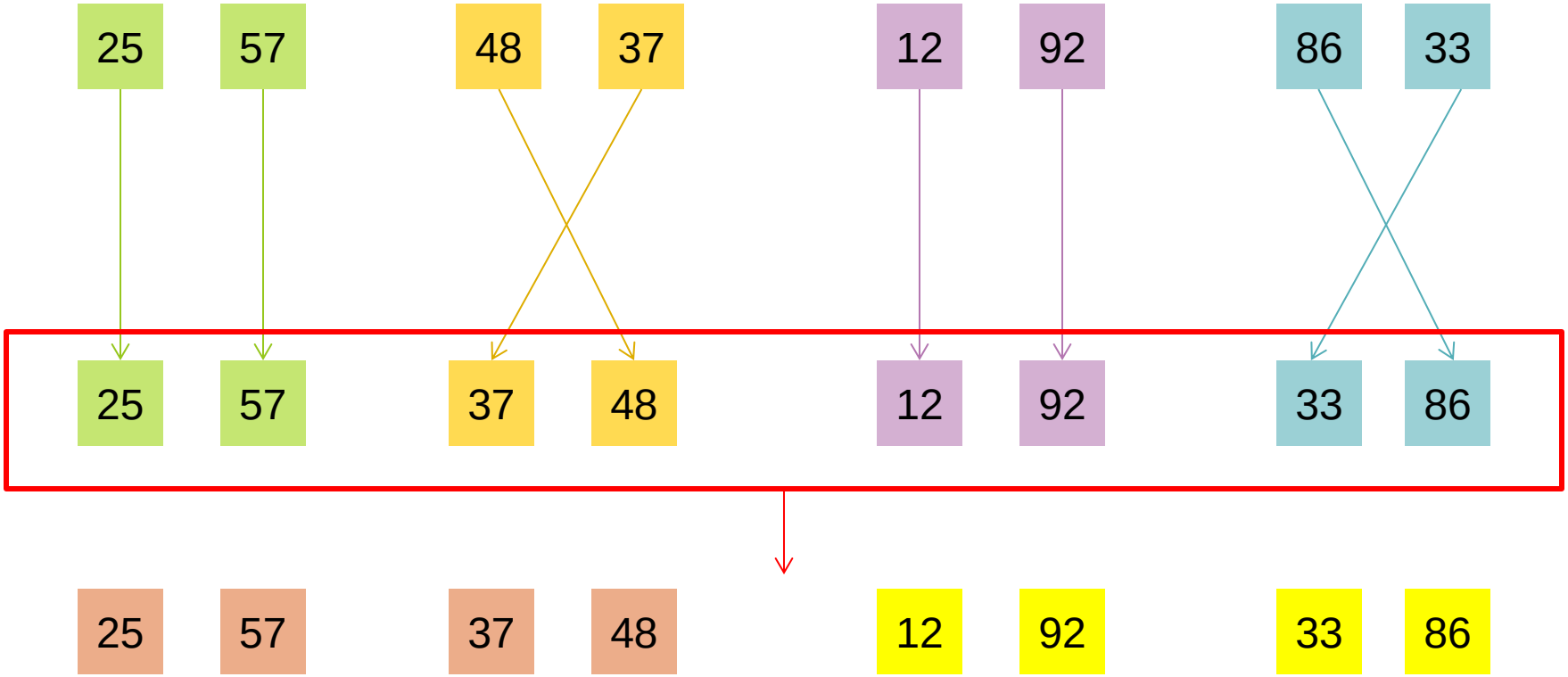
Έστω S = {25, 57, 48, 37, 12, 92, 86, 33}



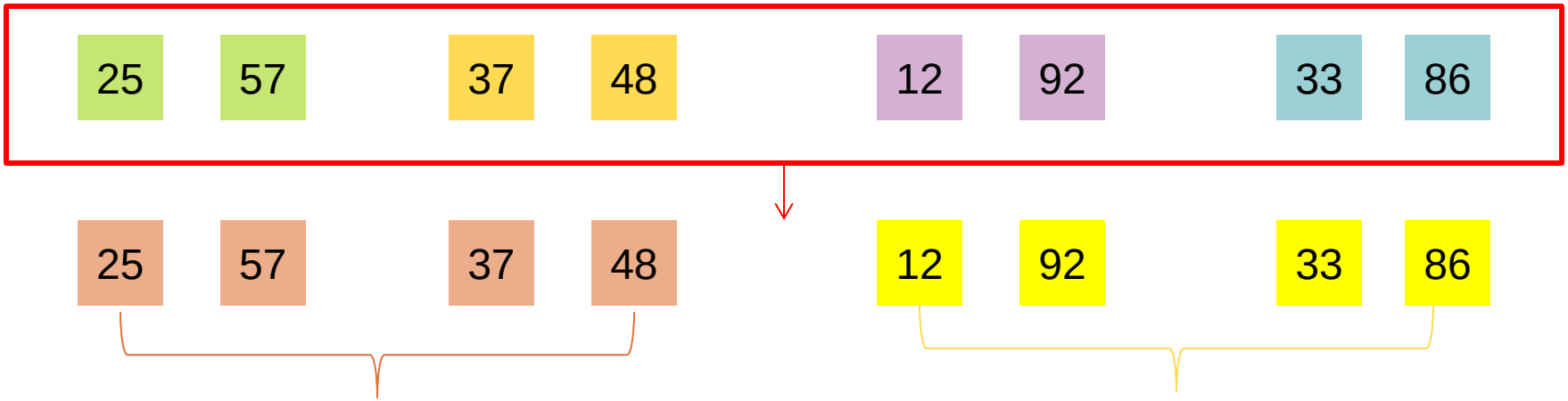
Merge Sort – Ταξινόμηση με Συμβολή

Παράδειγμα

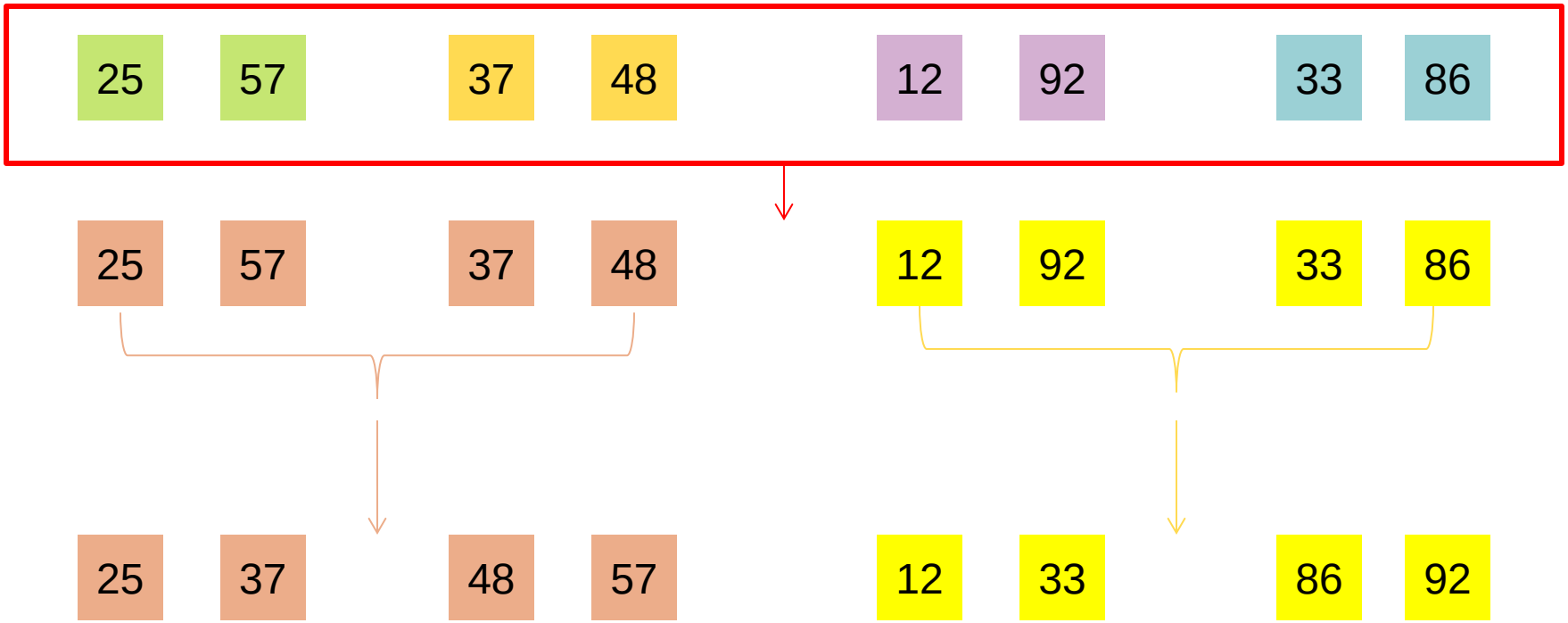
Έστω S = {25, 57, 48, 37, 12, 92, 86, 33}



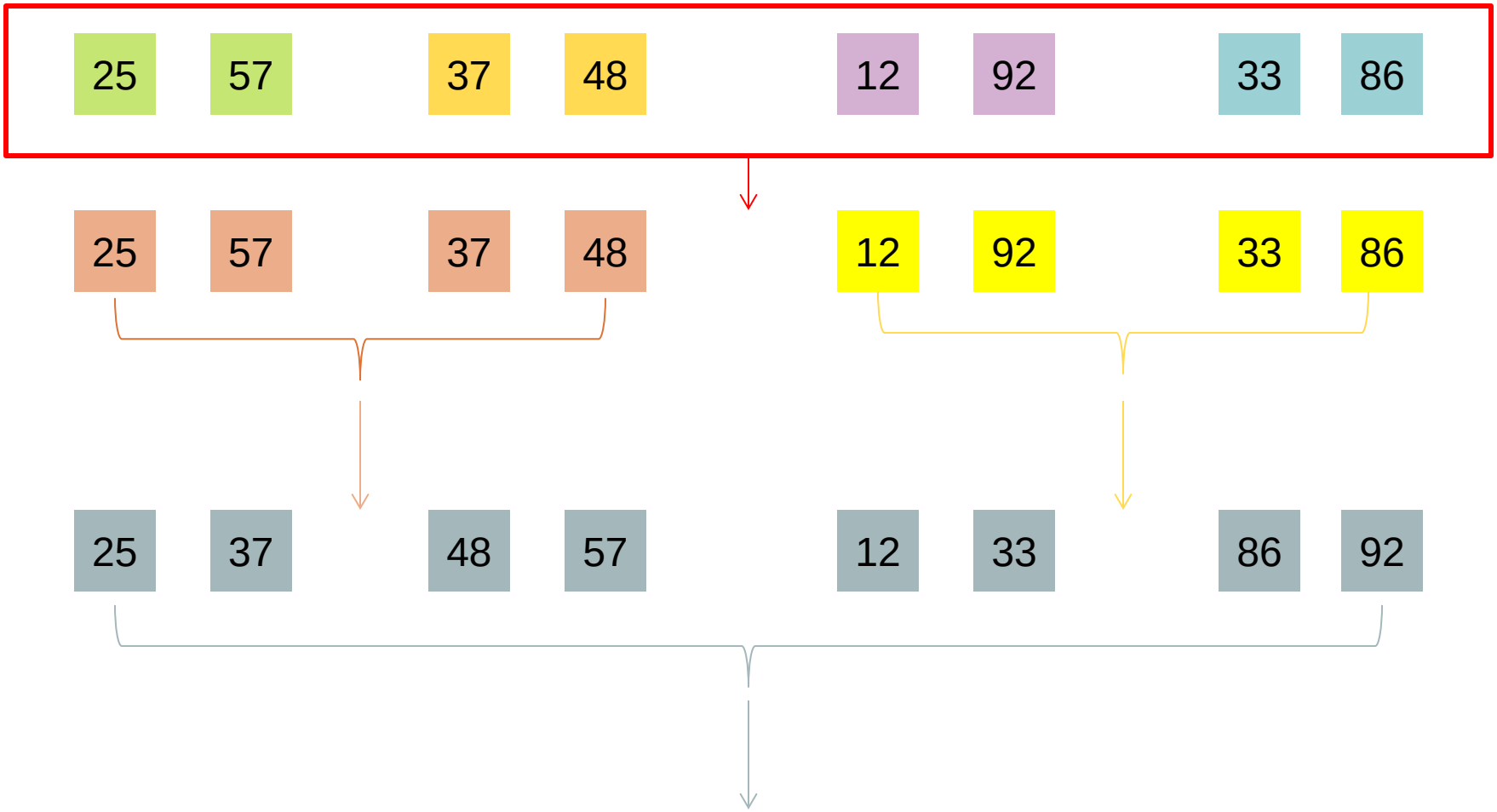
Merge Sort – Ταξινόμηση με Συμβολή



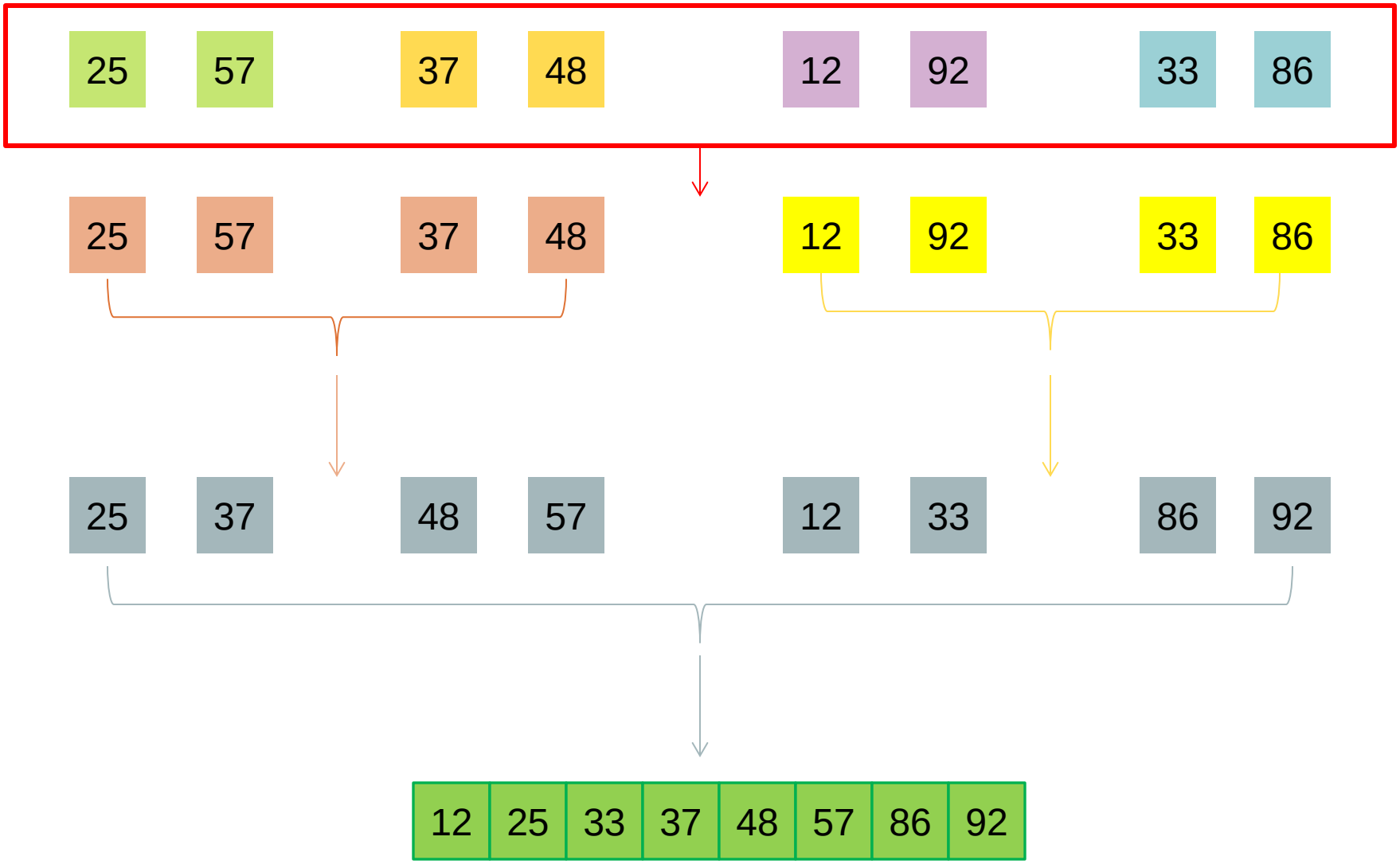
Merge Sort – Ταξινόμηση με Συμβολή



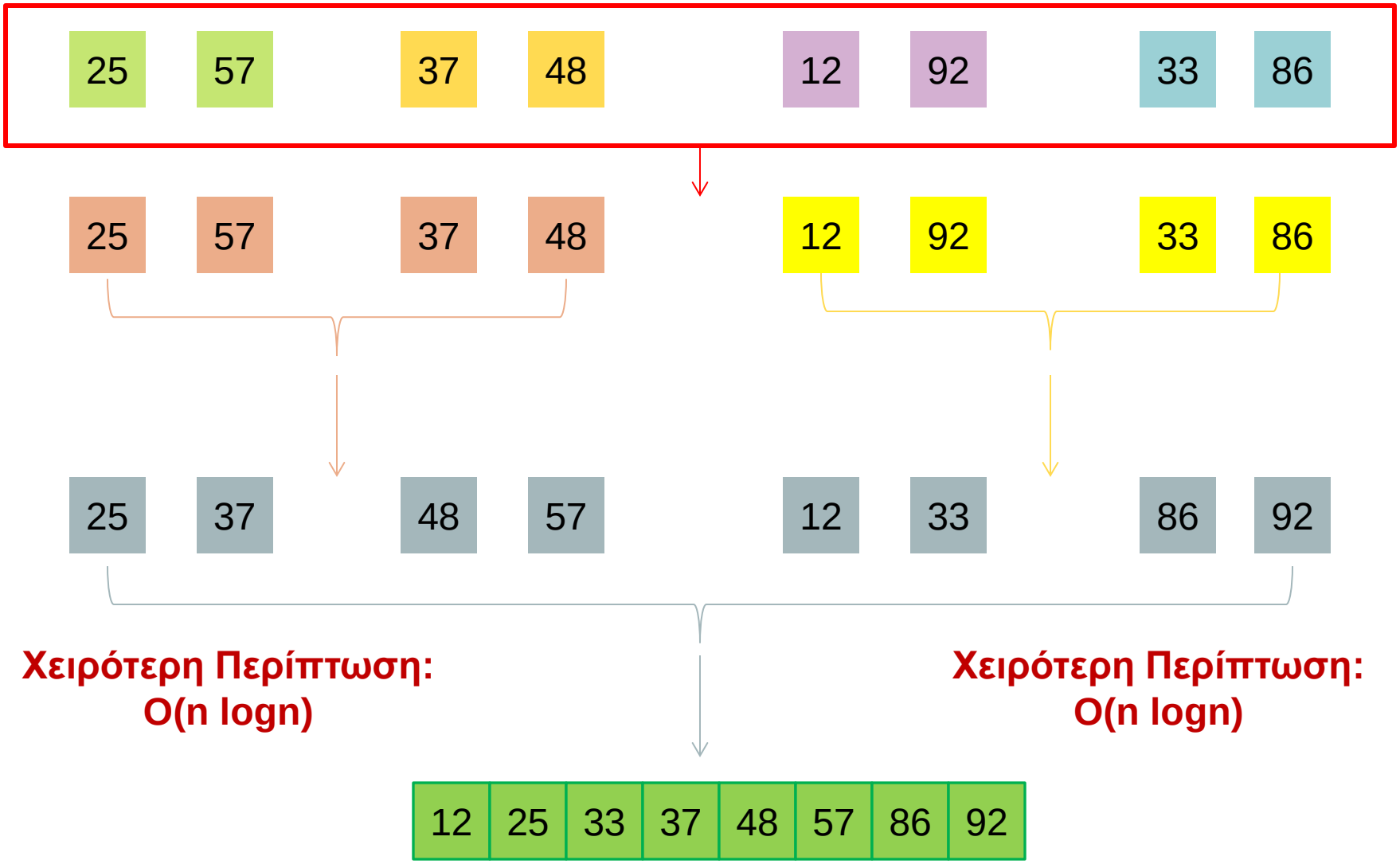
Merge Sort – Ταξινόμηση με Συμβολή



Merge Sort – Ταξινόμηση με Συμβολή



Merge Sort – Ταξινόμηση με Συμβολή



Χειρότερη Περίπτωση:
O(n logn)

Χειρότερη Περίπτωση:
O(n logn)

QuickSort – Γρήγορη Ταξινόμηση

- **Έχω τον πίνακα $S = \{s_1, s_2, \ldots, s_n\}$ προς ταξινόμηση**

  Η βασική ιδέα του αλγόριθμου βρίσκεται στη διαδικασία διαίρεση

  - Ένα στοιχείο S[i] τοποθετείται στην τελική θέση ταξινόμησης
  - Όλα τα στοιχεία S[0..i-1] είναι μικρότερα ή ίσα του S[i]
  - Όλα τα στοιχεία S[i+1..n-1] είναι μεγαλύτερα ή ίσα του S[i]

QuickSort – Γρήγορη Ταξινόμηση

- **Έχω τον πίνακα $S = \{s_1, s_2, \ldots, s_n\}$ προς ταξινόμηση**

  Η βασική ιδέα του αλγόριθμου βρίσκεται στη διαδικασία διαίρεση

  - Ένα στοιχείο S[i] τοποθετείται στην τελική θέση ταξινόμησης
  - Όλα τα στοιχεία S[0..i-1] είναι μικρότερα ή ίσα του S[i]
  - Όλα τα στοιχεία S[i+1..n-1] είναι μεγαλύτερα ή ίσα του S[i]
  - Υλοποίηση διαίρεσης:
    - Θέτω δείκτη *up* στο πρώτο στοιχείο του πίνακα και δείκτη *down* στο τελευταίο στοιχείο του πίνακα.
    - Θέτω *οδηγό (pivot)* το πρώτο στοιχείο του πίνακα
    - Διατρέχω τον πίνακα από αριστερά προς τα δεξιά με τον *up* και αναζητώ στοιχείο > *pivot* και από δεξιά προς τα αριστερά με τον *down* και αναζητώ στοιχείο ≤ *pivot.*
    - Εναλλάσσω αυτά τα στοιχεία

- Αλγόριθμος για Κύρια Μνήμη
- Αλγόριθμος βασιζόμενος σε συγκρίσεις

QuickSort – Γρήγορη Ταξινόμηση

Παράδειγμα

Έστω S = {44, 75, 23, 43, 55, 12, 64, 77, 33}

44	75	23	43	55	12	64	77	33

QuickSort – Γρήγορη Ταξινόμηση

Παράδειγμα

Έστω S = {44, 75, 23, 43, 55, 12, 64, 77, 33}

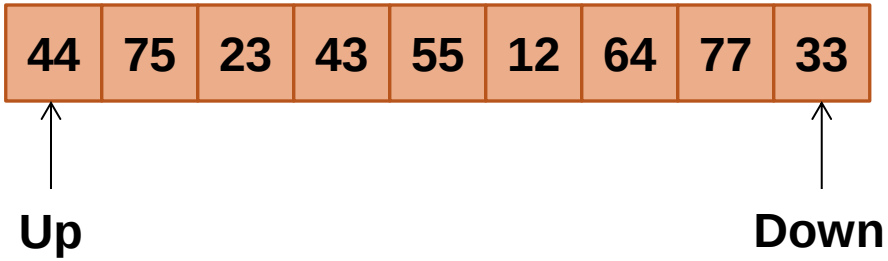
Pivot = 44

QuickSort – Γρήγορη Ταξινόμηση

Παράδειγμα

Έστω S = {44, 75, 23, 43, 55, 12, 64, 77, 33}

Pivot = 44

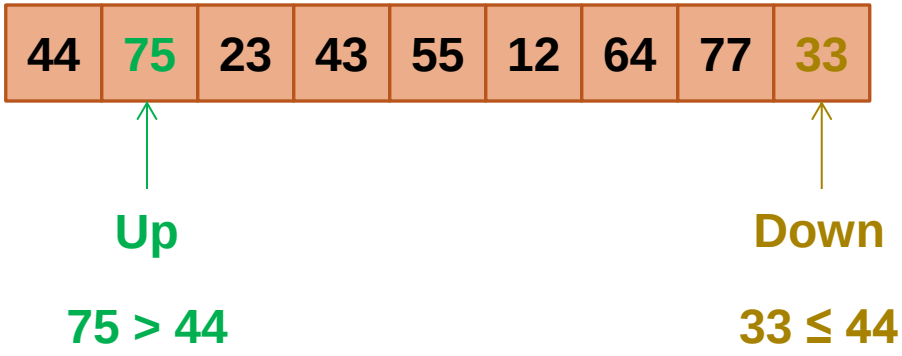


QuickSort – Γρήγορη Ταξινόμηση

Παράδειγμα

Έστω S = {44, 75, 23, 43, 55, 12, 64, 77, 33}

Pivot = 44

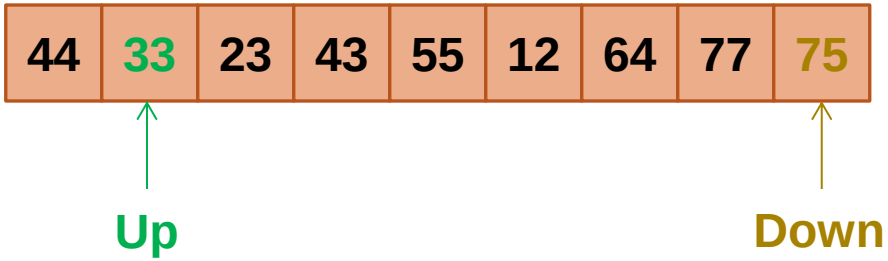


QuickSort – Γρήγορη Ταξινόμηση

Παράδειγμα

Έστω S = {44, 75, 23, 43, 55, 12, 64, 77, 33}

Pivot = 44

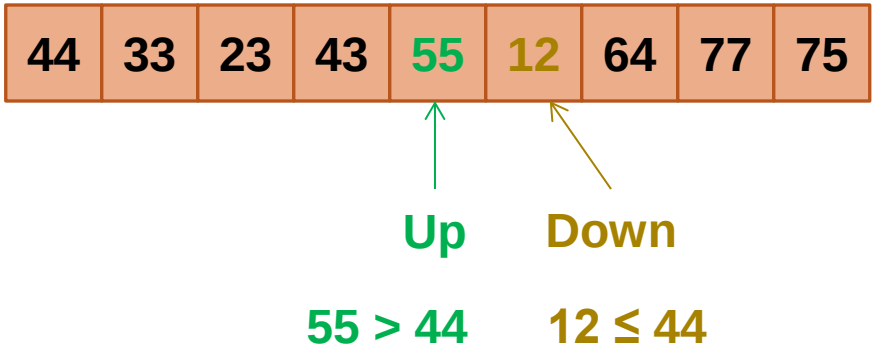


QuickSort – Γρήγορη Ταξινόμηση

Παράδειγμα

Έστω S = {44, 75, 23, 43, 55, 12, 64, 77, 33}

Pivot = 44

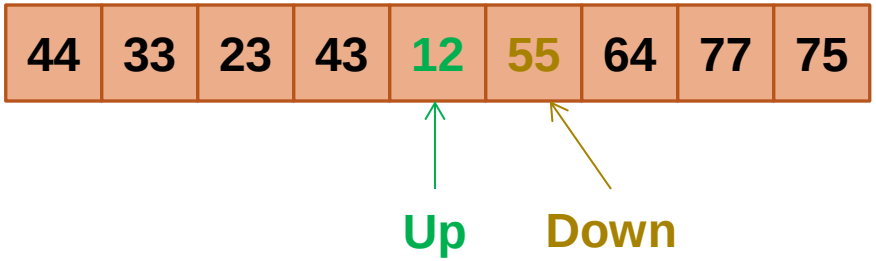


QuickSort – Γρήγορη Ταξινόμηση

Παράδειγμα

Έστω S = {44, 75, 23, 43, 55, 12, 64, 77, 33}

Pivot = 44

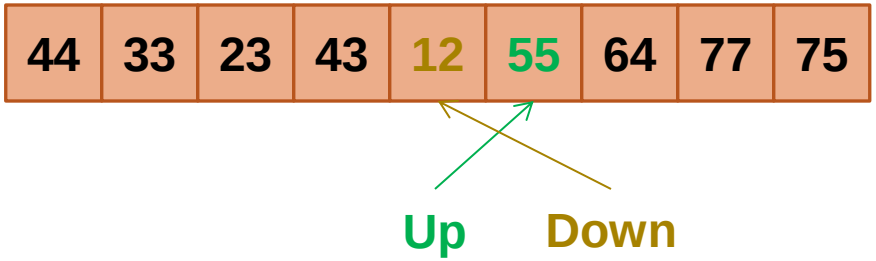


QuickSort – Γρήγορη Ταξινόμηση

Παράδειγμα

Έστω S = {44, 75, 23, 43, 55, 12, 64, 77, 33}

Pivot = 44



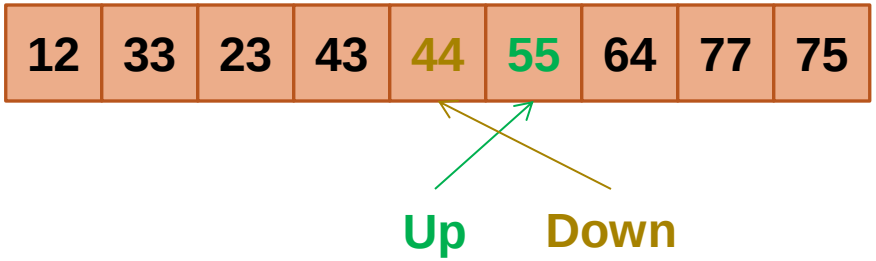
Οι δείκτες διασταυρώθηκαν οπότε εναλλάσσω τον οδηγό (pivot) με τον δείκτη down.

QuickSort – Γρήγορη Ταξινόμηση

Παράδειγμα

Έστω S = {44, 75, 23, 43, 55, 12, 64, 77, 33}

Pivot = 44



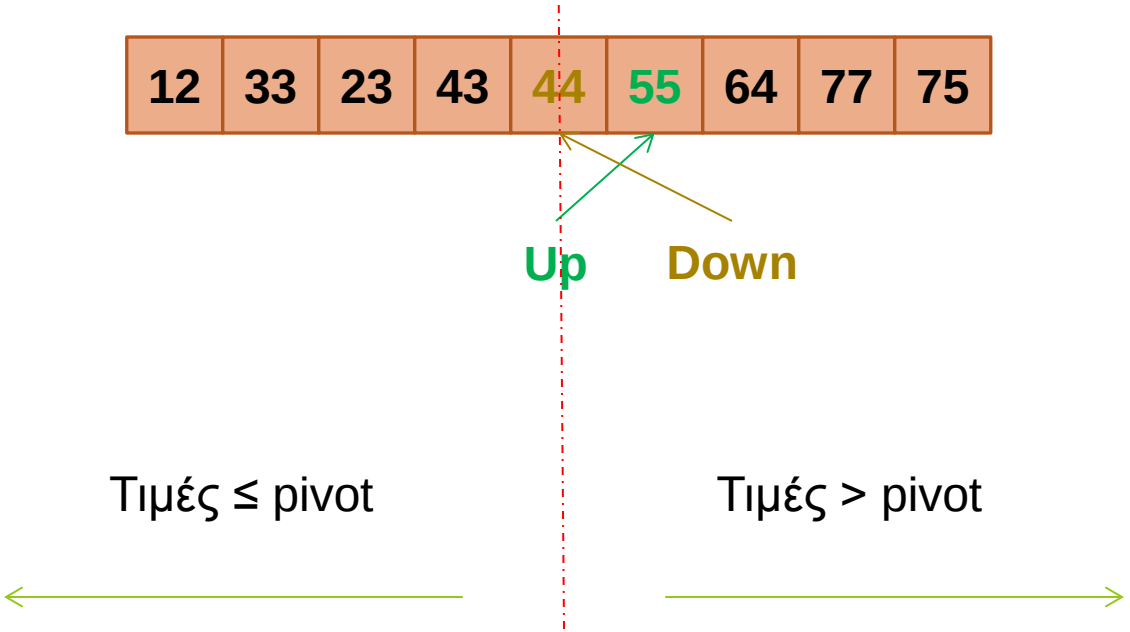
Οι δείκτες διασταυρώθηκαν οπότε εναλλάσσω τον οδηγό (pivot) με τον δείκτη down.

QuickSort – Γρήγορη Ταξινόμηση

Παράδειγμα

Έστω S = {44, 75, 23, 43, 55, 12, 64, 77, 33}

Pivot = 44

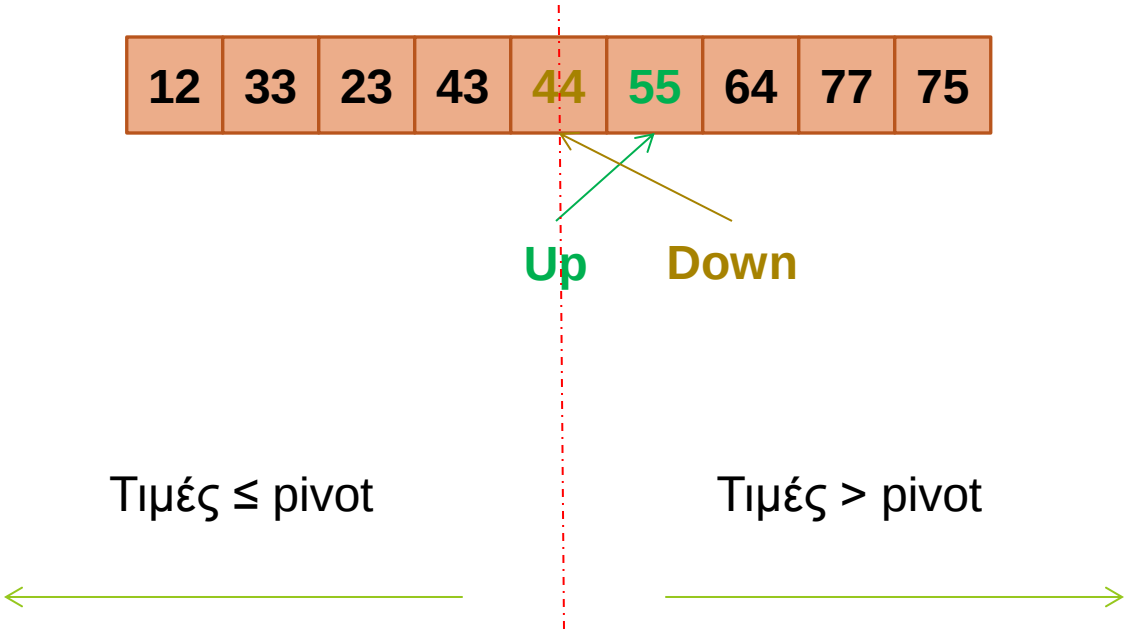


QuickSort – Γρήγορη Ταξινόμηση

Παράδειγμα

Έστω S = {44, 75, 23, 43, 55, 12, 64, 77, 33}

Pivot = 44



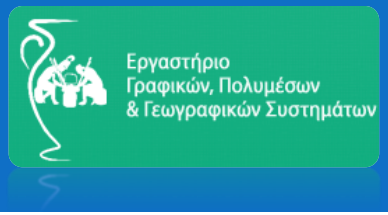
Ακολουθούμε την ίδια διαδικασία για τους υποπίνακες που προέκυψαν

Αναζήτηση - Searching

Το πρόβλημα του Λεξικού (Κεφ. 4)

Δομές Δεδομένων

Μάριος Κενδέα



3 Μαρτίου 2015

kendea@ceid.upatras.gr

Περιεχόμενα

Εισαγωγή (1/3)

- Έστω σύμπαν U πηγή ενδιαφέροντος
- Σύνολο $S \subseteq U$, $x \in S$ και $\text{info}(x)$
- Λεξικό (Dictionary) υποστηρίζει πράξεις:
  - Access(x): if $x \in S$ true{return info(x)} else false — Στατικές Δομές
  - Insert(x): $S \rightarrow S \cup \{x\}$
  - Delete(x): $S \rightarrow S - \{x\}$ — Δυναμικές Δομές

Εισαγωγή (2/3)

- Insert(x) και Delete(x) είναι καταστρεπτικές πράξεις
  - Εφήμερες Δομές: κάθε πράξη ενημέρωσης αλλάζει την δομή
- Διαχρονικές Δομές (Persistent): Δομές που υποστηρίζουν Insert(x) και Delete(x) με Μη-Καταστρεπτικό τρόπο και διατηρούν όλες τις εκδόσεις
  - Μόνο Access -> Μερικώς διαχρονικές
  - Insert, Delete σε προηγούμενες εκδόσεις -> Πλήρως διαχρονικές
- Κατηγοριοποίηση ως προς το χώρο
  - **Συνοπτικές Δομές**: ουρά με χρήση πίνακα -> n+O(1) χώρο
  - Εκτενείς Δομές: ουρά με χρήση δεικτών -> O(n) χώρο

Εισαγωγή (3/3)

- Πως κάνουμε αναζήτηση σε ένα λεξικό και πως επιλέγουμε κάθε φορά το επόμενο σημείο αν δεν έχουμε απάντηση?
  - Γραμμικό ψάξιμο (Linear Search)
    - $next \leftarrow \text{left} + 1$
    - O(n)
  - Δυαδικό ψάξιμο (binary search)
  - Ψάξιμο παρεμβολής (interpolation search)

Δυαδική Αναζήτηση

- $next \leftarrow \left\lfloor \frac{right+left}{2} \right\rfloor$
- Βρίσκουμε το μέσο του διαστήματος
- Πάμε στο υποδιάστημα που μας ενδιαφέρει
- Loop until
  - Μήκος υποδιαστήματος = 1 -> Λύση
  - Μήκος υποδιαστήματος = 0 -> Δεν Υπάρχει
- Χειρότερη περίπτωση χρόνος **O(log n)**

Αναζήτηση Παρεμβολής (1/2)

- Αναζήτηση ανάλογη της αίσθησης που έχει ο άνθρωπος όταν αναζητά σε ένα λεξικό.
- $next \leftarrow \left\lfloor \frac{x - S[left]}{S[right] - S[left]} (right - left) \right\rfloor + left$
- Πόσο μεγαλύτερο είναι το ζητούμενο στοιχείο από το αριστερό άκρο
- Πόσο μεγαλύτερο είναι το δεξί άκρο από το αριστερό
- Παίζει ρόλο στην απόδοση το πόσο διαφέρουν τα στοιχεία από τα γειτονικά
- Χειρότερη περίπτωση χρόνος O(n)
- Μέσος χρόνος O(log logn) -> Υπερισχύει της δυαδικής αναζήτησης

Αναζήτηση Παρεμβολής (2/2)

- Πρόβλημα Απόδοσης

Δυαδική Αναζήτηση Παρεμβολής (1/3)

- Χειρότερη περίπτωση χρόνος O($\sqrt{n}$)
- Μέσος χρόνος O(log logn)
- Κάνουμε γραμμική επανάληψη με βήμα $\sqrt{n}$

- Ακολουθεί ο αλγόριθμος
  - Προσοχή είναι ψευδοκώδικας! Δεν είναι έτοιμος για υλοποίηση
    - Θεωρεί 1^{η} θέση το 1 όχι το 0 και τελευταία το n όχι το n-1
    - Στη γραμμή 8 μικρά διαστήματα κάνει γραμμική αναζήτηση
    - Δεν χειρίζεται ακραίες περιπτώσεις

Δυαδική Αναζήτηση Παρεμβολής (2/3)

Δυαδική Αναζήτηση Παρεμβολής (3/3) Βελτίωση

- **Βελτίωση χρόνου χειρότερης περίπτωσης** σε O(logn) το βήμα στο loop από i=i+1 γίνεται **i=2*i**
- 1ο βήμα: Εκθετικά βήματα εξετάζοντας τα διαστήματα [next, next+$\sqrt{n}$], [next, next+$2\sqrt{n}$], [next, next+$2^2\sqrt{n}$], [next, next+$2^3\sqrt{n}$],…, [next, next+$2^{\mathrm{i}}\sqrt{n}$] μέχρι να βρεθεί το διάστημα που περιέχει το x: S[next+$2^{\mathrm{j}-1}\sqrt{n}$] $< x \leq$ [next+$2^j\sqrt{n}$]
  - Άρα κάνουμε εκθετική επανάληψη: $\sqrt{n}, 2\sqrt{n}, 4\sqrt{n}, 8\sqrt{n}, \ldots$
- 2ο βήμα: Ψάχνουμε **δυαδικά στα στοιχεία που απέχουν $\sqrt{n}$** στο διάστημα του 1ου βήματος και προκύπτει το διάστημα $\sqrt{n}$ που περιέχει το στοιχείο.

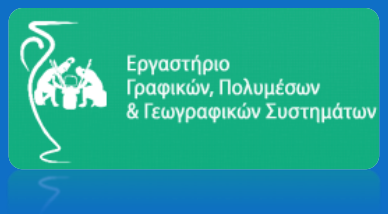
?

Standard Template Library (STL)

C++ library

Δομές Δεδομένων

Μάριος Κενδέα



10 Μαρτίου 2015

kendea@ceid.upatras.gr

Εισαγωγή

- Η Standard Βιβλιοθήκη προτύπων (STL) είναι μια βιβλιοθήκη λογισμικού η οποία περιλαμβάνεται στην C++ Standard Library.
- Παρέχει:
  - Containers
  - Iterators
  - Algorithms
  - Functors

Πλεονεκτήματα της Standard Template Library (STL) (1/2)

- Παρέχει δυνατότητα χρήσης γενικών δομών δεδομένων και αλγορίθμων.
- Αποτελείται από ένα ισχυρό και ευέλικτο σύνολο κλάσεων και λειτουργιών.
- Παρέχει ένα αποτελεσματικό, ελαφρύ, και επεκτάσιμο πλαίσιο για την ανάπτυξη εφαρμογών πληροφορικής.
- Προσφέρει ένα εκλεπτυσμένο επίπεδο αφαίρεσης που προωθεί τη χρήση των γενικών δομών δεδομένων και αλγορίθμων χωρίς την επιβάρυνση μιας γενικής λύσης.
- Είναι αρκετά ευέλικτη στην αντιμετώπιση των αναπτυξιακών αναγκών της για κάθε είδους εφαρμογές.

Πλεονεκτήματα της Standard Template Library (STL) (2/2)

- Οι εφαρμογές δεν υπόκεινται σε μείωση της απόδοσης τους μέσω της χρήσης της STL, δεδομένου ότι η STL παρέχει συγκεκριμένες εγγυήσεις απόδοσης για τους αλγόριθμους που προμηθεύει και μπορεί να επιλεγεί αυτός που ανταποκρίνεται καλύτερα στις ανάγκες της κάθε εφαρμογής
- Απαιτεί μικρότερο χρόνο ανάπτυξης και debugging
- Επιτυγχάνει καλή διαχείριση μνήμης
- Απαιτείται μικρό μέγεθος κώδικα.
- Επιτυγχάνει τα αποτελέσματά της μέσω της χρήσης των προτύπων templates.
- Η προσέγγιση χρήσης των προτύπων παρέχει πολυμορφισμό χρόνου μεταγλώττισης που είναι συχνά πιο αποδοτικός από ό, τι ο παραδοσιακός πολυμορφισμός χρόνου εκτέλεσης

Βασικά συστατικά της Standard Template Library (STL)

- Η STL παρέχει ένα σύνολο κλάσεων C++, όπως οι **Sequence Containers, οι Associative Containers και οι Container Adaptors**
- Ένας **container** είναι μια κλάση, μια δομή δεδομένων, ή ένας αφηρημένος τύπος στοιχείων (Abstract Data Type) του οποίου τα στιγμιότυπα είναι συλλογές άλλων αντικειμένων. Με άλλα λόγια χρησιμοποιούνται για να αποθηκεύσουν τα αντικείμενα με οργανωμένο τρόπο ακολουθώντας συγκεκριμένους κανόνες πρόσβασης.

Sequence Containers (1/8)

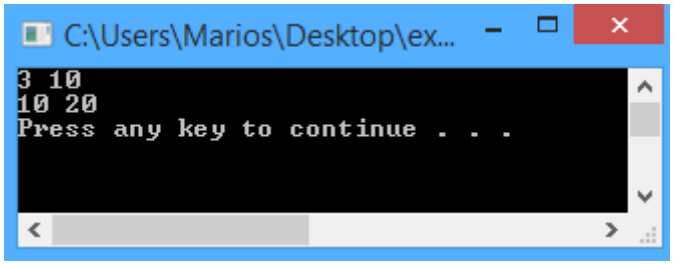
- Διατεταγμένοι πίνακες αντικειμένων μεταβλητού μήκους
  - vector
  - deque
  - list
- Επιτρέπουν εισαγωγή και διαγραφή στοιχείων σε οποιοδήποτε σημείο.
- **Βασική διαφορά:** Το κόστος εξαρτάται από τον τύπο.

Sequence Containers (2/8)
Vector (1/2)

- Είναι ένας δυναμικός πίνακας.
- Επιτρέπει τυχαία προσπέλαση, O(1)
- Υπάρχει δυνατότητα να επανα-ταξινομηθεί αυτόματα, κατά την προσθήκη ή τη διαγραφή ενός αντικειμένου.
- Το κόστος εισαγωγής στο τέλος ενός διανύσματος είναι σταθερό, O(1).
- Το κόστος εισαγωγής στην αρχή ενός διανύσματος είναι γραμμικό, O(n).
- Το κόστος εισαγωγής στη μέση ενός διανύσματος είναι γραμμικό, O(n).

Sequence Containers (3/8) Vector (2/2)

```
1 #include <vector>
2
3 int main(){
4   vector <int> example;
5   example.push_back(3);
6   example.push_back(10);
7
8   for(int x=0; x<example.size(); x++){
9       cout << example[x] << " ";
10  }
11  cout << endl;
12  if (!example.empty())
13      example.clear();
14
15  vector<int> another_vector;
16  another_vector.push_back(10);
17  example.push_back(10);
18  if (example == another_vector)
19      example.push_back(20);
20
21  for (int y=0; y<example.size(); y++){
22      cout << example[y] << " ";
23  }
24  cout << endl;
25  return 0;
26}
```



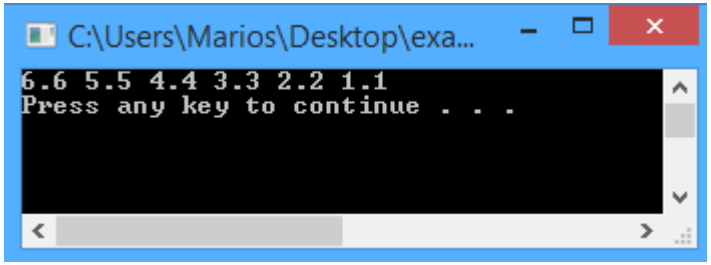
Sequence Containers (4/8)
Deque (1/3)

- Το κόστος εισαγωγής/διαγραφής στην αρχή είναι σταθερό, O(1).
- Το κόστος εισαγωγής/διαγραφής στο τέλος είναι σταθερό, O(1).
- Το κόστος εισαγωγής/διαγραφής στη μέση είναι γραμμικό, O(n).
- Επιτρέπει τυχαία προσπέλαση, O(1).
- Δεν παρέχει εγγυήσεις εγκυρότητας όταν έχει αλλάξει η σειρά.

Sequence Containers (5/8) Deque (2/3)

```
/*
 * Constructors
*/
    deque<int> first; //empty

    deque<int> second (4, 100); // 4 ints = 100

    deque<int> third(second.begin(), second.end()); //iterating through second

    deque<int> fourth(third); //a copy of third

    deque<double> fifth(5, 8.1); //doubles with 5 elements set to 8.1
```

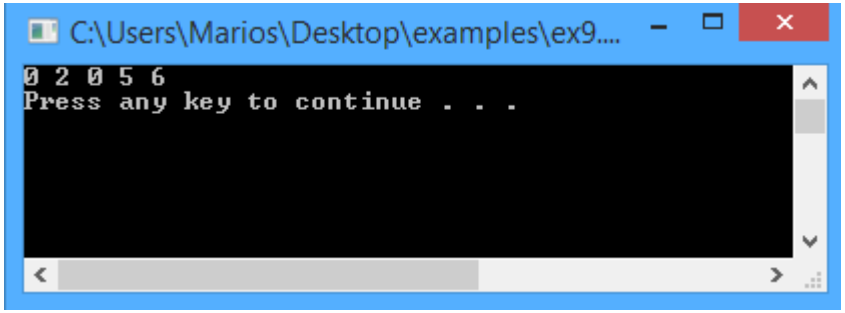
```
1 #include <deque>
2
3 int main(){
4
5   deque<float> example;
6   for(int i=1; i<=6; ++i){
7       example.push_front(i*1.1);
8   }
9
10  for(int i=0; i<example.size(); ++i){
11      cout << example[i] << " ";
12  }
13  cout << endl;
14  return 0;
15}
```



Sequence Containers (7/8)
List (1/2)

- Σε μία διπλά-συνδεμένη λίστα, τα στοιχεία δεν αποθηκεύονται στην παρακείμενη μνήμη.
- Επιτυγχάνει αντίθετη απόδοση από ένα διάνυσμα.
- Αργή αναζήτηση και προσπέλαση (γραμμικός χρόνος)…
- …αλλά μόλις βρεθεί μια θέση, γρήγορη εισαγωγή και διαγραφή (σταθερός χρόνος).
- Το κόστος εισαγωγής στην αρχή, στο τέλος, στη μέση είναι σταθερό.
- Επιτρέπει όμως **μόνο** σειριακή προσπέλαση.

```
1 #include <list>
2
3 int main()
4 {
5    list<int> L;
6    L.push_back(0);
7    L.push_front(0);
8    L.insert(++L.begin(),2);
9
10   L.push_back(5);
11   L.push_back(6);
12
13   list<int>::iterator i;
14
15   for(i=L.begin(); i != L.end(); ++i)
16      cout << *i << " ";
17   cout << endl;
18   system ("pause");
19   return 0;
20}
```



Associative Containers (1/5)

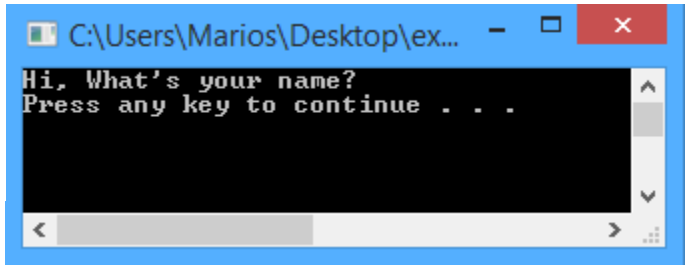
- Γενίκευση των sequences
  - map - multimap
  - set - multiset
- **Βασική διαφορά:** Δεν δεικτοδοτούνται από ακεραίους αλλά από κάποιο κλειδί.
- Το κλειδί αυτό μπορεί να είναι οποιουδήποτε τύπου

Associative Containers (2/5)
Map – MultiMap (1/2)

- Αποθηκεύει ζεύγη της μορφής <key, value>.
- Κάτι σαν πίνακας κατακερματισμού (Hash table), αλλά επιτρέπει εισαγωγή, διαγραφή και αναζήτηση με κόστος λογαριθμικό.
- Επιτρέπει ακόμα σειριακή προσπέλαση όλων των στοιχείων.
- Δεν επιτρέπονται διπλότυπα κλειδιά.
- MultiMap: Όμοιο με το map με τη διαφορά ότι επιτρέπονται διπλότυπα κλειδιά

hash set - hash multiset - hash map - hash multimap. Παρόμοια με τα set, multiset, map, multimap, αλλά υλοποιούνται με ένα hashtable. Τα κλειδιά δεν διατάσονται, αλλά μια hash λειτουργία είναι απαραίτητη για το βασικό τύπο. Αυτοί οι containers δεν είναι μέρος της C++ Standard Library, αλλά συμπεριλαμβάνονται στις επεκτάσεις STL SGI, και συμπεριλαμβάνονται σε βιβλιοθήκες όπως η GNU C++.

```
1 #include <map>
2
3 int main(){
4   typedef multimap<int, string> IntStringMMap;
5   IntStringMMap example;
6
7   example.insert(make_pair(2, "your"));
8   example.insert(make_pair(1, "Hi,"));
9   example.insert(make_pair(1, "What's"));
10  example.insert(make_pair(3, "name?"));
11
12  IntStringMMap::iterator pos ;
13  for(pos=example.begin(); pos!=example.end(); ++pos) {
14  cout << pos->second << " " ;
15  }
16  cout << endl;
17}
```

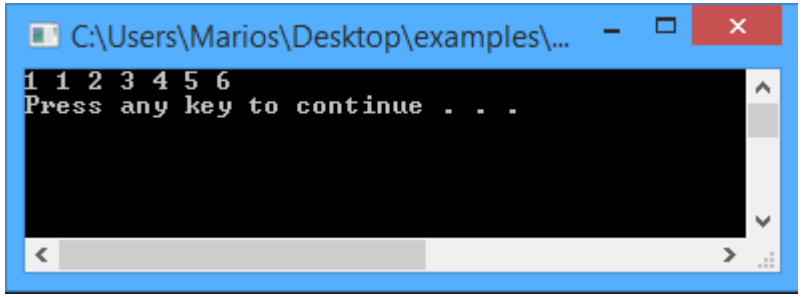


Associative Containers (4/5)
Set – MultiSet (1/2)

- Η εισαγωγή/διαγραφή των στοιχείων σε ένα σύνολο δεν ακυρώνει τους iterators που δείχνουν στο σύνολο.
- Παρέχει τις λειτουργίες: ένωση, τομή, διαφορά, συμμετρική διαφορά.
- Όμοιο με το map με τη διαφορά ότι οι ίδιες οι τιμές παίζουν και το ρόλο του κλειδιού.
- Επιτρέπει εισαγωγή, διαγραφή και αναζήτηση με κόστος λογαριθμικό.
- Επιτρέπει σειριακή προσπέλαση όλων των στοιχείων.
- Δεν επιτρέπονται διπλότυπα.
- Τα δεδομένα πρέπει να χρησιμοποιούν τον τελεστή σύγκρισης <.
- Έχει υλοποιηθεί χρησιμοποιώντας δυαδικό δέντρο αναζήτησης εξισορρόπησης.
- Multiset: Όμοιο με το set με τη διαφορά ότι επιτρέπονται διπλότυπα.

Associative Containers (5/5) Set - MultiSet (2/2)

```
1  #include <set>
2
3  int main() {
4    typedef std::multiset<int> IntSet;
5    IntSet myset;
6
7    myset.insert(3);
8    myset.insert(1);
9    myset.insert(5);
10   myset.insert(4);
11   myset.insert(1);
12   myset.insert(6);
13   myset.insert(2);
14
15       IntSet::const_iterator pos;
16     for (pos = myset.begin(); pos != myset.end(); ++pos) {
17         cout << *pos << ' ';
18     }
19     cout << endl;
20 }
```



Container Adaptors (1/2)

- **Ορισμός:** Η ουρά (queue), ουρά προτεραιότητας (priority queue) και η στοίβα (stack) αποτελούν Container Adaptors με συγκεκριμένη διεπαφή, που χρησιμοποιούν άλλους containers ως εφαρμογή.

- **Ουρά:**
  - Παρέχει μία διεπαφή ουράς FIFO για τις λειτουργίες push,pop,front,back.
  - Οποιεσδήποτε λειτουργίες υποστήριξης σειρών όπως η front(), back(), push_back(), pop_front() μπορεί να χρησιμοποιηθεί ως στιγμιότυπο της ουράς (π.χ. list, deque).

Container Adaptors (2/2)

- **Ουρά Προτεραιότητας:**
  - Παρέχει μία διεπαφή ουράς LIFO για τις λειτουργίες push, pop, top. (το στοιχείο με την υψηλότερη προτεραιότητα είναι στην κορυφή).
  - Οποιαδήποτε σειρά τυχαίας προσπέλασης που υποστηρίζει τις λειτουργίες front(), push_back(), pop_back() μπορεί να χρησιμοποιηθεί ως στιγμιότυπο της ουράς προτεραιότητας (π.χ. vector, deque).
  - Τα στοιχεία πρέπει, επίσης, να υποστηρίζουν τη σύγκριση (για να καθοριστεί ποιο στοιχείο έχει υψηλότερη προτεραιότητα και πρέπει να βγαίνει πρώτο από την ουρά).
- **Στοίβα:**
  - Παρέχει μία διεπαφή ουράς LIFO για τις λειτουργίες push,pop,top. (το τελευταίος-στοιχείο είναι στην κορυφή).
  - Οποιεσδήποτε λειτουργίες υποστήριξης σειρών όπως η back(), push_back(), pop_back() μπορεί να χρησιμοποιηθούν ως στιγμιότυπο της στοίβας (π.χ. vector, list, deque).

Iterators (1/4)

- Οι containers προσφέρουν δυνατότητα προσπέλασης των στοιχείων τους μέσω iterators.
- Πρόκειται για δείκτες στα στοιχεία ενός container.
- Βοηθούν στη σάρωση όλων των στοιχείων ενός container.
- Κάθε container ορίζει έναν ή περισσότερους τύπους iterators που μπορούμε να χρησιμοποιήσουμε.

Iterators (2/4)
Πέντε διαφορετικοί τύποι iterators

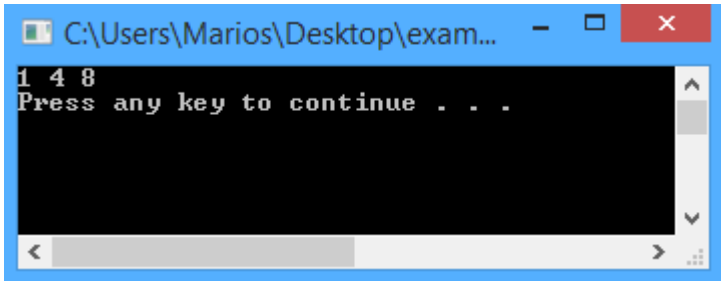
- input iterators (που μπορούν να χρησιμοποιηθούν μόνο για να διαβάσουν μια ακολουθία τιμών) *value = *myItr*
- output iterators (που μπορούν να χρησιμοποιηθούν μόνο για να γράψουν μια ακολουθία τιμών) **myItr = value*
- forward iterators (που μπορούν να διαβαστούν, να γράψουν, και να μετακινηθούν προς τα εμπρός) *++myItr is legal, but --myItr is not*
- bidirectional iterators (λειτουργούν όπως οι forward iterators, αλλά μπορούν επίσης να κινηθούν προς τα πίσω: list, set, multiset, map, and multimap)
- random access iterators (που μπορούν να κάνουν ελεύθερα οποιοδήποτε αριθμό βημάτων σε μια μόνο λειτουργία, vector and deque)

Iterators (3/4)
Μειονεκτήματα Χρήσης iterators

- Η χρήση των iterators επιφέρει κάποιο κόστος πολλές φορές.
- Π.χ. η εκτέλεση μιας αναζήτησης σε έναν associative container όπως σε ένα map ή σε ένα set μπορεί να είναι πολύ πιο αργή χρησιμοποιώντας τους iterators απ’ ό,τι θα ήταν αν χρησιμοποιούσαμε τις συναρτήσεις μέλη που προσφέρονται από τον ίδιο τον container.
- Αυτό συμβαίνει επειδή οι μέθοδοι ενός associative container μπορούν να εκμεταλλευτούν τη γνώση της εσωτερικής δομής, η οποία είναι αδιαφανής στους αλγορίθμους που χρησιμοποιούν τους iterators.

Iterators (4/4)

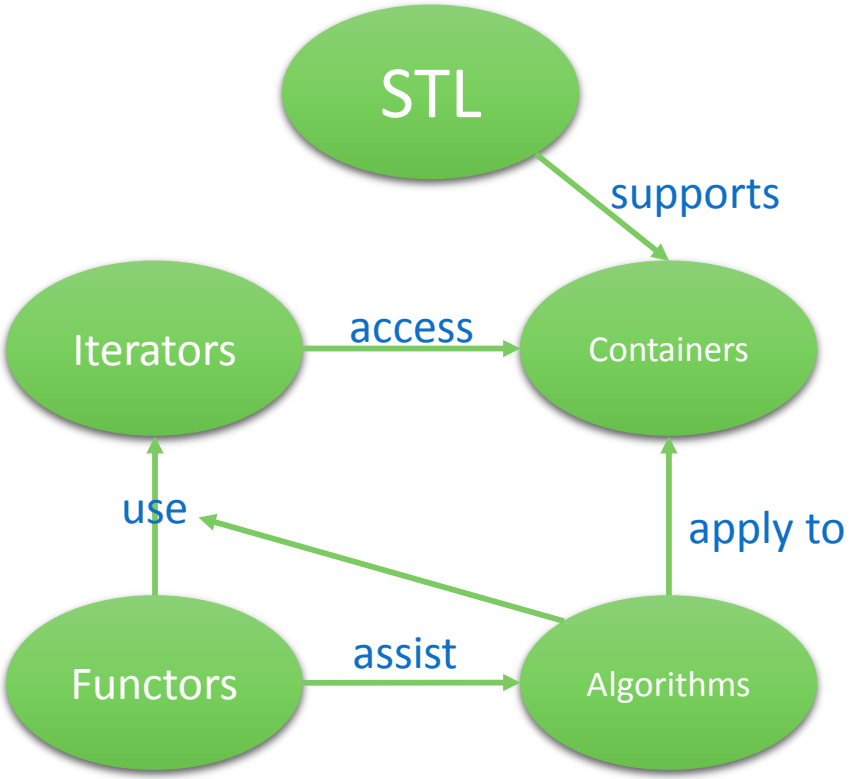
```
1 #include <vector>
2
3 int main(){
4   vector<int> myIntVector;
5   vector<int>::iterator myIntVectorIterator;
6
7   myIntVector.push_back(1);
8   myIntVector.push_back(4);
9   myIntVector.push_back(8);
10
11  for(myIntVectorIterator = myIntVector.begin(); myIntVectorIterator!=myIntVector.end(); myIntVectorIterator++) {
12      cout << *myIntVectorIterator << " " ;
13  }
14  cout << endl;
15}
```



Algorithms

- Πλούσια συλλογή από template C++ functions που λειτουργούν πάνω σε containers.
- Περιλαμβάνουν:
  - Ταξινόμηση (sort, merge, min, max,..)
  - Αναζήτηση (find, count, equal,..)
  - Μετασχηματισμούς (transform, replace, fill, rotate, nth element, random shuffle,..)
  - Πράξεις συνόλων (includes, set difference , set intersection, set union,..)

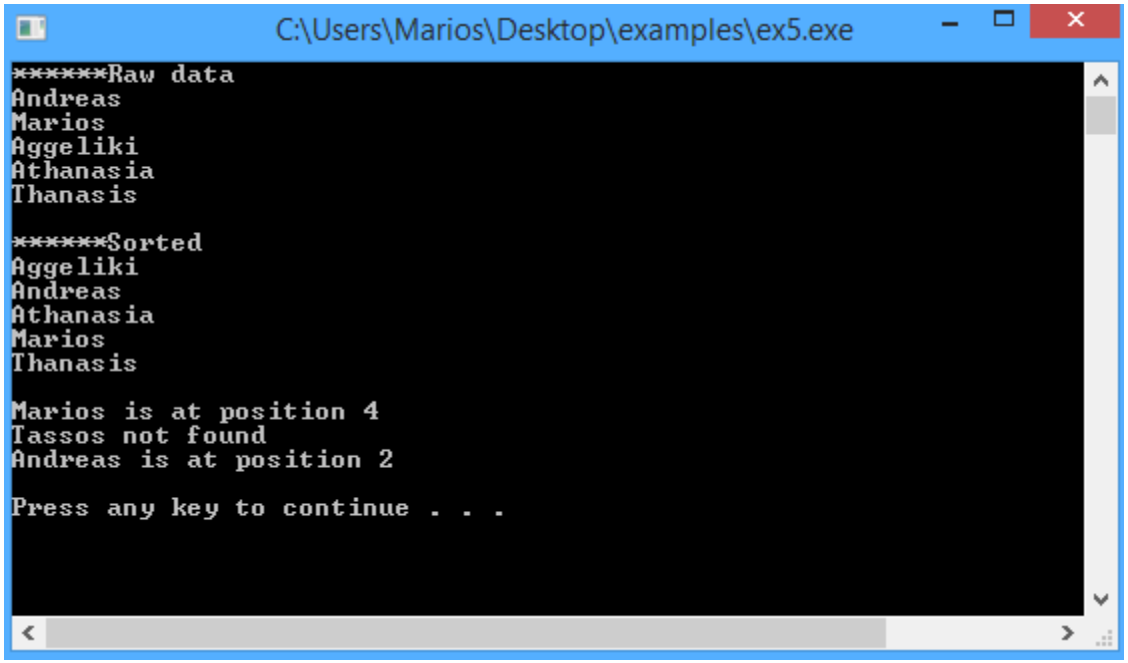
Ανακεφαλαίωση



Παράδειγμα (1)

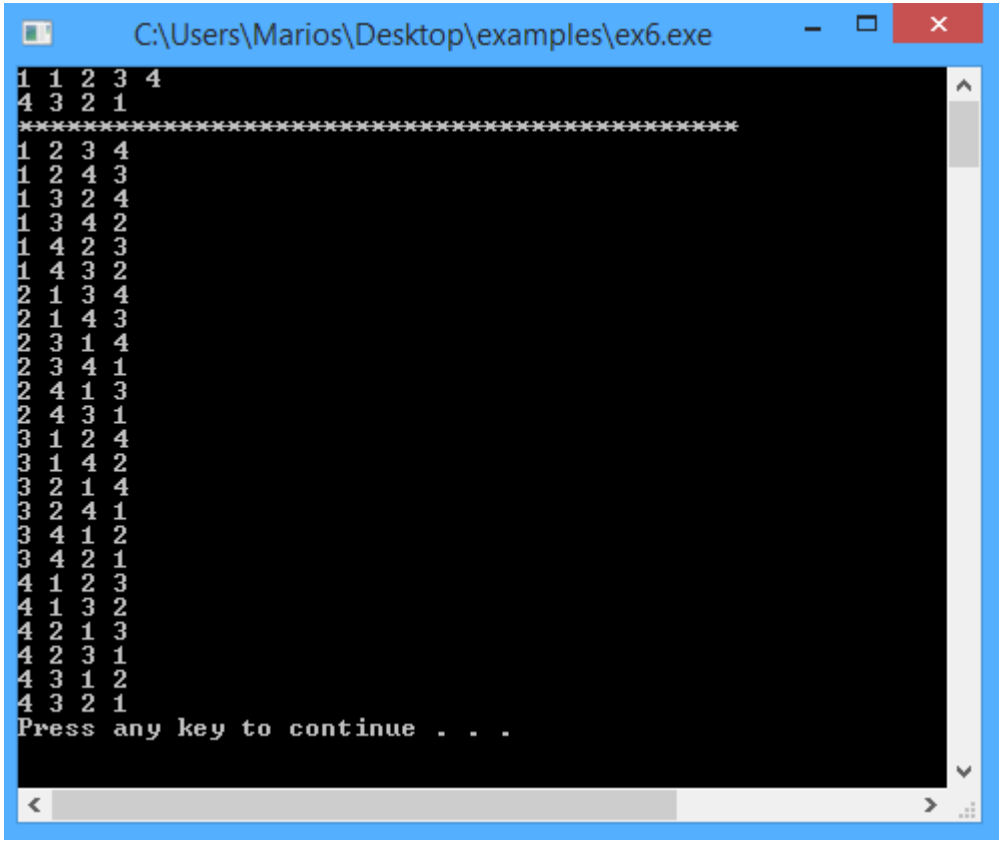
```
1 #include <iostream>
2 #include <vector>
3 #include <map>
4 #include <string>
5 #include <algorithm>
6
7 using namespace std;
8
9 void printAll(const vector<string> &sv){
10  for(int i=0; i<sv.size(); i++){
11      cout << sv[i].c_str() << endl;
12  }
13}
14
15 void fillMap(map<string, int> &theMap, const vector<string> &sv){
16  int i=0;
17  for(vector<string>::const_iterator it=sv.begin(); it!=sv.end(); it++){
18      theMap.insert(pair<string, int>(*it, i));
19      i++;
20  }
21}
22
23 void printPosition(const map<string, int> &theMap, string name){
24  map<string, int>::const_iterator it=theMap.find(name);
25  if(it==theMap.end())
26      cerr << name.c_str() << " not found" <<endl;
27  else
28      cout << name.c_str() << " is at position " << it->second +1 << endl;
29}
30
```

```
31 int main(int argc, char* argv[]){
32  vector<string> stringVector;
33
34  stringVector.push_back("Andreas");
35  stringVector.push_back("Marios");
36  stringVector.push_back("Aggeliki");
37  stringVector.push_back("Athanasia");
38  stringVector.push_back("Thanasis");
39
40  cout << "******Raw data" << endl;
41  printAll(stringVector);
42  cout << endl;
43  cout << "******Sorted" << endl;
44  sort(stringVector.begin(), stringVector.end());
45  printAll(stringVector);
46  cout << endl;
47  map<string, int> positionMap;
48  fillMap(positionMap, stringVector);
49  printPosition(positionMap, "Marios");
50  printPosition(positionMap, "Tassos");
51  printPosition(positionMap, "Andreas");
52
53  cout << endl;
54  system("pause");
55  return 0;
56}
```


C:\Users\Marios\Desktop\examples\ex5.exe
******Raw data
Andreas
Marios
Aggeliki
Athanasia
Thanasis
******Sorted
Aggeliki
Andreas
Athanasia
Marios
Thanasis
Marios is at position 4
Tassos not found
Andreas is at position 2
Press any key to continue . . .

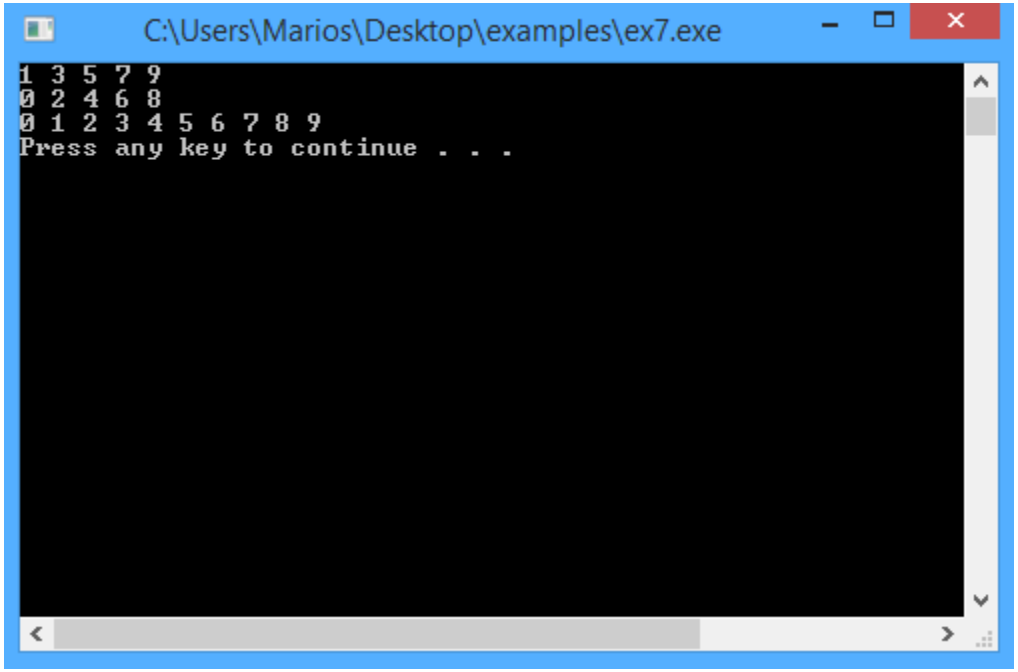
Παράδειγμα (2)

```
1 #include <iostream>
2 #include <list>
3 #include <algorithm>
4
5 using namespace std;
6
7 int main(int argc, char* argv[]){
8   list<int> numberList;
9
10  numberList.push_back(3);
11  numberList.push_back(4);
12  numberList.push_front(2);
13  numberList.push_front(1);
14  numberList.push_front(1);
15
16  for (list<int>::iterator it=numberList.begin(); it!=numberList.end(); it++){
17      cout << *it << " ";
18  }
19  cout << endl;
20
21  numberList.unique();
22  for (list<int>::reverse_iterator rit=numberList.rbegin(); rit!=numberList.rend(); rit++){
23      cout << *rit << " ";
24  }
25  cout << endl << "*****************************************" <<endl;
26
27  do{
28      for (list<int>::iterator it = numberList.begin(); it!=numberList.end(); it++){
29      cout << *it << " ";
30  }
31  cout << endl;
32  }while(next_permutation(numberList.begin(), numberList.end()));
33  system("pause");
34  return 0;
35}
```


C:\Users\Marios\Desktop\examples\ex6.exe
1 1 2 3 4
4 3 2 1
****
1 2 3 4
1 2 4 3
1 3 2 4
1 3 4 2
1 4 2 3
1 4 3 2
2 1 3 4
2 1 4 3
2 3 1 4
2 3 4 1
2 4 1 3
2 4 3 1
3 1 2 4
3 1 4 2
3 2 1 4
3 2 4 1
3 4 1 2
3 4 2 1
4 1 2 3
4 1 3 2
4 2 1 3
4 2 3 1
4 3 1 2
4 3 2 1
Press any key to continue . . .

Παράδειγμα (3)

```
1 #include <iostream>
2 #include <vector>
3 #include <algorithm>
4
5 using namespace std;
6
7 void printAll(vector<int>::iterator begin, vector<int>::iterator end){
8   for (vector<int>::iterator it=begin; it!=end; it++)
9       cout <<*it <<" ";
10  cout <<endl;
11}
12
13 int main(int argc, char* argv[]){
14  vector<int> set1;
15  vector<int> set2;
16  vector<int> set3(10);
17
18  vector<int>::iterator startit=set3.begin();
19  vector<int>::iterator endit;
20
21  for (int i=0; i<10; i++)
22      set1.push_back(i);
23
24  for (int i=0; i<10; i+=2)
25      set2.push_back(i);
26
27  endit = set_difference(set1.begin(), set1.end(), set2.begin(), set2.end(), startit);
28  printAll(startit, endit);
29  endit = set_intersection(set1.begin(), set1.end(), set2.begin(), set2.end(), startit);
30  printAll(startit, endit);
31  endit = set_union(set1.begin(), set1.end(), set2.begin(), set2.end(), startit);
32  printAll(startit, endit);
33
34  system("pause");
35  return 0;
36}
```

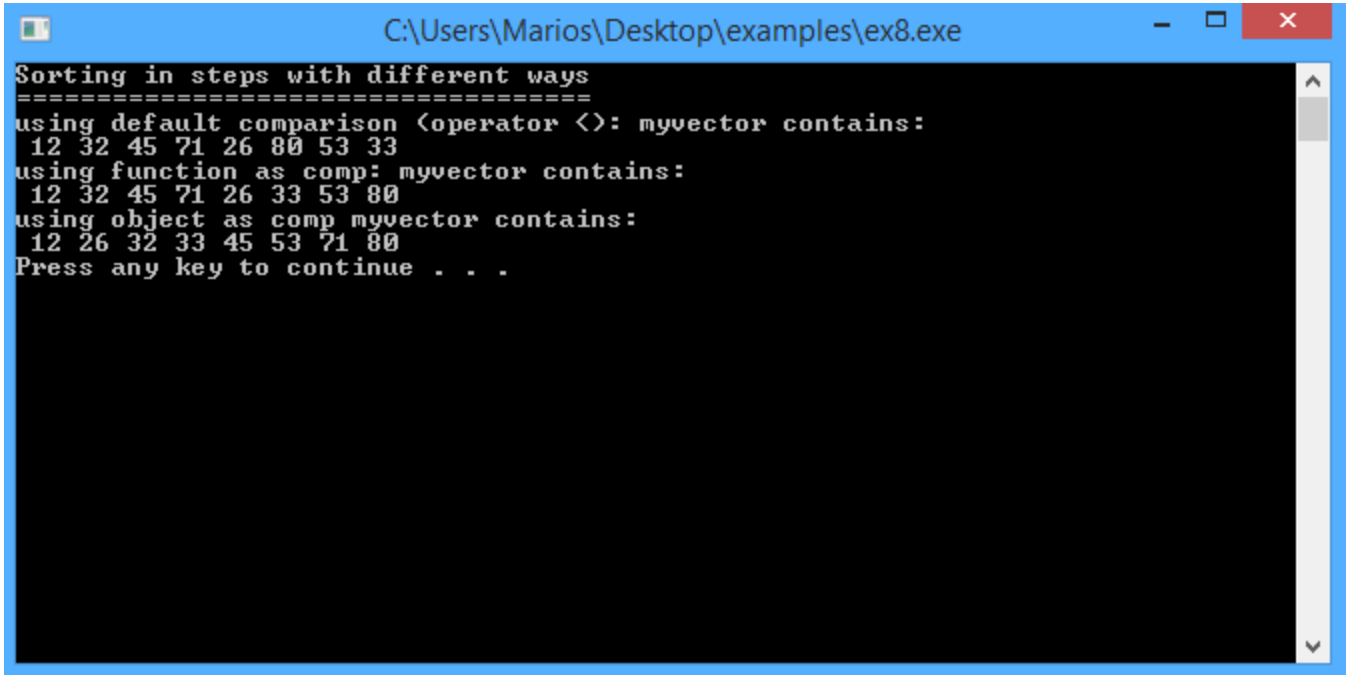

C:\Users\Marios\Desktop\examples\ex7.exe
1 3 5 7 9
0 2 4 6 8
0 1 2 3 4 5 6 7 8 9
Press any key to continue . . .

Παράδειγμα (4)

```
1  #include <iostream>
2  #include <algorithm>
3  #include <vector>
4
5  using namespace std;
6
7  bool myfunction (int i, int j){
8    return (i<j);
9    }
10
11 struct myclass {
12  bool operator() (int i, int j){
13       return (i<j);
14       }
15}      myobject;
16
```

```
17 int main () {
18  int myints[] = {32,71,12,45,26,80,53,33};
19  vector<int> myvector (myints, myints+8);
20
21  cout << "Sorting in steps with different ways" << endl;
22  cout << "====================================" << endl;
23  sort(myvector.begin(), myvector.begin()+4);
24  cout << "using default comparison (operator <): myvector contains:" << endl;
25  for (vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
26      cout << " " << *it;
27  cout << endl;
28
29  sort(myvector.begin()+4, myvector.end(), myfunction);
30  cout << "using function as comp: myvector contains:" << endl;
31  for (vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
32      cout << " " << *it;
33  cout << endl;
34
35  sort(myvector.begin(), myvector.end(), myobject);
36  cout << "using object as comp myvector contains:" << endl;
37  for (vector<int>::iterator it=myvector.begin(); it!=myvector.end(); ++it)
38      cout << " " << *it;
39  cout << endl;
40
41  system ("pause");
42  return 0;
43}
```

Functor (Function Object) Χρήση operator() ως συνάρτηση


C:\Users\Marios\Desktop\examples\ex8.exe
Sorting in steps with different ways
==========
using default comparison (operator <): myvector contains:
12 32 45 71 26 80 53 33
using function as comp: myvector contains:
12 32 45 71 26 33 53 80
using object as comp myvector contains:
12 26 32 33 45 53 71 80
Press any key to continue . . .

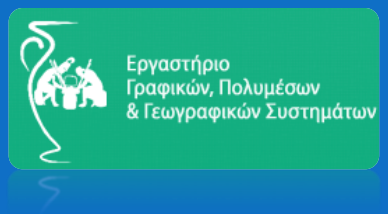
References

- http://www.cplusplus.com/reference/stl/
- http://www.cprogramming.com/tutorial/stl/stlintro.html
- http://www.yolinux.com/TUTORIALS/LinuxTutorialC++STL.html
- https://www.sgi.com/tech/stl/
- http://www.stanford.edu/class/cs106l/course-reader/Ch7_STLAlgorithms.pdf

Εκτενείς Δομές Δεδομένων(Κεφ. 5)

Δομές Δεδομένων

Τεστέμπασης Αθανάσιος
Ε.Τ.Υ.



17 Μαρτίου 2015

testebasis.thanos@gmail.com

Περιεχόμενα

1. Εισαγωγή
2. AVL Δέντρα
3. (a,b) Δέντρα

Εισαγωγή

- Δομές που βασίζονται σε συγκρίσεις : *Ισοζυγισμένα δέντρα εύρεσης* (δέντρα τα φύλλα των οποίων απέχουν της ίδιας τάξεως μεγέθους,απόσταση απο τη ρίζα)
  - Υψοζυγισμένα δέντρα (κριτήριο ζύγισης κάθε κόμβου αποτελεί το ύψος των υποδέντρων του) :
    - AVL
    - B-TREES
    - (a,b) TREES
    - RED – BLACK TREE
  - Βαροζυγισμένα δέντρα (κριτήριο ζύγισης κάθε κόμβου αποτελεί το βάρος των υποδέντρων του) :
    - BB[a] TREES
    - SKIP LISTS
    - INTERPOLATION SEARCH TREES
    - ΒΑΡΟΖΥΓΙΣΜΈΝΑ B-TREES

AVL TREE

- Δυαδικό ισοζυγισμένο δέντρο – σε κάθε κόμβο τα ύψη των υποδέντρων του διαφέρουν το πολύ κατα ένα.
- Υψοζύγιση του u : hb(u) = ύψος (R(u)) – ύψος (L(u)) hb(u) { +1 , 0 , -1 }
- Το ύψος h ενός δένδρου AVL με n στοιχεία είναι O(logn).

AVL TREE – Access(x)

- Ξεκινάμε απο τη ρίζα και ελέγχουμε σε κάθε κόμβο u :
  - αν x < val(u) συνεχίζουμε αριστερά
  - αν x> val(u) συνεχίζουμε δεξιά

  εως οτου βρούμε το x
- Ο χρόνος της Access(x) είναι Θ(logn)

AVL TREE – Insert(x , T) (1/2)

- Ο πρώτος κόμβος στο μονοπάτι από τον κόμβο v (που εισήχθη στο δένδρο) προς τη ρίζα, του οποίου το balance ήταν +1 ή −1 (πριν την εισαγωγή) ονομάζεται *κρίσιμος κόμβος* (κρίσιμο μονοπάτι αντίστοιχα).
- Αν ο κόμβος αυτός αποκτά balance +2 ή −2 μετά την εισαγωγή, τότε είναι ο πρώτος κόμβος στο μονοπάτι από τον v στη ρίζα για τον οποίο θα πρέπει να γίνουν κατάλληλες ενέργειες ώστε να διορθωθεί το balance του.
- Διακρίνουμε περιπτώσεις ανάλογα με το είδος των δύο πρώτων ακμών του μονοπατιού από τον κρίσιμο κόμβο w προς τον εισαχθέντα κόμβο v(απλή περιστροφή / διπλή περιστροφή)

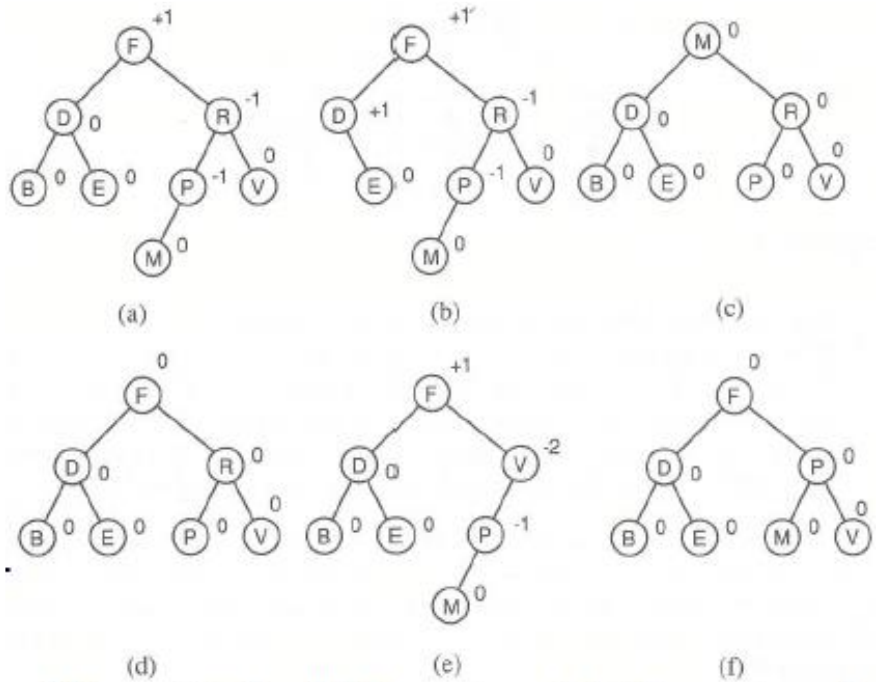
AVL TREE – Insert(x , T) (2/2)

- Περίπτωση RR (Right-Right)
  - Και οι δύο ακμές οδηγούν δεξιά.Εκτελούμε μια αριστερή περιστροφή γύρω από τον κρίσιμο κόμβο.
- Περίπτωση LL (Left-Left)
  - Και οι δύο ακμές οδηγούν αριστερά.Είναι συμμετρική της περίπτωσης RR!Μία δεξιά περιστροφή γύρω από τον κρίσιμο κόμβο αρκεί για να επιλυθεί το πρόβλημα με το balance του!
- Περίπτωση RL (Right-Left)
  - Η πρώτη ακμή οδηγεί δεξιά και η δεύτερη αριστερά.Απαιτούνται δύο περιστροφές, μια δεξιά περιστροφή γύρω από τον επόμενο του κρίσιμου κόμβου στο μονοπάτι του οδηγεί στον v και μια αριστερή περιστροφή γύρω από τον κρίσιμο κόμβο.
- Περίπτωση LR (Left-Right)
  - Η πρώτη ακμή οδηγεί αριστερά και η δεύτερη δεξιά.Είναι συμμετρική της περιπτώσεως RL.Απαιτούνται δύο περιστροφές, μια αριστερή περιστροφή γύρω από τον επόμενο του κρίσιμου κόμβου στο μονοπάτι που οδηγεί στον v και μια δεξιά περιστροφή γύρω από τον κρίσιμο κόμβο.

AVL TREE – Delete(x , T) (1/2)

- Ακολουθούμε τον ‘αλγόριθμο’ διαγραφής σε δυαδικό δένδρο:
  - Access(x)
  - Διαγραφή του ίδιου του κόμβου x αν είναι φύλλο.
  - Αντικατάστασή του από το παιδί του αν έχει μόνο ένα παιδί.
  - Αντικατάστασή του από τον επόμενό του στην ενδοδιατεταγμένη διάταξη αν έχει δύο παιδιά.
  - Balance

AVL TREE – Delete(x , T) (2/2)



(a) Αρχικό δένδρο, (b) Διαγραφή του B, (c) Διαγραφή του F,
(d) Διαγραφή του M, (e) Διαγραφή του R

(a,b) TREES

- Έστω a,b ακέραιοι τέτοιοι ώστε $a \geq 2$ και $b \geq 2a-1$. Ένα δέντρο T ειναι (a, b) αν
  - Όλα τα φύλλα του T έχουν το ίδιο βάθος (δηλαδή το δέντρο είναι πλήρως ζυγισμένο)
  - Για κάθε κόμβο u του T, ισχύει $p(u) \leq b$ {p(u) = αριθμός των παιδιών του u}
  - Για κάθε κόμβο u του T,με εξέρεση τη ρίζα,ισχύει $p(u) \geq a$
  - Για τη ρίζα r ,ισχύει $p(r) \geq 2$
- Όταν $b = 2a – 1$ τοτε το (a,b) tree ονομάζεται B-tree.

(a,b) TREES – Access(x)

- Ίδια διαδικασία οπως στα AVL trees

(a,b) TREES – Insert (x , T)

- Διαδικασία Access(x)
- Δημιουργία ενος νεου κενού φύλλου στο οποίο αποθηκεύουμε το x. Αν x<y τοτε τοποθετείται αριστερά, αν x>y τοποθετείτε δεξιά.
- Balance
  - Split

(a,b) TREES – Delete (x , T)

- Διαδικασία Access(x)
- Σβήνεται το φύλλο με τη τιμή x και στον πατέρα u του x διαγράφεται το κλειδί $k_i(u)$ αν το φύλλο x ειναι i-στο παιδί του u ή $k_{p(u)-1}$ (u) αν το p(u)-στο παιδί του.
- Balance
  - Share(διαμοιρασμός)
  - Fusion(σύμπτυξη)

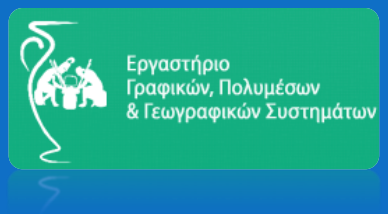
Boost - Boost Graph Library

C++ library

Δομές Δεδομένων

Τεστέμπασης Αθανάσιος

Ε.Τ.Υ.



17 Μαρτίου 2015

testebasis.thanos@gmail.com

Boost
(1/2)

- Η Boost δεν είναι μια βιβλιοθήκη, αλλά μια συλλογή με περισσότερες από 50 βιβλιοθήκες που λειτουργούν καλά μαζί.
- Πολλοί χρήστες ξεκινούν με μόνο μία ή δύο Boost βιβλιοθήκες, και στη συνέχεια επεκτείνουν την χρήση τους με την πάροδο του χρόνου.
- Όλες οι πληροφορίες για την Boost, συμπεριλαμβανομένου του κώδικα, των εγγράφων τεκμηρίωσης, και οτιδήποτε άλλο περιλαμβάνεται στην ιστοσελίδα της Boost είναι δωρεάν.
- Για να γίνουν αποδεκτές οι βιβλιοθήκες από την Boost, είναι απαραίτητο να περάσουν από δημόσια αξιολόγηση. Αυτή η διαδικασία αξιολόγησης έχει οδηγήσει σε πολύ υψηλής ποιότητας βιβλιοθήκες.
- Ο κώδικας υποστηρίζει φορητότητα, έτσι ώστε να επιτραπεί η χρήση σε πολλαπλούς compilers και λειτουργικά συστήματα.
- Ο πηγαίος κώδικας διανέμεται για όλες τις βιβλιοθήκες, ώστε οι χρήστες να μπορούν να ελέγχουν, και να τροποποιούν τον κώδικα για να ανταποκρίνονται στις εκάστοτε ανάγκες.

Boost
(2/2)

- Οι βιβλιοθήκες της Boost λειτουργούν άψογα με τις C + + Standard Βιβλιοθήκες, και περίπου δώδεκα από αυτές έχουν γίνει δεκτές για την επερχόμενη Τεχνική Έκθεση της Stardard Library. Αυτό σημαίνει ότι οι βιβλιοθήκες αυτές τελικά θα είναι συμβατές με τους περισσότερους μεταγλωττιστές της C++.
- Τυπικές Βιβλιοθήκες της Boost
  - Smart Pointers
  - Expressions
  - Boost Graph Library

Smart Pointers (1/3)

- Η διαχείριση της μνήμης στην C + + επαφίεται στον προγραμματιστή.
- Απαιτείται λήψη κρίσιμων αποφάσεων για τη διαχείριση μνήμης.
- Η απρόσεκτη χρήση της μνήμης σωρού μπορεί να οδηγήσει σε απώλειες μνήμης και αποτυχία του προγράμματος.
- Οι Smart Pointers αποτελούν τον ιδανικότερο τρόπο στην C + + για να διασφαλιστεί η ορθή διαχείριση της δυναμικής μνήμης που δεσμεύεται από εκφράσεις τύπου **new**.
- Οι Smart Pointers είναι παρόμοιοι με τους regular pointers, αλλά φροντίζουν για τις λεπτομέρειες της διαγραφής του αντικειμένου στο οποίο δείχνει ο pointer, όταν δεν είναι πλέον απαραίτητο.
- Η C + + πρότυπη βιβλιοθήκη περιέχει ήδη τον *std:: auto_ptr,* έναν Smart Pointer που μεταφέρει σημασιολογία ιδιοκτησίας.
- Η Boost προσθέτει επιπρόσθετους smart pointers για χειρισμό άλλων κοινών αναγκών, όπως η σημασιολογία της από κοινού-ιδιοκτησίας.

Smart Pointers (2/3)

- Η πρότυπη κλάση *shared_ptr* της Boost παρέχει έναν smart pointer κοινής ιδιοκτησίας που μπορεί να διαχειριστεί δυναμικά την κατανεμημένη μνήμη σε ένα ευρύ φάσμα εφαρμογών, και μπορεί να χρησιμοποιηθεί για να επικοινωνήσει με τρίτες βιβλιοθήκες.
- Σε αντίθεση με την *std:: auto_ptr,* η *shared_ptr* λειτουργεί καλά με τους STL containers.
- Ο ακόλουθος κώδικας αποσκοπεί στη δημιουργία ενός *std:: vector* της *shared_ptr σε* ένα τύπο που ονομάζεται *Foo*

```
typedef boost::shared_ptr<Foo>FooPtr;

std::vector<FooPtr> foo_vector;
```

- Το διάνυσμα μπορεί να γεμίσει ως εξής:

```
foo_vector.push_back( FooPtr(new Foo(3)) );

foo_vector.push_back( FooPtr(new Foo(2)) );

foo_vector.push_back( FooPtr(new Foo(1)) );.
```

Smart Pointers (3/3)

```
#include <boost/shared_ptr.hpp>
struct Foo
{
Foo( int _x ) : x(_x) {}
~Foo() { std::cout << "Destructing a Foo with x=" << x
<< "\n"; }
int x;
/* ... */
};
typedef boost::shared_ptr<Foo> FooPtr;
struct FooPtrOps
{
bool operator()( const FooPtr & a, const FooPtr & b )
{ return a->x < b->x; }
void operator()( const FooPtr & a )
{ std::cout << " " << a->x; }}
```

```
Original foo_vector: 3 2 1
    Sorted foo_vector: 1 2 3
    Destructing a Foo with x=1
    Destructing a Foo with x=2
    Destructing a Foo with x=3
```

```
int main()
{
std::vector<FooPtr> foo_vector;
foo_vector.push_back( FooPtr(new Foo(3)) );
foo_vector.push_back( FooPtr(new Foo(2)) );
foo_vector.push_back( FooPtr(new Foo(1)) );
std::cout << "Original foo_vector:";
std::for_each( foo_vector.begin(), foo_vector.end(),
FooPtrOps() );
std::cout << "\n";
std::sort( foo_vector.begin(), foo_vector.end(), FooPtrOps() );
std::cout << "Sorted foo_vector:";
std::for_each( foo_vector.begin(), foo_vector.end(),
FooPtrOps() );
std::cout << "\n";
return 0;
}
```

Boost και Regular Expressions

- Η Βιβλιοθήκη Regular Expression της Boost, η οποία συχνά αποκαλείται regex + +, έρχεται να καλύψει ένα σημαντικό κενό στην C + + Standard Library.
- Οι Regular Expressions επιτυγχάνουν ταίριασμα των τύπων strings (string pattern matching), και έτσι είναι απαραίτητες για διαχείρηση των strings.
- Ακολουθεί ένα απλό παράδειγμα:

```
bool validate_card_format(const std::string s)
{
static const boost::regex e("(\\d{4}[- ]){3}\\d{4}");
return regex_match(s, e);
}
```

- Αποτύπωση κώδικα: Να γίνεται δεκτό ένα string το οποίο αποτελείται από τρεις ομάδες των τεσσάρων ψηφίων που ακολουθείται από μια παύλα ή ένα κενό, και τελειώνει με άλλα τέσσερα ψηφία.

Γνωρίσματα Τύπων - Type Traits

- Βοηθάει τους προγραμματιστές να διαχειριστούν το είδος πληροφοριών κατά τη μεταγλώττιση.
- Κάποιες χρήσεις περιλαμβάνουν την υποστήριξη μιας ευρύτερης ποικιλίας τύπων που έχουν γραφτεί από χρήστες, προσθέτοντας ισχυρισμούς χρόνου μεταγλώτισσης για τη βελτίωση της ανίχνευσης σφαλμάτων, και την ενεργοποίηση βελτιστοποιήσεων.
- Τα γνωρίσματα τύπων παρέχουν πρότυπα με value που θα είναι αληθής ή ψευδής.
  - Π.χ. boost::is_integral<T>::value

    Εάν ο τύπος T αποτελεί αναπόσπαστο τύπο σύμφωνα με τους κανόνες της γλώσσας, η έκφραση αυτή θα αποτιμηθεί σε true, διαφορετικά θα αποτιμηθεί σε false.

Boost – MetaProgramming Library (MPL)

- Ο όρος Metaprogramming μπορεί να χαρακτηριστεί ως ≪προγράμματα που παράγουν προγράμματα≫.
- Τα μεταπρογράμματα τρέχουν σε χρόνο μεταγλώτισσης, προσαρμόζοντας τον κώδικα που μεταγλωτίζεται στο τελικό πρόγραμμα.
- Η MPL παρέχει μία πλούσια γκάμα λειτουργιών που εφαρμόζονται στους τύπους, συμπεριλαμβανομένου του ελέγχου της ροής, των containers, των iterators, και των αλγορίθμων, και μεταγλωτίζεται στο τελικό πρόγραμμα.

Boost – MetaProgramming Library (MPL)

- Παράδειγμα:

```
template< typename T >
class auto_size_example : private mpl::if_< sizeof(T) <= sizeof(double)
        , stack_implementation<T>
        , heap_implementation<T>
>::type
{
  // ...
};
```

- Επεξήγηση: Όταν ένα template auto_size_example αρχικοποιείται κατά τη μεταγλώττιση, το if_ της MPL καθορίζει αν θα κληρονομήσει από το stack_implementation ή από το heap_implementation ανάλογα με το μέγεθος της παραμέτρου T . Το καθαρό αποτέλεσμα είναι η βελτιστοποίηση που θα ήταν δύσκολο να εκτελεστεί διαφορετικά.

Boost Spirit Library

- Αποτελεί ένα σύνολο από C++ βιβλιοθήκες για το παρσάρισμα και την παραγωγή αποτελεσμάτων που υλοποιούνται ως ενσωματωμένες γλώσσες που εξαρτώνται από το πεδίο - Domain Specific Embedded Languages (DSEL) που χρησιμοποιούν πρότυπα έκφρασης και πρότυπο μετα- προγραμματισμό.
- Επιτρέπει σε BNF γραμματικές να εκφράζονται απευθείας στην C++, και ως εκ τούτου αποτελεί ένα σημαντικό επίτευγμα.
- Η γραμματική μπορεί να αναμιχθεί ελεύθερα με C++ κώδικα και, χάρη στην παραγωγική δύναμη των C++ προτύπων, είναι αμέσως εκτελέσιμη.

Boost Graph Library (1/7)

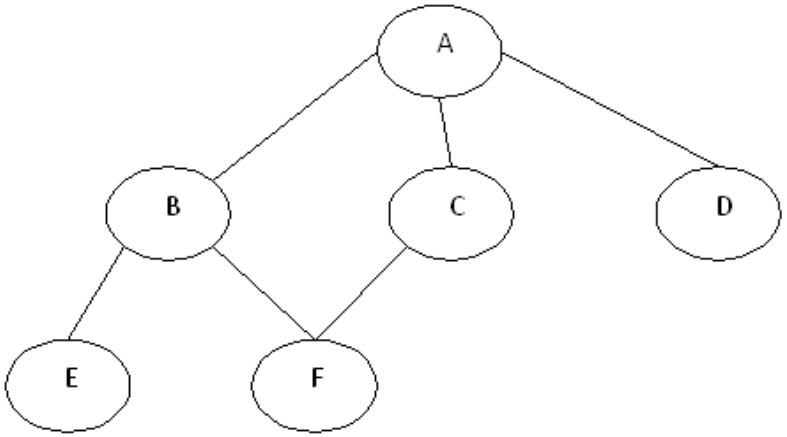
- Ένα μέρος της βιβλιοθήκης των γράφων της Boost είναι μια γενική διεπαφή που επιτρέπει πρόσβαση στη δομή ενός γράφου, αλλά κρύβει τις λεπτομέρειες της υλοποίησης.
- Είναι μια ≪ανοικτή≫ διεπαφή υπό την έννοια ότι οποιαδήποτε βιβλιοθήκη γράφων εφαρμόζει αυτήν την διεπαφή θα αλληλεπιδρά με τους BGL γενικούς αλγορίθμους και με άλλους αλγορίθμους που χρησιμοποιούν επίσης αυτήν την διεπαφή.
- Η βιβλιοθήκη BGL παρέχει μερικές κλάσεις γράφων γενικού σκοπού που προσαρμόζονται σε αυτήν την διεπαφή, αλλά δεν προορίζονται να είναι οι ≪μόνες≫ κλάσεις γράφων.

Boost Graph Library (2/7)

- Γράφημα G = (V, E)

  Όπου V ένα σύνολο κορυφών V = {v1, v2, v3, … , vn}

  Και E ένα σύνολο ακμών της μορφής {vi, vj} με 1 ≤ i, j ≤ n



Boost Graph Library (3/7)

- Τρόποι αναπαράστασης γραφημάτων:
  - Edge List: { (A,B), (A,C), (A,D), (B,E), (B,F), (C, F) }
  - Adjacency List:

    A: { B, C, D } B: { A, E, F }

    C: { A, F } D: { A }

    E: { B } F: { B, C }
- Adjacency Matrix :

	A	B	C	D	E	F
A	0	1	1	1	0	0
B	1	0	0	0	1	1
C	1	0	0	0	0	1
D	1	0	0	0	0	0
E	0	1	0	0	0	0
F	0	1	1	0	0	0

Boost Graph Library (4/7)

Expression	Return Type or Description
Graph	
graph_traits<G>::vertex_descriptor	The type of object used to identify vertices.
graph_traits<G>::directed_category	Directed or undirected edges?
graph_traits<G>::traversal_category	What kind of iterator traversal is supported?
graph_traits<G>::edge_parallel_category	Allow insertion of parallel edges?
IncidenceGraph refines Graph	
graph_traits<G>::edge_descriptor	The type of object used to identify edges.
graph_traits<G>::out_edge_iterator	Iterate through the out-edges.
graph_traits<G>::degree_size_type	The integer type for vertex degee.
out_edges(v, g)	*std::pair<out_edge_iterator, out_edge_iterator>*
source(e, g)	*vertex_descriptor*
target(e, g)	*vertex_descriptor*
out_degree(v, g)	*degree_size_type*
BidirectionalGraph refines IncidenceGraph	
graph_traits<G>::in_edge_iterator	Iterate through the in-edges.
in_edges(v, g)	*std::pair<in_edge_iterator, in_edge_iterator>*
in_degree(v, g)	*degree_size_type*
degree(e, g)	*degree_size_type*

Boost Graph Library (5/7)

AdjacencyGraph refines Graph	
graph_traits<G>::adjacency_iterator	Iterate through adjacent vertices.
adjacent_vertices(v, g)	*std::pair<adjacency_iterator, adjacency_iterator>*
VertexListGraph refines Graph	
graph_traits<G>::vertex_iterator	Iterate through the graph's vertex set.
graph_traits<G>::vertices_size_type	The unsigned integer type for representing the number of vertices.
num_vertices(g)	*vertices_size_type*
vertices(g)	*std::pair<vertex_iterator, vertex_iterator>*
EdgeListGraph refines Graph	
graph_traits<G>::edge_descriptor	The type of object used to identify edges.
graph_traits<G>::edge_iterator	Iterate through the graph's edge set.
graph_traits<G>::edges_size_type	The unsigned integer type for representing the number of edges.
num_edges(g)	*edges_size_type*
edges(g)	*std::pair<edge_iterator, edge_iterator>*
source(e, g)	*vertex_descriptor*
target(e, g)	*vertex_descriptor*
AdjacencyMatrix refines Graph	
edge(u, v, g)	*std::pair<edge_descriptor, bool>*

Boost Graph Library (6/7)

- Βασικοί αλγόριθμοι που προσφέρονται στην BGL:
  - breadth_first_search
  - depth_first_search
  - dijkstra_shortest_paths
  - bellman_ford_shortest_paths
  - kruskal_minimum_spanning_tree
  - prim_minimum_spanning_tree
  - connected_components
  - strong_components
  - edmunds_karp_max_flow
  - push_relabel_max_flow

Boost Graph Library (7/7)

```
template <typename UndirectedGraph> void undirected graph demo1() {

const int V = 3;

UndirectedGraph undigraph(V);

typename graph traits<UndirectedGraph>::vertex descriptor zero, one, two;

typename graph traits<UndirectedGraph>::out edge iterator out, out end;

typename graph traits<UndirectedGraph>::in edge iterator in, in end;

zero = vertex(0, undigraph);

one = vertex(1, undigraph);

two = vertex(2, undigraph);

add edge(zero, one, undigraph);

add edge(zero, two, undigraph);

add edge(one, two, undigraph);

std::cout << "out˙edges(0): ";

for (tie(out, out end) = out edges(zero, undigraph); out != out end; ++out)

std::cout << *out;

std::cout << std::endl << "in˙edges(0): ";

for (tie(in, in end) = in edges(zero, undigraph); in != in end; ++in)

std::cout << *in;

std::cout << std::endl;

}
```

```
out edges(0): (0,1) (0,2)
in edges(0): (1,0) (2,0)
```

References

- Boost Site: http://www.boost.org
- An Introduction to Boost: http://www.codeproject.com/KB/stl/boostintro.aspx
- List Of Libraries In Boost: http://en.wikipedia.org/wiki/Boost_%28programming%29
- Ενδιαφέρουσα Παρουσίαση για Generic Programming και BGL http://ecee.colorado.edu/~siek/boostcon2010bgl.pdf
- Επιπλέον δείτε το **The Boost Graph Library User Guide and Reference Manual, Jeremy Siek** για αναλυτική παρουσίαση, παραδείγματα, γενικό προγραμματισμό, class reference κ.ά.

JDSL

Java Data Structures Library

Δομές Δεδομένων

Μπαλτάς Αλέξανδρος

Εργαστήριο
Γραφικών, Πολυμέσων
& Γεωγραφικών Συστημάτων

24 Μαρτίου 2015

ampaltas@ceid.upatras.gr

Εισαγωγή

Η JDSL είναι μια βιβλιοθήκη Δομών Δεδομένων σε Java που αναπτύχθηκε από Brown University and Johns Hopkins University.

Είναι μια συλλογή από interfaces και κλάσεις σε Java, που υλοποιούν βασικές δομές δεδομένων και αλγορίθμους όπως:

- Sequences
- Binary trees
- Priority Queues
- αλγόριθμους αναζήτησης

Πλεονεκτήματα της JDSL

- Λειτουργικότητα: Μεγάλη ποικιλία υλοποιημένων δομών και αλγορίθμων.
- Απόδοση: Στις υλοποιήσεις των δομών της JDSL συναντούμε τις θεωρητικές ασυμπτωτικές πολυπλοκότητες χώρου και χρόνου.
- Ευελιξία: Η JDSL παρέχει διαφορετικές υλοποιήσεις για κάποιες δομές δεδομένων για διαφορετική χρήση ανάλογα σε εφαρμογή.
- Επεκτασιμότητα: Παρέχει λεπτομερή interfaces για πολλές δομές δεδομένων ώστε να μπορεί κανείς να κάνει μια νέα υλοποίηση.

Δομή της JDSL [Container & Element]

Container:

- Κάθε δομή δεδομένων αντιμετωπίζεται από την JDSL σαν μια οργανωμένη συλλογή αντικειμένων που καλούνται elements της δομής.
- Όλες οι δομές δεδομένων της JDSL υλοποιούν το Container Interface το οποίο ορίζει βασικές μεθόδους όπως την αναφορά του αριθμού των στοιχείων της δομής, και την επιστροφή ενός iterator για τα στοιχεία.

Element:

- Ένα element μιας δομής δεδομένων της JDSL μπορεί να είναι ένα οποιοδήποτε java.lang.Object.
- Το ίδιο αντικείμενο μπορεί να αποθηκευτεί σε πολλές δομές δεδομένων, η πολλές φορές στην ίδια.

Δομή της JDSL [Container]

Η JDSL υλοποιεί 2 ειδών containers:

- Key-based Containers

  Αποθηκεύουν ζευγάρια από key-element. Τα keys χρησιμοποιούνται συνήθως ως μηχανισμός για τη δημιουργία ευρετηρίου στα στοιχεία που συνδέονται. Π.χ. στο λεξικό, που είναι μια key-based δομή δεδομένων, κάποιες μέθοδοι που υποστηριζονται είναι η εισαγωγή ενός ζέυγους (key, element), η αναζήτηση σε ένα key κλπ.

- Positional Containers

  Είναι Δομές δεδομένων που δημιουργούν τοπολογικές σχέσεις μεταξύ των elements τους, όπως ακολουθίες, δέντρα κλπ. Τα στοιχεία προσπελαύνονται μέσω της θέσης τους.

Δομή της JDSL [Accessor]

- Η JDSL παρέχει πρόσβαση στα στοιχεία μιας δομής δεδομένων μέσω των accessors.
- Ένας accessor παρέχει πρόσβαση σε ένα στοιχείο μια δομής δεδομένων συνεχώς, ανεξάρτητα από την υλοποίηση της (πχ στην JDSL ένα sequence υλοποιείται είτε ως ένας πίνακας, είτε ως μια linked list).
- Κάθε element μιας δομής δεδομένων διαθέτει έναν accessor που σχετίζεται με αυτό.

Δομή της JDSL [Accessor]

- Positional Containers: Position

  Ένα position συμβολίζει τη «θέση» ενός element σε ενα positional container. Τα positional containers είναι δομές που μπορούμε να αναπαραστήσουμε με κόμβους (π.χ. Sequences, graphs κλπ), και το position ενός element εκφράζει τον κόμβο στον οποίο αυτό αποθηκεύεται.

- Key-based Containers: Locator

  Στα key-based Containers η έννοια της «θέσης» ενός element στο χώρο δεν υπάρχει, και κάθε στοιχείο εντοπίζεται από το κλειδί με το οποίο συνδέεται. Όταν προστίθεται ένα ζεύγος (key, element) σε έναν container αυτός θα επιστρέψει έναν locator για αυτό, και στη συνέχεια μπορεί να γίνει αναφορά στο ζεύγος μέσω του locator.

Δομή της JDSL [Iterator]

- Παρέχουν έναν απλό μηχανισμό για επαναλήψεις στα αντικείμενα μίας συλλογής.
- Οι containers παρέχουν μεθόδους που επιστρέφουν iterators που διατρέχουν όλα τα elements του container.
- Οι iterators μπορούν να κινηθούν μόνο προς τα εμπρός.
- Μπορούν να επανέλθουν στην αρχή για να επαναλάβουν την διαδικασία.

Positional Containers (Sequence)

- Ένα sequence είναι μια βασική δομή δεδομένων για την αποθήκευση στοιχείων με ένα γραμμικό τρόπο.
- InspectableSequence & Sequence: Τα interfaces της JDSL για sequences. Παρέχει μεθόδους για προσπέλαση και τροποποίηση των θέσεων στο τέλος του sequence αλλά και σε συγκεκριμένες θέσεις της.
- NodeSequence: Υλοποίηση του sequence με χρήση διπλά συνδεδεμένης λίστας.
- ArraySequence: Υλοποίηση του sequence με χρήση ενός πίνακα αυξανόμενου μεγέθους.

Positional Containers (Sorting Algorithms)

- Η JDSL παρέχει σειρά αλγόριθμων ταξινόμησης για sequences που υλοποιούν το interface SortObject. Ταξινομήσεις με πρόθεμα List είναι πιο αποτελεσματικές όταν χρησιμοποιούνται με την ακολουθία NodeSequence ενώ με πρόθεμα Array είναι πιο αποτελεσματικές με την ArraySequence.
- ListQuickSort/ArrayQuickSort: Γρήγορος αλγόριθμος, τρέχει σε O(NlogN). Η επίδοσή του υποβαθμίζεται σε μεγάλο βαθμό, αν η σειρά εχει σχεδόν ταξινομηθεί. Δεν είναι σταθερός , δεν εγγυάται ότι τα στοιχεία με ίδια τιμή θα παραμείνουν στην ίδια σειρά που είχαν πριν από τη ταξινόμηση.
- ListMergeSort/ArrayMergeSort: Πιο αργός από QuickSort, με την ίδια όμως πολυπλοκότητα O(NlogN) . Η απόδοση του δεν υποβαθμιζεται λόγω ιδιαιτεροτήτων στα δεδομένα εισόδου.
- HeapSort: Βασίζεται στο ArrayHeap. Η επίδοσή του επίσης δεν υποβαθμίζεται, λόγω ιδιαιτεροτήτων στην εισαγωγή δεδομένων.

Key-based Containers (PriorityQueue)

- Ένα PriorityQueue είναι μια δομή δεδομένων για την αποθήκευση μιας συλλογής στοιχείων στα οποία δίνεται προτεραιότητα με βάση τα κλειδιά, όπου η μικρότερη τιμή κλειδιού υποδεικνύει την υψηλότερη προτεραιότητα.
- Υποστηρίζει αυθαίρετες εισαγωγές , διαγραφές στοιχείων και παρακολουθεί τα κλειδιά υψηλότερης-προτεραιότητας.
- Είναι χρήσιμη, στις εφαρμογές όπου αποθηκεύεται μια ουρά διεργασιών με ποικίλες προτεραιότητες, και πάντα η επόμενη διεργασία που επεξεργάζεται είναι η σημαντικότερη.

Key-based Containers (PriorityQueue)

- PriorityQueue: Το interface για Priority Queues. Παρέχει μεθόδους για να προσπελάσει ή να αφαιρέσει το ζευγάρι κλειδί-στοιχείο με την πιό υψηλή προτεραιότητα και για να αλλάξει την προτεραιότητα ενός στοιχείου.
- ArrayHeap: αποδοτική υλοποίηση του PriorityQueue που χτίζεται επάνω σε έναν σωρό. Η εισαγωγή, η αφαίρεση, ή η αλλαγή του κλειδιού ενός ζευγαριού κλειδιού-στοιχείου παίρνουν λογαριθμικό χρόνο και η εξέταση του ζευγαριού κλειδιού-στοιχείου με το ελάχιστο κλειδί μπορεί να γίνει σε σταθερό χρόνο.

Demo

```
1
2   import jdsl.core.api.*;
3   import jdsl.core.ref.*;
4   import jdsl.core.algo.sorts.*;
5
6
7   public class SequenceExample {
8
9       private Sequence seq_;
10
11
12      public SequenceExample() { seq_ = new ArraySequence(); }
15
16      private void populateSequence() {
17          Position first = this.seq_.insertFirst("First");
18          Position second = this.seq_.insertLast("Third");
19          Position third = this.seq_.insertLast("Second");
20      }
21
22      private void sortSequence() {
23          SortObject sorter = new ArrayQuickSort();
24          sorter.sort(this.seq_, new ComparableComparator());
25      }
26
27      private void reverseSortSequence() {
28          SortObject sorter = new ArrayQuickSort();
29          sorter.sort(this.seq_, new ComparatorReverser(new ComparableComparator()));
30      }
31
32      private void printSequence() { System.out.println(this.seq_.toString()); }
35
36      public static void main(String args[]) {
37          SequenceExample s = new SequenceExample();
38          s.populateSequence();
39          s.printSequence();
40          s.sortSequence();
41          s.printSequence();
42          s.reverseSortSequence();
43          s.printSequence();
44
45      }
46
47
48  }
```

```
[ First, Third, Second ]
[ First, Second, Third ]
[ Third, Second, First ]
```

```java
import jdsl.core.api.*;
public class PriorityQueueExample {

    private PriorityQueue pq_;

    public PriorityQueueExample() { pq_ = new jdsl.core.ref.ArrayHeap(new jdsl.core.ref.IntegerComparator()); }

    public void populatePriorityQueue(){

        Object key = 1;
        Object element = "client1";
        Locator client1 = pq_.insert(key, element);

        key = 2;
        element = "client2";
        Locator client2 = pq_.insert(key, element);

        key = 3;
        element = "client3";
        Locator client3 = pq_.insert(key, element);
    }

    public void printPriorityQueue() { System.out.println(pq_.toString()); }

    public void printPriorityQueueWithIterator(){

        ObjectIterator keysIterator = pq_.keys();
        while(keysIterator.hasNext())
            System.out.println(keysIterator.nextObject());

        ObjectIterator elemIterator = pq_.elements();
        while(elemIterator.hasNext())
            System.out.println(elemIterator.nextObject());
    }

    public void printMinOfPriorityQueue()
    {
        System.out.println(pq_.min());
    }

    public void deleteClientWithHighestPriority(){
        pq_.removeMin();}

    public static void main(String args[]){
        PriorityQueueExample m = new PriorityQueueExample();
        m.populatePriorityQueue();
        m.printPriorityQueue();
        m.printPriorityQueueWithIterator();
        m.printMinOfPriorityQueue();
        m.deleteClientWithHighestPriority();
        m.printMinOfPriorityQueue();
    }
}
```

```
{ 1=client1, 2=client2, 3=client3 }
1
2
3
client1
client2
client3
Locator with key 1 = client1
Locator with key 2 = client2
```

References

- http://cs.brown.edu/cgc/jdsl/

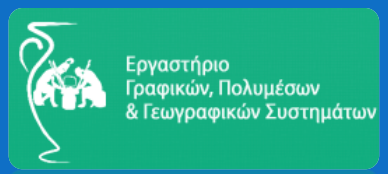
Ευχαριστώ!

?

Red-black δέντρα

Εκτενείς Δομές Δεδομένων (Κεφ. 5)

Δομές Δεδομένων



Περιεχόμενα

Εισαγωγή

Η δομή που θα παρουσιαστεί έχει τις εξής ιδιότητες:

- Είναι πλήρες δυαδικό δέντρο
- Υψοζυγισμένο δέντρο (& η ζύγιση επιτυγχάνεται με τη χρήση ενός μόνο bit)
- Αποθηκεύει δεδομένα μόνο στους εσωτερικούς κόμβους και όχι στα φύλλα

Ορισμός red-black tree

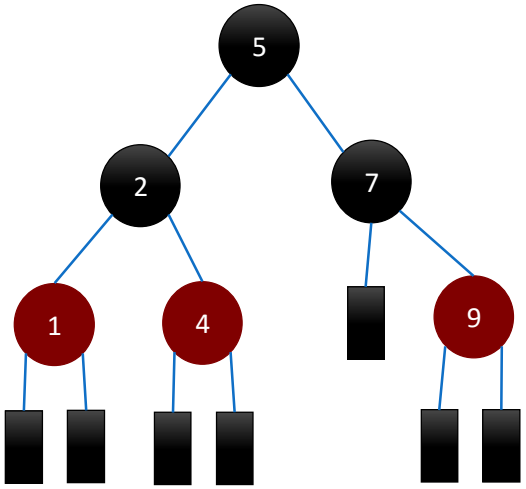
Ένα πλήρες δυαδικό δέντρο είναι red-black δέντρο όταν κάθε κόμβος είναι είτε κόκκινος είτε μαύρος και ικανοποιούνται οι εξής περιορισμοί:

Π1. Η ρίζα είναι μαύρη

Π2. Τα φύλλα είναι μαύρα

Π3. Κάθε μονοπάτι από τη ρίζα ως τα φύλλα έχει τον ίδιο αριθμό μαύρων κόμβων

Π4. Κάθε κόκκινος κόμβος έχει μαύρο πατέρα



Αναζήτηση (Access)

Για την αναζήτηση του στοιχείου x σε ένα red-black tree ακολουθούνται τα εξής βήματα:

1. Ξεκίνα από τη ρίζα.
2. Για κάθε κόμβο v:
    1. Αν x<v επισκέψου το αριστερό παιδί του v
    2. Αν x>v επισκέψου το δεξιό παιδί του v
    3. Αν x=v επέστρεψε

Ένθεση (Insertion)

Για την εισαγωγή του στοιχείου x σε ένα red-black tree ακολουθούνται τα εξής βήματα:

1. Εύρεση του κατάλληλου φύλλου y στο δέντρο για την εισαγωγή της τιμής x (Κλήση της Access(x)).
2. Αντικατάσταση του y με έναν κόκκινο κόμβο με τιμή x και 2 μαύρα παιδιά-φύλλα.
3. Εφαρμογή διάφορων μετασχηματισμών αναδρομικά από την ρίζα στον κόμβο x έτσι ώστε να ικανοποιούνται οι περιορισμοί.

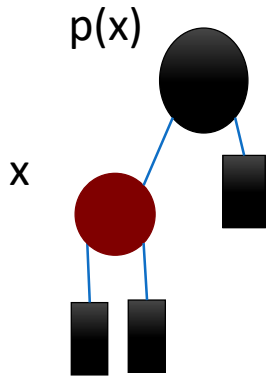
Ένθεση (Insertion)

Περίπτωση 1: Το x είναι η ρίζα. *[Καταπάτηση Π1]*

Λύση: Χρωματισμός της ρίζας. [Τερματική περίπτωση]

Ένθεση (Insertion)

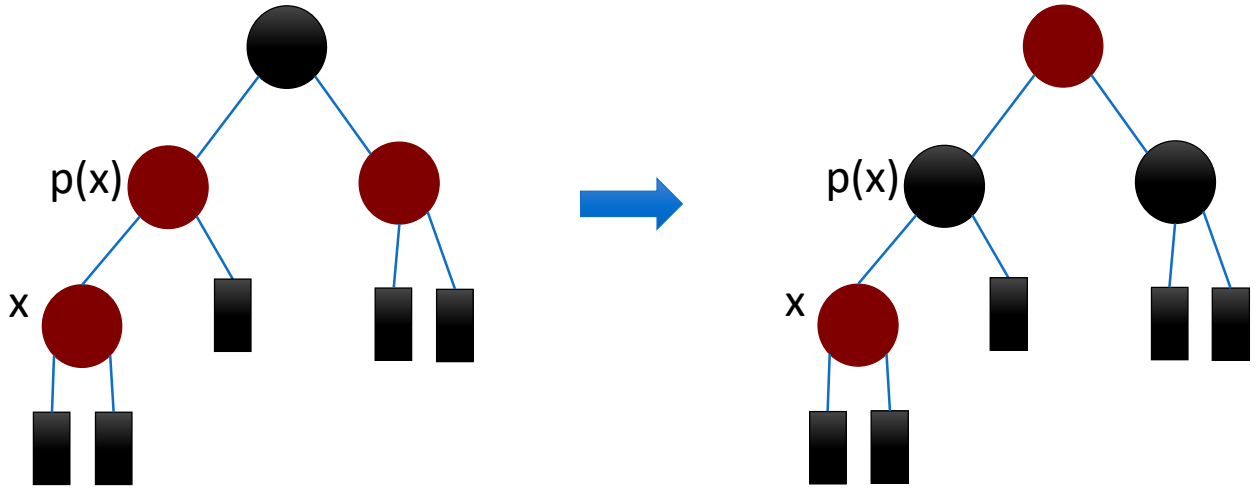
Περίπτωση 2: Ο πατέρας του x, p(x) είναι μαύρος. *[Καμμία Καταπάτηση]*



Ένθεση (Insertion)

Περίπτωση 3: Ο πατέρας του x, p(x) και ο αδερφός του p(x) είναι και οι 2 κόκκινοι. *[Καταπάτηση Π4]*

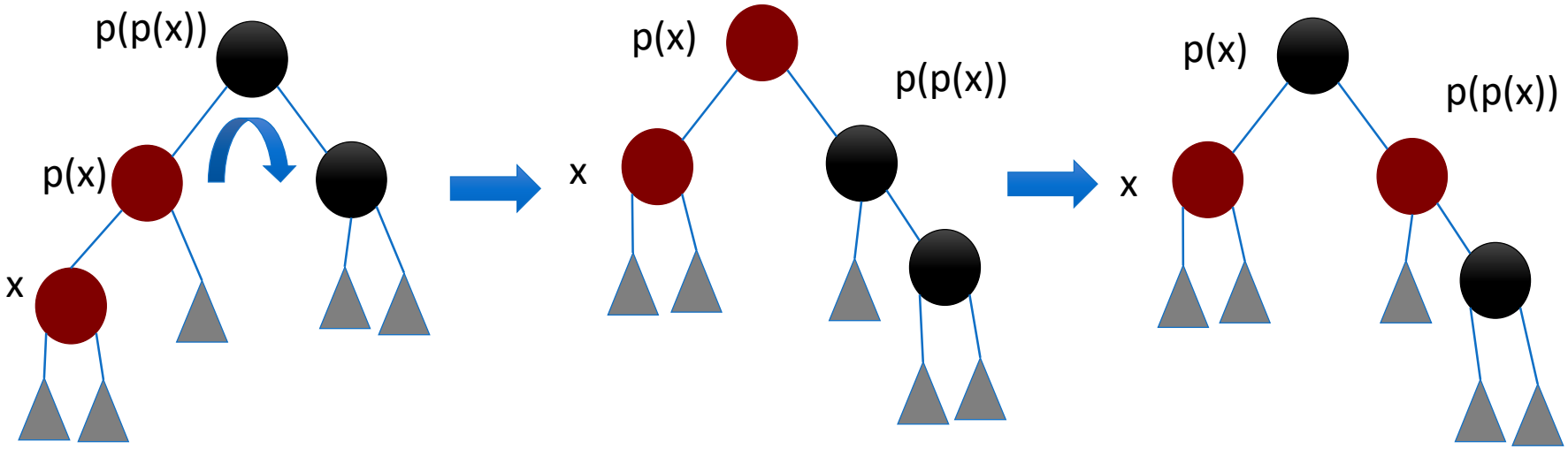
Λύση: Εναλλαγή χρωμάτων. [Μη τερματική περίπτωση]



Ένθεση (Insertion)

Περίπτωση 4: Ο πατέρας του x, p(x) είναι κόκκινος, ο αδερφός του p(x) είναι μαύρος και οι x και p(x) είναι και οι 2 παιδιά του ίδιου είδους (δηλαδή είναι και τα 2 αριστερά ή και τα 2 δεξιά παιδιά του πατέρα τους). *[Καταπάτηση Π4]*

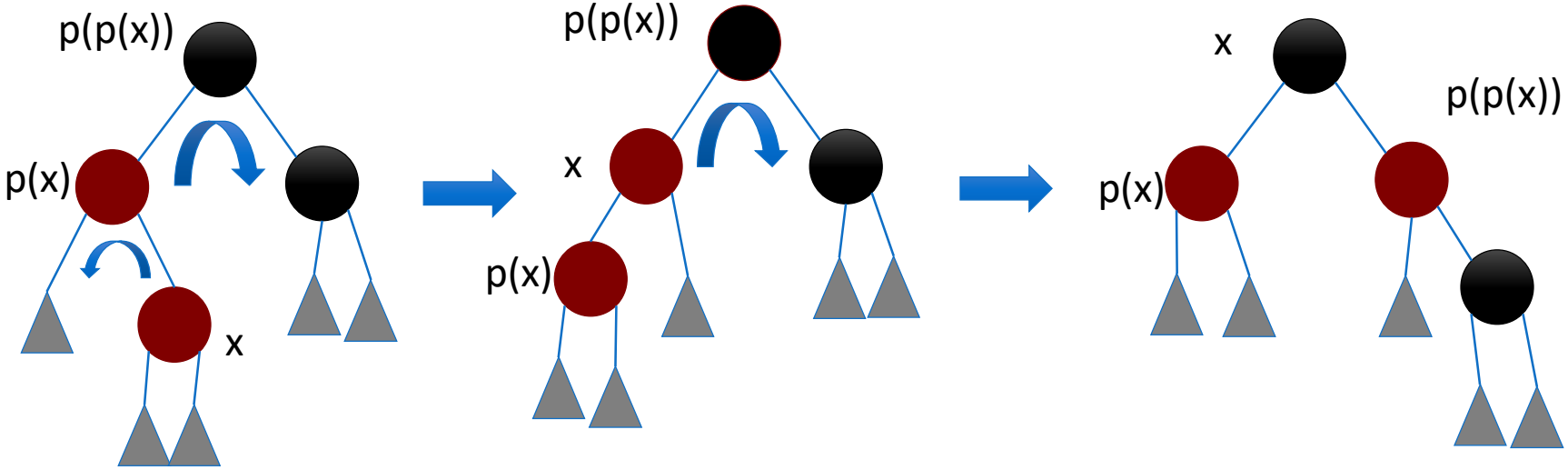
Λύση: Απλή περιστροφή. [Τερματική περίπτωση]



Ένθεση (Insertion)

Περίπτωση 5: Ο πατέρας του x, p(x) είναι κόκκινος, ο αδερφός του p(x) είναι μαύρος και οι x και p(x) είναι και οι 2 παιδιά διαφορετικού είδους (δηλαδή ο ένας είναι αριστερό και ο άλλος δεξιό παιδί του πατέρα τους). *[Καταπάτηση Π4]*

Λύση: Διπλή περιστροφή. [Τερματική περίπτωση]



Άσκηση (Insertion)

Να αποθηκευτούν οι τιμές 1,2,5,3,4 σε ένα red-black tree.

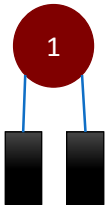
1. Insert(1)
2. Insert(2)
3. Insert(5)
4. Insert(3)
5. Insert(4)

Άσκηση (Insertion)

1) Insert(1)

Άσκηση (Insertion)

1) Insert(1)

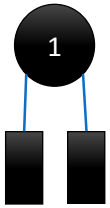


Άσκηση (Insertion)

1) Insert(1)

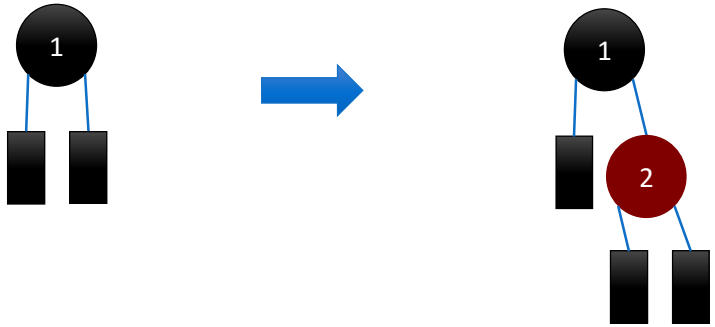
Άσκηση (Insertion)

2) Insert(2)



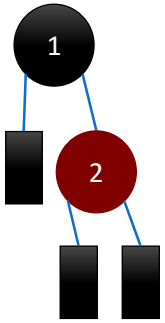
Άσκηση (Insertion)

2) Insert(2)



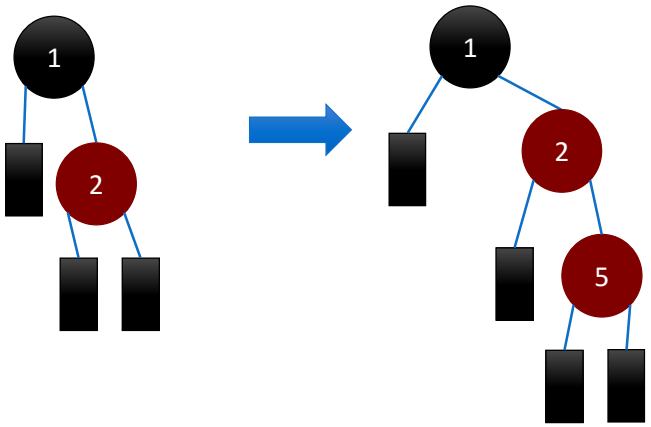
Άσκηση (Insertion)

3) Insert(5)



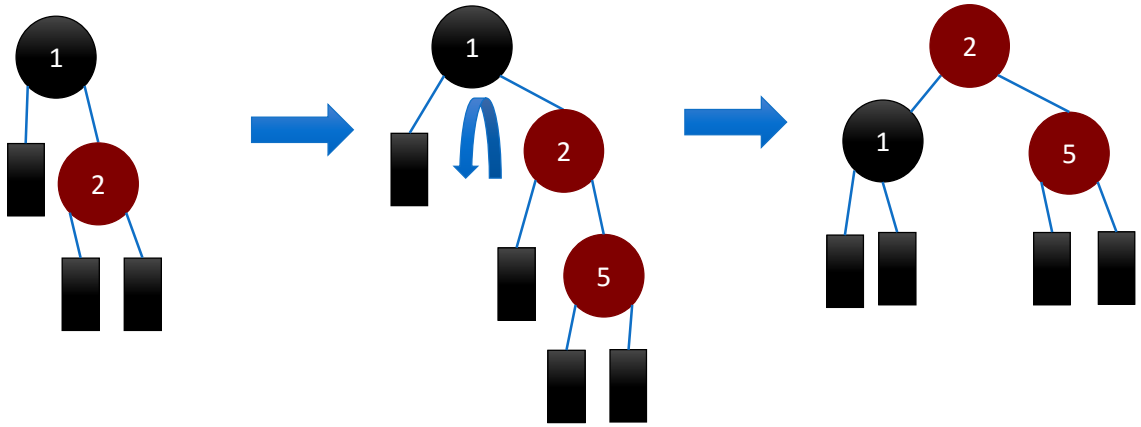
Άσκηση (Insertion)

3) Insert(5)



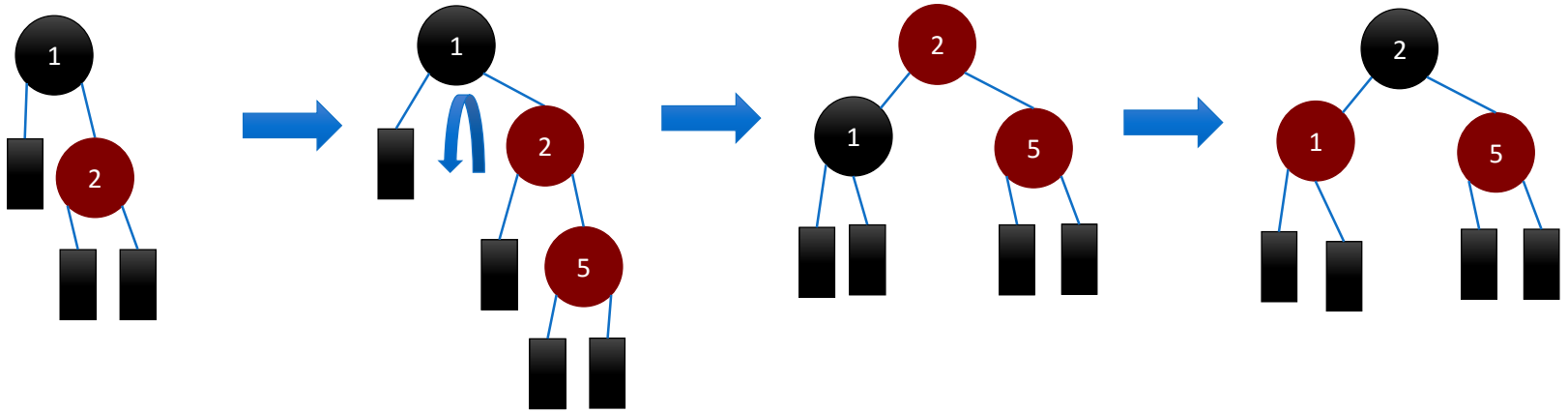
Άσκηση (Insertion)

3) Insert(5)



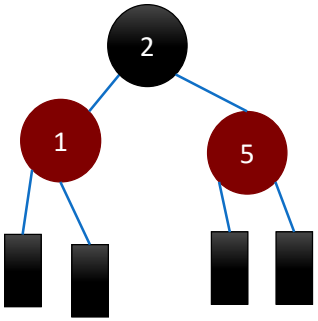
Άσκηση (Insertion)

3) Insert(5)



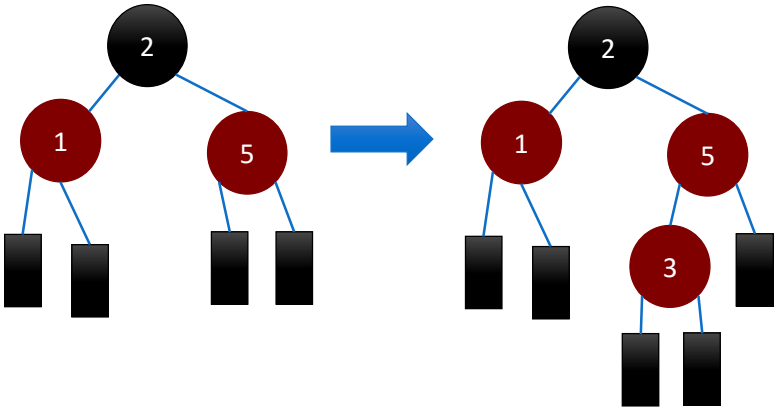
Άσκηση (Insertion)

4) Insert(3)



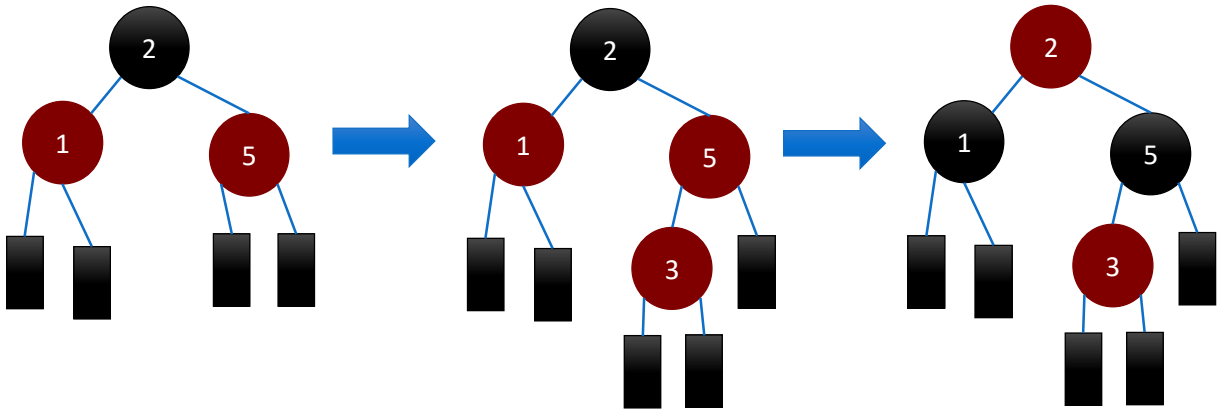
Άσκηση (Insertion)

4) Insert(3)



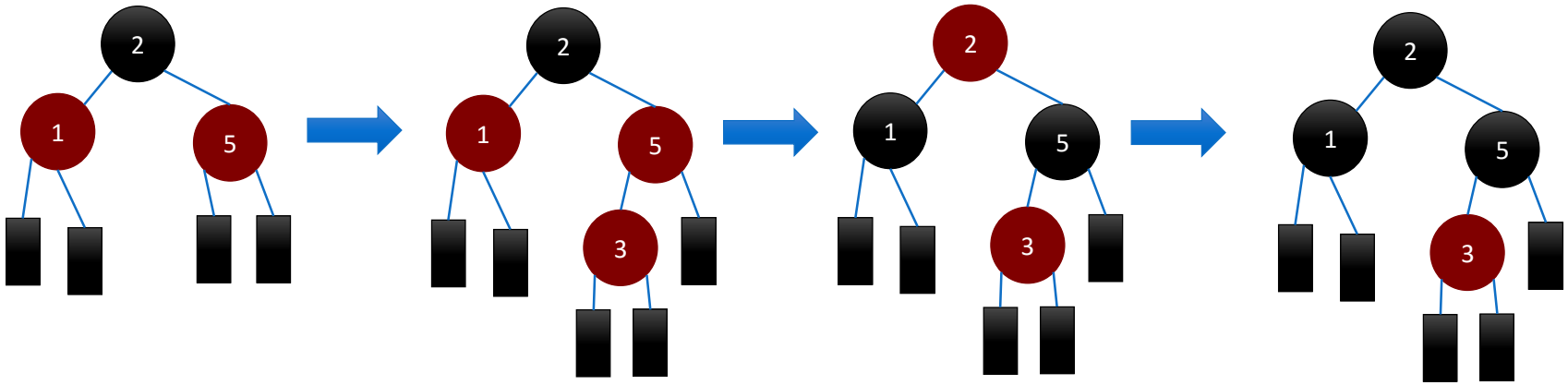
Άσκηση (Insertion)

4) Insert(3)



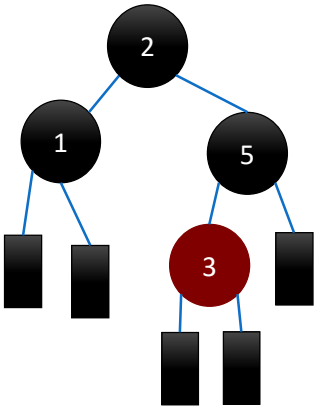
Άσκηση (Insertion)

4) Insert(3)



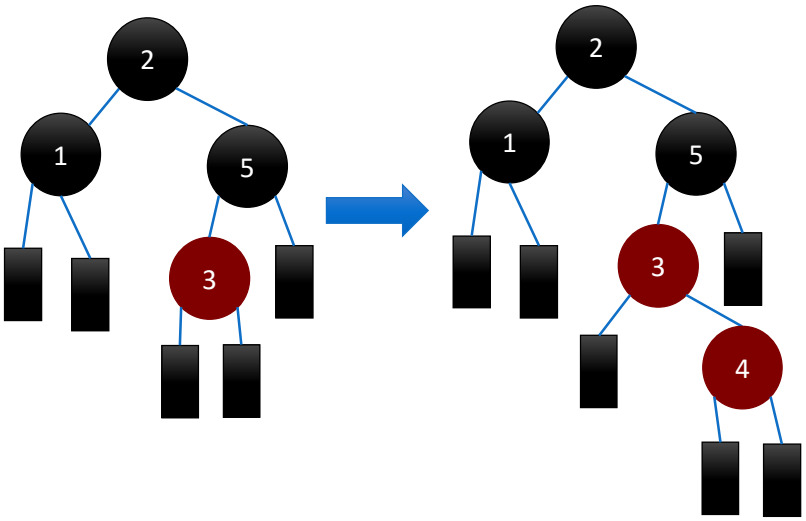
Άσκηση (Insertion)

5) Insert(4)



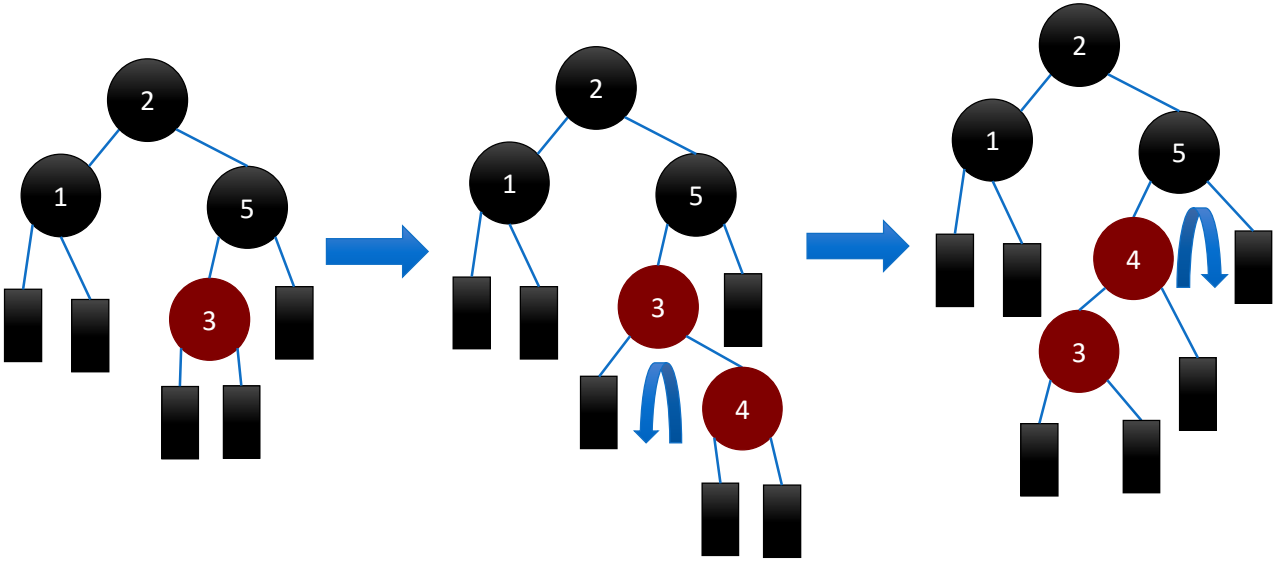
Άσκηση (Insertion)

5) Insert(4)



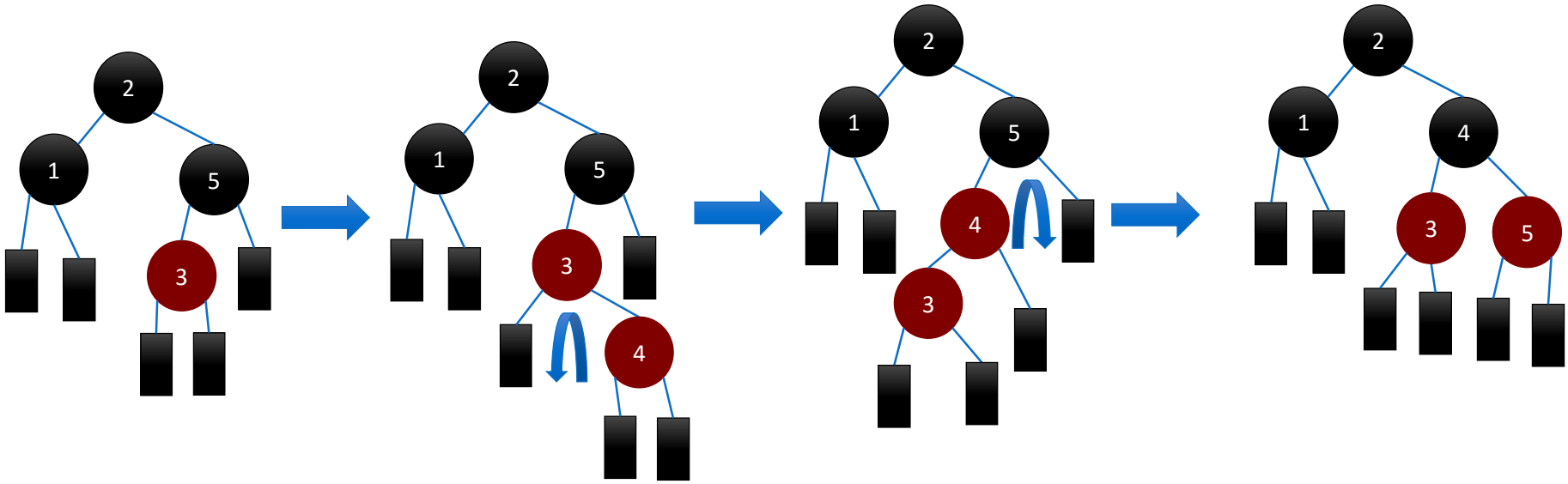
Άσκηση (Insertion)

5) Insert(4)



Άσκηση (Insertion)

5) Insert(4)



Απόσβεση (Deletion)

Η διαγραφή είναι μια πιο πολύπλοκη διαδικασία από την ένθεση.

Για την διαγραφή του στοιχείου x από ένα red-black tree ακολουθούνται τα εξής βήματα:

1. Εύρεση της κατάλληλης θέσης στο δέντρο για την απόσβεση της τιμής x (Κλήση της Access(x)).
2. Έστω v ο πατέρας του x και y ο αδερφός του. Αντικατάσταση του v με τον y και απόσβεση του v και του x από το δέντρο.
3. Εφαρμογή διάφορων μετασχηματισμών αναδρομικά από την ρίζα στον κόμβο x έτσι ώστε να ικανοποιούνται οι περιορισμοί.

Απόσβεση (Deletion)

Οι πιθανοί μετασχηματισμοί που μπορούν να γίνουν είναι:

1. Εναλλαγές χρώματος
    1. Μη τερματική
    2. Τερματική
2. Περιστροφές
    1. Μη τερματική απλή περιστροφή
    2. Τερματική διπλή περιστροφή
    3. Τερματική απλή περιστροφή

Αρχικά εφαρμόζονται οι μη τερματικοί μετασχηματισμοί επαναληπτικά, μέχρι να μπορεί να εφαρμοστεί κάποιος τερματικός μετασχηματισμός.

Απόδοση Red-Black tree

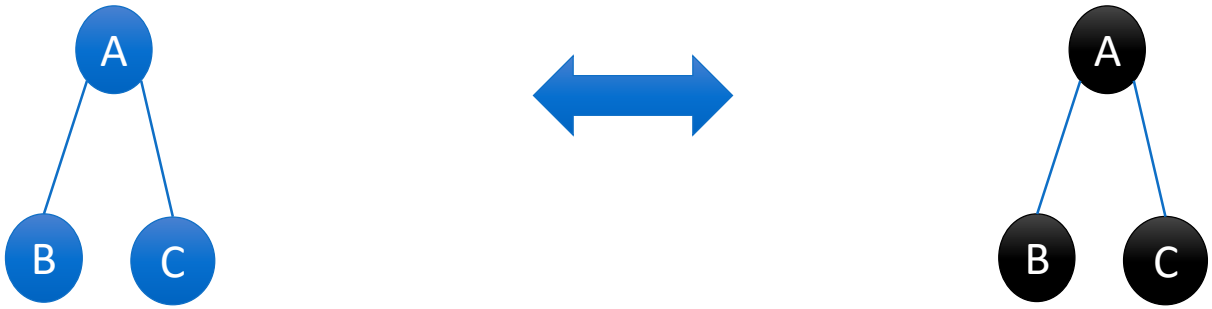
- Ύψος red-black δέντρου: **Θ(logn)**
- Για κάθε ένθεση ή απόσβεση απατούνται:
  - Πράξεις εναλλαγής χρωμάτων: **O(logn)**
  - Περιστροφές: **O(1)**
- Κόστος για Access, Insert, Delete: **O(logn)**

Ισοδυναμία red-black δέντων και (2,4) δέντρων

Από οποιοδήποτε (2,4) δέντρο μπορεί να προκύψει ένα ισοδύναμο red-black δέντρο (και αντίστροφα) εφαρμόζοντας κάποιους απλούς μετασχηματισμούς.

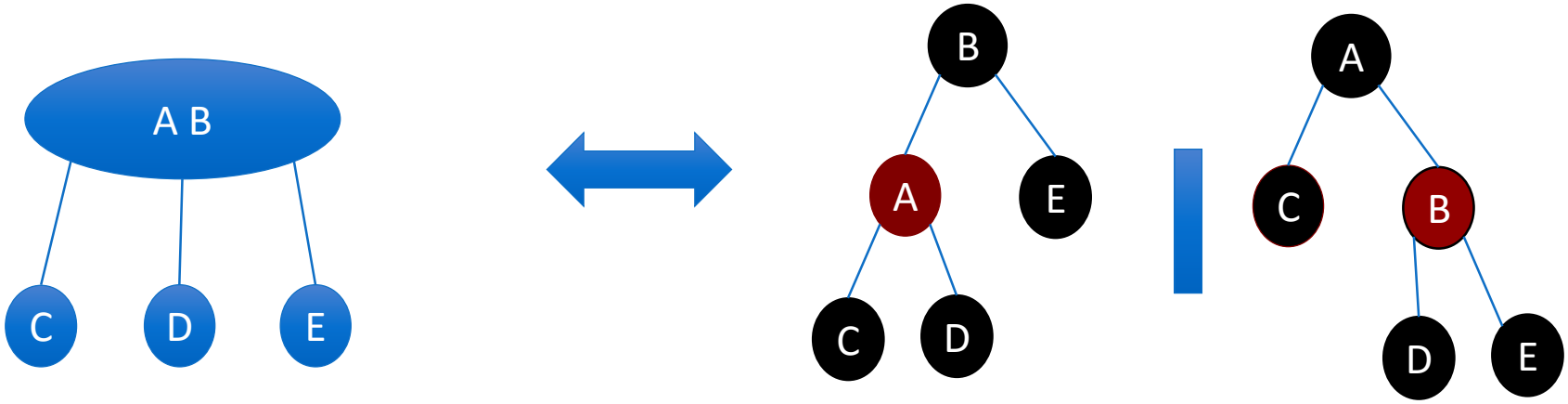
Ισοδυναμία red-black δέντων και (2,4) δέντρων

Μετασχηματισμός κόμβου με 2 παιδιά



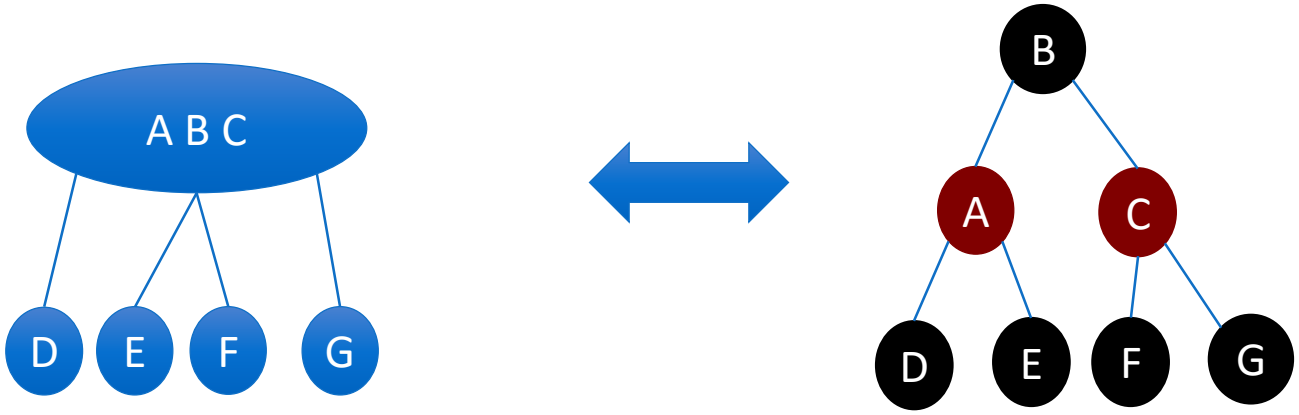
Ισοδυναμία red-black δέντων και (2,4) δέντρων

Μετασχηματισμός κόμβου με 3 παιδιά



Ισοδυναμία red-black δέντρων και (2,4) δέντρων

Μετασχηματισμός κόμβου με 4 παιδιά



References

- http://en.wikipedia.org/wiki/Red%E2%80%93black_tree
- https://www.cs.usfca.edu/~galles/visualization/RedBlack.html

Ευχαριστώ!

?

AVL-trees

C++ implementation

Δομές Δεδομένων

Μάριος Κενδέα

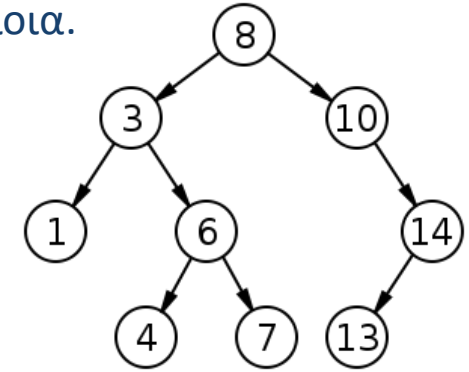
Εργαστήριο
Γραφικών, Πολυμέσων
& Γεωγραφικών Συστημάτων

31 Μαρτίου 2015

kendea@ceid.upatras.gr

Εισαγωγή (1/3)

- **Δυαδικά Δένδρα Αναζήτησης**: πολύ καλή δομή δεδομένων για την υλοποίηση maps, sets, και άλλα παρόμοια.



- Κύρια δυσκολία : είναι αποδοτικά μόνο όταν είναι ισοζυγισμένα
- Ορίζονται σχέσεις μεταξύ κόμβων
  - Parent: ο γονεϊκός κόμβος του κόμβου αναφοράς
  - LSON: το αριστερό παιδί του κόμβου αναφοράς
  - RSON: το δεξί παιδί του κόμβου αναφοράς
  - Ancestor: κάποιος κόμβος πρόγονος του κόμβου αναφοράς

Εισαγωγή (2/3)

- Ύψος δέντρου: για ένα δέντρο με n στοιχεία χρειάζονται O(logn) επίπεδα.
- **Το ύψος του δέντρου καθορίζει και το κόστος αναζήτησης γιατί έχουμε μια σύγκριση σε κάθε επίπεδο**
- **Ιδιότητα δυαδικού δέντρου αναζήτησης:**

Για να είναι ένα δέντρο δυαδικό δέντρο αναζήτησης (Binary Search Tree – BST) πρέπει να πληροί την ιδιότητα δυαδικού δέντρου αναζήτησης:

Έστω n κόμβος ενός δυαδικού δέντρου αναζήτησης. Τότε όλοι οι κόμβοι που βρίσκονται στο αριστερό υποδέντρο του n έχουν τιμή περιεχομένου μικρότερη ή ίση με το περιεχόμενο του n. Αντίστοιχα, οι κόμβοι που βρίσκονται στο δεξιό υποδέντρο του n έχουν τιμή περιεχομένου μεγαλύτερη ή ίση από το περιεχόμενο του n.

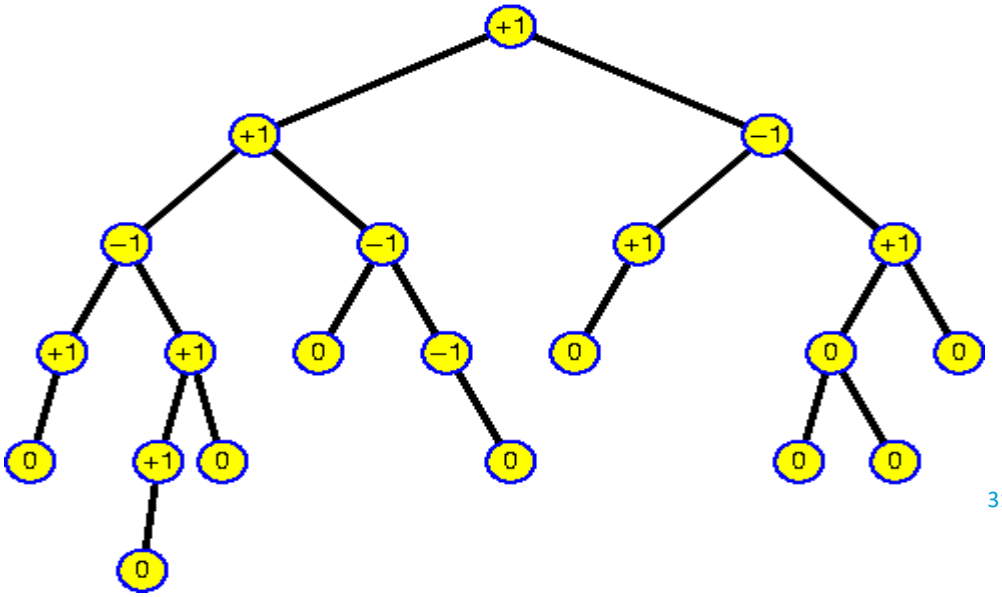
- **Έτσι το δυαδικό δέντρο αναζήτησης εγγυάται χρόνο αναζήτησης O(logn) με μια σύγκριση σε κάθε ένα από τα O(logn) επίπεδα. <u>Στη συνέχεια</u> η ένθεση (insert) και η διαγραφή απαιτούν σταθερό χρόνο.**

Εισαγωγή (3/3)

- Τα ισοζυγισμένα δέντρα απαντούν στο πρόβλημα του ύψους χειρότερης περίπτωσης ενός δυαδικού δέντρου αναζήτησης
  - Διατηρούν μια αναλογία μεταξύ του αριστερού και του δεξιού υποδέντρου σε κάθε κόμβο ώστε το συνολικό ύψος να είναι πολύ κοντά στο logn
  - Ισοζυγισμένα με βάρη (weight-balanced) και ισοζυγισμένα με ύψος (height – balanced)
- **Το AVL tree είναι ένα ισοζυγισμένο δέντρο βάσει ύψους (ισοσκελισμένο) που σε κάθε κόμβο του, το ύψος του αριστερού υποδέντρου επιτρέπεται να διαφέρει μόνο κατά 1 από το ύψος του δεξιού υποδέντρου.**

AVL Trees (1/8)

- Τα AVL trees, είναι μια απλή και αποδοτική δομή δεδομένων για την διατήρηση της ισορροπίας.
- Είναι η πρώτη που προτάθηκε (“An algorithm for the organisation of information” - Proceedings of the USSR Academy of Sciences 146: 263–266, 1962) από τους G.M. Adelson-Velskii και E.M. Landis

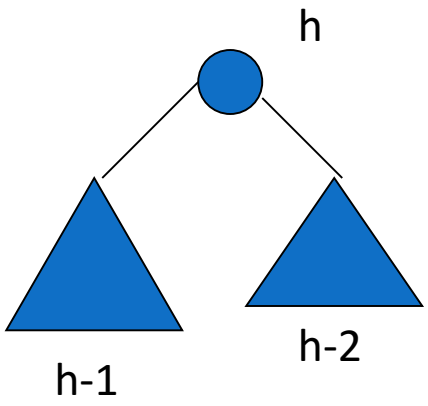


AVL Trees (2/8)

- Σε κάθε κόμβο του δέντρου αντιστοιχεί ένας αριθμός που δείχνει τον παράγοντα ισοσκελισμού.
  - Αν hleft είναι το ύψος του αριστερού υποδέντρου του κόμβου v και hright το ύψος του δεξιού υποδέντρου, τότε ο παράγοντας ισοσκελισμού υπολογίζεται από την πράξη hright - hleft
  - Εφόσον θέλουμε η διαφορά ύψους των υποδέντρων θέλουμε να είναι το πολύ 1, επιτρεπτές τιμές είναι τα -1, 0, 1.
  - Το AVL tree εγγυάται ύψος δέντρου h τέτοιο ώστε

$$\log n <= h < 1.44\log(n+2) -1$$

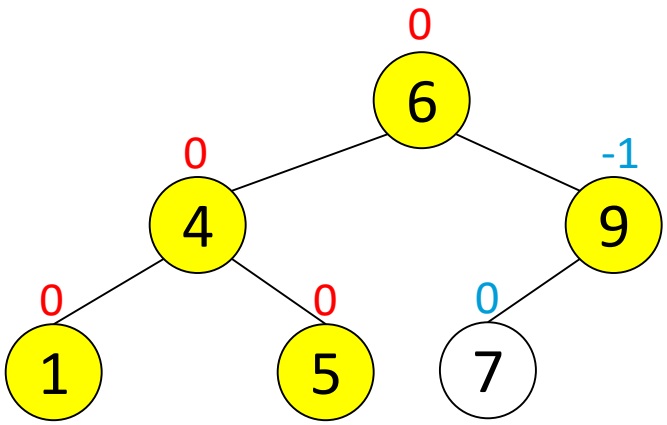
Άρα επί της ουσίας O(logn)!



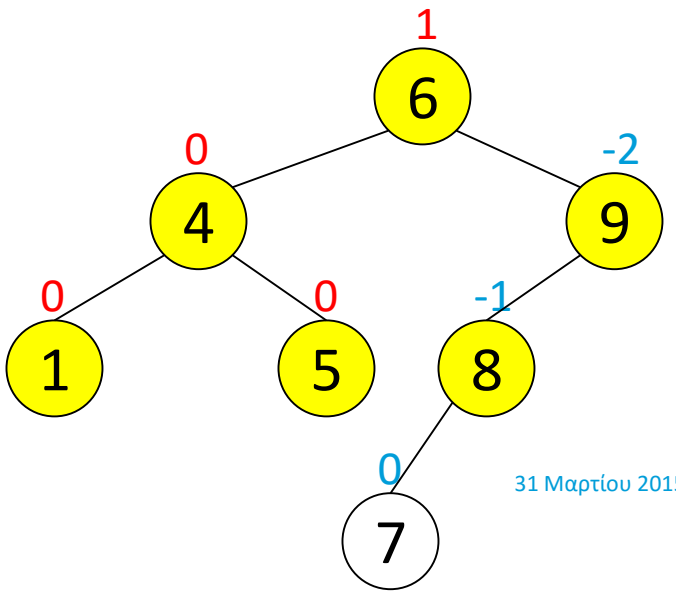
AVL Trees (3/8)
Insert

- Η εισαγωγή γίνεται κανονικά με εύρεση του κατάλληλου σημείου για ένθεση και στη συνέχεια κατασκευή νέου κόμβου στο σημείο αυτό.
  - Πολλές φορές χαλάει η ισορρόπία του AVL tree και εμφανίζονται βάρη 2 ή -2.
  - Για να αποκατασταθεί ο ισοσκελισμός και η ιδιότητα των AVL δέντρων πρέπει να γίνουν κάποιες πράξεις που ονομάζονται περιστρόφές

Tree A (AVL)



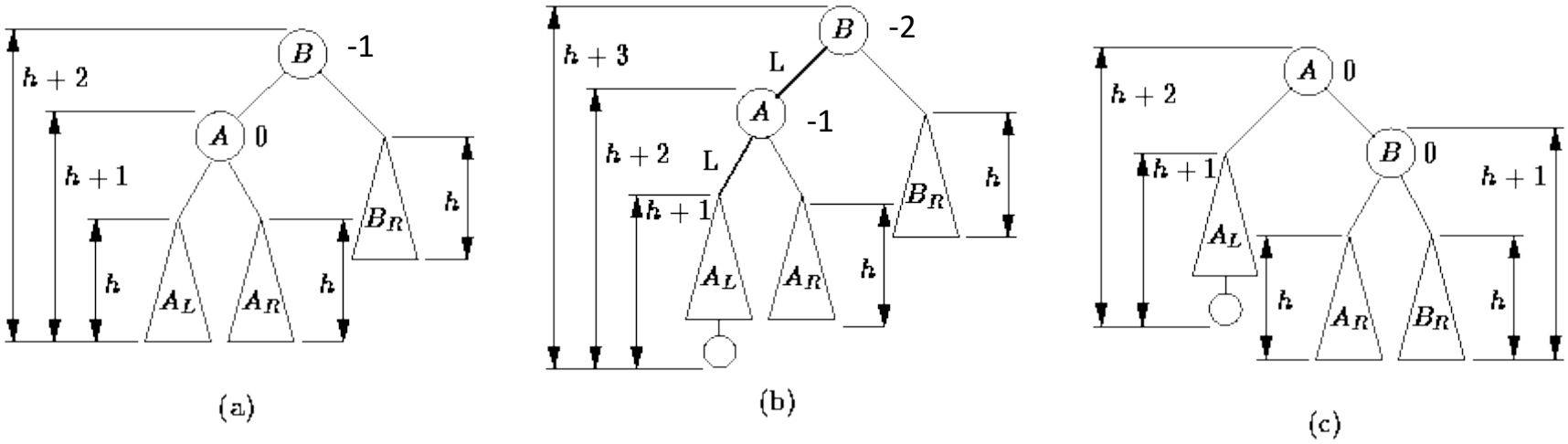
Tree B (not AVL)



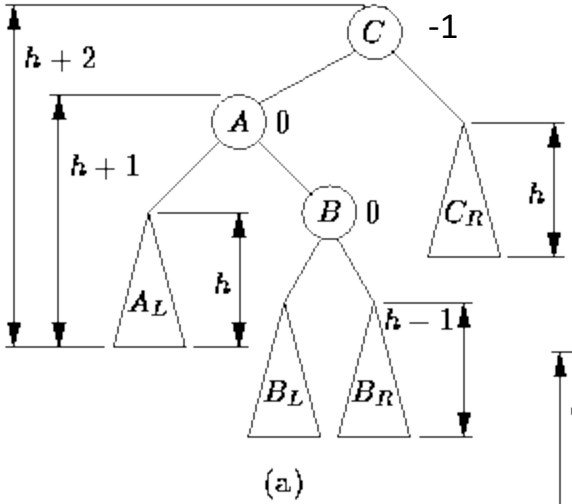
AVL Trees (4/8)
Insert

- Μόνο κόμβοι στο path από το σημείο εισαγωγής προς τον κόμβο της ρίζας μπορεί να αλλάξουν ύψος
- Οπότε, πάμε προς τα πάνω , διορθώνοντας τα ύψη
- $(h_{right}-h_{left})$ είναι 2 ή –2, κάνουμε περιστροφή γύρο από τον κόμβο
- Αν ο κόμβος που χρειάζεται περιστροφή είναι ο α.
- Υπάρχουν 4 περιπτώσεις:
  - Εξωτερικές περιπτώσεις (απαιτεί απλή περιστροφή) :
    1. Εισαγωγή στο αριστερό υποδέντρο του αριστερού παιδιού του α.
    2. Εισαγωγή στο δεξί υποδέντρο του δεξιού παιδιού του α.
  - Εσωτερικές περιπτώσεις (απαιτεί διπλή περιστροφή) :
    3. Εισαγωγή στο δεξί υποδέντρο του αριστερού παιδιού του α.
    4. Εισαγωγή στο αριστερό υποδέντρο του δεξιού παιδιού του α.
- Με αντίστοιχο τρόπο λειτουργεί και η διαγραφή: αναζήτηση, διαγραφή, εξισορρόπηση.

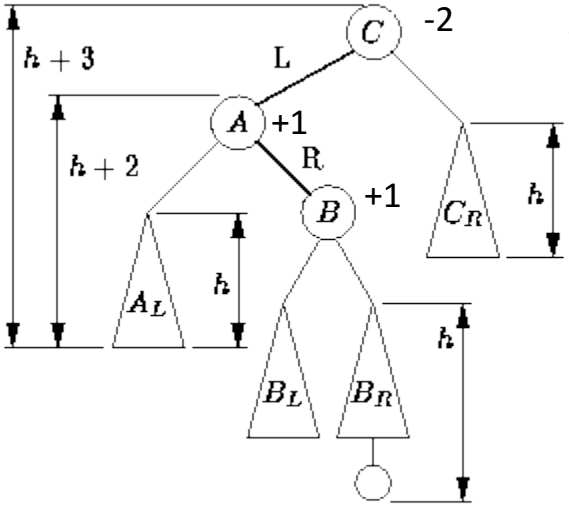
AVL Trees (5/8)
Απλή περιστροφή



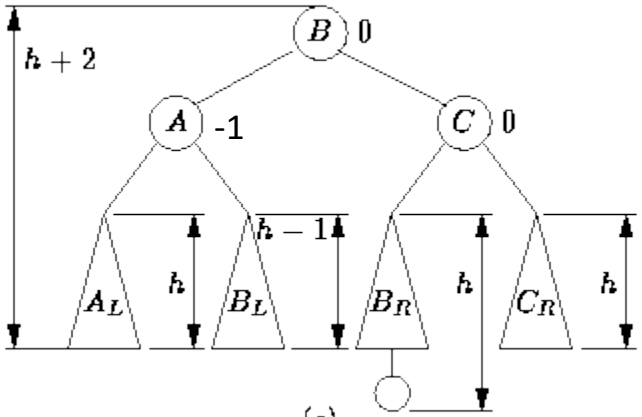
AVL Trees (6/8)
Διπλή περιστροφή



(a)



(b)



(c)

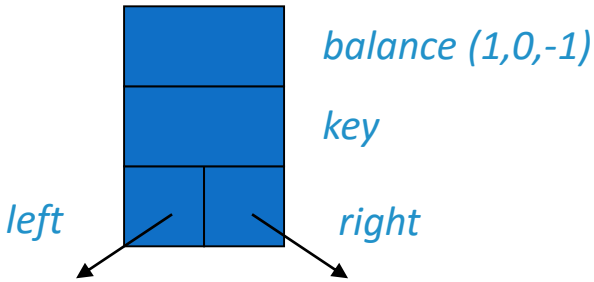
AVL Trees (7/8)
Περιστροφές

AVL Trees (8/8)
ΨΕΥΔΟΚΩΔΙΚΑΣ ΕΠΙΛΟΓΗΣ ΠΕΡΙΣΤΡΟΦΗΣ

```
IF tree is right heavy {
    IF tree's right subtree is left heavy
        Perform Double Left rotation
    ELSE
        Perform Single Left rotation
}
ELSE IF tree is left heavy {
    IF tree's left subtree is right heavy
        Perform Double Right rotation
    ELSE
        Perform Single Right rotation
}
```

AVL Trees & C++ (1/9)

- **Αρχικά, χρειαζόμαστε μια κλάση για τους κόμβους με βάση την οποία θα δομηθεί το δέντρο:**




```
template <class KeyType>
 class AvlNode {
 private:
      Comparable<KeyType> * myData; // Data field
      AvlNode<KeyType> * mySubtree[2]; // Subtree pointers
      short myBal; // Balance factor

      // ... many details omitted
};
```

- Παρατηρούμε ότι περιέχει ένα πίνακα για τα δύο παιδιά του κόμβου και μια μεταβλητή για τον παράγοντα ισοσκελισμού.
  - Η κλάση αυτή μας αρκεί για το δέντρο!

AVL Trees & C++ (2/9)

- **Ορίζουμε τα βοηθητικά στοιχεία σε μια κλάση Comparable**

```
enum cmp_t {
      MIN_CMP = -1, // less than
      EQ_CMP = 0, // equal to
      MAX_CMP = 1 // greater than
};
template <class KeyType>
  class Comparable {
  private:
      KeyType myKey;

  public:
      Comparable(KeyType key) : myKey(key) {};
      cmp_t Compare(KeyType key) const;
      KeyType Key() const { return myKey; }
};
```

AVL Trees & C++ (3/9)

- **Κώδικας για την αριστερή περιστροφή:**

```
enum dir_t { LEFT = 0, RIGHT = 1 };
template <class KeyType>
 void
 AvlNode<KeyType>::RotateLeft(AvlNode<KeyType> * & root) {
        AvlNode<KeyType> * oldRoot = root;
        root = root->mySubtree[RIGHT];
        oldRoot->mySubtree[RIGHT] = root->mySubtree[LEFT];
        root->mySubtree[LEFT] = oldRoot;
        // update balances
        oldRoot->myBal -= (1 + MAX(root->myBal, 0));
        root->myBal -= (1 - MIN(oldRoot->myBal, 0));
 }
```

AVL Trees & C++ (4/9)

- **Κώδικας για την δεξιά περιστροφή:**

```
template <class KeyType>
 void
 AvlNode<KeyType>::RotateRight(AvlNode<KeyType> * & root) {
        AvlNode<KeyType> * oldRoot = root;
        // perform rotation
        root = root->mySubtree[LEFT];
        oldRoot->mySubtree[LEFT] = root->mySubtree[RIGHT];
        root->mySubtree[RIGHT] = oldRoot;
        // update balances
        oldRoot->myBal += (1 - MIN(root->myBal, 0));
        root->myBal += (1 + MAX(oldRoot->myBal, 0));
 }
```

AVL Trees & C++ (5/9)

- **Ένωση των δύο προηγούμενων συναρτήσεων σε μια:**

```
template <class KeyType>
 void
 AvlNode<KeyType>::RotateOnce(AvlNode<KeyType> * & root, dir_t dir) {
      AvlNode<KeyType> * oldRoot = root;
      dir_t otherDir = Opposite(dir);
      short factor = (RIGHT - LEFT) * (1 - (2 * dir));

      // rotate
      root = tree->mySubtree[otherDir];
      oldRoot->mySubtree[otherDir] = tree->mySubtree[dir];
      root->mySubtree[dir] = oldRoot;
      // update balances
      oldRoot->myBal -= factor * (1 + MAX(factor * root->myBal, 0));
      root->myBal += factor * (1 + MAX(factor * oldRoot->myBal, 0));
 }
```

AVL Trees & C++ (6/9)

- **Διπλή περιστροφή με χρήση της RotateOnce:**

```
template <class KeyType>
 void
 AvlNode<KeyType>::RotateTwice(AvlNode<KeyType> * & root, dir_t dir) {
        dir_t otherDir = Opposite(dir);
        RotateOnce(root->mySubtree[otherDir], otherDir);
        RotateOnce(root, dir);
 }
```

AVL Trees & C++ (7/9)

- **Μέθοδος για τη σύγκριση του περιεχομένου ενός κόμβου με το ζητούμενο:**

```
template <class KeyType>
  cmp_t
  AvlNode<KeyType>::Compare(KeyType key, cmp_t cmp) const {
      switch (cmp) {
            case EQ_CMP : // Standard comparison
                  return myData->Compare(key);
            case MIN_CMP : // Find the minimal element in this tree
                  return (mySubtree[LEFT] == NULL) ? EQ_CMP : MIN_CMP;
            case MAX_CMP : // Find the maximal element in this tree
                  return (mySubtree[RIGHT] == NULL) ? EQ_CMP : MAX_CMP;
            }
}
```

AVL Trees & C++ (8/9)

- **Μέθοδος insert:**

```
template <class KeyType>
  Comparable<KeyType> *
  AvlNode<KeyType>::Insert(Comparable<KeyType> * item, AvlNode<KeyType> * & root,
int & change) {
        // See if the tree is empty
        if (root == NULL) {
                // Insert new node here
                root = new AvlNode<KeyType>(item);
                change = HEIGHT_CHANGE;
                return NULL;
        }
        // Initialize
        Comparable<KeyType> * found = NULL;
        int increase = 0;

        // Compare items and determine which direction to search
        cmp_t result = root->Compare(item->Key());
        dir_t dir = (result == MIN_CMP) ? LEFT : RIGHT;
```

AVL Trees & C++ (9/9)

- **Μέθοδος insert (συνέχεια):**

```
  if (result != EQ_CMP) {
      // Insert into "dir" subtree
      found = Insert(item, root->mySubtree[dir], change);
      if (found)
            return found; // already here - dont insert
      increase = result * change; // set balance factor increment
  }
  else { // key already in tree at this node
      increase = HEIGHT_NOCHANGE; // 0
      return root->myData;
  }

  root->myBal += increase; // update balance factor
  change = (increase && root->myBal) ? (1 - ReBalance(root)) : HEIGHT_NOCHANGE;
  return NULL; // the key was successfully inserted
}
```

Αναφορές

- Ολοκληρωμένος κώδικας AVL tree σε C++:

http://www.cmcrossroads.com/bradapp/ftp/src/libs/C++/AvlTrees.html

- Βασικές Πληροφορίες για AVL tree:

http://en.wikipedia.org/wiki/AVL_tree

- AVL με παραδείγματα:

http://www.cs.ucf.edu/~reinhard/classes/cop3503/lectures/AVLTrees02.pdf

- Standard AVL C++

http://sourceforge.net/projects/standardavl/

- Animated AVL Tree Java applet

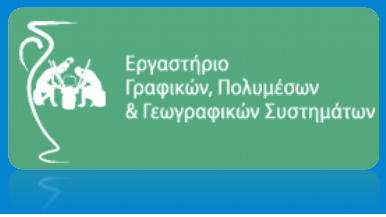
http://webpages.ull.es/users/jriera/Docencia/AVL/AVL%20tree%20applet.htm

?

Τεχνικές Επιμερισμένης Ανάλυσης

Δομές Δεδομένων

Μπαλτάς Αλέξανδρος



21 Απριλίου 2015

ampaltas@ceid.upatras.gr

Περιεχόμενα

Εισαγωγή

Γενικά στην ανάλυση Δομών Δεδομένων και Αλγορίθμων μας ενδιαφέρουν κυρίως 3 περιπτώσεις ως προς την Πολυπλοκότητα:

- Πολυπλοκότητα Χειρότερης Περίπτωσης [Worst Case Complexity]
- Πολυπλοκότητα Μέσης Περίπτωσης [Average Case Complexity]
- Επιμερισμένη Πολυπλοκότητα [Amortized Complexity]

Επιμερισμένη Πολυπλοκότητα

- Σε αρκετές περιπτώσεις η πολυπλοκότητα μιας πράξης μπορεί να έχει μεγάλες διακυμάνσεις.
- Επίσης σε κάποια προβλήματα θέλουμε να φράξουμε το κόστος μιας ακολουθίας πράξεων και όχι καθεμία πράξη ξεχωριστά.
- Στην ανάλυση επιμερισμένης πολυπλοκότητας υπολογίζεται το συνολικό κόστος μιας ακολουθίας πράξεων και επιμερίζεται το κόστος αυτό σε καθεμία πράξη.

Τεχινικές Επιμερισμένης Ανάλυσης

Παρουσιάζονται 3 τεχνικές επιμερισμένης ανάλυσης:

1. Μέθοδος Άθροισης
2. Μέθοδος Τραπεζίτη
3. Μέθοδος Φυσικού

Μέθοδος Άθροισης

- Ακριβής εφαρμογή του ορισμού της επιμερισμένης πολυπλοκότητας.
- Για μια ακολουθία πράξεων, καταγράφεται το ακριβές κόστος, υπολογίζεται το άθροισμα, και τέλος αυτό διαιρείται με τον αριθμό των πράξεων.

Μέθοδος Τραπεζίτη

- Η δομή που αναλύεται σχετίζεται με ένα ‘τραπεζικό λογαριασμό’.
- Κάθε πράξη έχει ένα πραγματικό κόστος, και παράλληλα με αυτό ‘καταθέτει’ ή ‘αποσύρει’ χρήματα από το λογαριασμό.
- Κάθε πράξη πληρώνει ένα ποσό. Συνήθως οι φθηνές πράξεις πληρώνουν παραπάνω από το πραγματικό τους κόστος, ενώ οι αριβές κάνουν ανάληψη, αρκεί να υπάρχει αρκετό υπόλοιπο στο λογαριασμό.
- Το επιμερισμένο κόστος δίνεται από το άθοισμα του πραγματικού κόστους της πράξης συν το ποσό που καταθέτει η αποσύρει από το λογαριασμό.

Μέθοδος Τραπεζίτη

Τυπικά:

- Δομή D
- Ακολουθία πράξεων O_1, O_2, …, O_m τέτοια ώστε η πράξη O_i να μετασχηματίζει τη δομή από τη φάση D_{i-1} σε D_i
- Συνάρτηση $Bal(D_i)$ που αναθέτει πραγματικές θετικές τιμές στα στιγμιότυπα της δομής, και $Bal(D_o) = 0$.
- Η επιμερισμένη πολυπλοκότητα AC(Oi) δίνεται ως εξής:

  $AC(O_i) = T(O_i) + Bal(D_i) – Bal(D_i\text{-}1)$, όπου $T(O_i)$ το πραγματικό κόστος της πράξης.

Μέθοδος Φυσικού

- Ισοδύναμη της Μεθόδου Τραπεζίτη
- Σε κάθε στιγμιότυπο της δομής ανατίθεται μια συνάρτηση που λαμβάνει πραγματικές θετικές τιμές και ονομάζεται συνάρτηση δυναμικού.
- Η συνάρτηση δυναμικού πρέπει να μετρά την ‘ανοχή’ της δομής σε ακριβές πράξεις, άρα όσο πιο μεγάλο το δυναμικό ενός στιγμιοτύπου μιας δομής, τόσο πιο ακριβή μπορεί να είναι μια πράξη σε αυτό.
- Οι φθηνές πράξεις πρέπει να αυξάνουν το δυναμικό, ενώ οι ακριβές να το μειώνουν.
- Το επιμερισμένο κόστος μιας πράξης είναι το άθροισμα του πραγματικού κόστους συν το ποσό του δυναμικού που συνεισφέρει ή καταναλώνει.

Μέθοδος Φυσικού

Τυπικά:

- Δομή D
- Ακολουθία πράξεων O_1, O_2, …, O_m τέτοια ώστε η πράξη O_i να μετασχηματίζει τη δομή από τη φάση D_{i-1} σε D_i
- Υπάρχει μια μη μειούμενη συνάρτηση Φ(Di) που αναθέτει πραγματιξκές τιμές στα σιγμυότυπα της δομής και $\Phi(D_o) = 0$.
- Η επιμερισμένη πολυπλοκότητα $AC(O_i)$ δίνεται ως εξής:

  $AC(O_i) = T(O_i) + \Phi(D_i) - \Phi(D_{i-1})$, όπου $T(O_i)$ το πραγματικό κόστος της πράξης.

Δυαδικός Μετρητής

- Δυαδικός μετρητής $c = < b_k, b_{k-1}, \ldots, b_1 >$.
- Μοναδική επιτρεπτή πράξη είναι η inc(c) δηλαδή η αύξηση του μετρητή κατά 1.
- Κάθε αλλαγή ψηφίου έχει κόστος 1.

Δυαδικός Μετρητής

Μέθοδος Άθροισης:

- Το ψηφίο b_0 αλλάζει σε κάθε αύξηση, το b_1 αλλάζει σε κάθε δεύτερη άυξηση, το b_3 αλλάζει σε κάθε τέταρτη άυξηση κλπ
- Κάθε αλλαγή κοστίζει O(1), άρα το b0 έχει κόστος O(n), το b_1 έχει O(n/2), το b_2 $O(n/2^2)$ κλπ
- Τελικά το κόστος για n αυξήσεις είναι:

  $C(n) = O(n/2^0)+O(n/2^1)+O(n/2^2)+ . ..= O(n)$

- Το επιμερισμένο κόστος είναι O(n)/n = **O(1)**

Δυαδικός Μετρητής

Μέθοδος Τραπεζίτη:

- Η πολιτική χρέωσης είναι η εξής:
  - Πληρώνουμε 1 μονάδα κάθε φορά που μετατρέπουμε ένα 0 σε 1
  - Κάνουμε ανάληψη 1 μονάδας κάθε φορά που μετατρέπουμε ένα 1 σε 0
- Σε μια τυχαία αύξηση i:
  - k διαδοχικά 1 θα γίνουν 0, και το k-sto ψηφίο θα γίνει από 0 σε 1.
  - Το πραγματικό κόστος είναι k+1, γίνεται ανάληψη k μονάδων, και κατάθεση 1 μονάδας
  - Άρα AC(incr$_i$) = T(incr$_i$) + Bal(i+1) – Bal(i) = k+1 – k+1 =2 = **O(1)**

Δυαδικός Μετρητής

Μέθοδος Φυσικού:

- Η συνάρτηση δυναμικού είναι Φ(μετρητή) = {πλήθος των ψηφίων με τιμή 1}.
- Σε μια τυχαία αύξηση i:
  - k διαδοχικά 1 θα γίνουν 0, και το k-sto ψηφίο θα γίνει από 0 σε 1.
  - Το πραγματικό κόστος είναι k+1, και ο συνολικός αριθμός των 1 μειώνεται κατά k-1 άρα η διαφορά δυναμικού ΔΦ = -k+1
  - Άρα AC(incri) = T(incri) + Φ(i+1) – Φ(i) = k+1 – k+1 =2 = **O(1)**

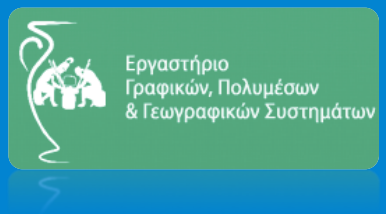
Ευχαριστώ!

?

B-Trees

Δομές Δεδομένων

Μπαλτάς Αλέξανδρος



21 Απριλίου 2015

ampaltas@ceid.upatras.gr

Περιεχόμενα

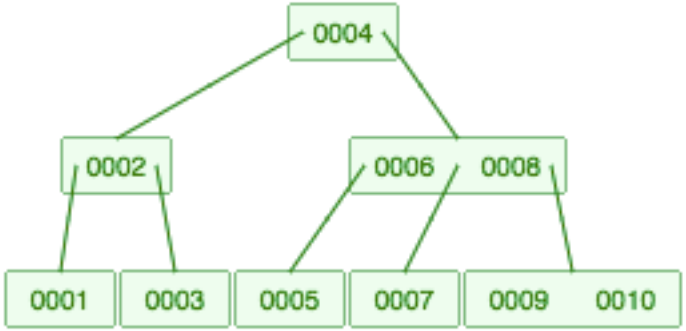
Εισαγωγή

B-tree:

- Δενδρική δομή
- Τα δεδομένα σε ένα B-tree αποθηκεύονται ταξινομημένα
- Χρησιμοποιείται σε υλοποιήσεις Βάσεων Δεδομένων και Filesystems

Ορισμός B-tree

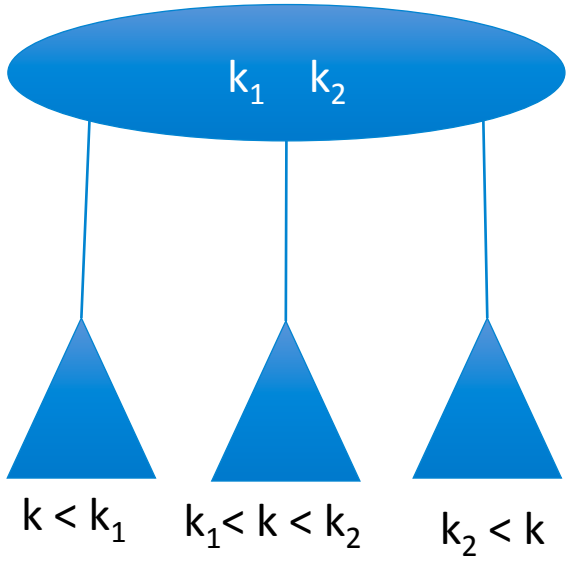
- Ένα B-tree τάξεως m, είναι ένα δέντρο με τα εξής χαρακτηριστικά:
  - Η ρίζα έχει τουλάχιστον 2 παιδιά, εκτός εάν είναι φύλλο
  - Κάθε κόμβος έχει το πολύ m παιδιά και m-1 κλειδιά
  - Κάθε κόμβος πλην της ρίζας και των φύλλων έχουν τουλάχιστον m/2 παιδιά
  - Όλα τα φύλλα έχουν το ίδιο ύψος
  - Ένας εσωτερικός κόμβος με k παιδιά έχει ακριβώς k-1 κλειδιά



[m = 3]

Ορισμός B-tree

Τα κλειδιά οργανώνονται σε υποδέντρα παρόμοια με ταν τρόπο που οργανώνονται στα δυαδικά δέντρα.



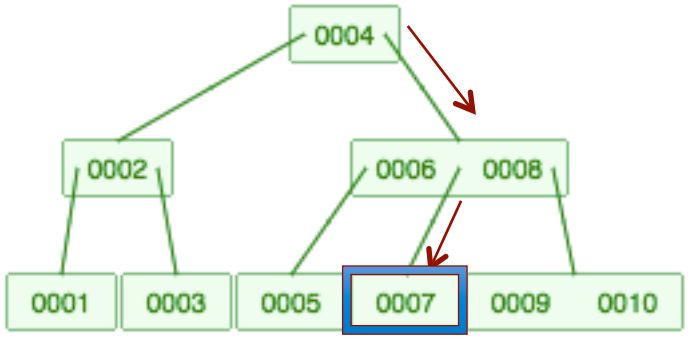
Πράξεις σε B-trees

Μας ενδιαφέρουν 3 πράξεις:

- Αναζήτηση (Access)
- Ένθεση (Insert)
- Διαγραφή (Delete)

Αναζήτηση (Access)

Η αναζήτηση του στοιχείου x σε ένα B-tree ξεκινά από τη ρίζα και σε κάθε επίπεδο επιλέγεται το αντίστοιχο υποδένδρο στο οποίο περιέχεται η τιμή που αναζητείται.



Access(7) [m=3]

Ένθεση (Insertion)

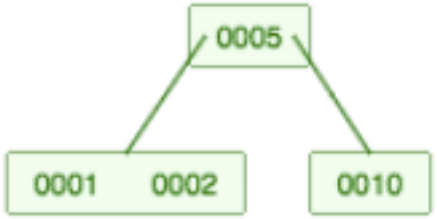
Για την εισαγωγή του στοιχείου x σε B-tree ακολουθούνται τα εξής βήματα:

- Εύρεση του κατάλληλου φύλλου y στο δέντρο για την εισαγωγή της τιμής x (Κλήση της Access(x)).
- Εισαγωγή του νέου κλειδιού στο φύλλο.
- Εφαρμογή επανορθωτικών πράξεων αν προκύπτει υπερχείλιση.

Ένθεση (Insertion)

Περίπτωση 1: Καμμία Υπερχείλιση

Εάν δεν υπάρχει υπερχείλιση, η ένθεση γίνεται κανονικά.

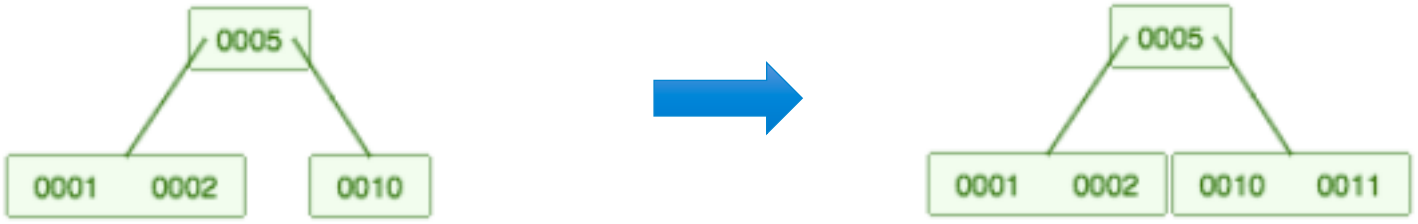


Insert(11) [m=3]

Ένθεση (Insertion)

Περίπτωση 1: Καμμία Υπερχείλιση

Εάν δεν υπάρχει υπερχείλιση, η ένθεση γίνεται κανονικά.



Insert(11) [m=3]

Ένθεση (Insertion)

Περίπτωση 2: Υπερχείλιση

Υπερχείλιση προκύπτει όταν μετά την ένθεση ένα φύλλο έχει m κλειδιά. Για να επιλυθεί το πρόβλημα εφαρμόζουμε τον μετασχηματισμό της διάσπασης ως εξής:

- Επιλέγεται το μεσαίο κλειδί x του κόμβου που είναι προς διάσπαση, μεταφέρεται στον γονέα, και ο κόμβος διασπάται σε 2 κόμβους.
- Τα μικρότερα κλειδιά από το x τοποθετούνται στο αριστερό υποδέντρο του x, ενώ τα μεγαλύτερα στο δεξιό.

Insert(15) *[m=3]*

Ένθεση (Insertion)

Περίπτωση 2: Υπερχείλιση

Υπερχείλιση προκύπτει όταν μετά την ένθεση ένα φύλλο έχει m κλειδιά. Για να επιλυθεί το πρόβλημα εφαρμόζουμε τον μετασχηματισμό της διάσπασης ως εξής:

- Επιλέγεται το μεσαίο κλειδί x του κόμβου που είναι προς διάσπαση, μεταφέρεται στον γονέα, και ο κόμβος διασπάται σε 2 κόμβους.
- Τα μικρότερα κλειδιά από το x τοποθετούνται στο αριστερό υποδέντρο του x, ενώ τα μεγαλύτερα στο δεξιό.

Insert(15) *[m=3]*

Ένθεση (Insertion)

Περίπτωση 2: Υπερχείλιση

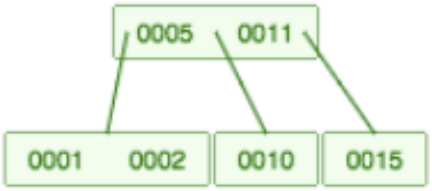
Υπερχείλιση προκύπτει όταν μετά την ένθεση ένα φύλλο έχει m κλειδιά. Για να επιλυθεί το πρόβλημα εφαρμόζουμε τον μετασχηματισμό της διάσπασης ως εξής:

- Επιλέγεται το μεσαίο κλειδί x του κόμβου που είναι προς διάσπαση, μεταφέρεται στον γονέα, και ο κόμβος διασπάται σε 2 κόμβους.
- Τα μικρότερα κλειδιά από το x τοποθετούνται στο αριστερό υποδέντρο του x, ενώ τα μεγαλύτερα στο δεξιό.

Insert(15) *[m=3]*

Ένθεση (Insertion)

Μετά τη διάσπαση ενός φύλλου, μπορεί να έχουμε διαδοχικές διασπάσεις μέχρι και τη ρίζα.



Insert(3) *[m=3]*

Ένθεση (Insertion)

Μετά τη διάσπαση ενός φύλλου, μπορεί να έχουμε διαδοχικές διασπάσεις μέχρι και τη ρίζα.

Insert(3) [m=3]

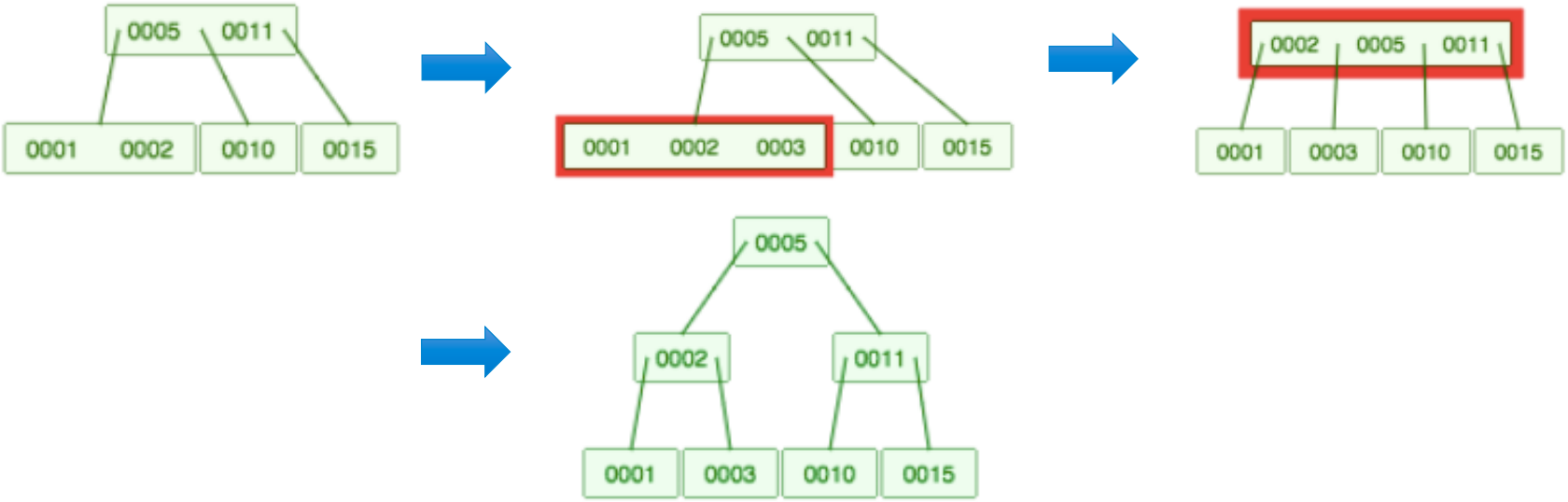
Ένθεση (Insertion)

Μετά τη διάσπαση ενός φύλλου, μπορεί να έχουμε διαδοχικές διασπάσεις μέχρι και τη ρίζα.

Insert(3) *[m=3]*

Ένθεση (Insertion)

Μετά τη διάσπαση ενός φύλλου, μπορεί να έχουμε διαδοχικές διασπάσεις μέχρι και τη ρίζα.



Insert(3) *[m=3]*

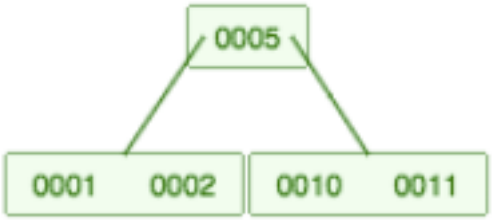
Διαγραφή (Deletion)

Για την διαγραφή ενος στοιχείου x σε B-tree ακολουθούνται τα εξής βήματα:

- Εύρεση του x σε ένα φύλλο του δέντρου (Κλήση της Access(x)).
- Διαγραφή του κλειδιού x
- Εφαρμογή επανορθωτικών πράξεων αν διαταραχθούν οι ιδιότητες του δέντρου.

Διαγραφή (Deletion)

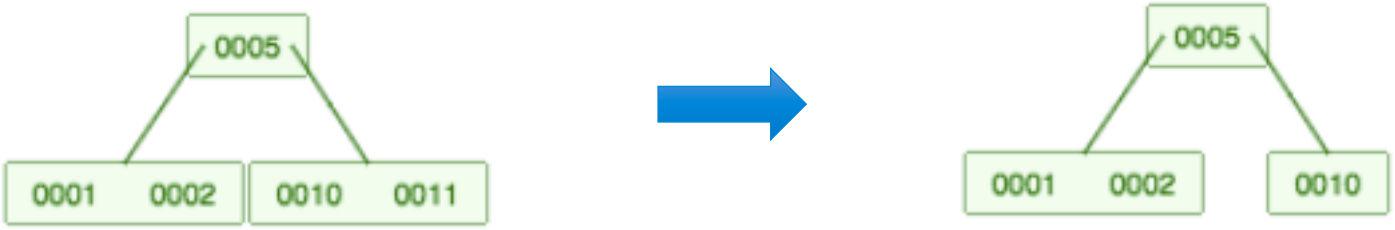
Περίπτωση 1: Η μορφή του δέντρου δε διαταράσσεται



Delete(11) [m=3]

Διαγραφή (Deletion)

Περίπτωση 1: Η μορφή του δέντρου δε διαταράσσεται

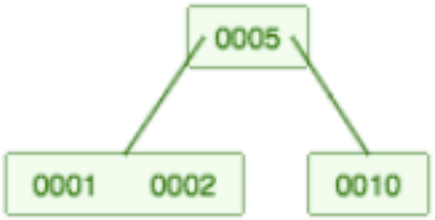


Delete(11) [m=3]

Διαγραφή (Deletion)

Περίπτωση 2: Η μορφή του δέντρου διαταράσσεται, και υπάρχει γειτονικός κόμβος του κόμβου που υπέστη διαγραφή, με παραπάνω από τα ελάχιστα δυνατά κλειδιά.

Λύση: Περιστροφή

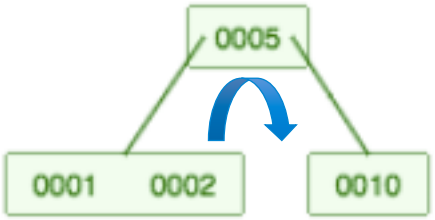


Delete(10) *[m=3]*

Διαγραφή (Deletion)

Περίπτωση 2: Η μορφή του δέντρου διαταράσσεται, και υπάρχει γειτονικός κόμβος του κόμβου που υπέστη διαγραφή, με παραπάνω από τα ελάχιστα δυνατά κλειδιά.

Λύση: Περιστροφή

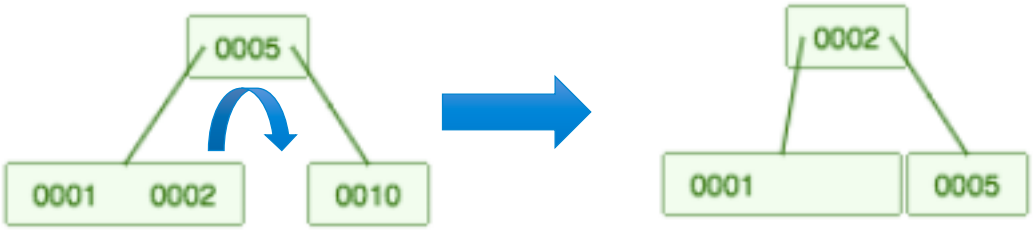


Delete(10) *[m=3]*

Διαγραφή (Deletion)

Περίπτωση 2: Η μορφή του δέντρου διαταράσσεται, και υπάρχει γειτονικός κόμβος του κόμβου που υπέστη διαγραφή, με παραπάνω από τα ελάχιστα δυνατά κλειδιά.

Λύση: Περιστροφή

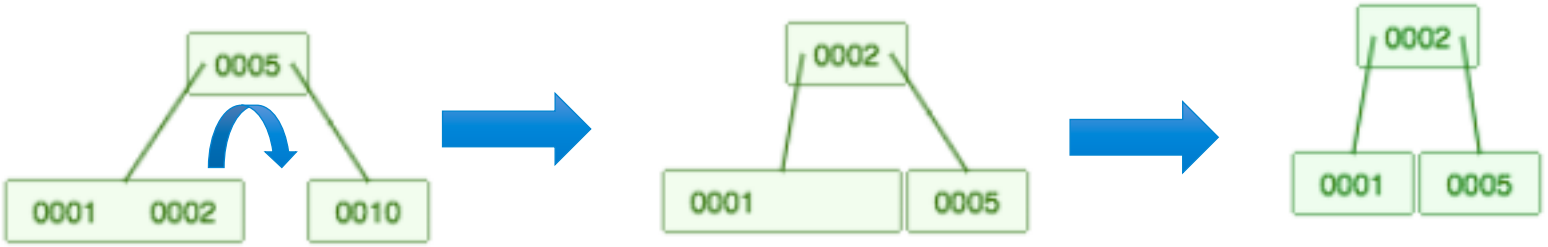


Delete(10) *[m=3]*

Διαγραφή (Deletion)

Περίπτωση 2: Η μορφή του δέντρου διαταράσσεται, και υπάρχει γειτονικός κόμβος του κόμβου που υπέστη διαγραφή, με παραπάνω από τα ελάχιστα δυνατά κλειδιά.

Λύση: Περιστροφή



Delete(10) [m=3]

Διαγραφή (Deletion)

Περίπτωση 3: Η μορφή του δέντρου διαταράσσεται, και **ΔΕΝ** υπάρχει γειτονικός κόμβος του κόμβου που υπέστη διαγραφή, με παραπάνω από τα ελάχιστα δυνατά κλειδιά.

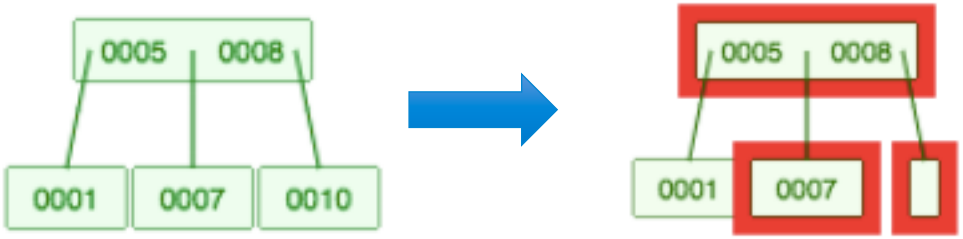
Λύση: Συγχώνευση

Delete(10) *[m=3]*

Διαγραφή (Deletion)

Περίπτωση 3: Η μορφή του δέντρου διαταράσσεται, και **ΔΕΝ** υπάρχει γειτονικός κόμβος του κόμβου που υπέστη διαγραφή, με παραπάνω από τα ελάχιστα δυνατά κλειδιά.

Λύση: Συγχώνευση

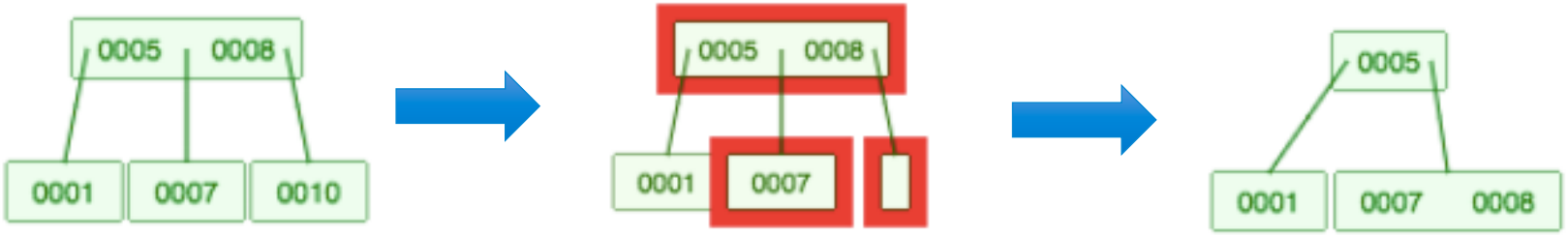


Delete(10) *[m=3]*

Διαγραφή (Deletion)

Περίπτωση 3: Η μορφή του δέντρου διαταράσσεται, και **ΔΕΝ** υπάρχει γειτονικός κόμβος του κόμβου που υπέστη διαγραφή, με παραπάνω από τα ελάχιστα δυνατά κλειδιά.

Λύση: Συγχώνευση



Delete(10) [m=3]

Απόδοση

Πολυπλοκότητες χειρότερης περίπτωσης:

- Search: O(log n)
- Insert: O(log n)
- Delete: O(log n)

Παρουσίαση Υλοποίησης

```
public class BTree<Key extends Comparable<Key>, Value>  {
    private static final int M = 3;     // max children per B-tree node = M-1

    private Node root;              // root of the B-tree
    private int HT;                 // height of the B-tree
    private int N;                  // number of key-value pairs in the B-tree

    // helper B-tree node data type
    private static final class Node {
        private int m;                                  // number of children
        private Entry[] children = new Entry[M];    // the array of children
        private Node(int k) { m = k; }              // create a node with k children
    }

    // internal nodes: only use key and next
    // external nodes: only use key and value
    private static class Entry {
        private Comparable key;
        private Object value;
        private Node next;     // helper field to iterate over array entries
        public Entry(Comparable key, Object value, Node next) {
            this.key   = key;
            this.value = value;
            this.next  = next;
        }
    }

    // constructor
    public BTree() { root = new Node(0); }
```

Παρουσίαση Υλοποίησης

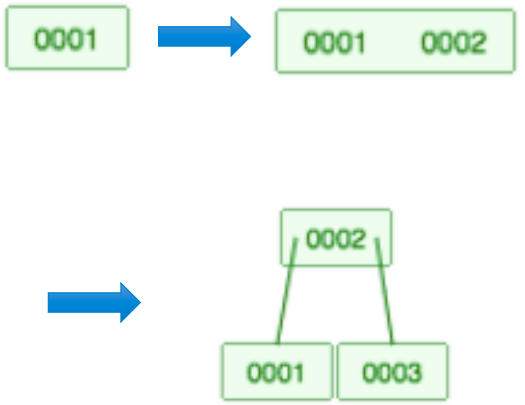
```
private Value search(Node x, Key key, int ht) {
    Entry[] children = x.children;

    // external node
    if (ht == 0) {
        for (int j = 0; j < x.m; j++) { // for all children of Node x
            // if the key is found, return the value
            if (eq(key, children[j].key)) return (Value) children[j].value;
        }
    }

    // internal node
    else {
        for (int j = 0; j < x.m; j++) { // for all children of Node x
            //if i'm inspecting the last child, or the key i'm searching is smaller than the next one..
            if (j+1 == x.m || less(key, children[j+1].key))
                // ... call search recursively
                return search(children[j].next, key, ht-1);
        }
    }
    return null;
}
```

Παρουσίαση Υλοποίησης

```
public static void main(String[] args) {
    BTree<String, String> st = new BTree<String, String>();

    st.put("1", "value1");
    st.put("2", "value2");

    System.out.println("Tree Size:    " + st.size());
    System.out.println("Tree Height:  " + st.height());
    System.out.println("-----");

    st.put("3", "value3");

    System.out.println("Tree Size:    " + st.size());
    System.out.println("Tree Height:  " + st.height());
    System.out.println("-----");
}
```

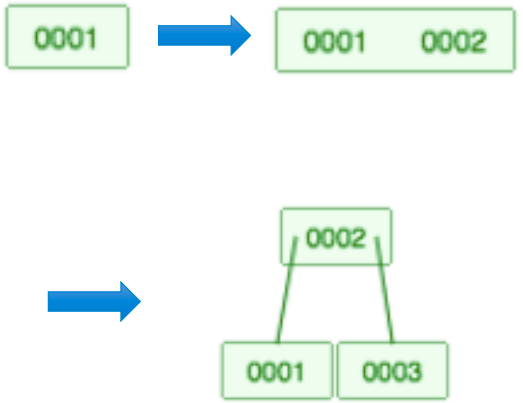
Παρουσίαση Υλοποίησης

```
public static void main(String[] args) {
    BTree<String, String> st = new BTree<String, String>();

    st.put("1", "value1");
    st.put("2", "value2");

    System.out.println("Tree Size:    " + st.size());
    System.out.println("Tree Height:  " + st.height());
    System.out.println("-----");

    st.put("3", "value3");

    System.out.println("Tree Size:    " + st.size());
    System.out.println("Tree Height:  " + st.height());
    System.out.println("-----");
}
```



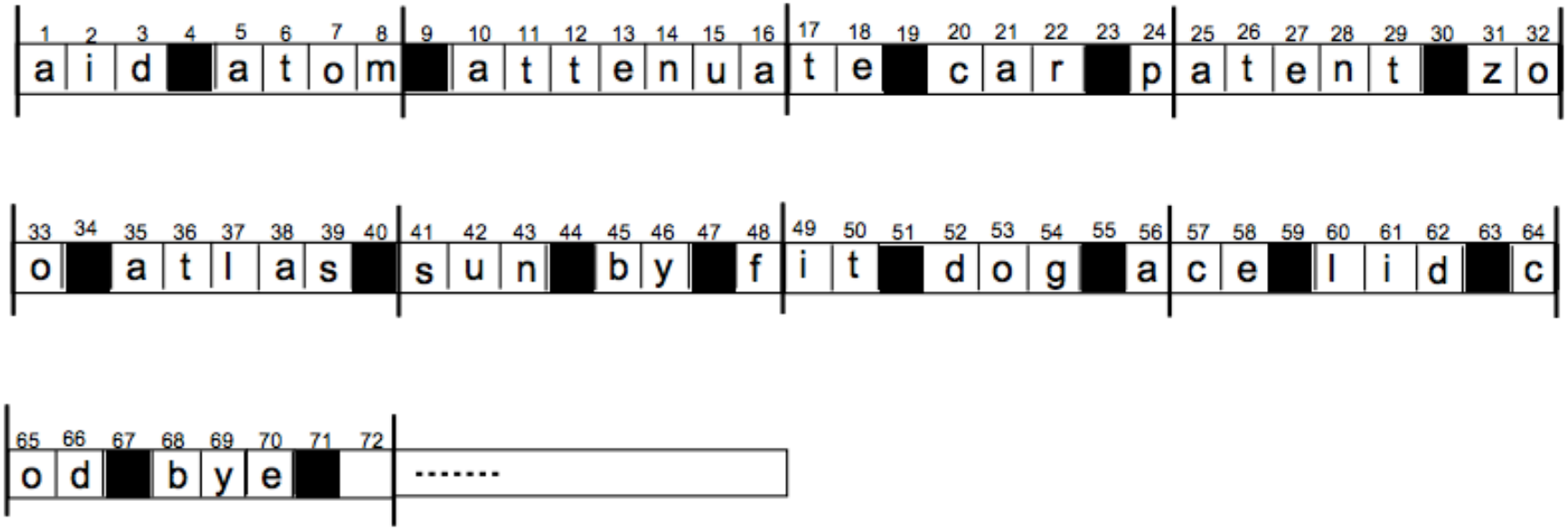
Παρουσίαση Υλοποίησης

```
public static void main(String[] args) {
    BTree<String, String> st = new BTree<String, String>();

    st.put("1", "value1");
    st.put("2", "value2");

    System.out.println("Tree Size:    " + st.size());
    System.out.println("Tree Height:  " + st.height());
    System.out.println("-----");

    st.put("3", "value3");

    System.out.println("Tree Size:    " + st.size());
    System.out.println("Tree Height:  " + st.height());
    System.out.println("-----");
}
```



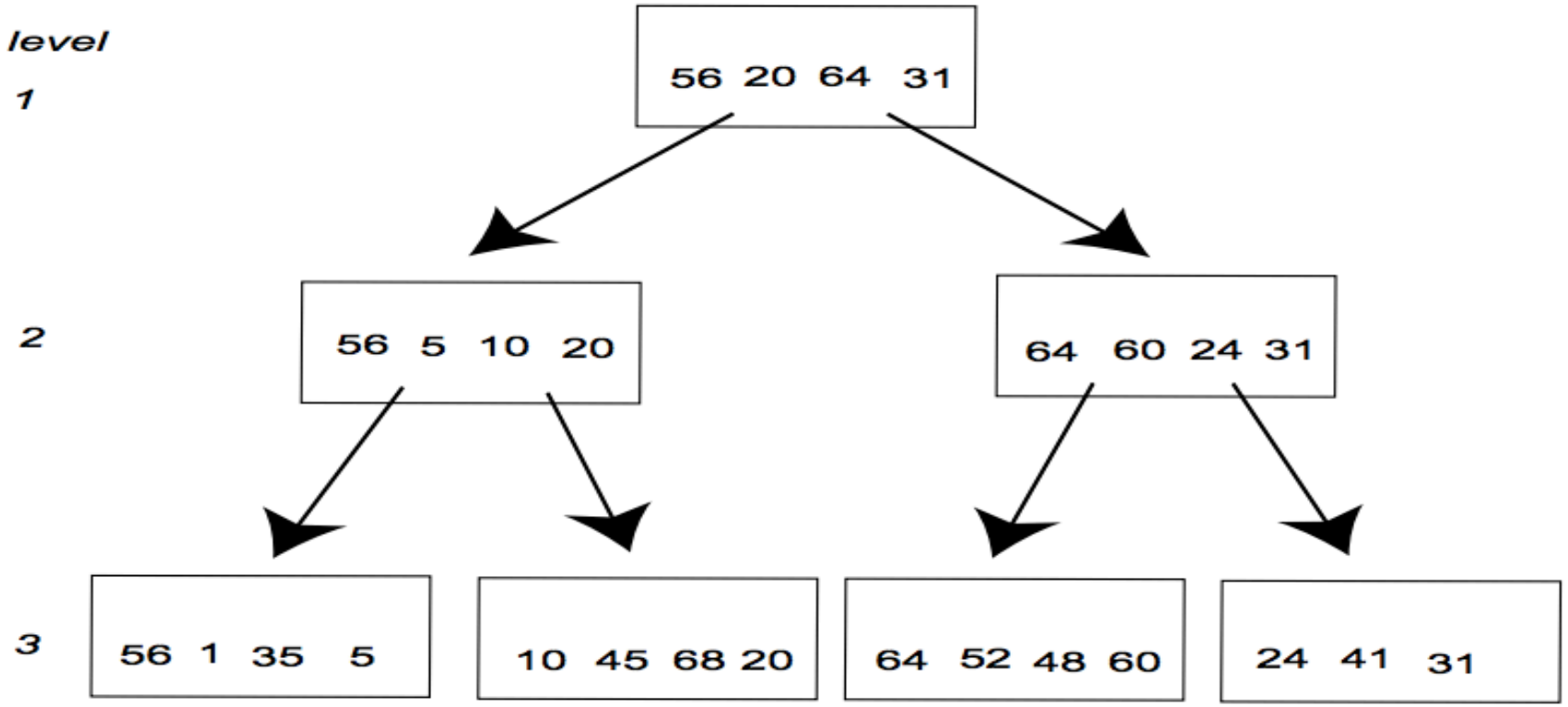

```
Alex-Air:src Alex$ java BTree
Tree Size:    2
Tree Height:  0
-----
Tree Size:    3
Tree Height:  1
-----
```

String B-tree



Δ = { ace, aid, atlas, atom, attenuate, by, bye, car, cod, dog, fit, lid, patent, sun, zoo }

String B-tree



References

- https://www.cs.usfca.edu/~galles/visualization/BTree.html
- http://algs4.cs.princeton.edu/62btrees/BTree.java.html

Ευχαριστώ!

?

ΑΥΤΟΡΓΑΝΟΥΜΕΝΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Splay Δέντρα

ΕΚΤΕΝΕΙΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Γενικά στοιχεία

- Μία τέτοιου είδους δομή βρίσκεται σε **τυχαία αρχική κατάσταση**, αλλά κατά τη διάρκεια κάθε πράξης εφαρμόζεται ένας **απλός ανακατασκευαστικός κανόνας** με σκοπό την **βελτίωση της αποδοτικότητας των μελλοντικών πράξεων**

Πλεονεκτήματα - Μειονεκτήματα

- ✓ Επίτευξη αποδοτικότητας με την επιμερισμένη έννοια
- ✓ Εφόσον προσαρμόζονται ανάλογα με τη χρήση, μπορούν να είναι πολύ πιο αποδοτικές αν το μοτίβο χρήσης είναι παράξενο
- ✓ Δεν αποθηκεύουν καμία πληροφορία ισορροπίας → Απαίτηση για μικρότερο χώρο
- ✓ Έχουν απλούς και εύκολα υλοποιήσιμους αλγόριθμους προσπέλασης και ανανέωσης
- — Απαιτούν περισσότερες τοπικές αναπροσαρμογές → μεγαλύτερα έξοδα αναδιοργάνωσης

Αυτοργανούμενες γραμμικές λίστες

- ✓ Πολύ απλές λίστες που αποθηκεύουν στοιχεία
- ✓ Πράξεις:
  - Access(x)
  - Insert(x)
  - Delete(x)
- Για την υλοποίησή τους μπορούμε να χρησιμοποιήσουμε συνδεδεμένες λίστες ή πίνακες

Αυτοργανούμενες γραμμικές λίστες

- ✓ pos(i) → η θέση του στοιχείου x_i στη λίστα
- ✓ Κάθε πράξη προσπελάζει γραμμικά τα στοιχεία από το αριστερότερο άκρο έως και την εύρεση του κατάλληλου στοιχείου ή το τέλος της λίστας
- ✓ Άρα,
  - ➢ Οι Access(x_i) και Delete(x_i) κοστίζουν pos(x_i)
  - ➢ Η Insert(x_i) κοστίζει |S|+1.

Αυτοργανούμενες γραμμικές λίστες

- ✓ pos(i) → η θέση του στοιχείου x_i στη λίστα
- ✓ Κάθε πράξη προσπελάζει γραμμικά τα στοιχεία από το αριστερότερο άκρο έως και την εύρεση του κατάλληλου στοιχείου ή το τέλος της λίστας
- ✓ Άρα,
  - ➢ Οι Access(x_i) και Delete(x_i) κοστίζουν pos(x_i)
  - ➢ Η Insert(x_i) κοστίζει |S|+1. Γιατί?

Στρατηγικές αυτοργάνωσης

- ✓ **Κανόνας μετακίνησης στην αρχή (Move to Front Rule - MFR)**
  - Οι πράξεις Access(x_i) και Insert(x_i) μετακινούν το x στην αρχή της λίστας → Δεν μεταβάλλεται η σειρά των υπόλοιπων στοιχείων
  - Η Delete(x_i) διαγράφει το στοιχείο x από τη λίστα

Στρατηγικές αυτοργάνωσης

- ✓ **Κανόνας μετακίνησης στην αρχή (Move to Front Rule - MFR)**
  - ➢ Οι πράξεις Access(x_i) και Insert(x_i) μετακινούν το x στην αρχή της λίστας → Δεν μεταβάλλεται η σειρά των υπόλοιπων στοιχείων
  - ➢ Η Delete(x_i) διαγράφει το στοιχείο x από τη λίστα
- ✓ Κανόνας Αντιμετάθεσης – Transposition Rule
  - ➢ Η Access(x_i) εναλλάσσει το x με το προηγούμενο στοιχείο
  - ➢ Η Insert(x_i) κάνει το x προτελευταίο στοιχείο της λίστας

Splay Δέντρα

- ✓ **Επιτυγχάνουν πολύ καλή επιμερισμένη πολυπλοκότητα για κάθε πράξη χωρίς να αποθηκεύουν τα βάρη κάθε στοιχείου.**
- ✓ Στρατηγική επανοργάνωσης: ***splaying***
- ✓ Κατά τη διάρκεια κάθε πράξης το εμπλεκόμενο στοιχείο μεταφέρεται στη ρίζα *με διαδοχικές περιστροφές*

Splay Δέντρα

- ✓ **Επιτυγχάνουν πολύ καλή επιμερισμένη πολυπλοκότητα για κάθε πράξη χωρίς να αποθηκεύουν τα βάρη κάθε στοιχείου.**
- ✓ Στρατηγική επανοργάνωσης: ***splaying***
- ✓ Κατά τη διάρκεια κάθε πράξης το εμπλεκόμενο στοιχείο μεταφέρεται στη ρίζα *με διαδοχικές περιστροφές → οι επόμενες πράξεις που θα χρειαστούν αυτό το στοιχείο θα είναι πιο φθηνές*

Splay Δέντρα

- ✓ Έστω ένα στοιχείο x ε S και p(x) ο πατέρας του x.
- ✓ Για να μεταφέρω το x στη ρίζα επαναλαμβάνουμε το splaying ως εξής:

- ❑ **ΠΕΡΙΠΤΩΣΗ 1**: Ο p(x) είναι ρίζα
- ❑ **ΠΕΡΙΠΤΩΣΗ 2**: Ο p(x) δεν είναι ρίζα & οι x, p(x) είναι και οι 2 αριστερά ή δεξιά παιδιά του γονέα τους
- ❑ **ΠΕΡΙΠΤΩΣΗ 3**: Ο p(x) είναι ρίζα & οι x, p(x) δεν είναι του ίδιου είδους παιδιά

ΠΕΡΙΠΤΩΣΗ 1

- **Ο p(x) είναι ρίζα**
  - ✓ Κάνουμε απλή περιστροφή και φέρνουμε το x στη ρίζα

ΠΕΡΙΠΤΩΣΗ 2

- **Ο p(x) είναι ρίζα**
  - ✓ Κάνουμε απλή περιστροφή και φέρνουμε το x στη ρίζα

- **Ο p(x) δεν είναι ρίζα & οι x, p(x) είναι και οι 2 αριστερά ή δεξιά παιδιά του γονέα τους**
  - ✓ Κάνουμε απλή περιστροφή στο p(x) με τον πατέρα του p(p(x)) και μετά κάνουμε άλλη μία περιστροφή της ίδιας μορφής με την πρώτη στο x και το p(x)

ΠΕΡΙΠΤΩΣΗ 3

- **Ο p(x) είναι ρίζα**
  - ✓ Κάνουμε απλή περιστροφή και φέρνουμε το x στη ρίζα

- Ο p(x) δεν είναι ρίζα & οι x, p(x) είναι και οι 2 αριστερά ή δεξιά παιδιά του γονέα τους
  - ✓ Κάνουμε απλή περιστροφή στο p(x) με τον πατέρα του p(p(x)) και μετά κάνουμε άλλη μία περιστροφή της ίδιας μορφής με την πρώτη στο x και το p(x)

- **Ο p(x) είναι ρίζα & οι x, p(x) δεν είναι του ίδιου είδους παιδιά**
  - ✓ Εκτελούμε διπλή περιστροφή στον x

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Υβριδικές Δομές Δεδομένων

Κεφάλαιο 6

ΥΒΡΙΔΙΚΕΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

Συνδυάζουν τη χρήση **δεικτών** και **πινάκων**

- Ψηφιακά Δένδρα – TRIES
- Interpolation Search Tree

TRIE

Το ζητούμενο:

Αποθήκευση και ανάκτηση πληροφορίας κειμένου εύκολα.

- Λέξεις
- Συμβολοσειρές
- Επιθέματα κλπ.

TRIE

Το ζητούμενο:

Αποθήκευση και ανάκτηση πληροφορίας κειμένου εύκολα.

- Λέξεις
- Συμβολοσειρές
- Επιθέματα κλπ.

Ταίριασμα Προτύπου ή Συμβολοσειράς

Εφαρμογές σε αναζήτηση/επεξεργασία κειμένου, data mining, βιοπληροφορική κλπ.

TRIE

Το ζητούμενο:

Αποθήκευση και ανάκτηση πληροφορίας κειμένου εύκολα.

- Λέξεις
- Συμβολοσειρές
- Επιθέματα κλπ.

Ταίριασμα Προτύπου ή Συμβολοσειράς

Εφαρμογές σε αναζήτηση/επεξεργασία κειμένου, data mining, βιοπληροφορική κλπ.

Μια απλή λύση είναι τα ψηφιακά δέντρα (TRIEs)

ΛΟΓΙΚΗ ΤΟΥ TRIE

$S = \{x_1, \ldots, x_n\}$

- Θέλουμε να αναπαραστήσω το S σε μια δομή
- Δεν στηριζόμαστε στις τιμές x_i
- Χρησιμοποιούμε **αναπαράσταση των στοιχείων** σαν μια ακολουθία χαρακτήρων

ΛΟΓΙΚΗ ΤΟΥ TRIE

$S = \{x_1, \ldots, x_n\}$

- Θέλουμε να αναπαραστήσω το S σε μια δομή
- Δεν στηριζόμαστε στις τιμές x_i
- Χρησιμοποιούμε **αναπαράσταση των στοιχείων** σαν μια ακολουθία χαρακτήρων

Μοιάζει με ΛΕΞΙΚΟ!!

- Οι λέξεις βρίσκονται ανάλογα με το γράμμα με το οποίο αρχίζουν
- Το ίδιο για το 2ο, 3ο, 4ο κλπ χαρακτήρα
- Είναι **φυλλοπροσανατολισμένο**!

ΟΡΙΣΜΟΣ

“Έστω σύμπαν U του οποίου τα στοιχεία είναι συμβολοσειρές μήκους λ πάνω σε ένα αλφάβητο K με |K| = k. Ένα σύνολο $S \subseteq U$ αναπαρίσταται σαν ένα k-δικο δένδρο που περιέχει όλα τα προθέματα των στοιχείων του S”

ΟΡΙΣΜΟΣ

“Έστω σύμπαν U του οποίου τα στοιχεία είναι συμβολοσειρές μήκους λ πάνω σε ένα αλφάβητο K με |K| = k. Ένα σύνολο $S \subseteq U$ αναπαρίσταται σαν ένα k-δικο δένδρο που περιέχει όλα τα προθέματα των στοιχείων του S”

Τι είναι πρόθεμα;

ΥΛΟΠΟΙΗΣΗ ΤΟΥ TRIE

1. Κάθε εσωτερικός κόμβος του δέντρου είναι ένας πίνακας μήκους k από δείκτες.

ΥΛΟΠΟΙΗΣΗ ΤΟΥ TRIE

1. Κάθε εσωτερικός κόμβος του δέντρου είναι ένας πίνακας μήκους k από δείκτες.
2. Κάθε θέση του πίνακα αντιστοιχίζεται σε ένα γράμμα του αλφαβήτου.

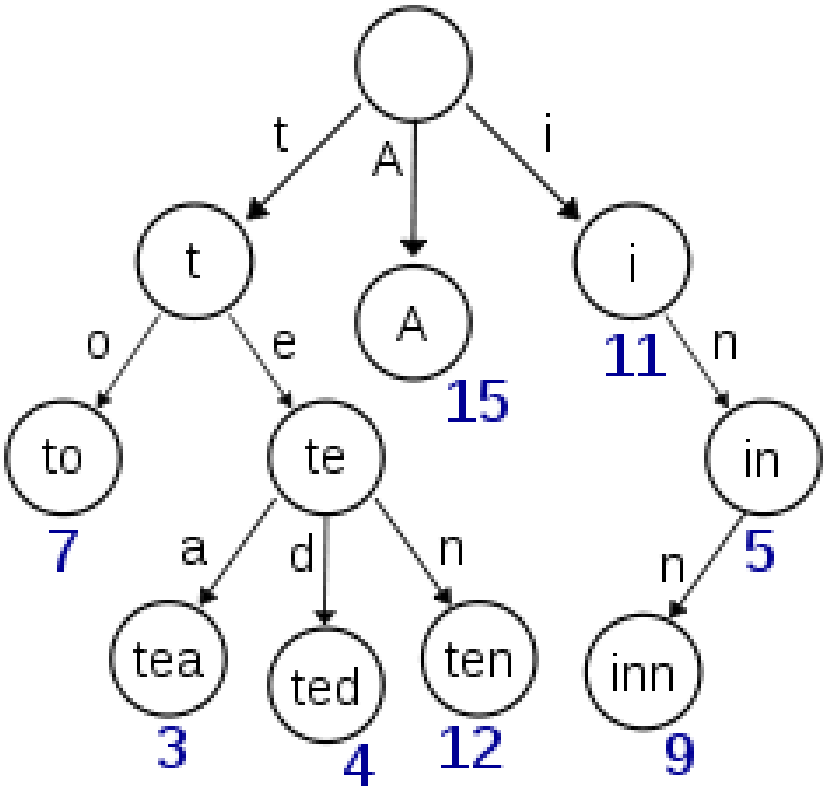
ΥΛΟΠΟΙΗΣΗ ΤΟΥ TRIE

1. Κάθε εσωτερικός κόμβος του δέντρου είναι ένας πίνακας μήκους k από δείκτες.
2. Κάθε θέση του πίνακα αντιστοιχίζεται σε ένα γράμμα του αλφαβήτου.
3. Κάθε θέση του πίνακα σε ένα κόμβο u σε βάθος i θα λάβει τιμή αν κάποιο από τα στοιχεία του S στην i-οστη θέση έχει τον αντίστοιχο χαρακτήρα

ΥΛΟΠΟΙΗΣΗ ΤΟΥ TRIE

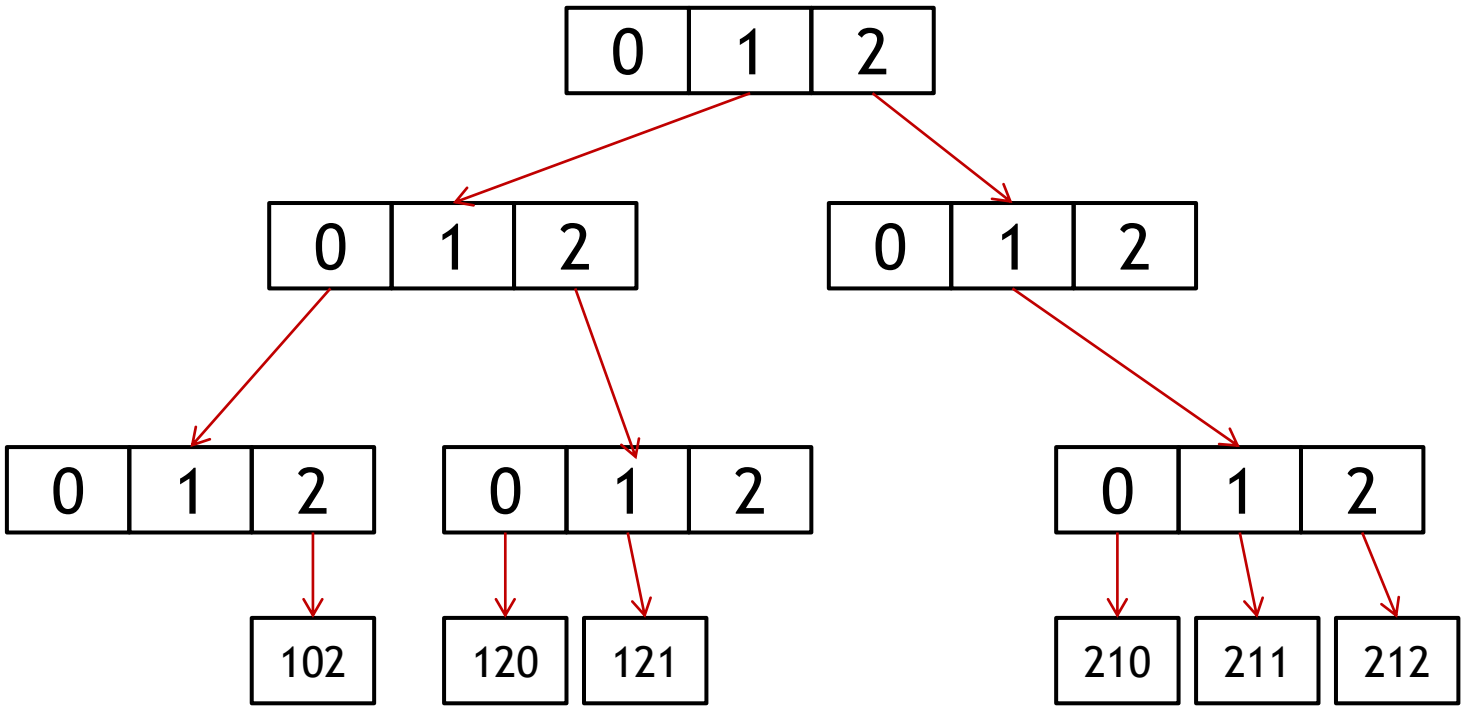
1. Κάθε εσωτερικός κόμβος του δέντρου είναι ένας πίνακας μήκους k από δείκτες.
2. Κάθε θέση του πίνακα αντιστοιχίζεται σε ένα γράμμα του αλφαβήτου.
3. Κάθε θέση του πίνακα σε ένα κόμβο u σε βάθος i θα λάβει τιμή αν κάποιο από τα στοιχεία του S στην i-οστη θέση έχει τον αντίστοιχο χαρακτήρα
4. Ύψος λ!
5. Χώρος O(k)

1ο ΠΑΡΑΔΕΙΓΜΑ



2ο ΠΑΡΑΔΕΙΓΜΑ

S = {102, 120, 121, 210, 211, 212}



ΠΟΛΥΠΛΟΚΟΤΗΤΕΣ

1. Οι **βασικές πράξεις** χρειάζονται χρόνο Ο(λ)

ΠΟΛΥΠΛΟΚΟΤΗΤΕΣ

1. Οι **<u>βασικές πράξεις</u>** χρειάζονται χρόνο O(λ)
2. Εξαρτάται από το μέγεθος του αλφάβητου και του σύμπαντος U: $O(\lambda) = O(\log_k N)$

ΠΟΛΥΠΛΟΚΟΤΗΤΕΣ

1. Οι **βασικές πράξεις** χρειάζονται χρόνο O(λ)
2. Εξαρτάται από το μέγεθος του αλφάβητου και του σύμπαντος U: $O(\lambda) = O(\log_k N)$
3. Ο χώρος που χρησιμοποιεί ένα TRIE όταν αναπαριστά |S| = n στοιχεία είναι O(nλk) στην χειρότερη περίπτωση. **Ποια είναι αυτή η περίπτωση;;**

ΠΟΛΥΠΛΟΚΟΤΗΤΕΣ

1. Οι **<u>βασικές πράξεις</u>** χρειάζονται χρόνο O(λ)
2. Εξαρτάται από το μέγεθος του αλφάβητου και του σύμπαντος U: $O(\lambda) = O(\log_k N)$
3. Ο χώρος που χρησιμοποιεί ένα TRIE όταν αναπαριστά |S| = n στοιχεία είναι O(nλk) στην χειρότερη περίπτωση. **<u>Ποια είναι αυτή η περίπτωση;;</u>**

Όταν τα στοιχεία δεν έχουν κανένα κοινό πρόθεμα. Οπότε:

- ***n* πλήρη μονοπάτια**
- **λ* *n* κόμβους συνολικά**
- **K * λ * *n* συνολικός χώρος**

ΛΥΣΗ – ΣΥΜΠΑΓΕΣ TRIE

Αποθηκεύονται ΜΟΝΟ οι κόμβοι του TRIE με βαθμό μεγαλύτερο ή ίσο του 2, ενώ οι αλυσίδες των κόμβων με βαθμό 1 αντικαθίστανται από έναν απλό αριθμό.

ΛΥΣΗ – ΣΥΜΠΑΓΕΣ TRIE

Αποθηκεύονται ΜΟΝΟ οι κόμβοι του TRIE με βαθμό μεγαλύτερο ή ίσο του 2, ενώ οι αλυσίδες των κόμβων με βαθμό 1 αντικαθίστανται από έναν απλό αριθμό.

Ο απλός αριθμός αποθηκεύεται στην (πρώτη) πλευρά που οδηγεί στην αλυσίδα και είναι ίσος με το πλήθος των κόμβων σε αυτή.

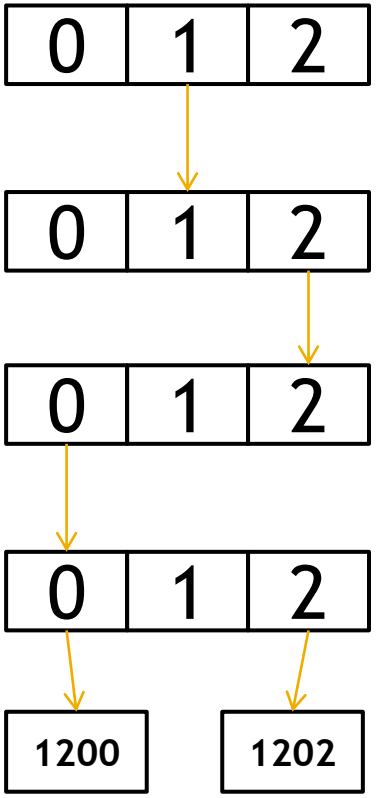
ΛΥΣΗ – ΣΥΜΠΑΓΕΣ TRIE

Αποθηκεύονται ΜΟΝΟ οι κόμβοι του TRIE με βαθμό μεγαλύτερο ή ίσο του 2, ενώ οι αλυσίδες των κόμβων με βαθμό 1 αντικαθίστανται από έναν απλό αριθμό.

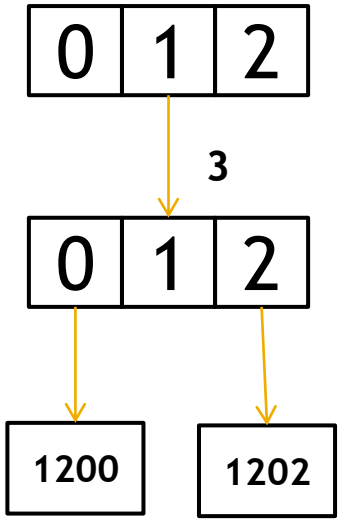
Ο απλός αριθμός αποθηκεύεται στην (πρώτη) πλευρά που οδηγεί στην αλυσίδα και είναι ίσος με το πλήθος των κόμβων σε αυτή.

Ο χώρος από O(nλk) γίνεται O(nk)

ΠΑΡΑΔΕΙΓΜΑ



TRIE



ΣΥΜΠΑΓΕΣ TRIE

ΔΕΝΔΡΟ ΕΠΙΘΕΜΑΤΩΝ – Suffix Tree

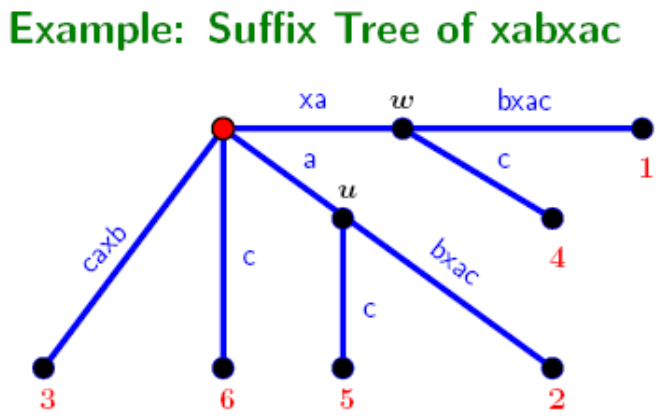
Ορισμός

Αποθηκεύει όλα τα δυνατά επιθέματα μιας συμβολοσειράς S.

ΔΕΝΔΡΟ ΕΠΙΘΕΜΑΤΩΝ – Suffix Tree

Ορισμός

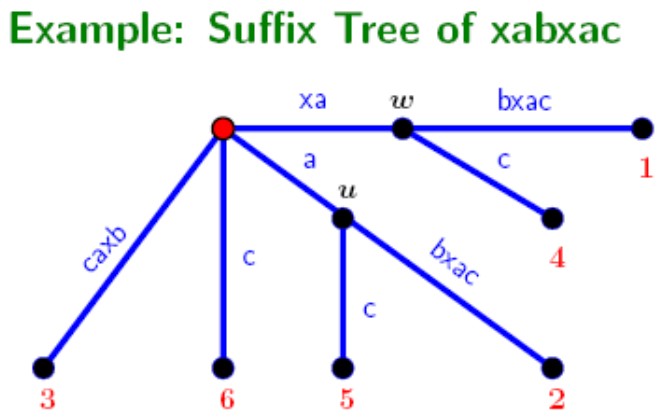
Αποθηκεύει όλα τα δυνατά επιθέματα μιας συμβολοσειράς S.



ΔΕΝΔΡΟ ΕΠΙΘΕΜΑΤΩΝ – Suffix Tree

Ορισμός

Αποθηκεύει όλα τα δυνατά επιθέματα μιας συμβολοσειράς S.



Το suffix tree μιας συμβολοσειράς S[1…n] είναι ένα *συμπαγές* **TRIE,** που περιέχει ως κλειδιά όλα τα επιθέματα S[i…n], 1≤i≤n.

ΚΑΤΑΣΚΕΥΗ Suffix Tree

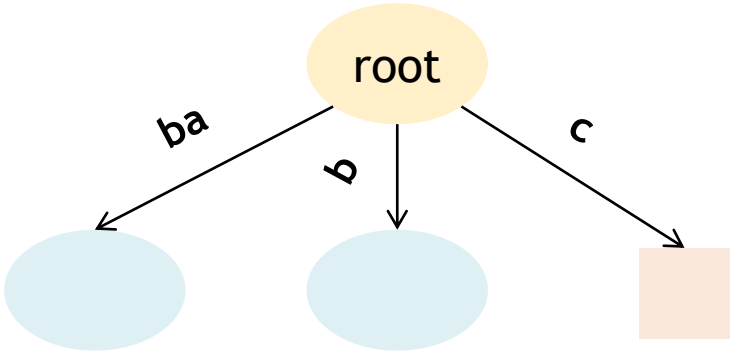
Έστω συμβολοσειρά X = ababc

ΚΑΤΑΣΚΕΥΗ Suffix Tree

Έστω συμβολοσειρά X = ababc

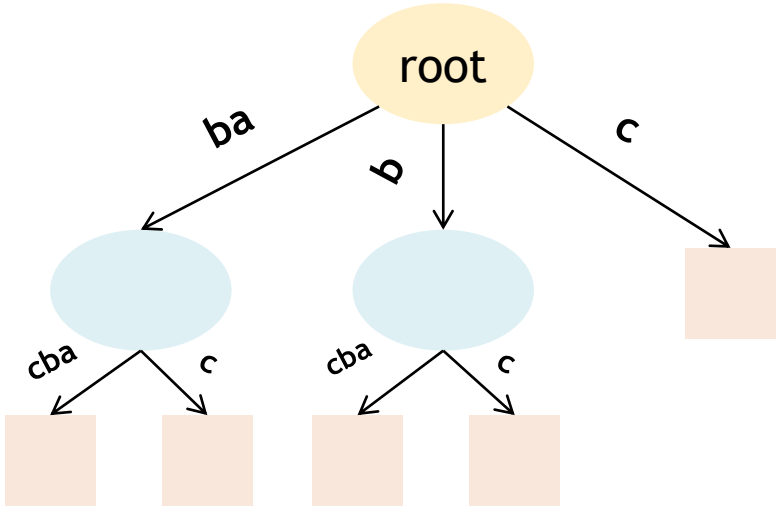
ΚΑΤΑΣΚΕΥΗ Suffix Tree

Έστω συμβολοσειρά X = ababc



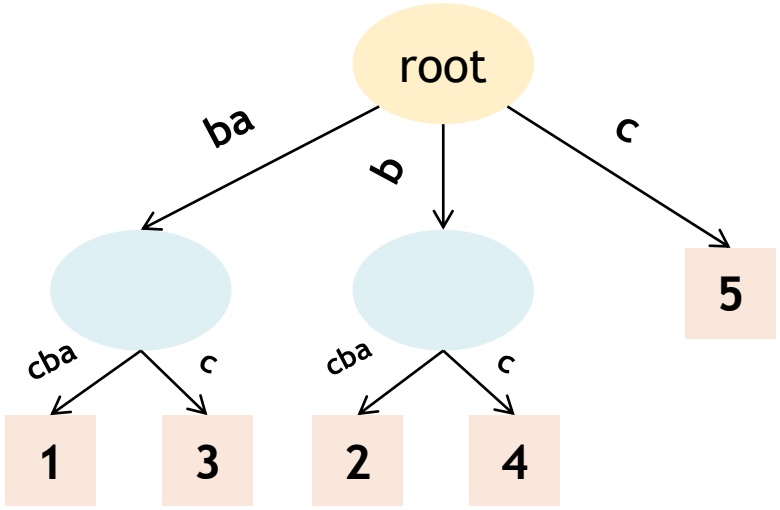
ΚΑΤΑΣΚΕΥΗ Suffix Tree

Έστω συμβολοσειρά X = ababc



ΚΑΤΑΣΚΕΥΗ Suffix Tree

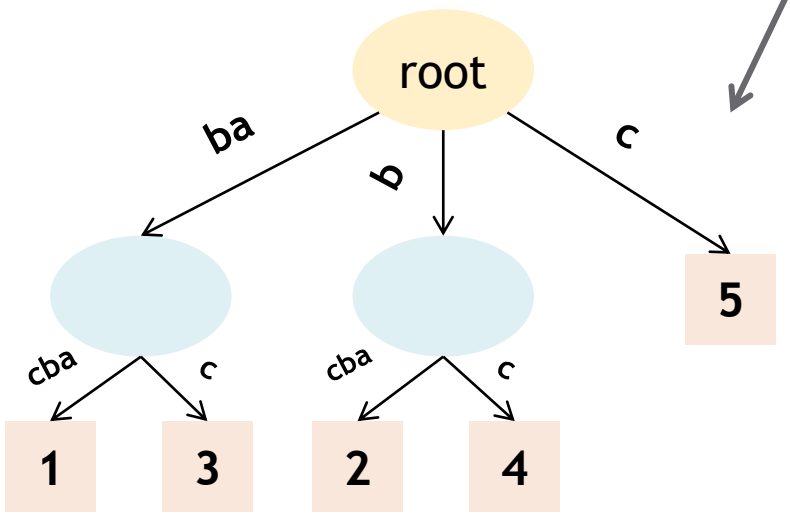
Έστω συμβολοσειρά X = ababc



ΚΑΤΑΣΚΕΥΗ Suffix Tree

Έστω συμβολοσειρά X = ababc

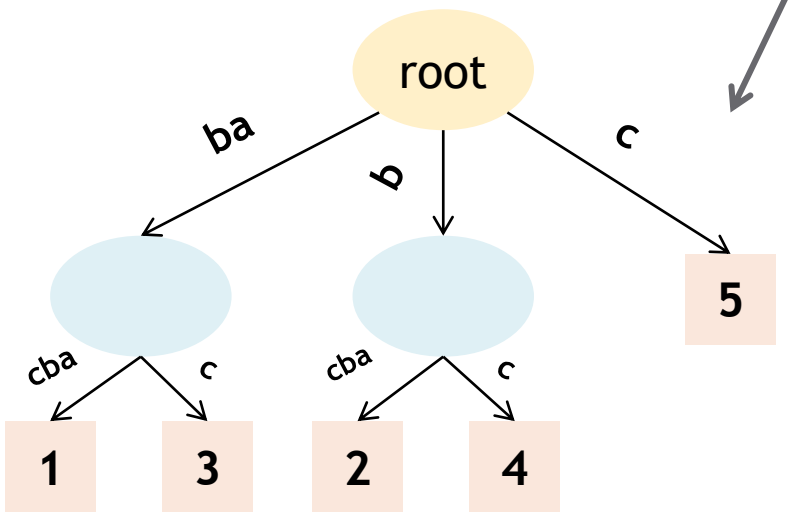
ΠΡΟΣΟΧΗ!
Από τον ίδιο κόμβο δεν εξέρχονται υπο-συμβολοσειρές με κοινό πρώτο χαρακτήρα



ΚΑΤΑΣΚΕΥΗ Suffix Tree

Έστω συμβολοσειρά X = ababc

ΠΡΟΣΟΧΗ!
Από τον ίδιο κόμβο δεν εξέρχονται υπο-συμβολοσειρές με κοινό πρώτο χαρακτήρα



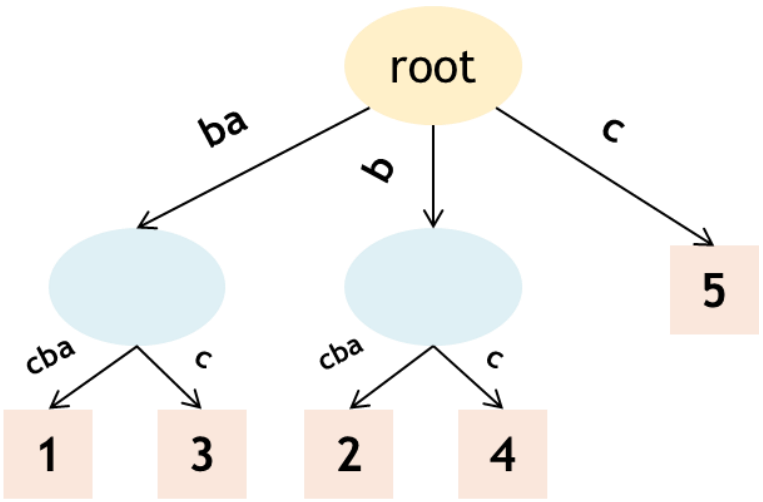
Γιατί αριθμήσαμε με αυτή τη σειρά τα φύλλα;

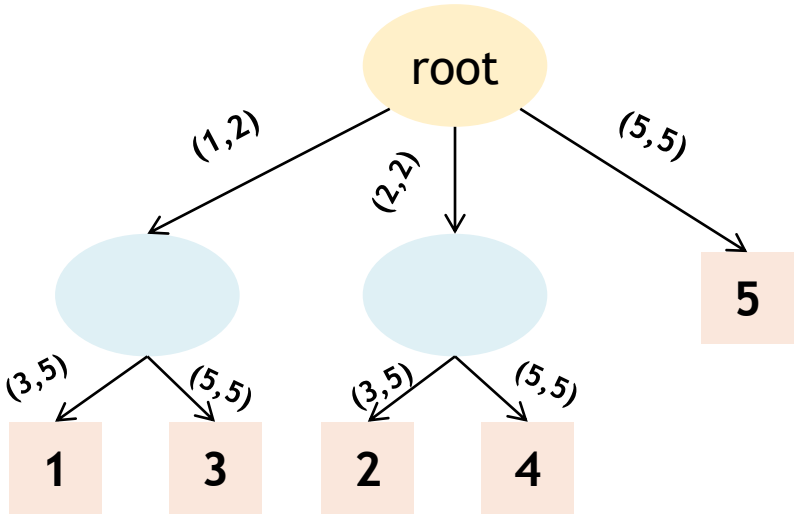
ΚΑΤΑΣΚΕΥΗ Suffix Tree – Αλλιώς

Έστω συμβολοσειρά X = ababc

ΚΑΤΑΣΚΕΥΗ Suffix Tree - Αλλιώς

Έστω συμβολοσειρά X = ababc





Εφαρμογές Suffix Tree

1. Ταίριασμα Προτύπου – Pattern matching
2. Μέγιστη Επαναλαμβανόμενη Υποσυμβολοσειρά – Longest Repeated Substring
3. Μέγιστη Κοινή Υποσυμβολοσειρά – Longest Common Substring

ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ

HASHING

ΣΑΛΤΟΓΙΑΝΝΗ ΑΘΑΝΑΣΙΑ

saltogiann@ceid.upatras.gr

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ

ΤΟ ΠΡΟΒΛΗΜΑ ΤΟΥ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΥ

- Θέλουμε **τα δεδομένα που διαθέτουμε** να μπορούν να αποθηκευτούν σε κάποιο πίνακα ή άλλη δομή και **να χωράνε στη μνήμη του υπολογιστή** που χρησιμοποιούμε.
- Ένα προς ένα αντιστοίχιση στοιχείων - **Απευθείας διευθυνσιοδότηση**
- Η διαθέσιμη μνήμη του υπολογιστή δεν μπορεί να ανταποκριθεί στον τεράστιο όγκο δεδομένων.

ΛΥΣΗ: Η μέθοδος του κατακερματισμού

Προσπαθεί να δώσει λύση στο πρόβλημα έλλειψης θέσεων σε μία δομή ή στη μνήμη του υπολογιστή, αντιστοιχίζοντας θέσεις της δομής/μνήμης στα δεδομένα με βάση κάποια συνάρτηση, ώστε να αποφεύγονται οι συγκρούσεις.

ΟΡΙΣΜΟΣ

Έστω:

- **σύμπαν** U = {0,1,..., N-1}, |U| = N
- **πίνακας κατακερματισμού** T[0,1,...,m-1], |T| = m
- **συνάρτηση κατακερματισμού** h:U → [0,1,...,m-1], |h| = m

Το στοιχείο x ε U θα αποθηκευτεί στην θέση T[h(x)]

ΕΙΔΗ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΎ

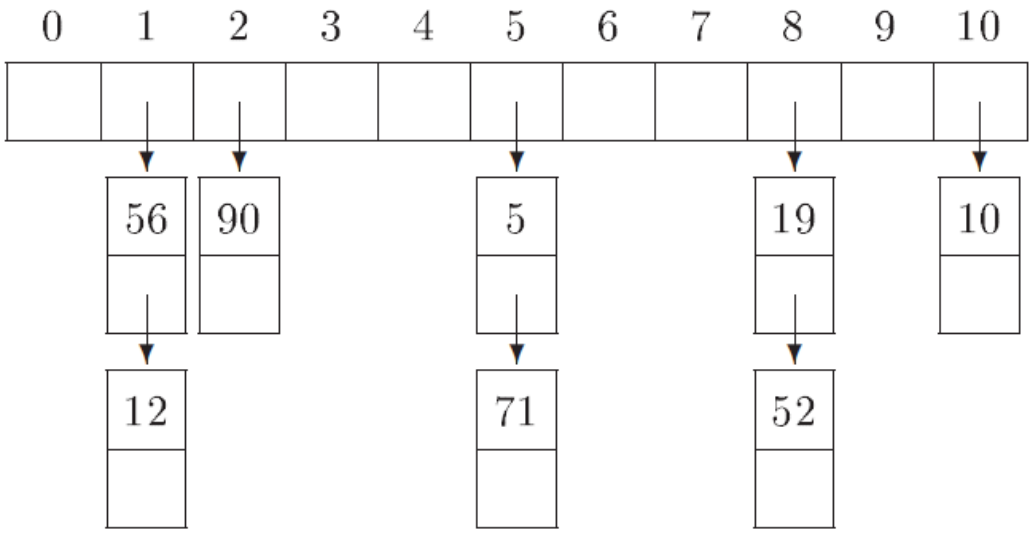
- **Κατακερματισμός με αλυσίδες**
  - Σε κάθε θέση του πίνακα υπάρχουν συνδεδεμένες αλυσίδες και μπορούν να επεκταθούν δυναμικά
  - **Τεχνική κλειστής διεύθυνσης**

- **Κλειστός κατακερματισμός**
  - Δεν χρησιμοποιεί δείκτες για το χειρισμό των πινάκων
  - **Τεχνική ανοικτής διεύθυνσης**
  - Γραμμική Δοκιμή, Τετραγωνική Δοκιμή, Διπλός κατακερματισμός

ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ ΜΕ ΑΛΥΣΙΔΕΣ

- Στις αλυσίδες, υπάρχει μία δομή record όπου το πρώτο πεδίο χρησιμοποιείται για την αποθήκευση των δεδομένων ενώ το δεύτερο πεδίο, αποτελεί δείκτη προς το επόμενο ζευγάρι της αλυσίδας.

- π.χ. για h(x) = x mod m



- Search(x)
- Insert(x)
- Delete(x)

ΚΛΕΙΣΤΟΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ

- Αντιμετωπίζει τις συγκρούσεις χωρίς να χρησιμοποιεί επιπλέον χώρο
- Το x ϵ U αντιστοιχίζεται σε σε μία ακολουθία θέσεων του πίνακα κατακερματισμού
- Πιο σύνθετη συνάρτηση κατακερματισμού
  - h(x,0), …, h(x,m-1)
  - I = 0,…,m-1 → προσπάθεια εύρεσης κατάλληλης θέσης

ΚΛΕΙΣΤΟΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ

- **ΓΡΑΜΜΙΚΗ ΔΟΚΙΜΗ**
  - εξετάζονται διαδοχικές θέσεις του πίνακα για την ανεύρεση ελεύθερης θέσης χρησιμοποιώντας συνάρτηση κατακερματισμού που δέχεται δύο ορίσματα, την τιμή έναρξης και το βήμα
  - $h(x,i) = [\, h_1(x) + i \,]\ mod\ x,\ \ i = 0, 1, \ldots, m-1$
  - $h_1(x) = x\ mod\ m$

Παράδειγμα

- m=11
- S = {52, 12, 71, 56, 5, 10, 19, 90}

	0	1	2	3	4	5	6	7	8	9	10
(α)		12				71			52		
(β)		12	56			71			52		
(γ)		12	56			71	5		52		10
(δ)		12	56			71	5		52	19	10
(ε)		12	56	90		71	5		52	19	10

ΚΛΕΙΣΤΟΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ

- **ΤΕΤΡΑΓΩΝΙΚΗ ΔΟΚΙΜΗ**
  - Η συνάρτηση κατακερματισμού είναι της μορφής:

$$(h_1(x) + c_1 i^2 + c_2 i^2)\ mod\ m$$

- Το i δηλώνει τις προσπάθειες για την εύρεση κενής θέσης και ισχύει $i = 0,1,2 \ldots$ και οι c_1, c_2είναι σταθερές.
- Εξετάζεται η πρώτη θέση για $i = 0$ και έπειτα αναζητούνται θέσεις σε αποστάσεις ανάλογες του τετραγώνου του i.
- Με αυτόν τον τρόπο, αποφεύγεται να καταλαμβάνονται συνεχόμενα μεγάλα τμήματα του πίνακα A στον οποίο εισάγονται τα κλειδιά.
- Μεγαλύτερη απόδοση από την γραμμικτή δοκιμή

ΚΛΕΙΣΤΟΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ

ΤΕΤΡΑΓΩΝΙΚΗ ΔΟΚΙΜΗ

Παράδειγμα

- m=11
- S = {52, 12, 71, 56, 5, 10, 19, 90}
- $c_1 = 1, c_2 = 2.$

	0	1	2	3	4	5	6	7	8	9	10
(a)		12				71			52		
(b)		12			56	71			52		
(g)	5	12			56	71			52		10
(d)	5	12	90	19	56	71			52		10

ΚΛΕΙΣΤΟΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ

- **ΔΙΠΛΟΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ**
  - Πιο αποτελεσματική μέθοδος από τις άλλες 2.
  - $h(x,i) = (h_1(x) + ih_2(x)) \; mod \; m$
  - $h_1, h_2 \rightarrow$ κλασικές συναρτήσεις κατακερματισμού
  - Αν προκύψει νέα σύγκρουση, τότε υπολογίζεται η επόμενη θέση σε ίση απόσταση από τη δεύτερη θέση σύγκρουσης.
  - Στην περίπτωση που το πηλίκο που προκύπτει είναι ίσο με 0, εξισώνεται με 1 ώστε αν κάποια κλειδιά συγκρούονται στην ίδια αρχική θέση, να μην συγκρουστούν σε επόμενη θέση.
  - Το μέγεθος του πίνακα να είναι πρώτος αριθμός, για να προσπελαστούν όλες οι θέσεις του πίνακα.

ΚΛΕΙΣΤΟΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ

Παράδειγμα

Ας εισάγουμε στην παρακάτω δομή με m=11 και n=8, τα στοιχεία S = {52, 12, 71, 56, 5, 10, 19, 90} με $h_1(x) = x \; mod \; m$ και $h_2(x) = (x \div m) \; mod \; m$.

	0	1	2	3	4	5	6	7	8	9	10
(a)		12				71			52		
(b)		12				71	56		52		
(g)		12				71	56	5	52		10
(d)		12	90			71	56	5	52	19	10

ΙΔΑΝΙΚΟΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ

- **ΙΔΑΝΙΚΟΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ – perfect hashing**
- Μπορούμε κατά τη σχεδίαση μίας δομής να επιλέξουμε συνάρτηση κατακερματισμού που να αποφεύγει τις συγκρούσεις.

- Μία συνάρτηση $h:U = [0,1,\ldots,N-1] \rightarrow [0,1,\ldots,m-1]$ είναι ιδανική συνάρτηση κατακερματισμού για το $S \subseteq [0,1,\ldots,N-1]$ αν για όλα τα x, $y \in S$ ισχύει $h(x) \neq h(y)$.

ΠΑΓΚΟΣΜΙΟΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ

- **ΠΑΓΚΟΣΜΙΟΣ ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ – universal hashing**
- Μπορούμε να επιλέξουμε για συνάρτηση κατακερματισμού μέσα από μία κλάση ομοιόμορφων συναρτήσεων.

- Μία συλλογή συναρτήσεων H={h : U =[0,1,…,N-1]→ =[0,1,…,m-1]} καλείται c-universal με c ∈ R αν για κάθε x, y ∈ U, x≠y, ισχύει:

$$|h: h \in H, h(x)=h(y)| \leq \frac{c|H|}{m}$$

25/5/2016

ΟΥΡΕΣ ΠΡΟΤΕΡΑΙΟΤΗΤΑΣ

Δομές Δεδομένων

Τι θα δούμε

- **Ουρές προτεραιότητας – Πράξεις**
- **Διωνυμικές Ουρές**
  - Διωνυμικά Δέντρα
  - Διωνυμικοί Σωροί
- **Ουρές Fibonacci**
  - Αναπαράσταση
  - Πράξεις
  - Ανάλυση
- **Συγκρίσεις**

Ουρές προτεραιότητας – Priority Queue

Συχνά χρειαζόμαστε δομές δεδομένων για την εισαγωγή στοιχείων που:

- το καθένα έχει κάποια **προτεραιότητα**
- η σειρά διαγραφής να καθορίζεται από **προτεραιότητα** (**μεγαλύτερη** - μικρότερη), χωρίς να μας ενδιαφέρει η σειρά με την οποία έγινε η εισαγωγή

Ουρές προτεραιότητας – Priority Queue

Συχνά χρειαζόμαστε δομές δεδομένων για την εισαγωγή στοιχείων που:

- το καθένα έχει κάποια **προτεραιότητα**
- η σειρά διαγραφής να καθορίζεται από **προτεραιότητα** (**μεγαλύτερη** - μικρότερη), χωρίς να μας ενδιαφέρει η σειρά με την οποία έγινε η εισαγωγή

Μια τέτοια ουρά ονομάζεται ουρά προτεραιότητας

Ορισμός

Τι είναι η Ουρά Προτεραιότητας;

Ορισμός

Τι είναι η Ουρά Προτεραιότητας;

Είναι ένας Αφηρημένος Τύπος Δεδομένων που:

- Διατηρεί ένα **σύνολο στοιχείων S**, όπου κάθε στοιχείο έχει μια **συσχετισμένη τιμή key(v),** η οποία υποδηλώνει την προτεραιότητα του στοιχείου.

Εφαρμογές

Τι είναι η Ουρά Προτεραιότητας;

Είναι ένας Αφηρημένος Τύπος Δεδομένων που:

- Διατηρεί ένα **σύνολο στοιχείων S**, όπου κάθε στοιχείο έχει μια **συσχετισμένη τιμή key(v),** η οποία υποδηλώνει την προτεραιότητα του στοιχείου.

- Υποστηρίζει τις πράξεις:
  - **MakeQueue( )**: Δημιουργία κενής ουράς
  - **Insert(Q,x)**: Εισαγωγή του στοιχείου x στην ουρά Q
  - **Delete(Q,x)**: Διαγραφή του στοιχείου x από την ουρά Q
  - **FindMin(Q)**: Επιστρέφει ένα δείκτη στην ελάχιστη τιμή της Q
  - **DeleteMin(Q)**: Διαγραφή του στοιχείου που περιέχει την μικρότερη τιμή
  - **Meld(Q_1, Q_2)**: Ένωση των ουρών Q_1 και Q_2
  - **DecreaseKey(Q,x,k)**: Ανάθεση της τιμής k στο κλειδί που είναι αποθηκευμένο στο στοιχείο x της ουράς Q.

Εφαρμογές

- **Άμεσες εφαρμογές:**
  - Υλοποίηση ουρών αναμονής με προτεραιότητες
    - Δρομολόγηση με προτεραιότητες
    - Largest (Smallest) Processing Time First
- **Έμμεσες εφαρμογές:**
  - Βασικό συστατικό πολλών ΔΔ και αλγορίθμων:
    - HeapSort
    - Αλγόριθμος Huffman
    - Αλγόριθμος Prim

Πιθανές υλοποιήσεις

1. Συνδεδεμένη λίστα
2. Ταξινομημένη συνδεδεμένη λίστα
3. Δυαδικό δένδρο αναζήτησης

Πιθανές υλοποιήσεις

1. Συνδεδεμένη λίστα
2. Ταξινομημένη συνδεδεμένη λίστα
3. Δυαδικό δένδρο αναζήτησης

Υπάρχει καλύτερη υλοποίηση;

Πιθανές υλοποιήσεις

1. Συνδεδεμένη λίστα
2. Ταξινομημένη συνδεδεμένη λίστα
3. Δυαδικό δένδρο αναζήτησης

Υπάρχει καλύτερη υλοποίηση;

Οι ουρές προτεραιότητας είναι μία παραλλαγή της κλασσικής σωρού

Τι είναι η Διωνυμική Ουρά Προτεραιότητας;

Είναι ένα δάσος που αποτελείται από ένα αριθμό δένδρων, τα οποία ονομάζονται **Διωνυμικά Δένδρα (Binomial Trees)**

Διωνυμικό Δέντρο

Συμβολίζω ένα **Διωνυμικό Δέντρο** έστω $\mathbf{B_k}$

Ορισμός

i. Το δέντρο $\mathbf{B_0}$ αποτελείται από **ένα μόνο στοιχείο**

ii. Το δέντρο $\mathbf{B_k}$, $\mathbf{k \geq 1}$, αποτελείται από **2** $\mathbf{B_{k-1}}$ **δέντρα** που συνδέονται μεταξύ τους έτσι ώστε **η ρίζα του ενός να είναι το αριστερότερο παιδί της ρίζας του άλλου**

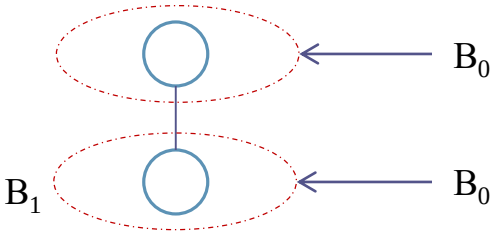
Διωνυμικό Δέντρο

B_0

Διωνυμικό Δέντρο

B_1

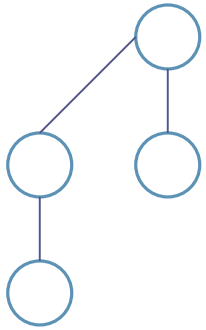
Διωνυμικό Δέντρο



Διωνυμικό Δέντρο

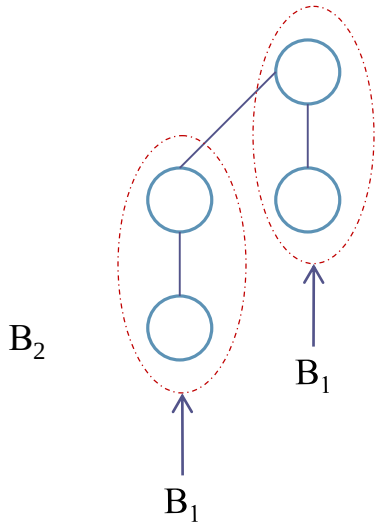
B_1

Διωνυμικό Δέντρο

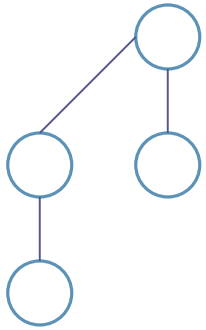


B_2

Διωνυμικό Δέντρο

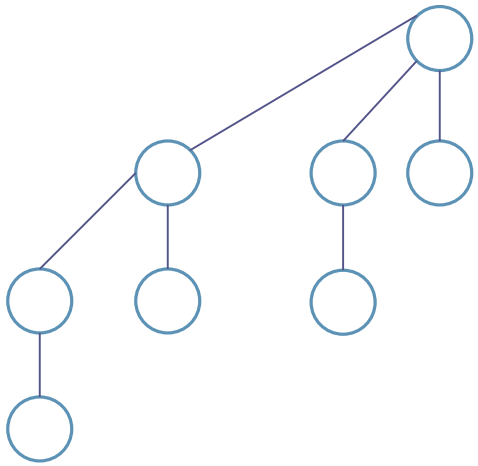


Διωνυμικό Δέντρο



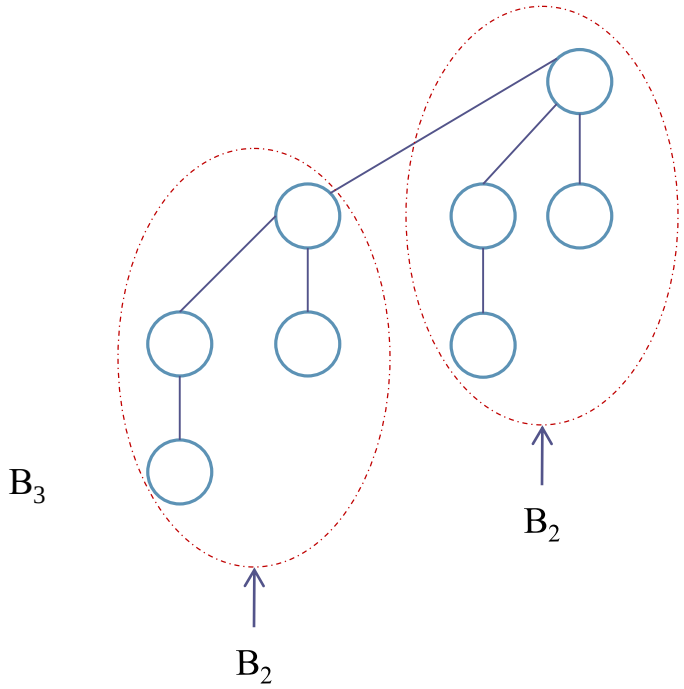
B_2

Διωνυμικό Δέντρο

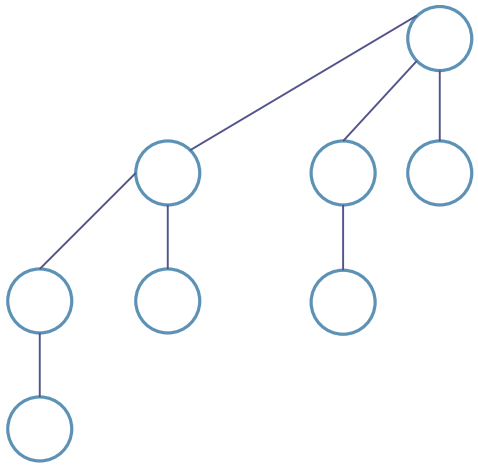


B_3

Διωνυμικό Δέντρο

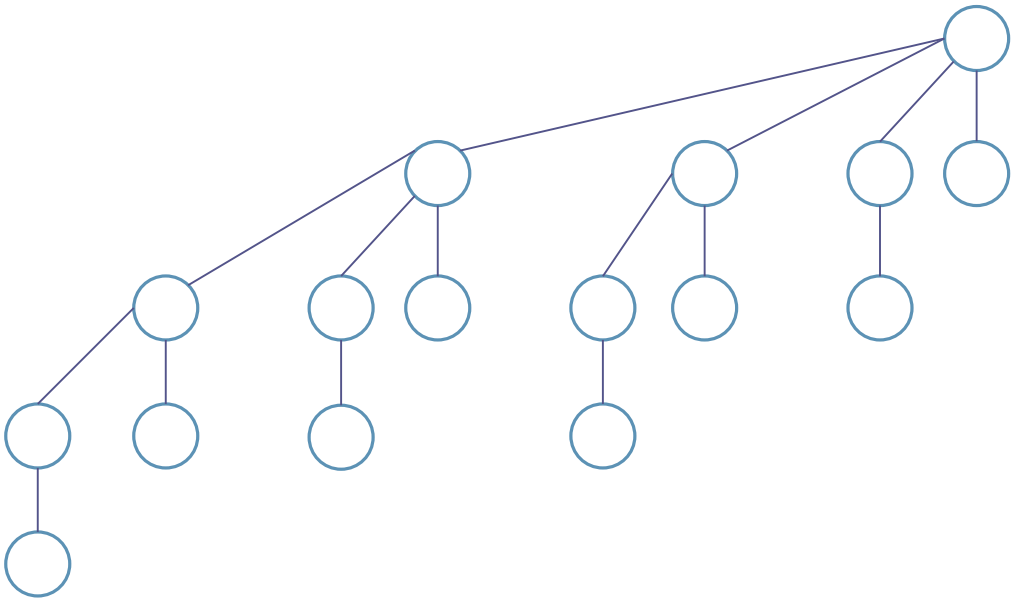


Διωνυμικό Δέντρο



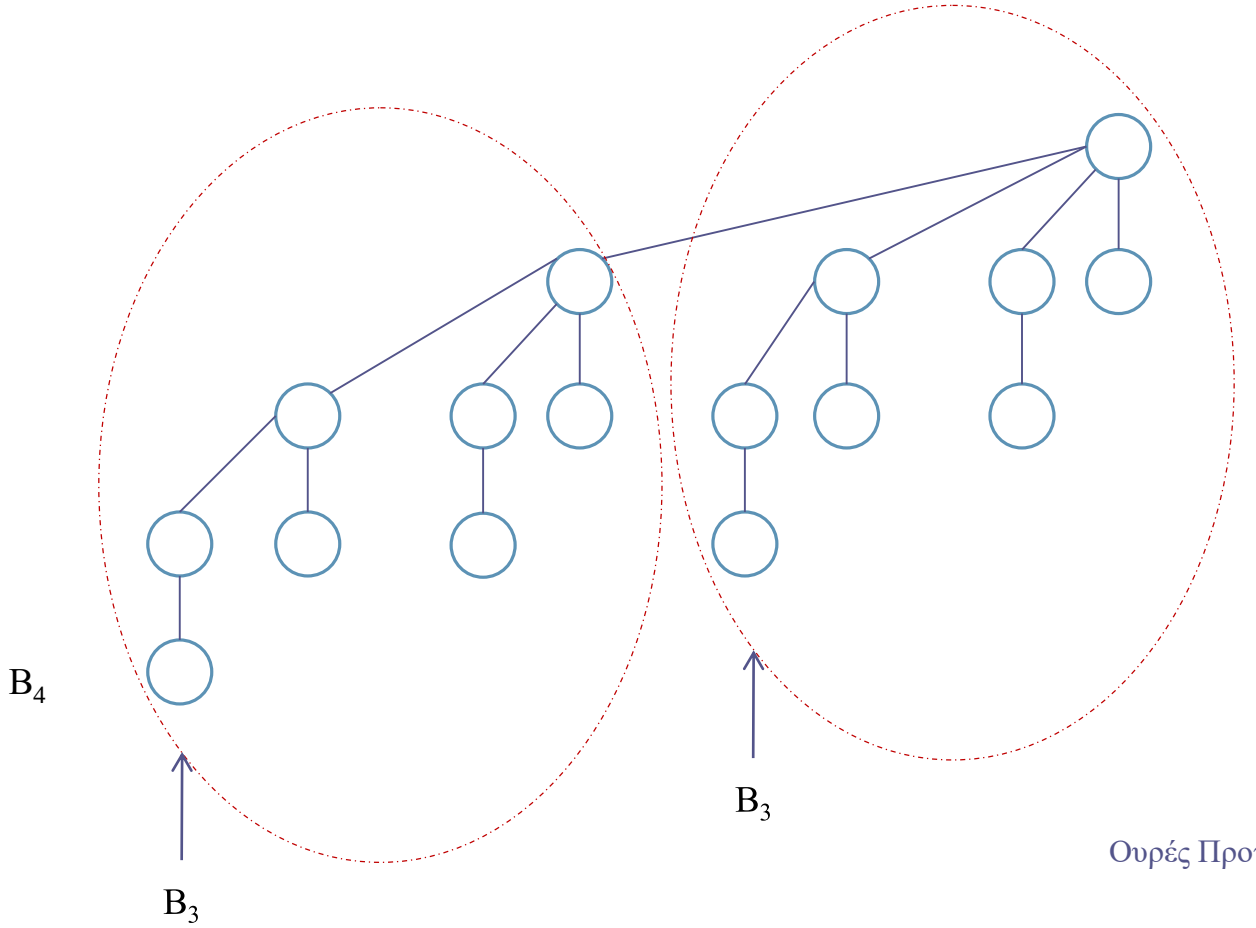
B_3

Διωνυμικό Δέντρο

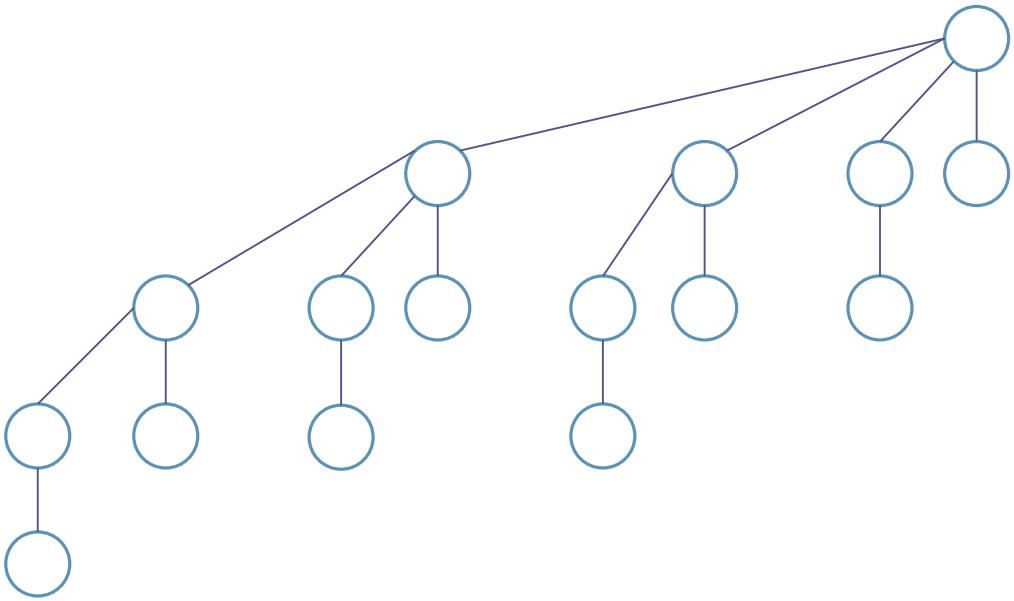


B_4

Διωνυμικό Δέντρο



Διωνυμικό Δέντρο



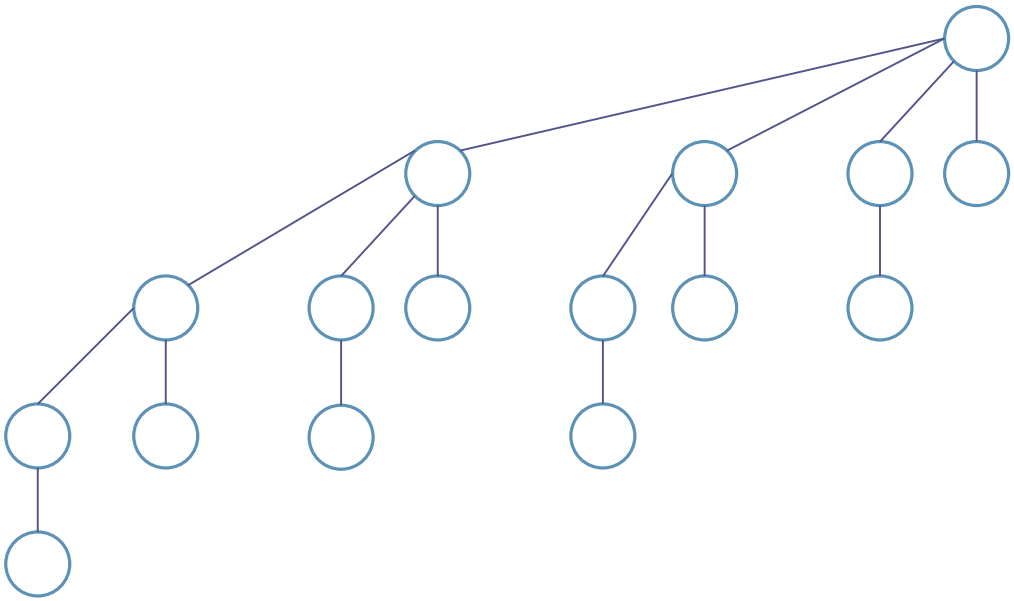
B_4

Διωνυμικό Δέντρο

Τι παρατηρούμε;

1. Το δέντρο $\mathbf{B_k}$ έχει 2^k κόμβους
2. Το δέντρο έχει ύψος k
3. Στο επίπεδο i υπάρχουν $\binom{k}{i}$ κόμβοι
4. Η ρίζα του δέντρου $\mathbf{B_k}$ έχει k παιδιά

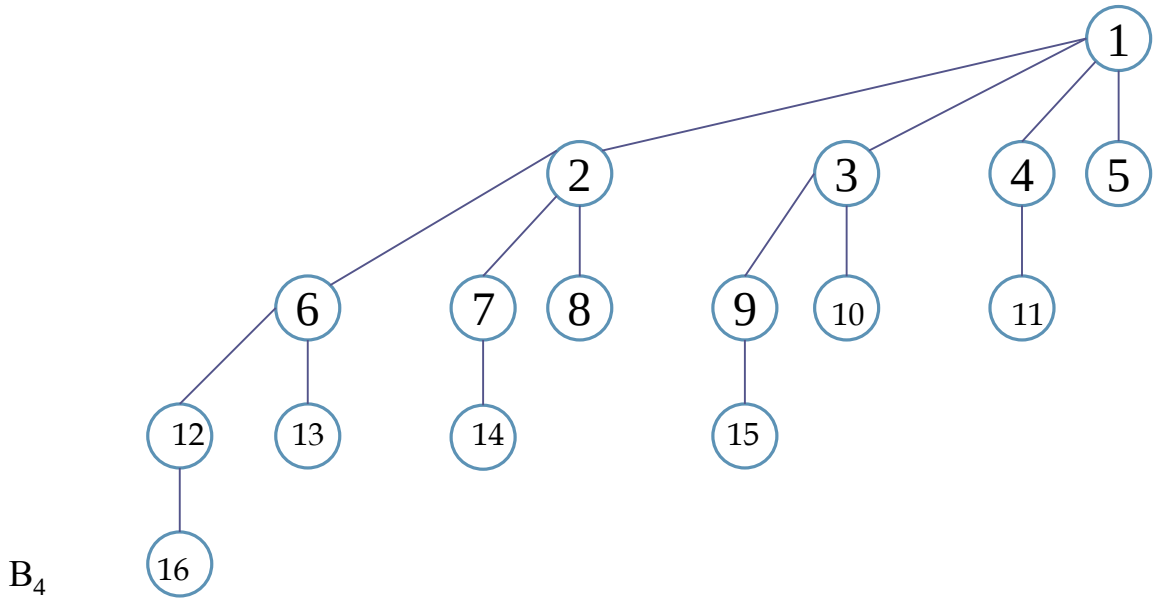
Διωνυμικό Δέντρο



B_4

Διωνυμικό Δέντρο

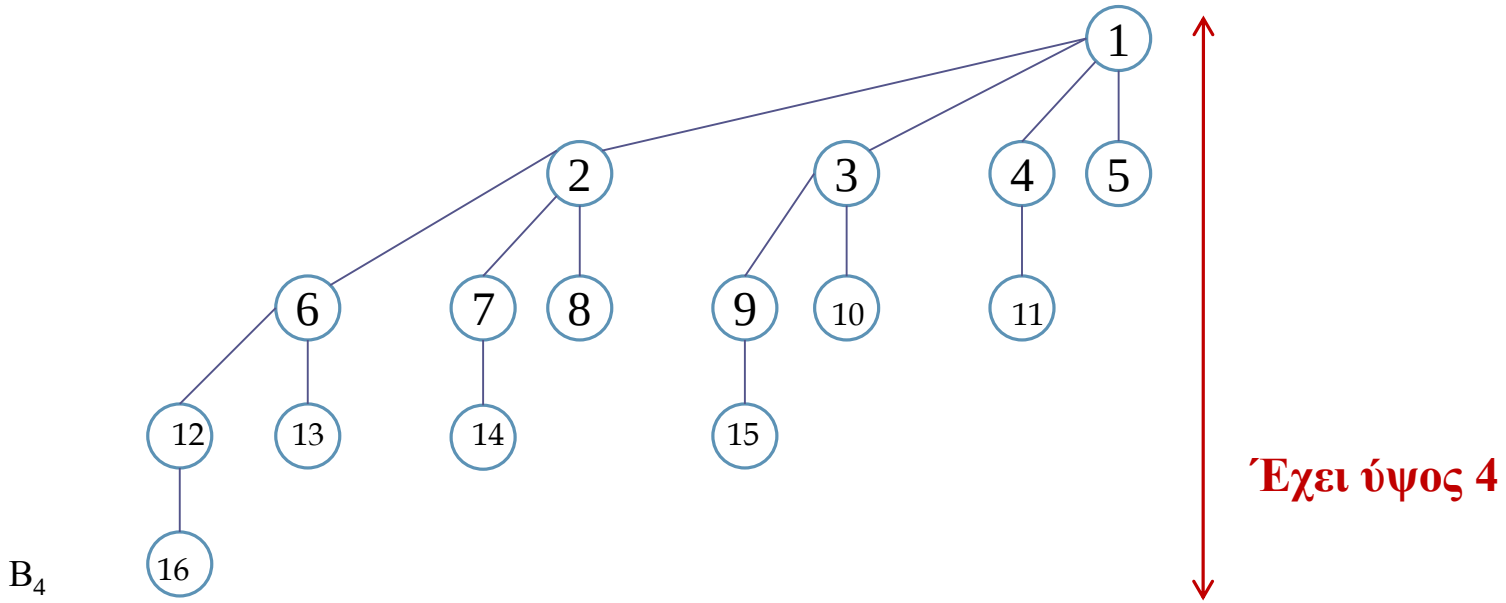
Το B_4 έχει 16 κόμβους



B_4

Διωνυμικό Δέντρο

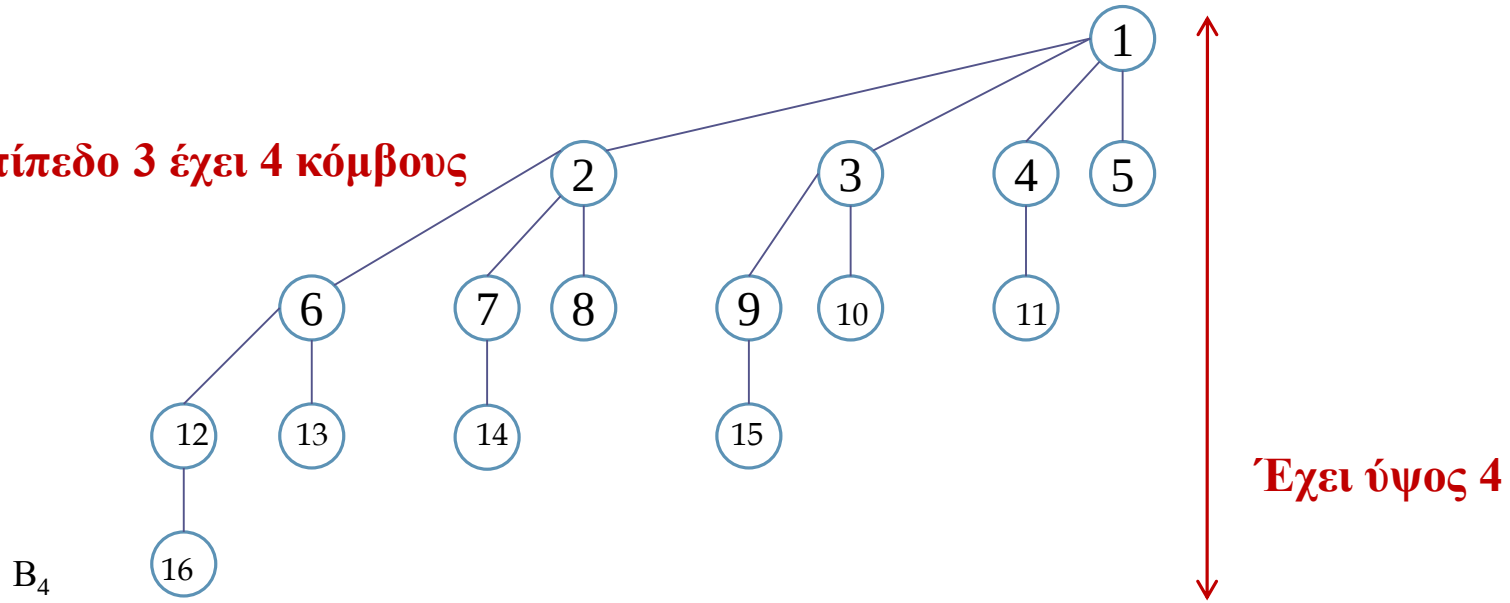
Το B_4 έχει 16 κόμβους



Διωνυμικό Δέντρο

Το B_4 έχει 16 κόμβους

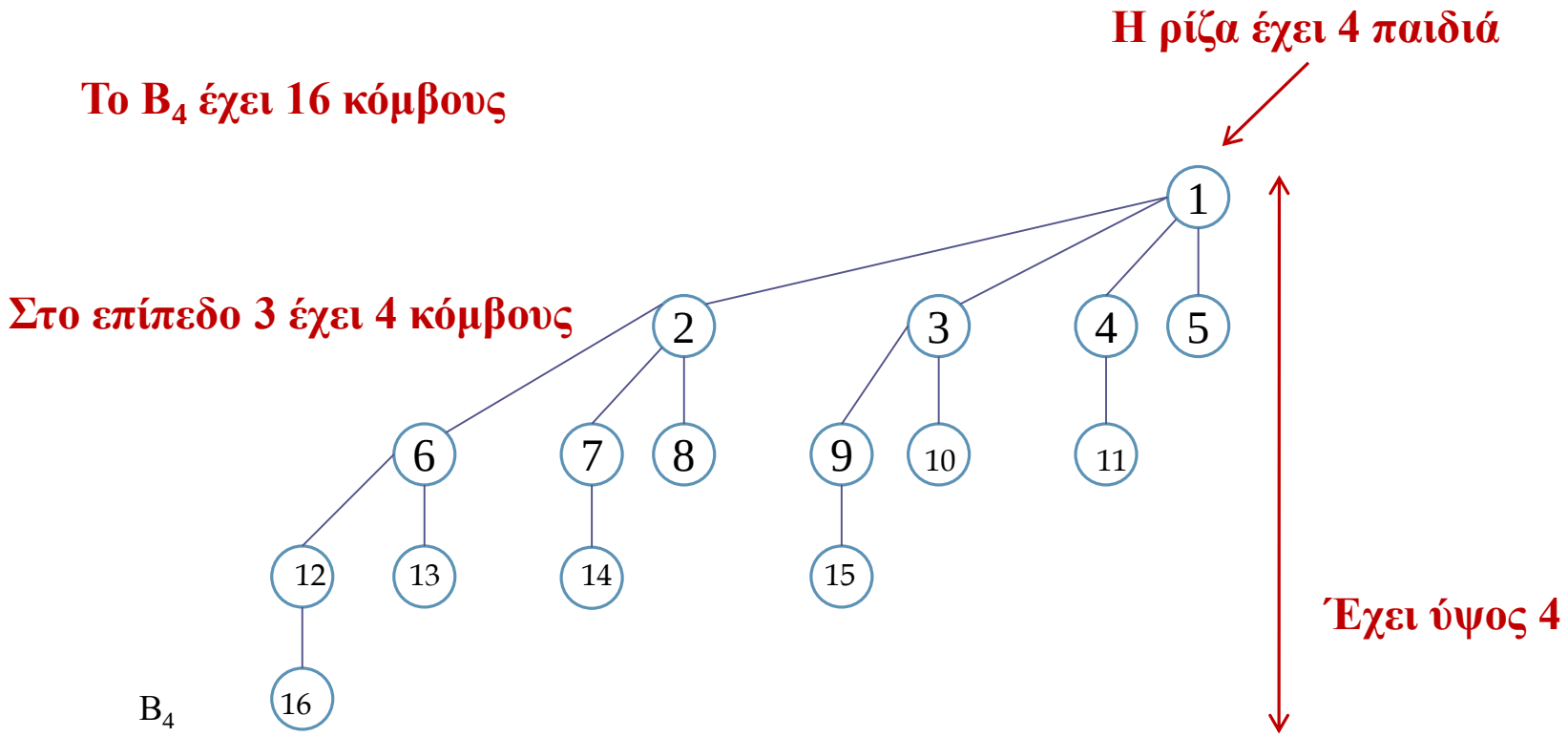
Στο επίπεδο 3 έχει 4 κόμβους



B_4

Έχει ύψος 4

Διωνυμικό Δέντρο



Τι είναι ο Διωνυμικός Σωρός;

Ένα σύνολο Διωνυμικών Δέντρων

ή αλλιώς

Ένα δάσος Διωνυμικών Δέντρων

Διωνυμικός Σωρός

- Οι ρίζες των δέντρων είναι οργανωμένες σε διασυνδεδεμένη λίστα → ***Λίστα Ριζών***
- Σε κάθε Διωνυμικό Δέντρο ενός Διωνυμικού Σωρού η τιμή που είναι αποθηκευμένη στο **γονέα** είναι **μικρότερη** από την τιμή/τις τιμές που είναι αποθηκευμένη/ες **στα παιδιά του**.

Διωνυμικός Σωρός

- Οι ρίζες των δέντρων είναι οργανωμένες σε διασυνδεδεμένη λίστα → ***Λίστα Ριζών***
- Σε κάθε Διωνυμικό Δέντρο ενός Διωνυμικού Σωρού η τιμή που είναι αποθηκευμένη στο **γονέα** είναι **μικρότερη** από την τιμή/τις τιμές που είναι αποθηκευμένη/ες **στα παιδιά του**.

Που θα αποθηκευτεί η μικρότερη τιμή;;

Διωνυμικός Σωρός

- Οι ρίζες των δέντρων είναι οργανωμένες σε διασυνδεδεμένη λίστα → ***Λίστα Ριζών***
- Σε κάθε Διωνυμικό Δέντρο ενός Διωνυμικού Σωρού η τιμή που είναι αποθηκευμένη στο **γονέα** είναι **μικρότερη** από την τιμή/τις τιμές που είναι αποθηκευμένη/ες **στα παιδιά του**.

Που θα αποθηκευτεί η μικρότερη τιμή;;

- **Ανάλογα με τον αριθμό κόμβων δημιουργούμε τον Διωνυμικό Σωρό:**
  - Μετατρέπουμε τον αριθμό σε δυαδικό
  - Με βάση την δύναμη του 2 που αντιστοιχεί σε 1 φτιάχνουμε τα αντίστοιχα Διωνυμικά Δέντρα
  - Τοποθετούμε τα Διωνυμικά Δέντρα από αριστερά προς δεξιά κατά αύξουσα τάξη

Παράδειγμα - Διωνυμικός Σωρός 7 στοιχείων

$7_{(10)} = 0111_{(2)} \rightarrow 7_{(10)} = 2^2 + 2^1 + 2^0$

Παράδειγμα - Διωνυμικός Σωρός 7 στοιχείων

$7_{(10)} = 0111_{(2)} \rightarrow 7_{(10)} = 2^2 + 2^1 + 2^0$

Άρα ο Διωνυμικός Σωρός 7 στοιχείων θα αποτελείται από:

- 1 Διωνυμικό Δέντρο $\mathbf{B_0}$

Παράδειγμα - Διωνυμικός Σωρός 7 στοιχείων

$7_{(10)} = 0111_{(2)} \rightarrow 7_{(10)} = 2^2 + 2^1 + 2^0$

Άρα ο Διωνυμικός Σωρός 7 στοιχείων θα αποτελείται από:

- 1 Διωνυμικό Δέντρο $\mathbf{B_0}$

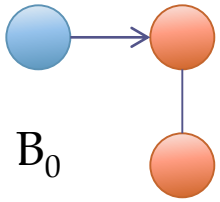
B_0

Παράδειγμα - Διωνυμικός Σωρός 7 στοιχείων

$7_{(10)} = 0111_{(2)} \rightarrow 7_{(10)} = 2^2 + 2^1 + 2^0$

Άρα ο Διωνυμικός Σωρός 7 στοιχείων θα αποτελείται από:

- 1 Διωνυμικό Δέντρο $\mathbf{B_0}$
- 1 Διωνυμικό Δέντρο $\mathbf{B_1}$

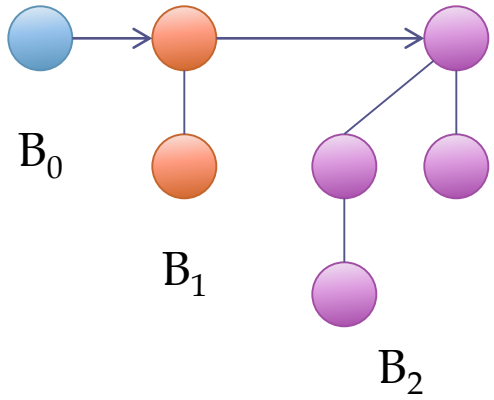


Παράδειγμα - Διωνυμικός Σωρός 7 στοιχείων

$7_{(10)} = 0111_{(2)} \rightarrow 7_{(10)} = 2^2 + 2^1 + 2^0$

Άρα ο Διωνυμικός Σωρός 7 στοιχείων θα αποτελείται από:

- 1 Διωνυμικό Δέντρο $\mathbf{B_0}$
- 1 Διωνυμικό Δέντρο $\mathbf{B_1}$
- 1 Διωνυμικό Δέντρο $\mathbf{B_2}$

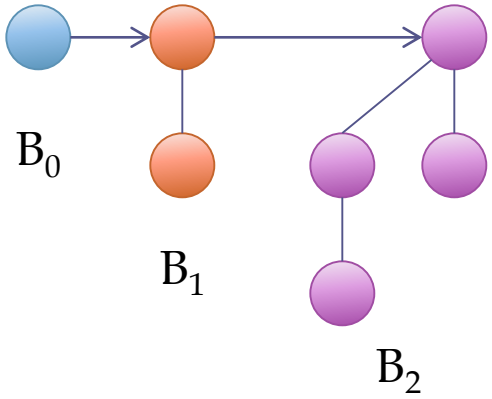


Παράδειγμα - Διωνυμικός Σωρός 7 στοιχείων

$7_{(10)} = 0111_{(2)} \rightarrow 7_{(10)} = 2^2 + 2^1 + 2^0$

Άρα ο Διωνυμικός Σωρός 7 στοιχείων θα αποτελείται από:

- 1 Διωνυμικό Δέντρο $\mathbf{B_0}$
- 1 Διωνυμικό Δέντρο $\mathbf{B_1}$
- 1 Διωνυμικό Δέντρο $\mathbf{B_2}$



Παρατηρούμε ότι

ΚΑΘΕ ΡΙΖΑ ΕΧΕΙ ΔΙΑΦΟΡΕΤΙΚΟ ΒΑΘΜΟ!

Παράδειγμα - Διωνυμικός Σωρός 13 στοιχείων

$13_{(10)} = 1101_{(2)} \rightarrow 13_{(10)} = 2^3 + 2^2 + 2^0$

Παράδειγμα - Διωνυμικός Σωρός 13 στοιχείων

$13_{(10)} = 1101_{(2)} \rightarrow 13_{(10)} = 2^3 + 2^2 + 2^0$

Άρα ο Διωνυμικός Σωρός 13 στοιχείων θα αποτελείται από:

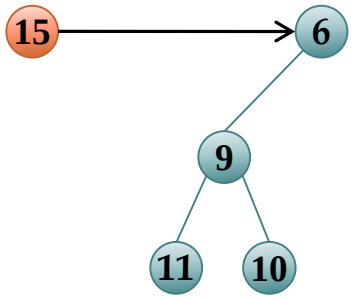
- 1 Διωνυμικό Δέντρο $\mathbf{B_0}$

Παράδειγμα - Διωνυμικός Σωρός 13 στοιχείων

$13_{(10)} = 1101_{(2)} \rightarrow 13_{(10)} = 2^3 + 2^2 + 2^0$

Άρα ο Διωνυμικός Σωρός 13 στοιχείων θα αποτελείται από:

- 1 Διωνυμικό Δέντρο $\mathbf{B_0}$
- 1 Διωνυμικό Δέντρο $\mathbf{B_2}$

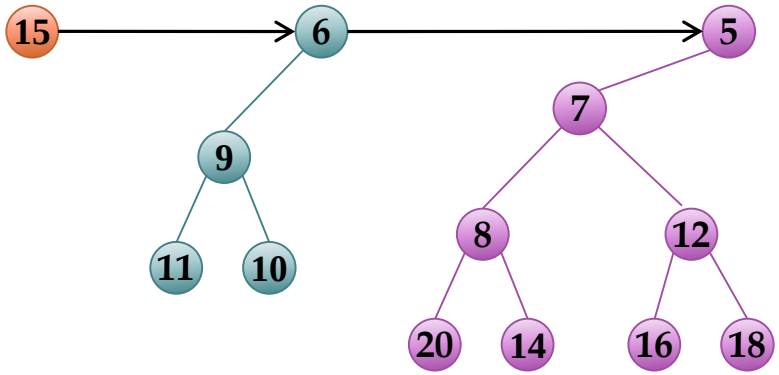


Παράδειγμα - Διωνυμικός Σωρός 13 στοιχείων

$13_{(10)} = 1101_{(2)} \rightarrow 13_{(10)} = 2^3 + 2^2 + 2^0$

Άρα ο Διωνυμικός Σωρός 13 στοιχείων θα αποτελείται από:

- 1 Διωνυμικό Δέντρο $\mathbf{B_0}$
- 1 Διωνυμικό Δέντρο $\mathbf{B_2}$
- 1 Διωνυμικό Δέντρο $\mathbf{B_3}$



Οι τιμές που είναι αποθηκευμένες στους γονείς είναι μικρότερη από τις τιμές στα παιδιά

Αναπαράσταση Διωνυμικών Σωρών

Χρησιμοποιείται η τεχνική **αριστερό παιδί - δεξιός γείτονας,** δηλαδή κάθε κόμβος αποθηκεύει

- **ένα δείκτη για το αριστερό παιδί του**
- **ένα δείκτη για το δεξιό γείτονά του**

Και επιπλέον

- **ένα δείκτη στο γονέα του**
- **ένα πεδίο με τον βαθμό του**
- **ένα πεδίο με την τιμή που έχει αποθηκευμένη σε αυτόν**
- **ένα πεδίο με διάφορα δεδομένα που δεν χρησιμοποιούνται στο σωρό**

Πράξεις σε Διωνυμικούς Σωρούς

1. **MakeQueue( ):** Δημιουργεί έναν κενό σωρό σε Θ(1) χρόνο και μας επιστρέφει ένα δείκτη σε αυτόν το σωρό.
2. **FindMin(Q):** Για να βρούμε το ελάχιστο στοιχείο κάνουμε σειριακή αναζήτηση στη λίστα ριζών και βάζουμε ένα δείκτη στη ρίζα με την μικρότερη τιμή.
3. **Meld(Q_1, Q_2)**: Για να ενώσουμε 2 ουρές (2 διωνυμικούς σωρούς) εκτελούμε 2 στάδια
   - Συγχωνεύουμε τις λίστες ριζών, προσέχοντας κάθε λίστα των ουρών να έχουν μοναδικό βαθμό
   - Μετά τη συγχώνευση οι βαθμοί στη λίστα που προκύπτει εμφανίζονται σε αύξουσα σειρά

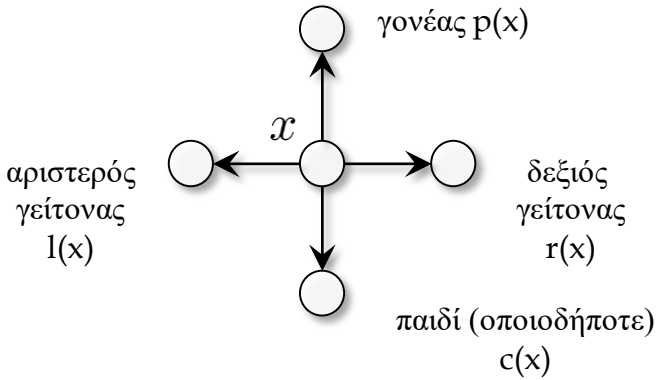
Για πιο εύκολα παραλληλίστε την παραπάνω διαδικασία με την πρόσθεση δυαδικών αριθμών

Ουρά Fibonacci

- Βασίζεται στο Διωνυμικό Σωρό (δηλαδή αποτελεί μια συλλογή από δένδρα) αλλά έχει πιο χαλαρή δομή.
- Καθένα από τα δέντρα ικανοποιεί την ιδιότητα του σωρού.
- Η αναπαράσταση του δάσους γίνεται με μία διπλά συνδεδεμένη κυκλική λίστα των ριζών των δέντρων.

Αναπαράσταση Ουράς Fibonacci

- **Σε κάθε κόμβο x έχουμε:**
  - **d(x): αριθμός παιδιών**
  - **mark(x): boolean πεδίο, 1 όταν ο x χάνει ένα παιδί**
  - **p(x): δείκτης προς τον γονέα**
  - **c(x): δείκτης προς κάποιο παιδί**
  - **l(x) και r(x): δείκτες στον αριστερό και δεξιό του γείτονα αντίστοιχα**

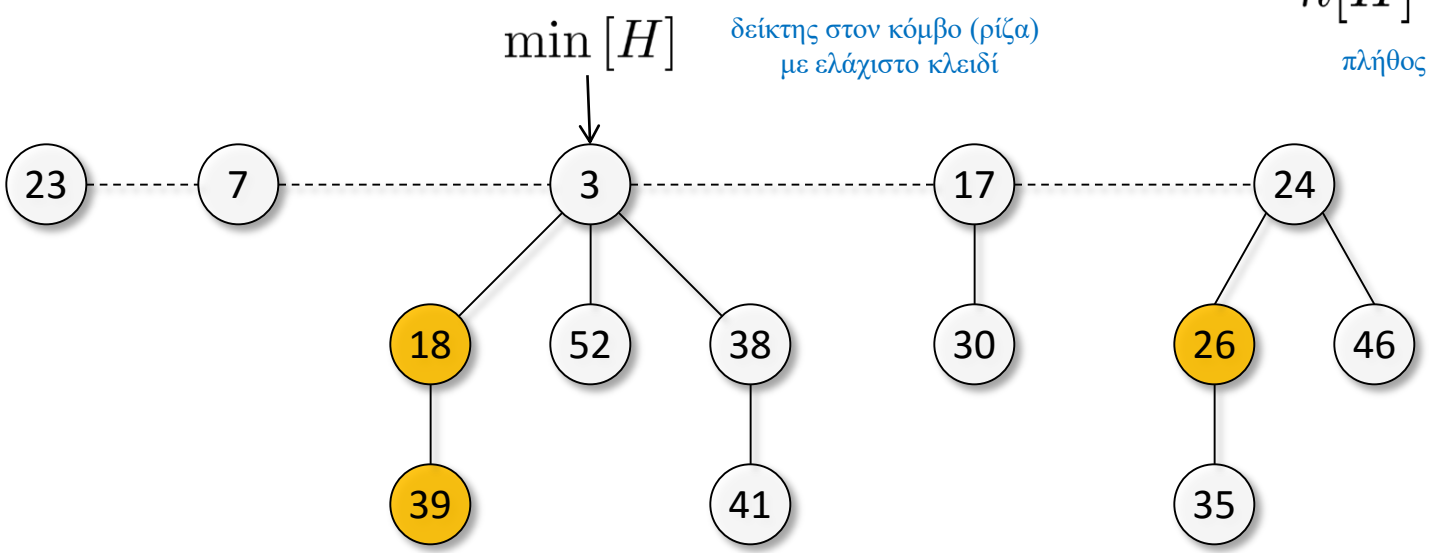


αριθμός παιδιών d(x)
bit επισήμανσης mark(x)

Αναπαράσταση Ουράς Fibonacci

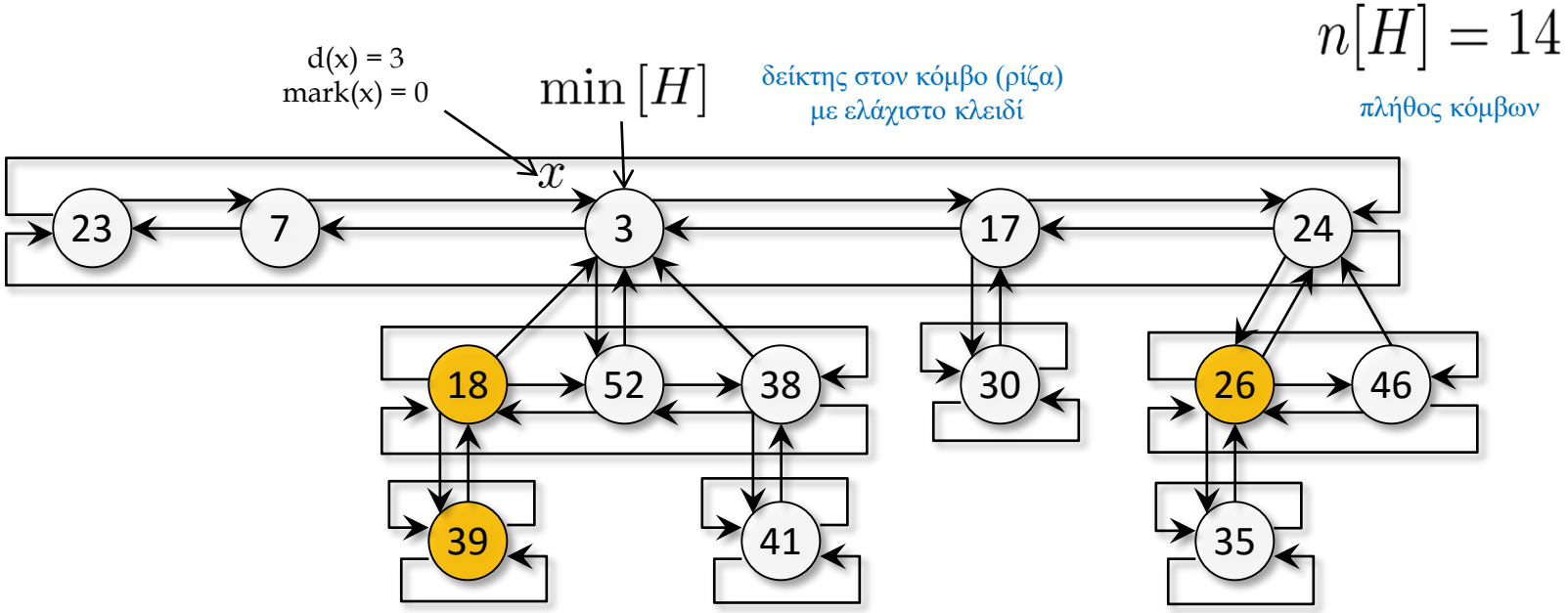
$n[H] = 14$

πλήθος κόμβων



Η σωρός Fibonacci είναι πιο χαλαρή δομή από την διωνυμική σωρό

Αναπαράσταση Ουράς Fibonacci



Η σωρός Fibonacci είναι πιο χαλαρή δομή από την διωνυμική σωρό

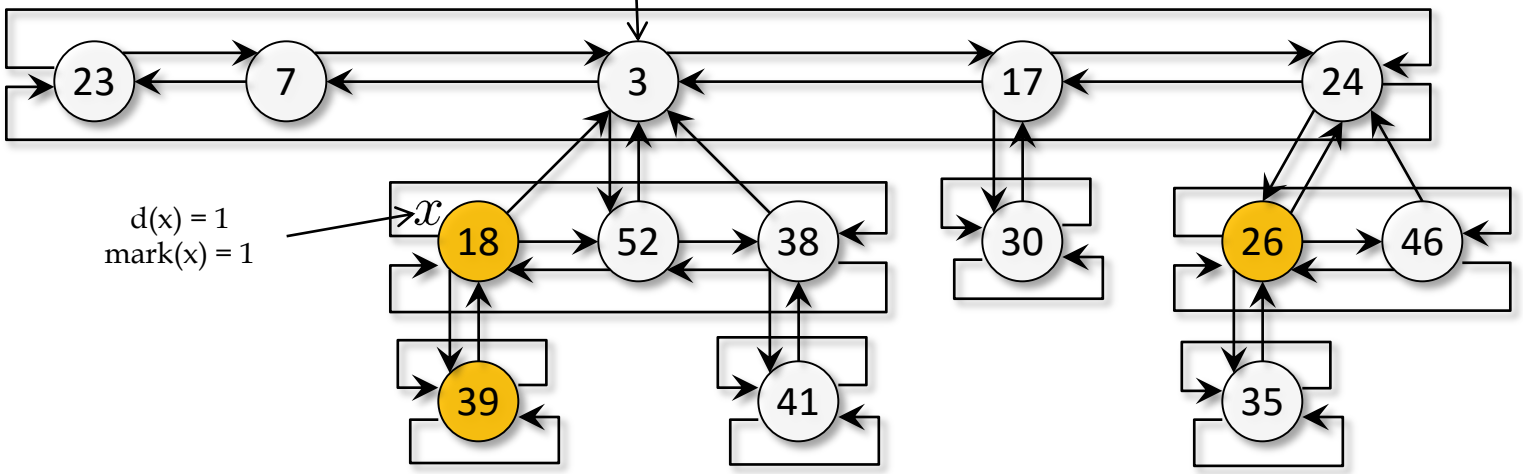
Αναπαράσταση Ουράς Fibonacci

$n[H] = 14$

πλήθος κόμβων

$\min[H]$

δείκτης στον κόμβο (ρίζα) με ελάχιστο κλειδί

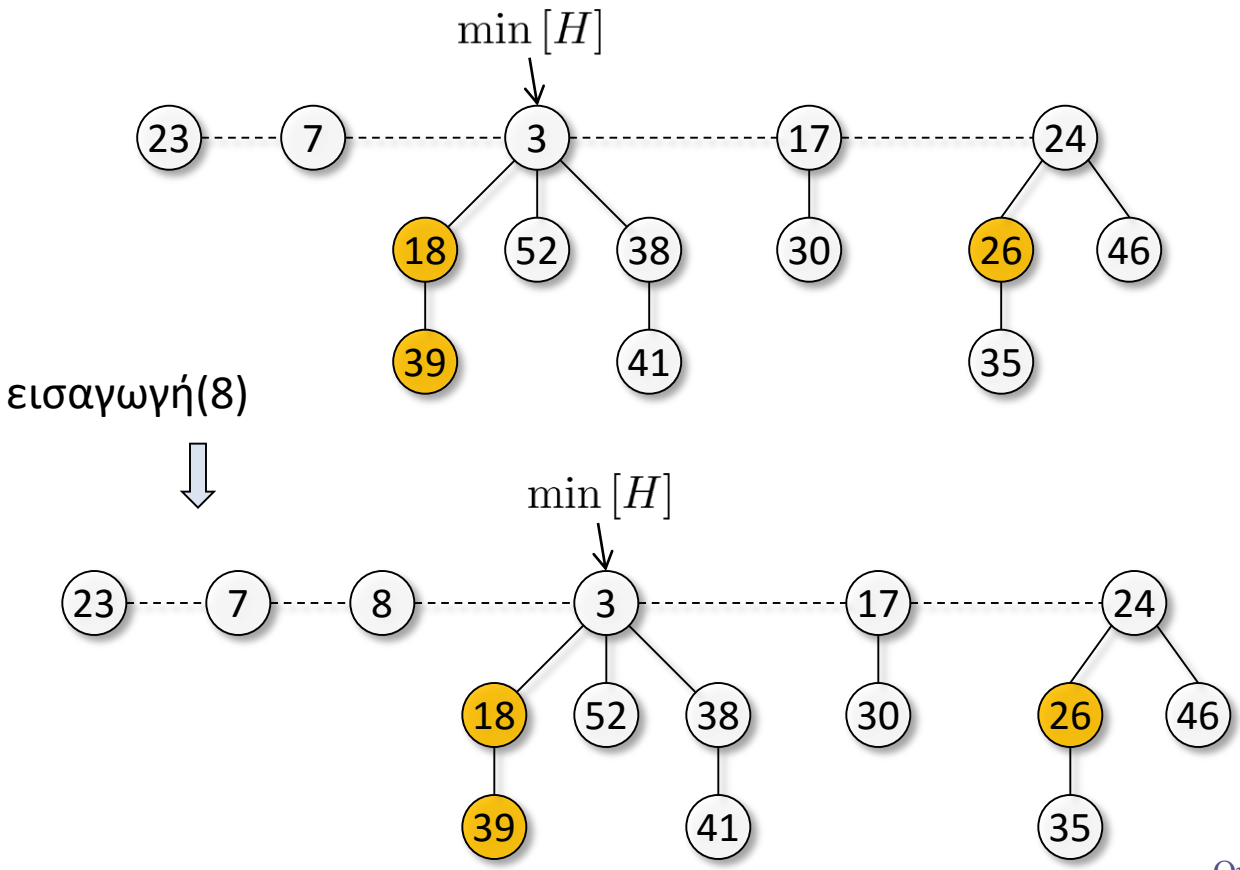


Η σωρός Fibonacci είναι πιο χαλαρή δομή από την διωνυμική σωρό

Πράξεις σε σωρούς Fibonacci

- Όταν θέλουμε να ενώσουμε 2 σωρούς Fibonacci (Meld) ενώνουμε τις λίστες ριζών και καθορίζουμε τον δείκτη ελαχίστου στη νέα δομή που δημιουργήθηκε.
- Όταν θέλουμε να εισάγουμε ένα στοιχείο σε ένα σωρό δημιουργούμε ένα διωνυμικό δέντρο για αυτό το στοιχείο μόνο και μετά το συνενώνουμε με το σωρό Fibonacci που έχουμε ήδη.
- **Επανορθωτικές πράξεις εκτελούμε όταν διαγράφουμε κάποιο στοιχείο από τον σωρό.**

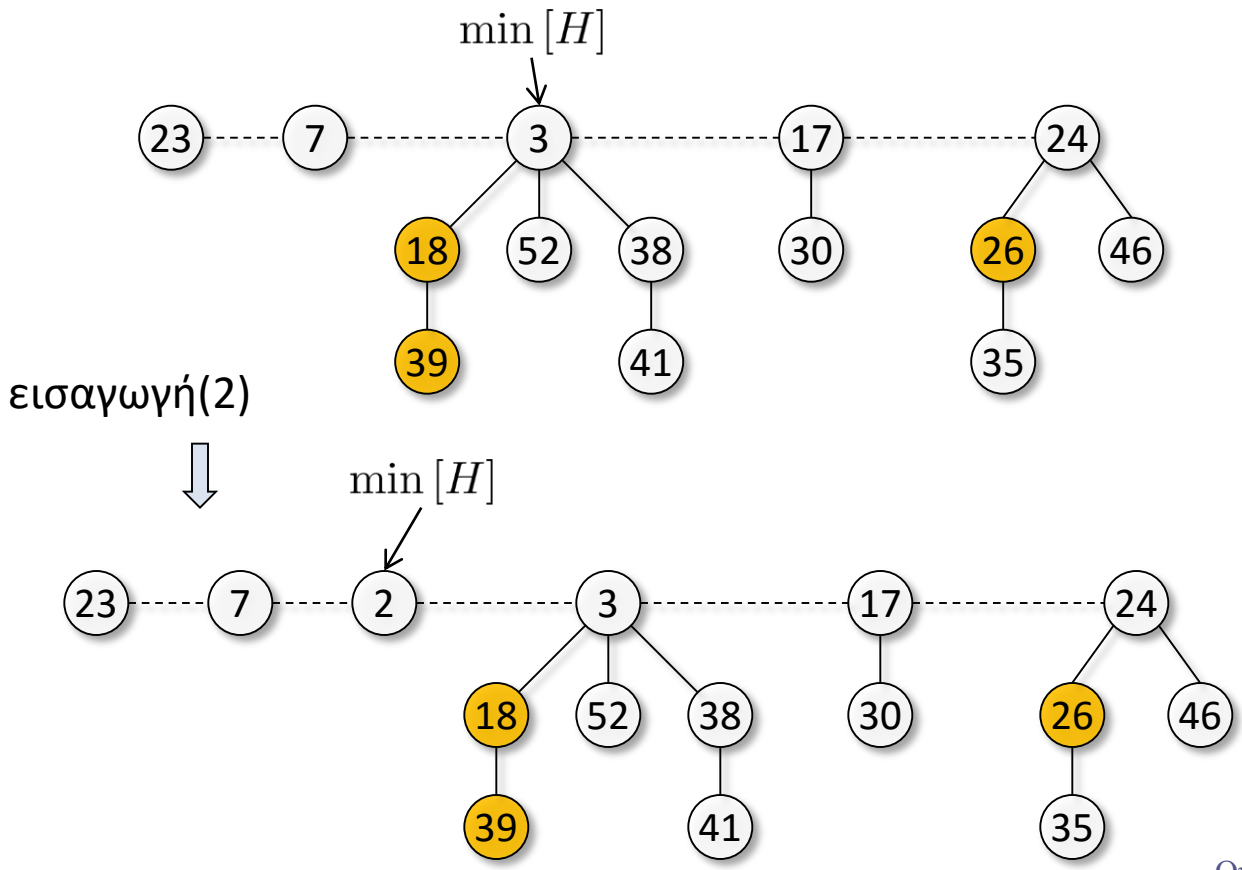
Εισαγωγή κόμβου σε σωρό Fibonacci



Δημιουργείται νέο δένδρο με μόνο ένα κόμβο και εισάγεται στη λίστα των ριζών δίπλα από το min[H]

Αν το εισαγόμενο κλειδί είναι το ελάχιστο τότε ο δείκτης min[H] δείχνει στο νέο κόμβο

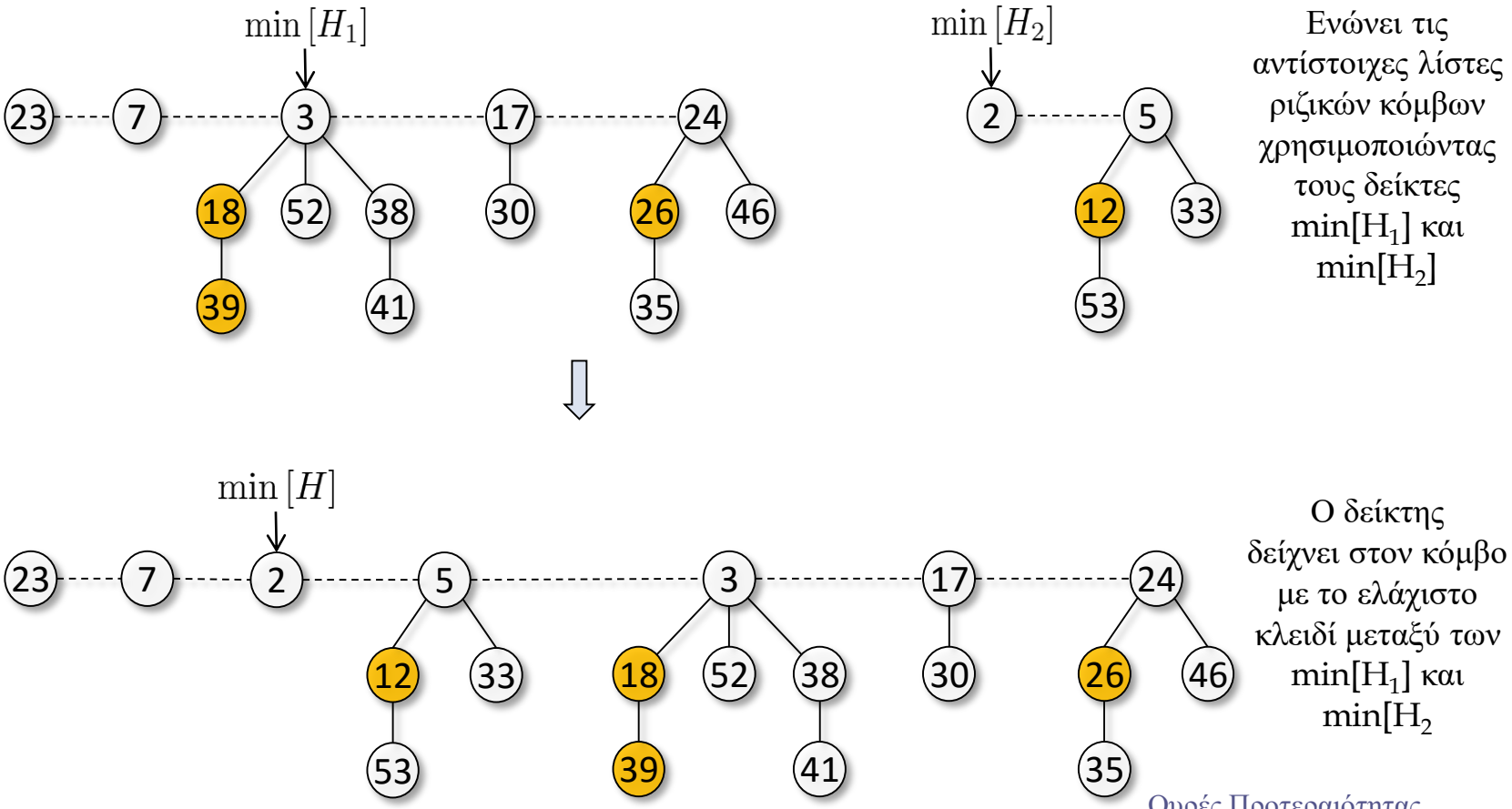
Εισαγωγή κόμβου σε σωρό Fibonacci



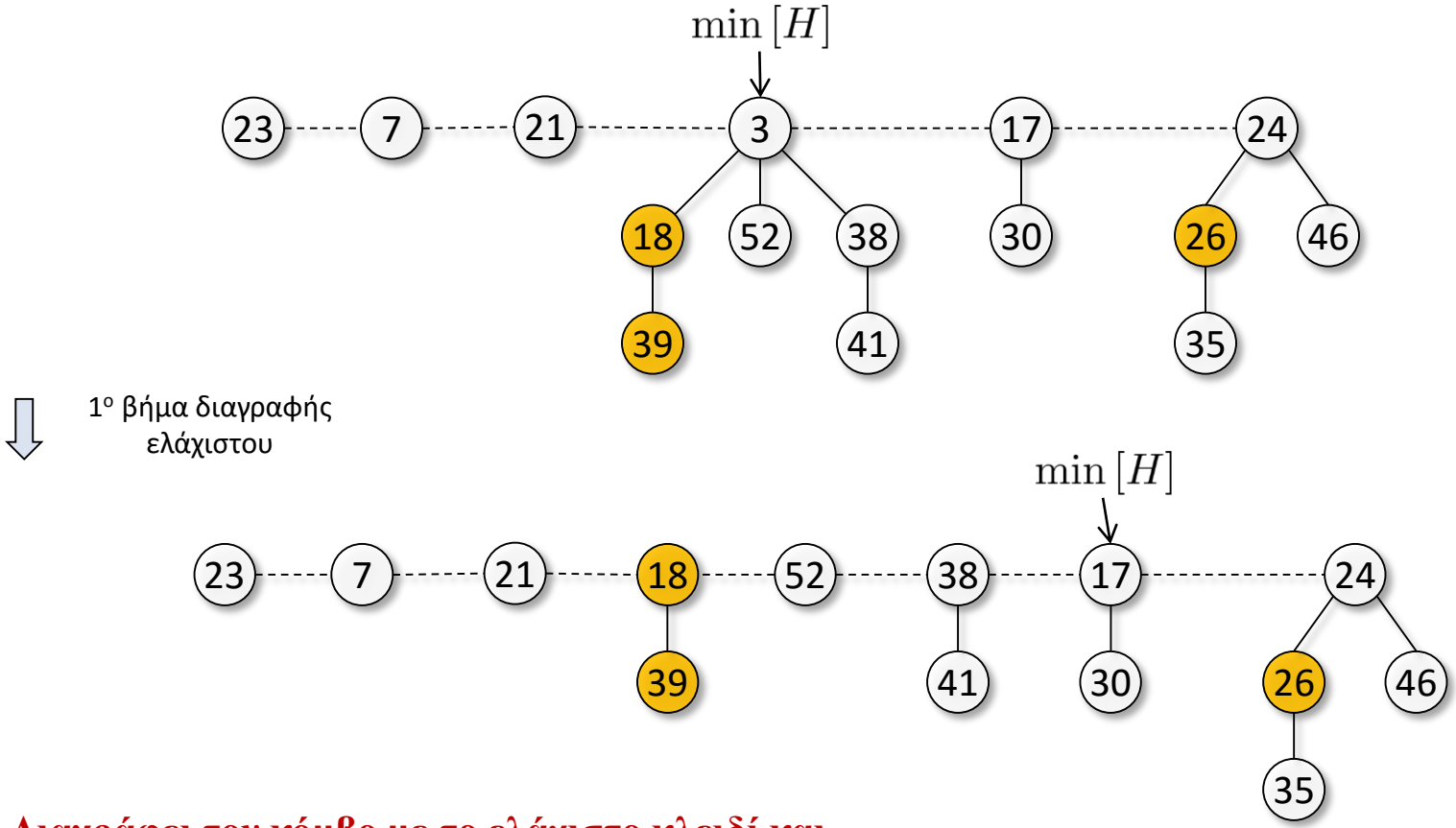
Δημιουργείται νέο δένδρο με μόνο ένα κόμβο και εισάγεται στη λίστα των ριζών δίπλα από το min[H]

Αν το εισαγόμενο κλειδί είναι το ελάχιστο τότε ο δείκτης min[H] δείχνει στο νέο κόμβο

Ένωση δύο σωρών Fibonacci



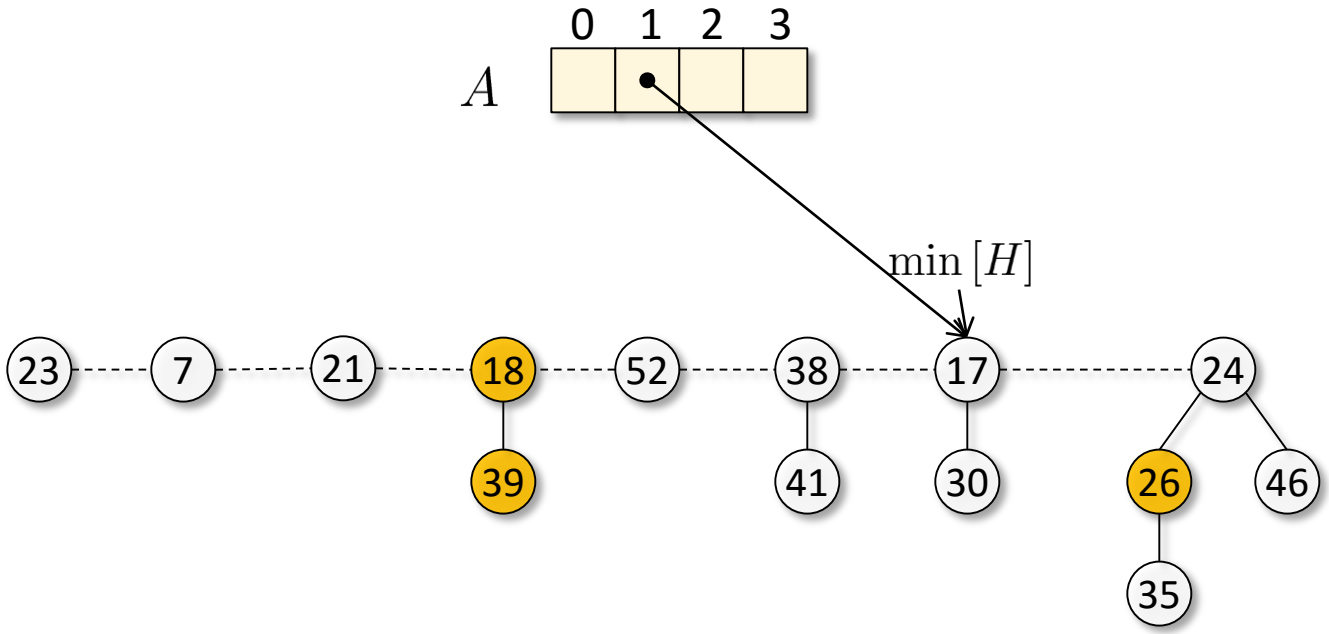
Εξαγωγή ελάχιστου σωρού Fibonacci



Διαγράφει τον κόμβο με το ελάχιστο κλειδί και ενοποιεί δένδρα στο ριζικό επίπεδο

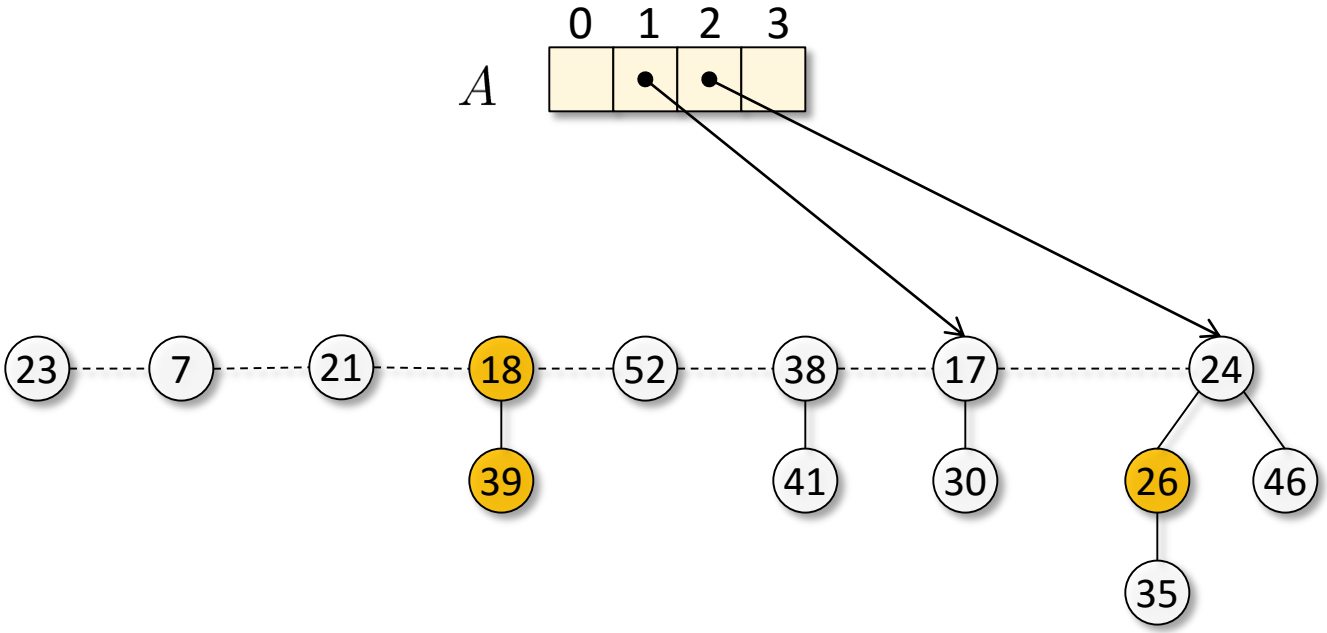
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



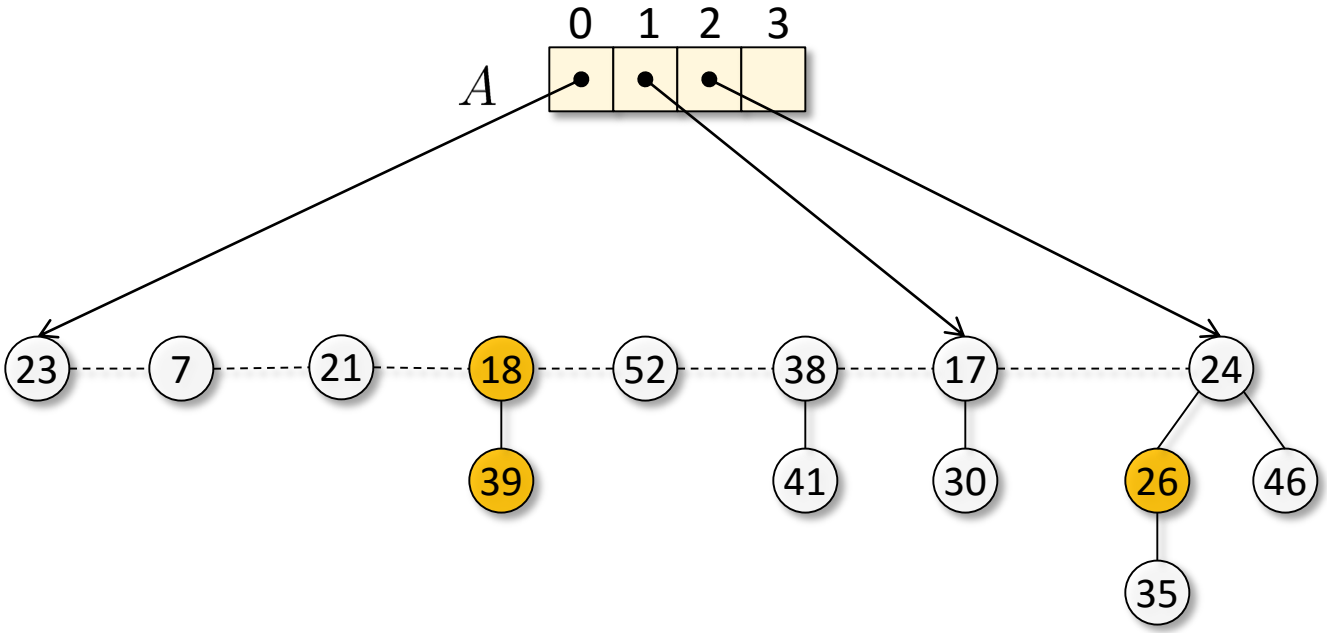
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



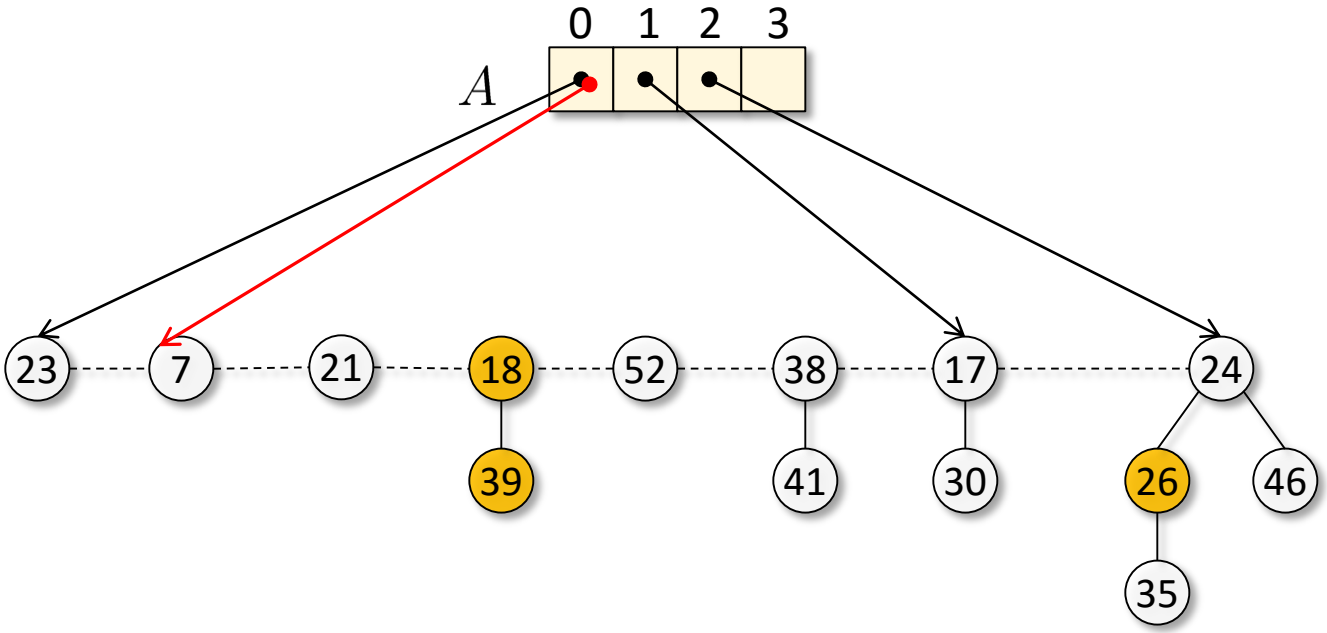
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



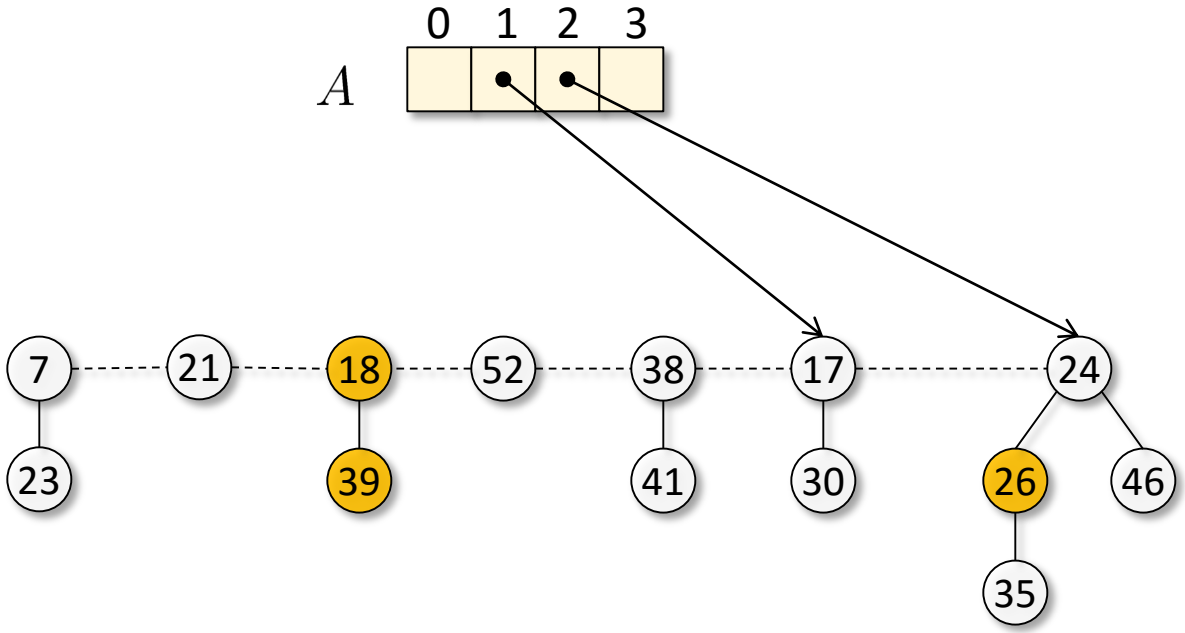
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



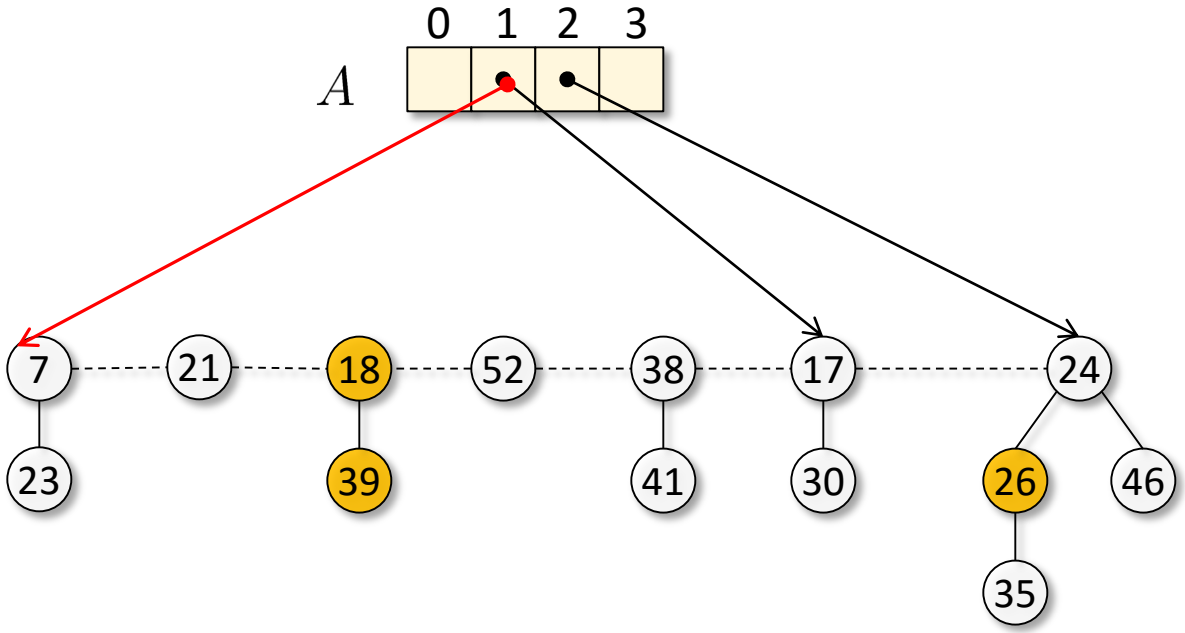
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



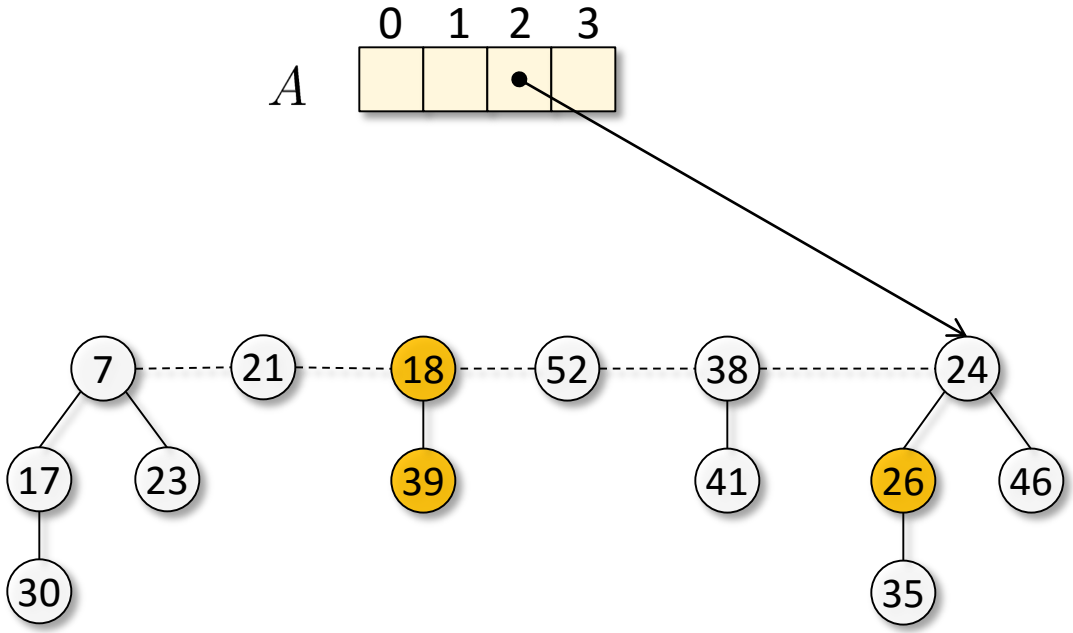
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



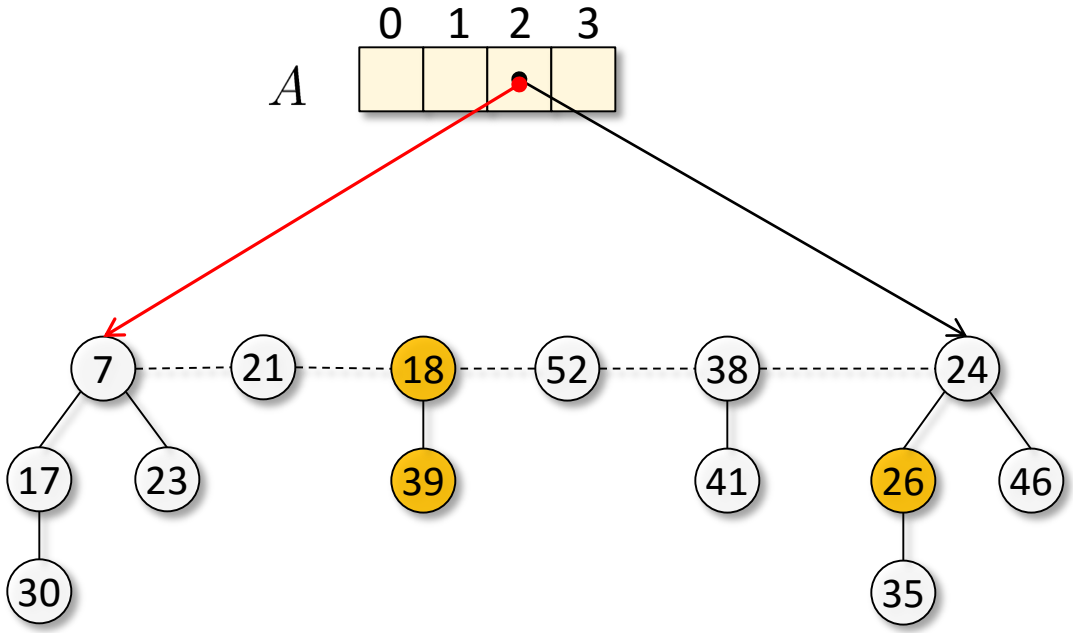
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



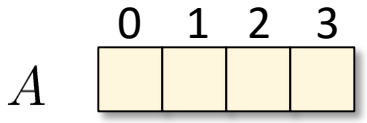
Εξαγωγή ελάχιστου σωρού Fibonacci

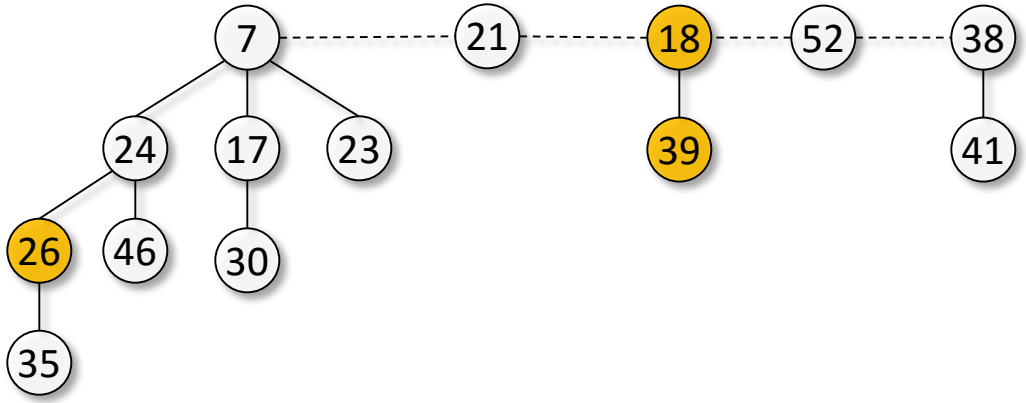
Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



Εξαγωγή ελάχιστου σωρού Fibonacci

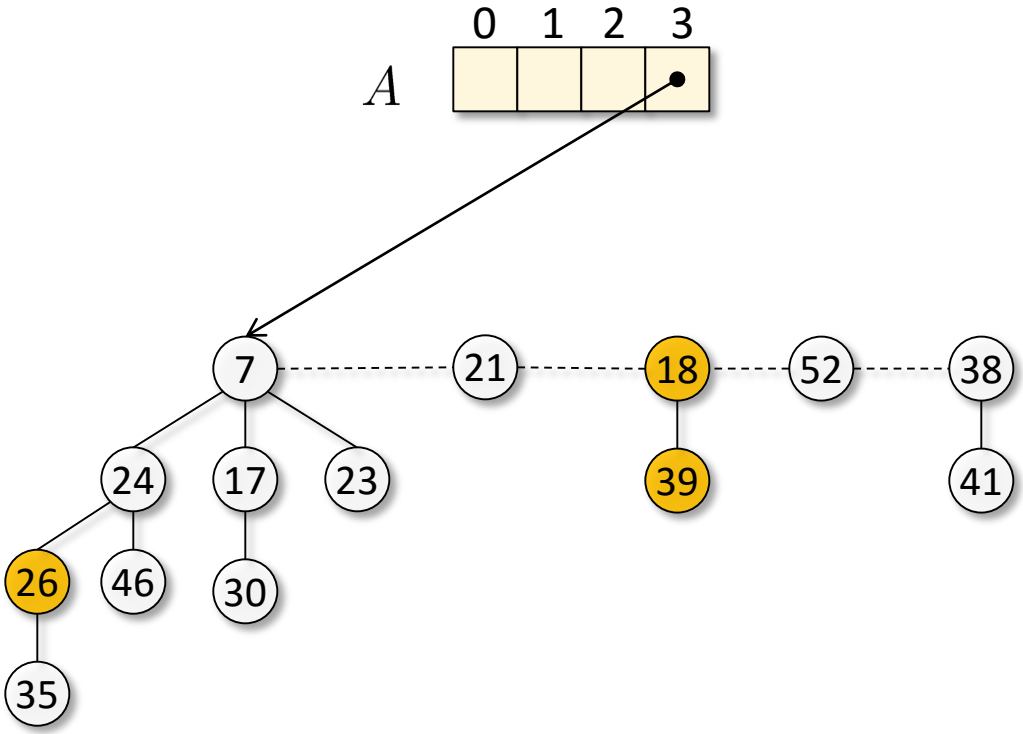
Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών





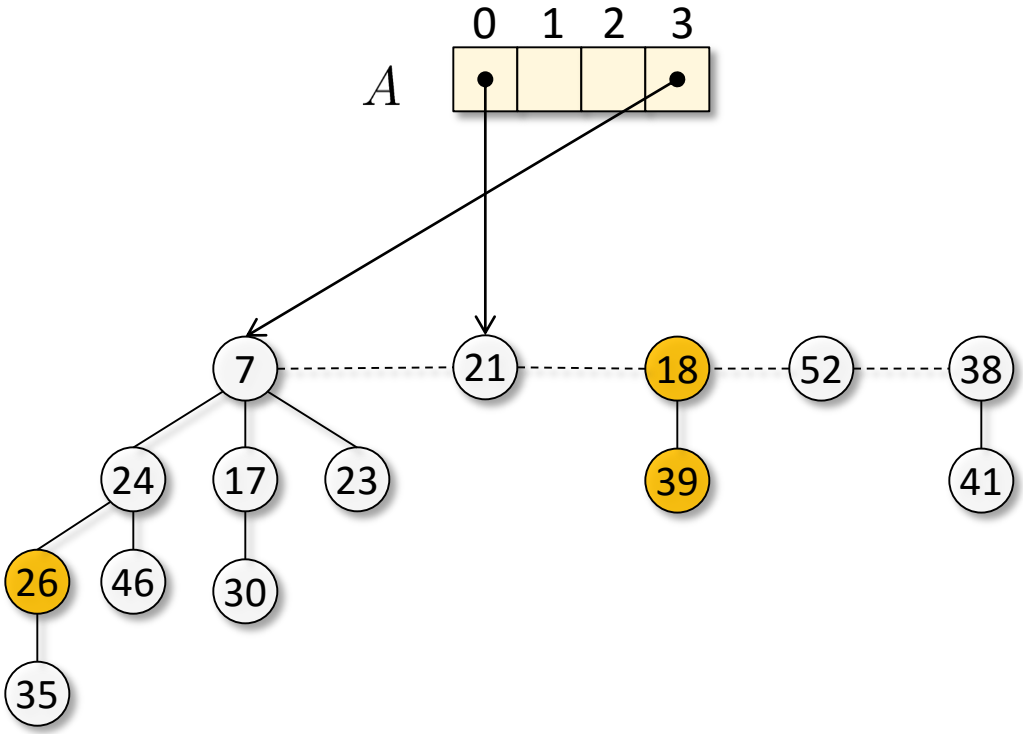
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



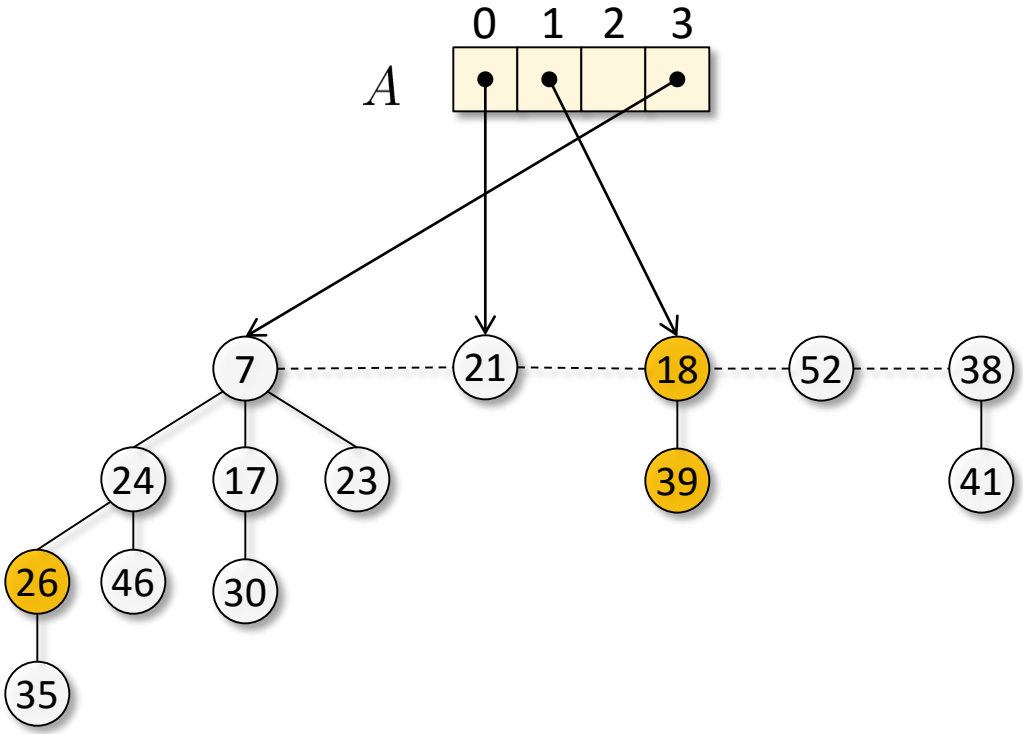
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



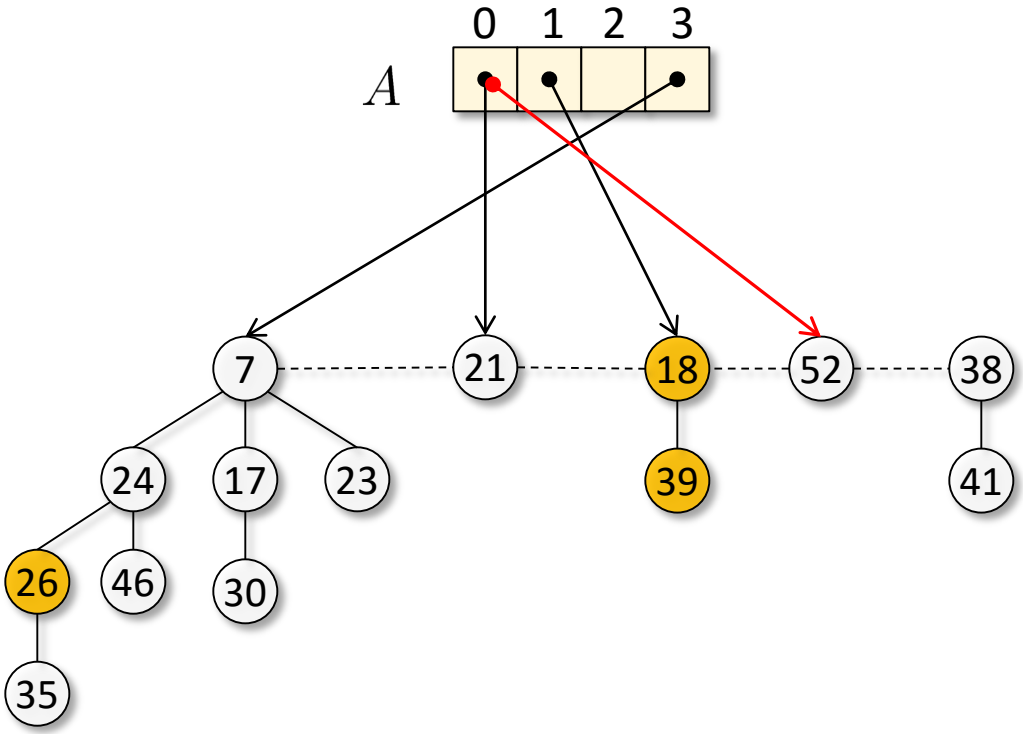
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



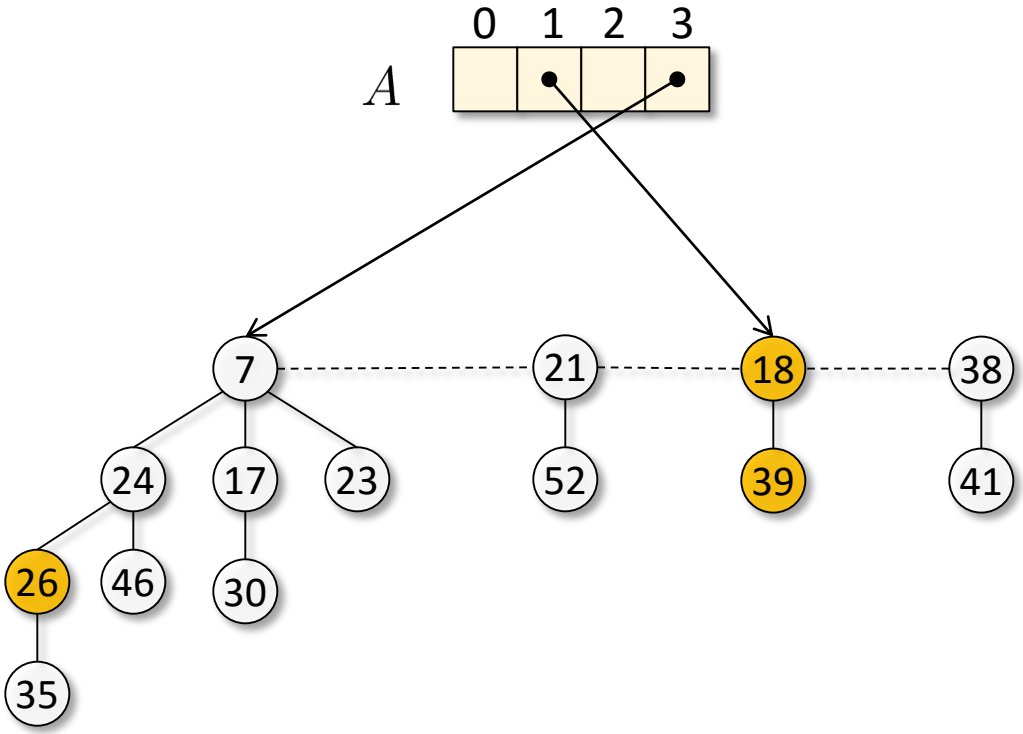
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



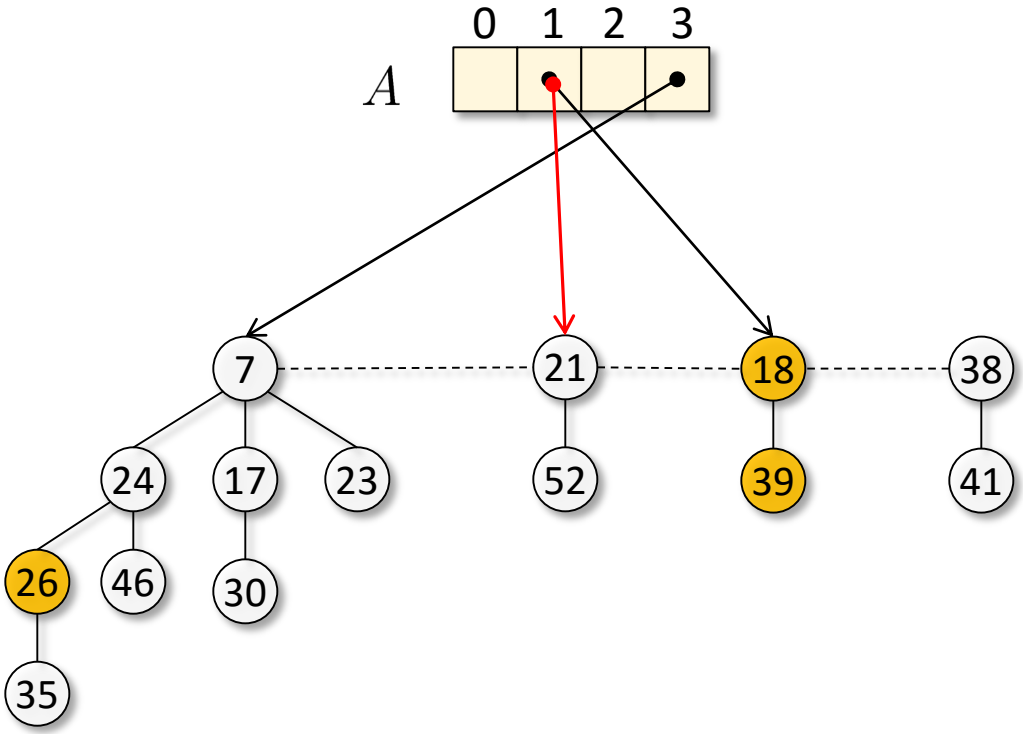
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



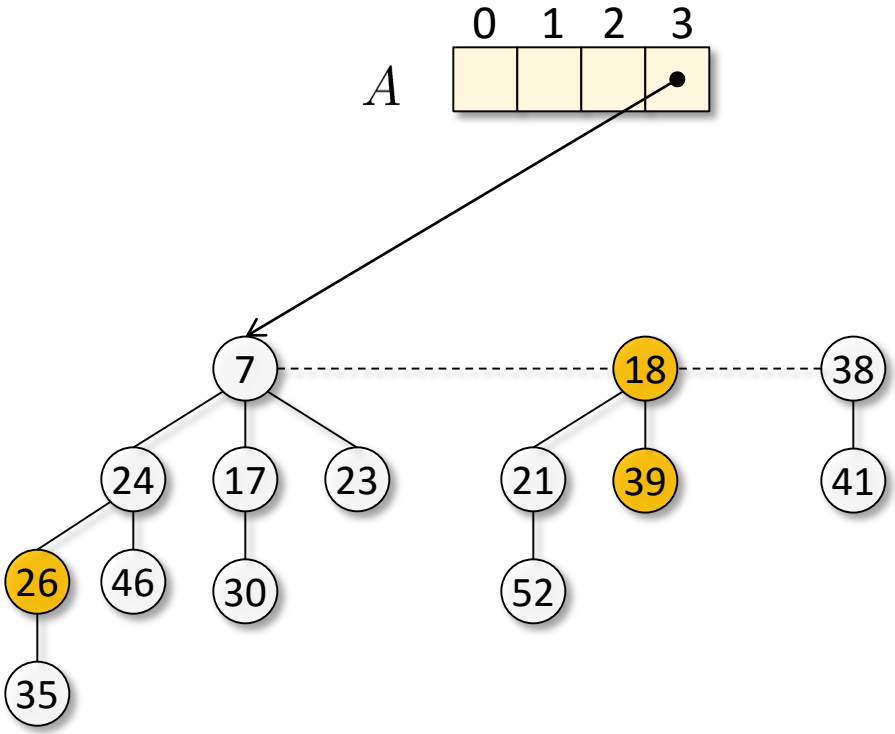
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



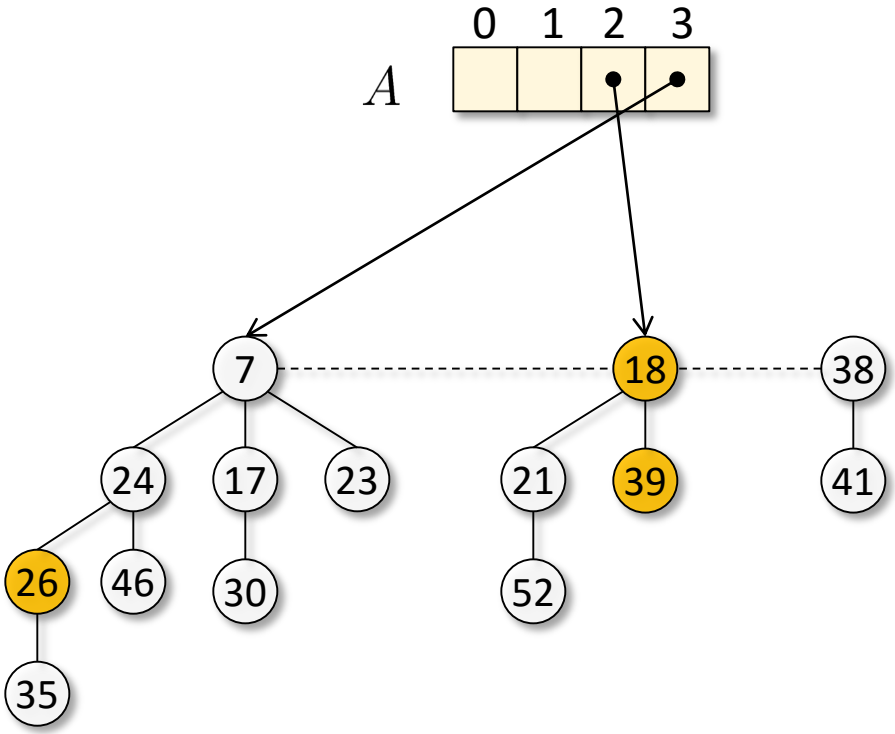
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



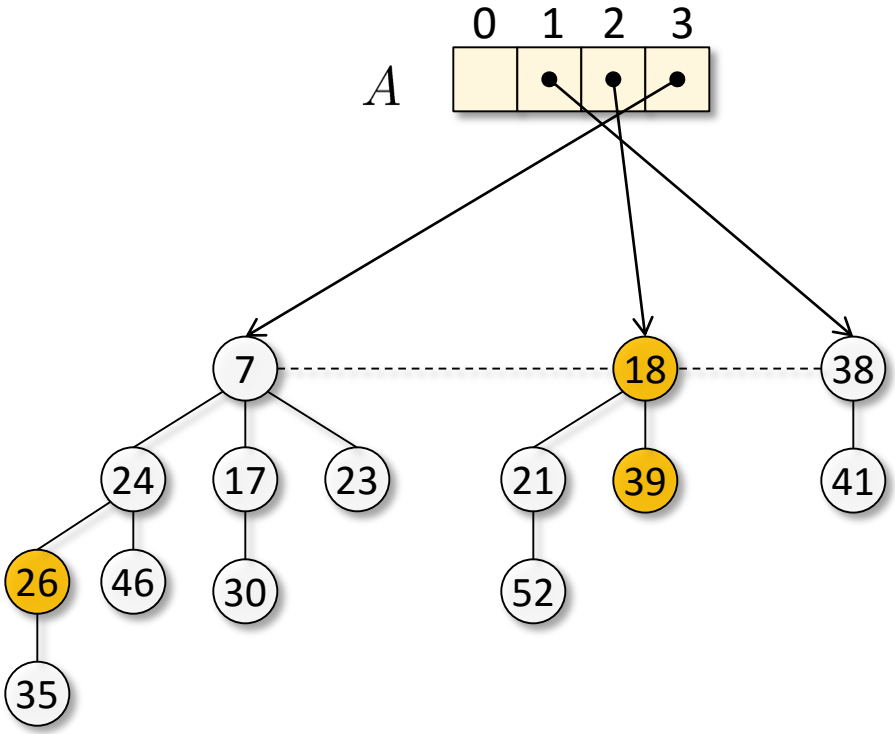
Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών

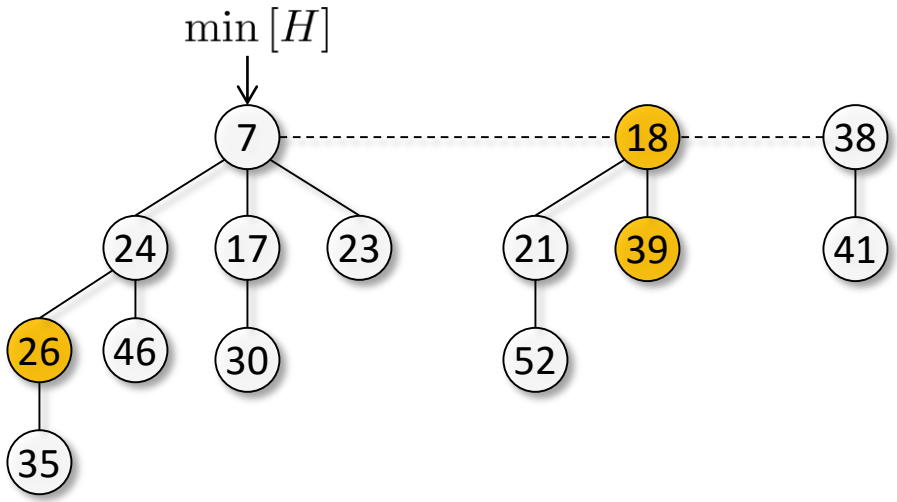


Εξαγωγή ελάχιστου σωρού Fibonacci

Η ρουτίνα ενοποίησης χρησιμοποιεί ένα βοηθητικό πίνακα δεικτών



Εξαγωγή ελάχιστου σωρού Fibonacci

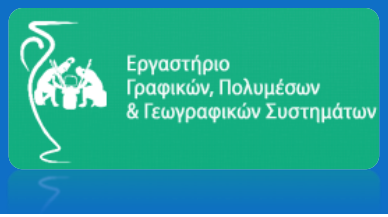


Εισαγωγή στις Διαχρονικές Δομές Δεδομένων

Persistent Data Structures

Δομές Δεδομένων

Μάριος Κενδέα



19 Μαΐου 2015

kendea@ceid.upatras.gr

Εισαγωγή (1/3)

- **Εφήμερες δομές:**
  - Δεν έχουμε πρόσβαση στις προηγούμενες καταστάσεις (εκδοχές) της δομής.
  - Η παλιά έκδοση καταστρέφεται μετά από μία αλλαγή.

- **Διαχρονικές Δομές - Persistent:**
  - Επιτρέπει την πρόσβαση σε όλες τις εκδοχές της δομής.
  - Που χρειαζόμαστε τις διαχρονικές δομές;
    - Επεξεργασία κειμένου και αρχείων.
    - Υπολογιστική Γεωμετρία κ.α.

Εισαγωγή (2/3)

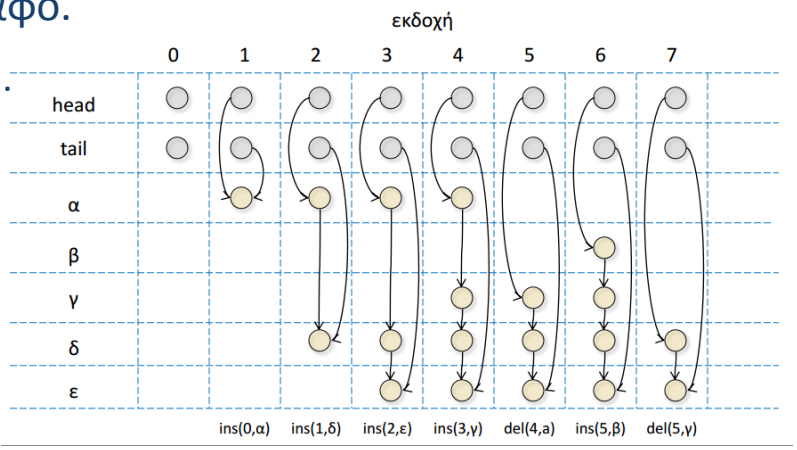
- **Μερική Διαχρονικότητα – Partial Persistent:**
  - Επιτρέπει την πρόσβαση σε όλες τις εκδοχές της δομής, αλλά είναι δυνατόν να τροποποιηθεί μόνο η τελευταία εκδοχή.
- **Πλήρης Διαχρονικότητα – Fully Persistent:**
  - Επιτρέπει τόσο την πρόσβαση όσο την τροποποίηση σε όλες τις εκδοχές της δομής.

Εισαγωγή (3/3)

- Για να υποστηρίξουμε τη διαχρονικότητα της δομής εισάγουμε την ακόλουθη παράμετρο :
  - **Αριθμός εκδοχής:**
    - Η αρχική δομή αποτελεί την εκδοχή 0.
    - Η i-οστή πράξη τροποποίησης δημιουργεί την εκδοχή i.
- **Παράμετροι απόδοσης:**
  - n = αριθμός στοιχείων στην τρέχουσα εκδοχή
  - m = συνολικός αριθμός τροποποιήσεων

Απλοϊκές Λύσεις

1. Κάθε εκδοχή δημιουργεί ένα πλήρες αντίγραφο.
   - Απαιτεί Ω(n) χρόνο και χώρο ανά τροποποίηση.

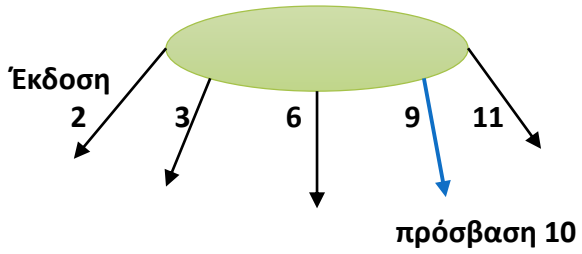


http://www.cs.uoi.gr/~loukas/courses/grad/Data_Structures_and_Algorithms/index.files/Persistence.pdf

2. Κάθε εκδοχή δημιουργεί ένα πλήρες αντίγραφο.
   - Απαιτεί Ω(n) χρόνο και χώρο ανά τροποποίηση.
3. Δεν αποθηκεύουμε καμία εκδοχή, αλλά μόνο την ακολουθία τροποποιήσεων. Για πρόσβαση στην i-οστή εκδοχή κατασκευάζουμε τη δομή έως αυτή την εκδοχή από την αρχή.
   - Απαιτεί Ω(i) χρόνο για την κάθε πρόσβαση στην i-οστή εκδοχή.
4. Συνδυασμός των δύο παραπάνω

Μερική Διαχρονικότητα (1/2)

- **Μέθοδος παχιού κόμβου (Fat Node)**
  - Κάθε κόμβος αποθηκεύει αυθαίρετο αριθμό από τιμές (ετικέτες έκδοσης)
  - Έχει ετικέτα που υποδεικνύει την έκδοση που δημιουργήθηκε ο ίδιος
  - Εύρεση της i-οστής έκδοσης:
    - Ξεκινάμε από τον κατάλληλο δείκτη πρόσβασης της έκδοσης
    - Όταν βρισκόμαστε σε ένα διαχρονικό κόμβο P και θέλουμε να ανακτήσουμε μία τιμή ενός πεδίου αναζητούμε την τιμή του πεδίου με μέγιστη ετικέτα <=i.
      - O(log m) χρόνος για m εκδόσεις αν οι τιμές εκδόσεων είναι οργανωμένες σε δυαδικό δέντρο)
      - Χωρική Επιβάρυνση: O(1) ανά έκδοση



Μερική Διαχρονικότητα (2/2)

- **Μέθοδος αντιγραφής κόμβων (Node Copying)**
  - Καλύτερη χρονική επιβάρυνση κατά την διάρκεια της προσπέλασης.
  - Σε αυτή την τεχνική, ο κάθε κόμβος έχει σταθερό χώρο.
  - Όταν γίνεται μία αλλαγή έκδοσης
    - Καταγράφεται στον κόμβο αν έχει χώρο
    - Αλλιώς δημιουργείται ένας νέος κόμβος που περιέχει μόνο την τελευταία έκδοση του κόμβου.
  - Πλέον η ιστορικότητα καταγράφεται σε λίστα από κόμβους.
  - Δημιουργούνται δείκτες από τους προηγούμενους- προγόνους στον νέο κόμβο.
    - Αν δεν έχουν χώρο δημιουργούνται αντίγραφα.
  - Χρονικό κόστος:
    - Αναζήτηση και Ενημέρωση O(1) επιμερισμένη
    - Μετακίνηση στις εκδόσεις είναι O(1)
  - Χωρική Επιβάρυνση:
    - O(1)

Πλήρης Διαχρονικότητα

- **Χρήση μεθόδου fat node**
  - Χρονικό κόστος:
    - O(log m) επιβάρυνση για πρόσβαση
    - και O(1) για ενημέρωση
  - Χωρικό κόστος: O(1)
- **Χρήση μεθόδου διάσπασης κόμβου**
  - Χρονικό κόστος:
    - O(1) επιμερισμένη επιβάρυνση για αναζήτηση
    - και O(1) για ενημέρωση
  - Χωρικό κόστος: O(1)

Union – Find

Δομές Δεδομένων

Μάριος Κενδέα

Εργαστήριο
Γραφικών, Πολυμέσων
& Γεωγραφικών Συστημάτων

19 Μαΐου 2015

kendea@ceid.upatras.gr

ΤΟ ΠΡΟΒΛΗΜΑ

- Πως χειριζόμαστε τις διαμερίσεις πάνω στα στοιχεία του σύμπαντος U;
- Έστω ότι $U = \{0,1, \ldots, N-1\}$, $x \in U$ και A,B,C ονόματα συνόλων
  - Find(x): Επιστρέφει το όνομα του συνόλου στο οποίο ανήκει το x.
  - Union(A, B, C): Ενώνει τα σύνολα A και B κάτω από το κοινό όνομα C.

Υλοποίηση Find (1)

- **Απλή Λύση**
  - Για κάθε στοιχείο έχουμε το σύνολο που ανήκει
  - Πράξη Union(A,B,C) -> απλή με κόστος O(N)

0	A
1	A
2	B
3	F
.	.
.	.
.	.
N-1	B

Υλοποίηση Find (2)

- **Χρήση ανεστραμμένων λιστών**
  - Μία για κάθε σύνολο
  - $L(A) = \{0,1, \dots\}$
  - $L(B) = \{2, \dots, N-1\}$
- Find είναι το ίδιο με πριν
- Union εκτελείται σε χρόνο $O(\{A \cup B\})$
  - Πιο αποδοτικό αλλαγή μόνο στο όνομα των στοιχείων του μικρότερου συνόλου στο όνομα του άλλου εσωτερικά
  - Και κρατάμε και απεικόνιση για τον εξωτερικό παρατηρητή
    - MAPOUT: πχ το B εσωτερικά είναι το C εξωτερικά
    - MAPIN: πχ το C εξωτερικά είναι το B εσωτερικά
      - Γιατί στο union(A,B,C)
        - Τα A έγιναν B
        - Και κρατάμε ότι το B είναι το C

0	A
1	A
2	B
3	F
.	.
.	.
.	.
N-1	B

Θεώρημα Αποδοτικού Find

- **Μία ακολουθία από**
  - **n-1 Unions και (Union κόστος επιμερισμένο O(log n))**
  - **m Find (Find κόστος O(1))**
  - **σε σύνολα αρχικού μεγέθους 1**
    - **έχει κόστος $O(m + n\ logn)$**
      - Απόδειξη στο βιβλίο

Αποδοτική Υλοποίηση Union

- Θυσιάζει χρόνο Find για σταθερό χρόνο στο Union
- 3 πίνακες
  - Father [1…N] : το i-οστό στοιχείο δείχνει τον πατέρα j στον ίδιο πίνακα
  - Name [1…N] : το i-οστό στοιχείο δείχνει την ομάδα του i αν είναι η ρίζα στην ομάδα αλλιώς κενό
  - Root [] : το i-οστό στοιχείο του πίνακα δείχνει την ρίζα της ομάδας με όνομα i.
- Find(x): αναζήτηση πατέρα μέχρι πριν να καταλήξουμε σε κενό. Τότε εκτυπώνουμε το αντίστοιχο όνομα
- Union(A,B,B) : i <- Root[A], j <- Root[B], Father[i] <- j

Weighted Union Rule

- Για αποφυγή μεγάλων βαθών δέντρων
- Η ρίζα μικρότερου γίνεται γιός του μεγαλύτερου
- Χρειάζεται πίνακας size[1…N]: i-οστή θέση πλήθος στοιχείων με ρίζα το στοιχείο i.
- Εφαρμογή: Οι αριθμοί στις παρενθέσεις δηλώνουν τον αριθμό των στοιχείων κάθε συνόλου. Τα κεφαλαία είναι ονόματα συνόλων
  1. U(1,2,A)
  2. U(3,4,B)
  3. U(A,B,C)
  4. U(5,6,D)
  5. U(7,8,E)
  6. U(D,C,F)
  7. U(E,F,G)