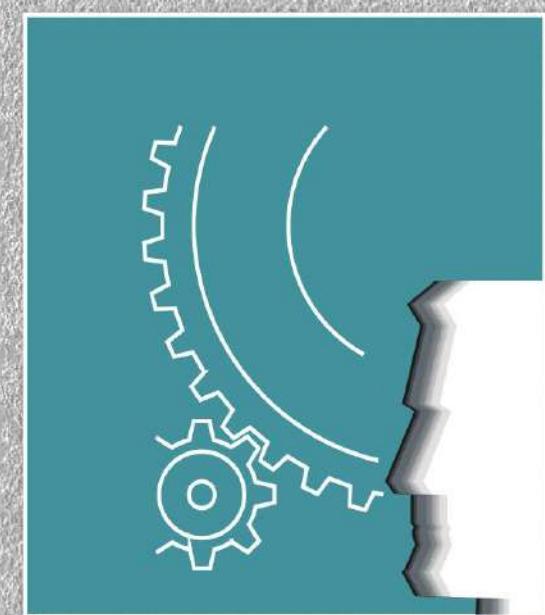


ΣΗΜΕΙΩΣΕΙΣ ΣΤΙΣ ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ



ανάλυση
Σ Ο Π Ρ Σ Τ Ε Ρ

ΕΡΓΑΣΤΗΡΙΟ ΕΛΕΥΘΕΡΩΝ ΣΠΟΥΔΩΝ
ΣΤΗΝ ΕΦΑΡΜΟΣΜΕΝΗ ΠΛΗΡΟΦΟΡΙΚΗ

*...λυση
σπουδων*

ΑΓ. ΑΝΔΡΕΟΥ 130-132 & ΓΟΥΝΑΡΗ, Τ.Κ. 26222, ΠΑΤΡΑ
ΤΗΛ: 310-097, 324-900 - ΤΗΛ/FAX: 313-547
E-MAIL: jianelisj@otenet.gr, jianelisk@otenet.gr

Computer – Ανάλυση

Δομές Δεδομένων –Computer Ανάλυση
Περιεχόμενα

1	ΜΟΝΤΕΛΟ RAM KAI PM.....	20
1.1	Θέμα 1 Ιούνιος 2010.....	20
1.2	ΠΙΝΑΚΕΣ.....	21
1.3	ΣΤΟΙΒΑ.....	22
1.4	Πιθανό Θέμα με Στοίβα για σωστό ταίριασμα παρενθέσεων και αγκυλών	23
2	ΟΥΡΑ.....	26
3	ΛΙΣΤΑ.....	28
4	ΔΕΝΤΡΑ.....	29
4.1	Ορισμός Δέντρου.....	29
4.2	Χαρακτηριστικά Δυαδικού Δέντρου.....	29
4.2.1	Πράξεις Δυαδικών Δέντρων	29
4.3	Πιθανό Θέμα με Ακμές και Κορυφές Δέντρου	30
4.4	Θέμα Σεπτέμβριος 2018	30
4.5	Διασχίσεις Δέντρων.....	31
4.5.1.1	Θέμα 4 Ιούνιος 2008.....	31
4.5.1.2	Θέμα 1 Ιούνιος 2009.....	31
4.6	Ανακατασκευή Δέντρου από Διασχίσεις.....	35
4.6.1	Πιθανό Θέμα 1 με Κατασκευή Δέντρου από τις Διατρέξεις του	35
4.7	Πιθανό Θέμα 2 με Κατασκευή Δέντρου από τις Διατρέξεις του	41
4.8	Θέμα 3 Ιούνιος 2018 με Κατασκευή Δέντρου από τις Διατρέξεις του	43
4.9	Θέμα 3 Σεπτέμβριος 2018 με Κατασκευή Δέντρου από τις Διατρέξεις του	43
4.10	Πιθανό θέμα με Κώδικα Διαπεράσεων Δέντρου	52
4.11	Πιθανό θέμα Διαπεράσεων	53
4.11.1	Πιθανό θέμα Εύρεσης Μέγιστου Βάθους ή Ύψους ενός δέντρου	53
4.12	Κατασκευή Φυλλοπροσανατολισμένου Δέντρου Εύρεσης.....	54
4.12.1	Θέμα 2 Ιούνιος 2009.....	54
4.12.2	Δυαδικά Δέντρα Αναζήτησης - Εύρεσης ((Binary Search Tree – BST).....	55
4.12.3	Παράδειγμα 1 με Κατασκευή Δυαδικού Δέντρου Αναζήτησης.....	55
4.12.4	Παράδειγμα 2 με Κατασκευή Δυαδικού Δέντρου Αναζήτησης.....	55
4.12.5	Περιπτώσεις Διαγραφών σε Δυαδικά Δέντρα Αναζήτησης.....	57
4.12.6	Παράδειγμα με Διαγραφή στοιχείων από Δυαδικό Δέντρα Αναζήτησης.....	58
4.12.7	Θέμα 4 Ιούνιος 2019.....	59
4.12.8	Βασικές πράξεις σε Φυλλοπροσανατολισμένα Δέντρα Αναζήτησης	61
5	ΤΑΞΙΝΟΜΗΣΗ.....	63
5.1	Τι ονομάζεται ταξινόμηση. Σε ποιες κατηγορίες χωρίζονται οι αλγόριθμοι ταξινόμησης; Ποιοι αλγόριθμοι ανήκουν σε κάθε κατηγορία;.....	63
5.2	Ταξινόμηση Φυσαλίδας (Bubble Sort)	63
5.2.1.1	Κώδικας Bubble Sort.....	63
5.2.1.2	Βελτιωμένος Κώδικας Bubble Sort	63
5.2.2	Παράδειγμα Εφαρμογής BubbleSort	64
5.3	Ταξινόμηση με Εισαγωγή (Insertion Sort)	65
5.3.1.1	Κώδικας Insertion Sort	65
5.3.1.2	Παράδειγμα Εφαρμογής InsertionSort	65
5.4	Ταξινόμηση με Επιλογή (Selection Sort)	66

Δομές Δεδομένων –Computer Ανάλυση

5.4.1.1	Κώδικας Selection Sort.....	66
5.4.2	Παράδειγμα Εφαρμογής SelectionSort	66
5.5	Ταξινόμηση με Συγχώνευση (MergeSort)	67
5.5.1.1	Παράδειγμα 1 Εφαρμογής MergeSort.....	67
5.5.1.2	Θέμα 1 Ιούνιος 2019.....	68
5.5.1.3	Κώδικας MergeSort.....	68
5.6	Ταξινόμηση Σωρού (HeapSort)	70
5.6.1	Παράδειγμα Εφαρμογής HeapSort	70
5.6.2	Ψευδοκώδικας για τον αλγόριθμο HeapSort	86
5.7	Γρήγορος Αλγόριθμος (Quicksort)	88
5.7.1.1	Θέμα 3 Ιούνιος 2017 και Σεπτέμβριος 2018.....	88
5.7.1.2	Θέμα 1 Ιούνιος 2019.....	92
5.7.2	Θέμα 3 Σεπτέμβριος 2016 και Φεβρουάριος 2019	94
5.8	Πολυπλοκότητες Αλγορίθμων Ταξινόμησης	102
6	ΑΝΑΖΗΤΗΣΗ	103
6.1	Σειριακή (Γραμμική) Αναζήτηση.....	103
6.2	Δυαδική Αναζήτηση (Binary Search).....	104
6.3	Αναζήτηση Παρεμβολής (Interpolation Search).....	106
6.4	Θέμα 1 Ιούνιος 2018 με Αναζήτηση Παρεμβολής.....	107
6.5	Θέμα 5 Σεπτέμβριος 2018 με Αναζήτηση Παρεμβολής.....	111
6.6	Δυική Αναζήτηση Παρεμβολής (Binary Interpolation Search - BIS).....	114
6.7	Θέμα 3 Ιούνιος 2019 με Δυαδική Αναζήτηση Παρεμβολής.....	Error! Bookmark not defined.
6.8	Θέμα 7 Ιούνιος 2009 με Δυαδική Αναζήτηση Παρεμβολής.....	119
6.9	Θέμα 12 Ιούνιος 2008 με Δυαδική Αναζήτηση Παρεμβολής.....	121
7	ΑΝΑΠΑΡΑΣΤΑΣΗ ΓΡΑΦΩΝ ΜΕ ΠΙΝΑΚΑ ΚΑΙ ΛΙΣΤΑ ΓΕΙΝΙΑΣΗΣ.....	123
7.1	Αναπαράσταση με Πίνακες.....	123
7.2	Αναπαράσταση με Λίστες Γειτνίασης	123
7.3	Παραδείγματα Αναπαράστασης Γράφων με Λίστες Γειτνίασης και Πίνακες	124
8	ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ	125
8.1	Θέμα 6 Σεπτέμβριος 2016 και Ιούνιος 2017.....	128
8.2	Θέμα 3-Ιούνιος 2011.....	131
8.3	Θέμα 9-Ιούνιος 2008.....	135
8.4	Θέμα 7 Σεπτέμβριος 2018 σε Extensible Hashing	144
9	UNION FIND	150
9.1	Θεωρία για weighted-union rule.....	150
9.2	Θέμα 8 Σεπτέμβριος 2016 και Ιούνιος 2017.....	151
9.3	Θέμα 8 Ιούνιος 2008, Θέμα 8 Ιούνιος 2009, Θέμα 3 Ιούνιος 2010, Θέμα 5 Ιούνιος 2011 και Θέμα 5 Ιούνιος 2015.....	153
10	AVL ΔΕΝΤΡΑ.....	158
10.1	Χαρακτηριστικά AVL Δέντρου.....	158
10.2	Διορθωτικές Πράξεις σε AVL για την επαναφορά της υψηζόγισης	159
10.3	Θέμα 2 Φεβρουάριος και Ιούνιος 2017.....	160
10.4	Θέμα 5 Σεπτέμβριος 2016.....	170
10.5	Θέμα 10 – Ιούνιος 2009.....	176
10.6	Θέμα 7 Ιούνιος 2011	180
10.7	Θέμα 5 Σεπτέμβριος 2012.....	183
10.8	Θέμα 10 με AVL Ιούνιος 2008	195

Δομές Δεδομένων –Computer Ανάλυση

10.9	Ερωτήσεις Θεωρίας στα AVL Δέντρα	207
11	(a,b) ΔΕΝΤΡΑ	208
11.1	Χαρακτηριστικά (a, b) δέντρων	208
11.2	Φροντιστήριο 2018	209
11.3	Θέμα 7 Ιούνιος 2015	211
11.4	Θέμα 7 Ιούνιος 2016	215
11.5	Θέμα 2 Ιούνιος 2008	224
11.6	Θέμα 2 Ιούνιος 2017	229
11.7	Θέμα 4 Φεβρουάριος 2019.....	232
12	TRIE ΔΕΝΤΡΑ	240
12.1	Χαρακτηριστικά TRIE Δέντρου	240
12.2	Πολυπλοκότητες TRIE Δέντρων.....	240
12.3	Θέμα 5 Ιούνιος 2008	241
12.4	Θέμα 4 Ιούνιος 2011	242
12.5	Θέμα 3 Ιούνιος 2012	243
12.6	Θέμα 9 Ιούνιος 2015	244
12.7	Θέμα 7 Σεπτέμβριος 2016 και Ιούνιος 2017	245
13	ΘΕΜΑΤΑ ΜΕ B+ TREES.....	247
13.1	Παράδειγμα 1 Κατασκευής B+ Δέντρου	248
13.2	Παράδειγμα 2 Κατασκευής B+ Δέντρου	249
13.3	Παράδειγμα 1 με Διαγραφές από B+ Δέντρο.....	251
14	EXTERNAL SORTING	253
14.1	Παράδειγμα με External Sorting	253
14.2	ΘΕΜΑΤΑ ΜΕ EXTERNAL SORTING KAI REPLACEMENT SELECTION	254
14.3	Θέμα 6 Ιούνιος 2011	254
14.4	Θέμα 5 Ιούνιος 2016 με External Sorting και Replacement Selection	256
14.5	Θέμα 4 Σεπτέμβριος 2016.....	259
14.6	Θέμα 3 Φεβρουάριος 2017.....	261
14.7	Θέμα 6 Σεπτέμβριος 2018.....	263
15	ΘΕΜΑΤΑ ΜΕ B TREES.....	275
15.1	Παράδειγμα 1 Κατασκευής B-Tree με m=5.....	275
15.2	Παράδειγμα 2 Κατασκευής B-Tree τάξης 4.....	285
16	ΑΛΓΟΡΙΘΜΟΣ KRUSKAL.....	288
16.1	Παράδειγμα 1	288
16.2	Παράδειγμα 2	290
17	ΕΡΩΤΗΣΕΙΣ ΘΕΩΡΙΑΣ	292
17.1	Αρχικοποίηση Πίνακα σε χρόνο O(1) – Σεπτέμβριος 2018	292
17.2	Τι είναι το Λεξικό Δεδομένων και ποιες πράξεις υποστηρίζει;	294
17.3	Ποιοι οι τρόποι αναζήτησης στο Λεξικό Δεδομένων;	294
17.4	Ποια η διαφορά του μοντέλου RAM από το μοντέλο PM. Σε ποια βασική υπόθεση στηρίζεται η ανάλυση πολυπλοκότητας των αλγορίθμων που μελετάμε;	294
17.5	Ποιες δομές ονομάζονται στατικές και ποιες δυναμικές; Ποιες ονομάζονται εφήμερες και ποιες διαχρονικές (πλήρως και μερικώς διαχρονικές); Ποιες δομές ονομάζονται εκτενείς και ποιες συνοπτικές;	294
17.6	Τι ονομάζεται Ουρά Προτεραιότητας, ποιες πράξεις υποστηρίζει και ποιες οι εφαρμογές της; (Ιούνιος 2014).....	295
17.7	Ποια η διαφορά των διαχρονικών από τις εφήμερες δομές;	296
17.8	Ποια η διαφορά της μερικής από την πλήρη διαχρονικότητα;	296
17.9	Ποια παράμετρο εισάγουμε για να υποστηρίξουμε τη διαχρονικότητα;	296

Δομές Δεδομένων –Computer Ανάλυση

17.10	Με ποιες μεθόδους υλοποιείται η μερική διαχρονικότητα;.....	296
17.11	Με ποιες μεθόδους υλοποιείται η πλήρη διαχρονικότητα;.....	297
17.12	Ποιες οι εφαρμογές της ουράς προτεραιότητας; (Σεπτέμβριος 2016)	298
17.13	Τι είναι η Διωνυμική Ουρά Προτεραιότητας και τι το Διωνυμικό Δέντρο;	298
17.14	Ποιες πράξεις υποστηρίζονται από τις ισότητας και τι κάνει η καθεμία; Ποιες είναι οι πολυπλοκότητες των πράξεων αυτών στις Διωνυμικές Σωρούς μεγέθους n ; (Σεπτέμβριος 2016 και Φεβρουάριος 2019)	299
17.15	Τι ονομάζεται Διωνυμικός Σωρός; Ποιες οι πράξεις ενός Διωνυμικού Σωρού(Σεπτέμβριος 2016).....	299
17.16	Ποιες οι Βασικές πράξεις σε δέντρα Αναζήτησης.....	300
17.17	Τι γνωρίζετε για το πρόβλημα Union Find;	301
17.18	Τι γνωρίζετε για τη μέθοδο weighted union rule;	301
17.19	Πόσο κοστίζει μια ακολουθία από $n-1$ unions και m Find όταν χρησιμοποιείται μόνο η τεχνική weighted union rule;	301
17.20	Τι γνωρίζετε για τη μέθοδο path compression;.....	301
17.21	Πόσο κοστίζει μια ακολουθία από $n-1$ union και m Find όταν χρησιμοποιούνται και οι δύο τεχνικές;.....	302
17.22	Ποιες περιπτώσεις ως προς την πολυπλοκότητα μελετάμε; (Θέμα 3 Ιούνιος 2015)	302
17.23	Δώστε τον ορισμό της επιμερισμένης πολυπλοκότητας με μια μικρή περιγραφή. Γιατί είναι διαφορετική από την πολυπλοκότητα μέσης περίπτωσης;.....	302
17.24	Ποιές πράξεις υποστηρίζονται σε σωρούς και ποιοι οι χρόνοι τους; (Ιούνιος 2015).....	303
17.25	Ποιες πράξεις υποστηρίζονται σε ουρές και ποιοι οι χρόνοι τους; (Ιούνιος 2014).....	303
17.26	Ποια τα χαρακτηριστικά του IST δέντρου;	303
17.27	Να δώσετε σε ψευδοκώδικα όλες τις πράξεις στο Hashing με αλυσίδες;.....	304
17.28	Να δώστε τον ορισμό του Red-Black Tree;	304
17.29	Θέματα Θεωρίας Άτυπης 2017.....	305
18	ΘΕΜΑΤΑ ΜΕ ΣΤΟΙΒΑ ΚΑΙ ΟΥΡΑ ΜΕ ΚΩΔΙΚΑ	306
18.1	Ολοκληρωμένο πρόγραμμα διαχείρισης στοιβας.....	306
18.2	Ολοκληρωμένο πρόγραμμα διαχείρισης ουράς.....	308
18.3	Ολοκληρωμένο πρόγραμμα διαχείρισης απλά συνδεδεμένης λίστας (simple-linked list)	309
19	ΘΕΜΑΤΑ ΜΕ ΔΙΚΟ ΜΑΣ ΑΛΓΟΡΙΘΜΟ	317
	ΘΕΜΑΤΑ ΦΕΒΡΟΥΑΡΙΟΣ 2019	318
	ΘΕΜΑΤΑ ΙΟΥΝΙΟΣ 2019	319
	ΘΕΜΑΤΑ ΣΕΠΤΕΜΒΡΙΟΣ 2019	322
4.1.1	Θέμα 2 Σεπτέμβριος 2018.....	328
4.1.2	Θέμα 6 Σεπτέμβριος 2012.....	328
4.1.3	Θέμα 7 Σεπτέμβριος 2012.....	329
4.1.4	Θέμα 8 Σεπτέμβριος 2012.....	330
4.1.5	Θέμα 2 Ιούνιος 2013	330
4.1.6	Θέμα 2 Σεπτέμβριος 2007.....	332

Εξέταση Μαθήματος «ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ» (ΟΜΑΔΑ 1)

ΕΞΕΤΑΣΤΙΚΗ ΦΕΒΡΟΥΑΡΙΟΥ 2021

Διδάσκοντες: Σιούτας Σπύρος, Μακρής Χρήστος, Καναβός Ανδρέας

ΔΙΑΡΚΕΙΑ 1:30 ΩΡΑ

ΘΕΜΑ 1 (3 μονάδες)

Ταξινομήστε τον πίνακα ακεραίων: {45, 13, 35, 23, 1, 9, 93, 3, 24, 5} με χρήση του **Heap Sort**. Αναφέρετε την πολυπλοκότητα χειρότερης περίπτωσης.

ΘΕΜΑ 2 (3 μονάδες)

Να σχεδιαστεί βήμα-βήμα το AVL δέντρο που προκύπτει από την εισαγωγή σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων {91, 2, 8, 7, 12, 3, 1, 4, 22, 8}. Διαγράψτε 22 και 7 και δείξτε τη μορφή του δένδρου.

ΘΕΜΑ 3 (3 μονάδες)

Δίνονται οι ακόλουθες διαπεράσεις ενός δυαδικού δένδρου:

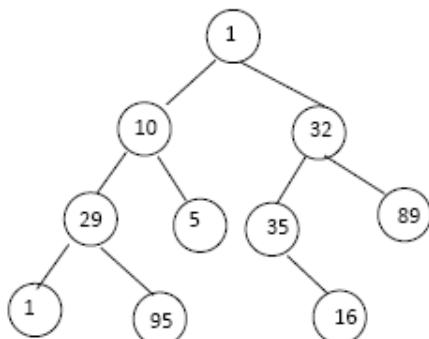
ΜΕΤΑΔΙΑΤΑΞΗ: QLDEFGKHIJA

ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ: QDLAEKGFIJI

Ανασχηματίστε τη μορφή του συγκεκριμένου δυαδικού δένδρου.

ΘΕΜΑ 4 (1 μονάδες)

Σας δίνεται το ακόλουθο δέντρο. Δείξτε το αποτέλεσμα της μεταδιάταξης



Δομές Δεδομένων –Computer Ανάλυση
Εξέταση Μαθήματος «ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ» (ΟΜΑΔΑ 2)

ΕΞΕΤΑΣΤΙΚΗ ΦΕΒΡΟΥΑΡΙΟΥ 2021

Διδάσκοντες: Σιούτας Σπύρος, Μακρής Χρήστος, Καναβός Ανδρέας

ΔΙΑΡΚΕΙΑ 1:30 ΏΡΑ

ΘΕΜΑ 1 (3 μονάδες)

Να σχεδιαστεί βήμα-βήμα το AVL-δένδρο που προκύπτει από την εισαγωγή σε ένα άδειο δένδρο με τη σειρά των εξής στοιχείων {2, 23, 5, 17, 1, 11, 3, 33, 4, 35} . Στην συνέχεια διαγράψτε βήμα-βήμα τα στοιχεία {11, 23}.

ΘΕΜΑ 3 (3 μονάδες)

Σας δίνετε το αλφάριθμο $\Sigma = \{A, T, C, G\}$ που αποτελεί τις βάσεις του ανθρώπινου γονιδιώματος (DNA). Σύμφωνα με το αλφάριθμο αυτό δημιουργούνται τα στοιχεία {CTATA, CTACA, CATC, CAACG, CTCGC, CACTT, CSTATG, GTCTA} τα οποία καλείστε να αποθηκεύστε σε δομή Trie χτισμένη στο παραπάνω αλφάριθμο. Σχεδιάστε:

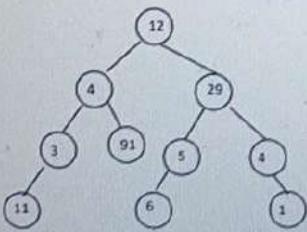
- i) Την πλήρη αποθήκευση
- ii) Την αποθήκευση που καταλαμβάνει το μικρότερο χώρο

ΘΕΜΑ 3 (3 μονάδες)

Ταξινομήστε την ακολουθία: SIGCOMPWORD2VEC12LEARNING με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection, δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος $M=3$, και ότι έχετε 6 τανίες στις διάθεσή σας ($r=3$). (Υπόδειξη: Προτεραιότητα ταξινόμησης A B C D E F G H I J K L M N O P Q R S T U V X Y Z 01...9)

ΘΕΜΑ 4 (1 μονάδες)

Σας δίνεται το ακόλουθο δέντρο. Δείξτε το αποτέλεσμα της μεταδιάταξης.



1 ΘΕΜΑΤΑ ΙΟΥΝΙΟΣ 2023**ΕΡΩΤΗΜΑ 1 (1 μονάδες)**

Ο παρακάτω ψευδοκώδικας αντιστοιχεί στον αναδρομικό αλγόριθμο γρήγορης ταξινόμησης (quick sort) ενός πίνακα Α μεγέθους $n=right-left+1$. Αρχικά $left=0$ και $right=n-1$. Συμπληρώστε τα κομμάτια κώδικα που λείπουν:

```
void QuickSort(int list[], int left, int right)
{
    int pivot, leftArrow, rightArrow;
    leftArrow = left;
    rightArrow = right;
    pivot = list[(left + right) / 2];
    do
    {
        while (list[rightArrow] > pivot)
            ;
        while (.....)
            ++
        if (leftArrow <= rightArrow)
        {
            Swap_Data(list[leftArrow], list[rightArrow]);
            .....
        }
    } while (rightArrow >= leftArrow);
    if (left < rightArrow)
        if (leftArrow < right)
            .....
}
```

ΘΕΜΑ 2 (1 μονάδες)

Ταξινομήστε την ακολουθία: CHATGPTQUANTUMCOMPUTERS με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection, δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος $M=4$, και ότι συνολικά έχετε 8 ταυτίες στη διάθεσή σας ($p=4$). (Υπόδειξη: Προτεραιότητα ταξινόμησης ABCDEFGHIJKLMNOPQRSTUVWXYZ)

ΘΕΜΑ 3 (2 μονάδες)

- Περιγράψτε το Binary Interpolation Search. Δώστε τις πολυπλοκότητες μέσης και χειρότερης περίπτωσης. Περιγράψτε αναλυτικά πώς θα γίνει ο χειρότερος χρόνος ψαζίματος ίσος με $O(\log n)$
- Σχεδιάστε το δέντρο σωρό, στο οποίο η μεταδιατεταγμένη διαπέραση να επισκέπτεται τους κόμβους με τη σειρά: 8, 10, 6, 5, 3, 4, 7, 2, 1. Γράψτε τον πίνακα της συνεχόμενης αναπαράστασης που αντιστοιχεί στο συγκεκριμένο δέντρο σωρό και αναφέρετε αν πρόκειται για σωρό ελαχίστων ή μεγίστων.

ΘΕΜΑ 4 (1,5 μονάδες)

Να σχεδιαστεί βήμα-βήμα το (2,3)-δένδρο που προκύπτει από την εισαγωγή σε ένα άδειο δένδρο με τη σειρά των εξής στοιχείων {17, 12, 11, 2, 30, 41, 20, 32, 19, 6}. Στην συνέχεια διαγράψτε βήμα-βήμα τα στοιχεία {11, 2}.

ΘΕΜΑ 5 (1,5 μονάδες)

Να σχεδιαστεί βήμα-βήμα το AVL δέντρο που προκύπτει από την εισαγωγή σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων {17, 12, 11, 2, 30, 41, 20, 32, 19, 6}. Στην συνέχεια διαγράψτε βήμα-βήμα τα στοιχεία {11, 2}.

ΘΕΜΑ 6 (1 μονάδες)

Ταξινομήστε τον πίνακα ακεραίων: {20, 2, 4, 3, 18, 8, 1, 23, 29, 21} με χρήση Ταξινόμησης Σωρού (Heap Sort). Αναφέρετε την πολυπλοκότητα χειρότερης περίπτωσης.

ΘΕΜΑ 7 (1 μονάδες)

Ταξινομήστε τον πίνακα ακεραίων: {7, 5, 2, 2, 4, 3, 3, 6, 4, 5} με χρήση ταξινόμησης με μέτρηση (Counting Sort). Αναφέρετε την πολυπλοκότητα χειρότερης περίπτωσης.

ΘΕΜΑ 8 (1 μονάδες)

Ιστω ότι θέλουμε να ενθέσουμε τα κλειδιά 8,20,29,2,13,26,15,86 σε ένα πίνακα ήκουνς $m=10$ με τη σειρά που δίνονται χρησιμοποιώντας hashing με αλυσίδες και chaining με open addressing.
ι στις δύο μεθόδους η βασική hash συνάρτηση είναι $h_1(k) = k \bmod m$. Στο hashing open addressing έστω η $h_2(k) = (p + 7 - k \bmod 7) \bmod m$ (μέθοδος με double hashing), όπου p το τελευταίο ψηφίο του αριθμού μητρώου σας. Δώστε σχηματικά hashing πίνακες και αναλυτικά τις πράξεις.

Εξέταση Μαθήματος «ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ»

ΕΞΕΤΑΣΤΙΚΗ ΣΕΠΤΕΜΒΡΙΟΣ 2023

Διδάσκοντες:

Σιούτας Σπύρος, Μακρής Χρήστος, Ηλίας Αριστείδης
ΔΙΑΡΚΕΙΑ 3 ΩΡΕΣ (2 σελίδες – 7 θέματα)

ΘΕΜΑ 1 (1 μονάδα)

Συγκρίνετε μεταξύ τους, τους αλγορίθμους *Quicksort* και *Median* (αλγόριθμος *select*) ως προς το πρόβλημα που αντιμετωπίζουν, την λογική τους και την ασυμπτωτική συμπεριφορά τους. Μπορούν να συνδυαστούν;

ΘΕΜΑ 2 (1.5 μονάδες)

- (α) Σχεδιάστε το δυαδικό δέντρο αναζήτησης, στο οποίο η προδιατεταγμένη διαπέραση να επισκέπτεται τους κόμβους με τη σειρά 15, 7, 3, 1, 5, 11, 9, 13, 23, 19, 21, 27, 29.
 (β) Προσθέστε στο δέντρο τα στοιχεία {25, 8} και στη συνέχεια διαγράψτε τα στοιχεία {7, 19}.
 (γ) Στο νέο δέντρο που δημιουργήθηκε, δώστε τα αποτελέσματα της μεταδιάταξης και της συμμετρικής διάταξης.

ΘΕΜΑ 3 (1.5 μονάδες)

Ταξινομήστε την ακολουθία: TRANSFORMERARCHITECTUREBERT με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection, δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος $M=4$, και ότι έχετε 8 ταινίες στις διάθεσή σας ($p=4$). (Υπόδειξη: Προτεραιότητα ταξινόμησης ABCDEFGHIJKLMNOPQRSTUVWXYZ)

ΘΕΜΑ 4 (1.5 μονάδες)

Ταξινομήστε τον πίνακα ακεραίων: {1, 21, 14, 3, 31, 89, 11, 23, 24, 8} με χρήση ταξινόμησης με συμβολή (Merge Sort). Αναφέρετε την πολυπλοκότητα χειρότερης περίπτωσης.

ΘΕΜΑ 5 (1.5 μονάδες)

Ταξινομήστε τον πίνακα ακεραίων: {78, 51, 4, 37, 81, 18, 11, 29, 19, 21} με χρήση γρήγορης ταξινόμησης (Quick Sort). Αναφέρετε την πολυπλοκότητα χειρότερης περίπτωσης.

ΘΕΜΑ 6 (1.5 μονάδες)

Να σχεδιαστεί βήμα-βήμα το (a,b)-δέντρο που προκύπτει από την εισαγωγή σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων {96, 12, 6, 71, 5, 3, 11, 13, 1, 20, 31, 99, 122} με $a=2$ και $b=4$. Διαγράψτε 20 και 3 και δείξτε τη μορφή του δένδρου.

ΘΕΜΑ 7 (1.5 μονάδες)

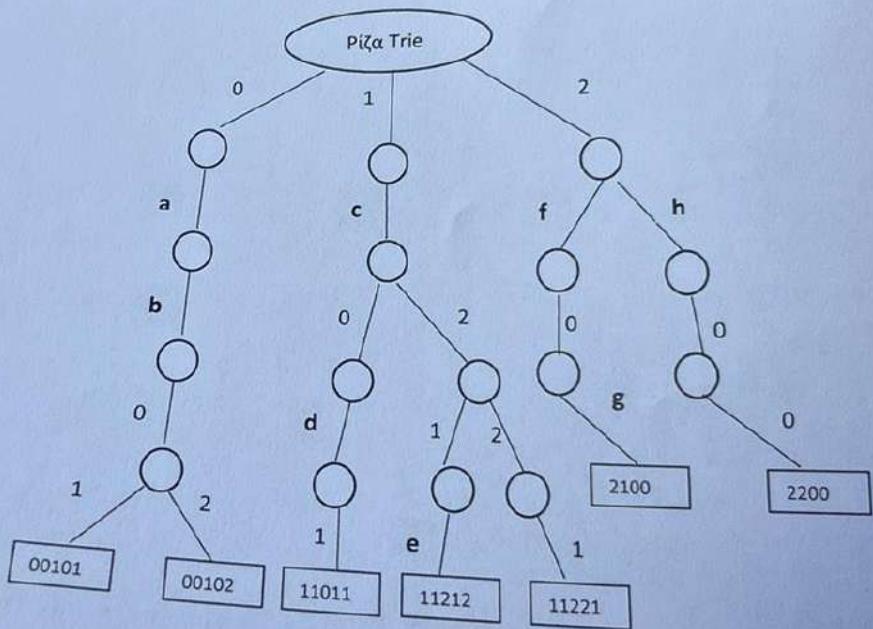
1. Σας δίνετε το αλφάριθμο $\Sigma = \{A, T, C, G\}$. Σύμφωνα με το αλφάριθμο αυτό δημιουργούνται τα στοιχεία {AAAA, AAC, AATA, ATCG, TACG, TAGC, TCGC, CATT, CATG, GCTA} τα οποία καλείστε να αποθηκεύστε σε δομή Trie χτισμένη στο παραπάνω αλφάριθμο.

Σχεδιάστε:

- i) Την πλήρη αποθήκευση
- ii) Την αποθήκευση που καταλαμβάνει το μικρότερο χώρο.

2. Σας δίνεται η παρακάτω δομή **Trie**, χτισμένη στο τριαδικό αλφάριθμο $\Sigma = \{0, 1, 2\}$. Με βάση τα στοιχεία που αποθηκεύονται στα φύλλα, βρείτε τις τιμές που πρέπει να έχουν τα σύμβολα a, b, c, d, e, f, g, h και συμπληρώστε τον παρακάτω πίνακα.

a	b	c	d	e	f	g	h

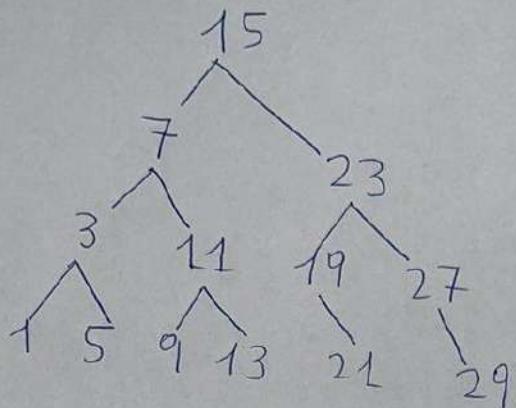


Απαντήσεις

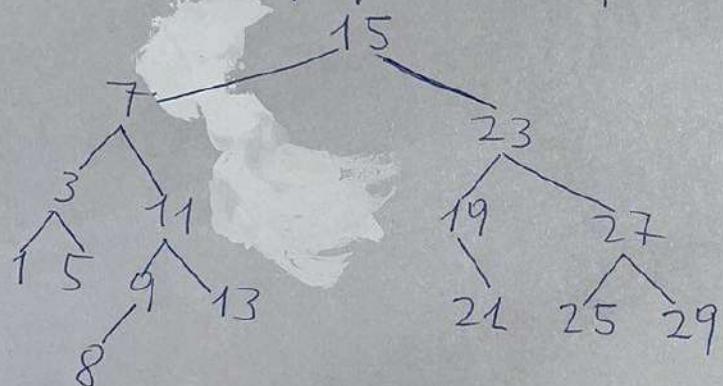
ΘΕΜΑ 2

Θέμα 2

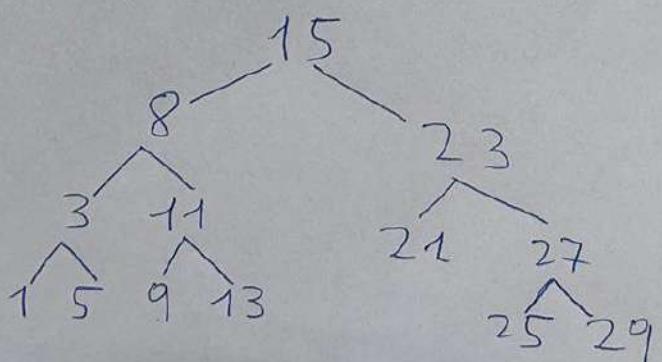
(a) Διαδικός Δέντρο Αναζήτησης



(b) Διαδικός δέντρο με όλη την προσθήτη 25 και 8



Διαδικός Δέντρο με όλη τη διαγραφή 7 και 19



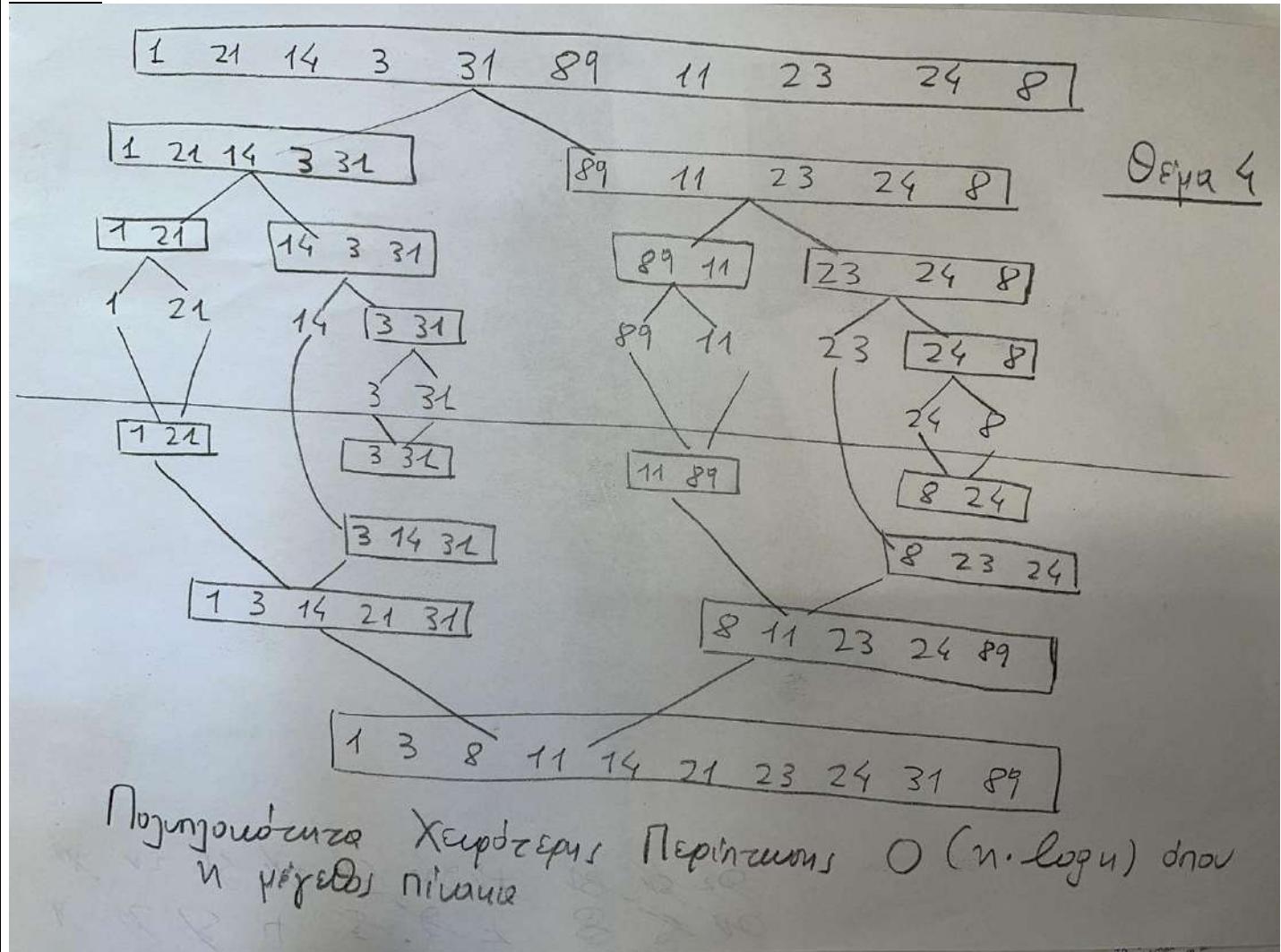
(g) Μεταδιάταξη

1-5-3-9-13-11-8-21-25-29-27-23-15

Συμμετρική Διάταξη

1-3-5-8-9-11-13-15-21-23-25-27-29

ΘΕΜΑ 4



Πορευόμαστε χωρίς εργασία Περιήγησης $O(n \cdot \log n)$ στην
η μέγεθος πώλησης

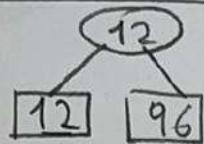
ΘΕΜΑ 6

Θεώρηα 6

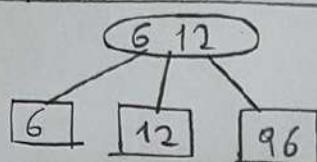
+ 96



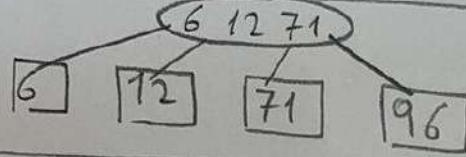
+ 12



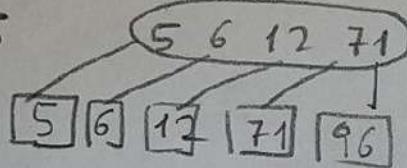
+ 6



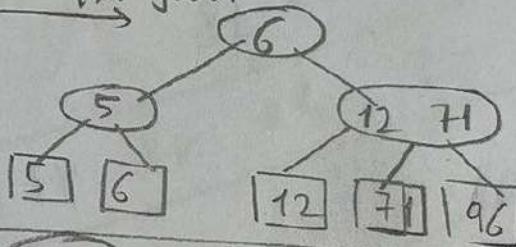
+ 71



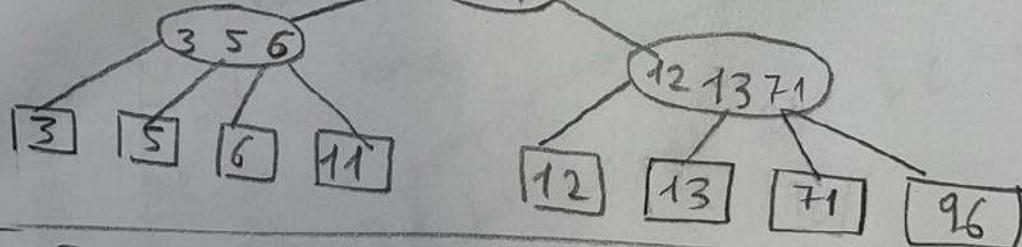
+ 5



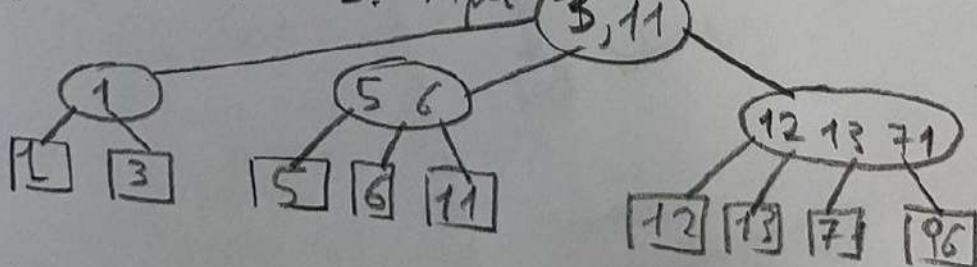
υπερχείλιον



+3, +11, +13

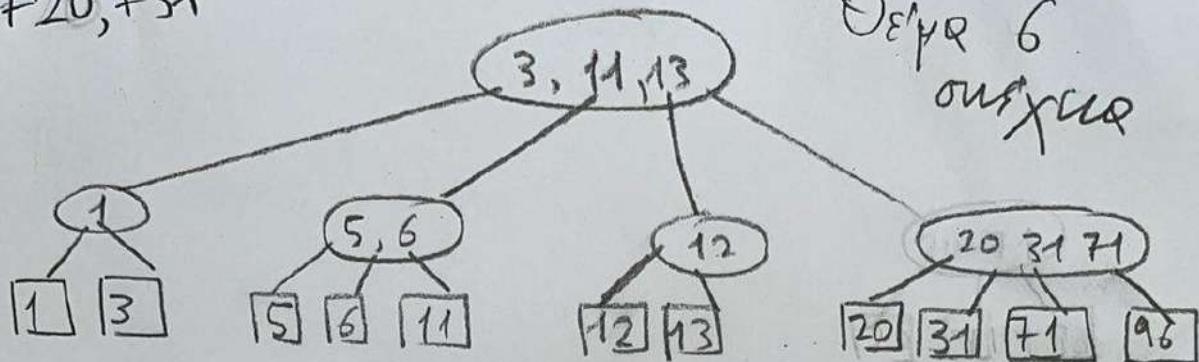


+13 Πρέπει υπερχείλιον στον κόρυφο (3 5 6) με -mv
ηρθετικόν - του L. Αρχ:

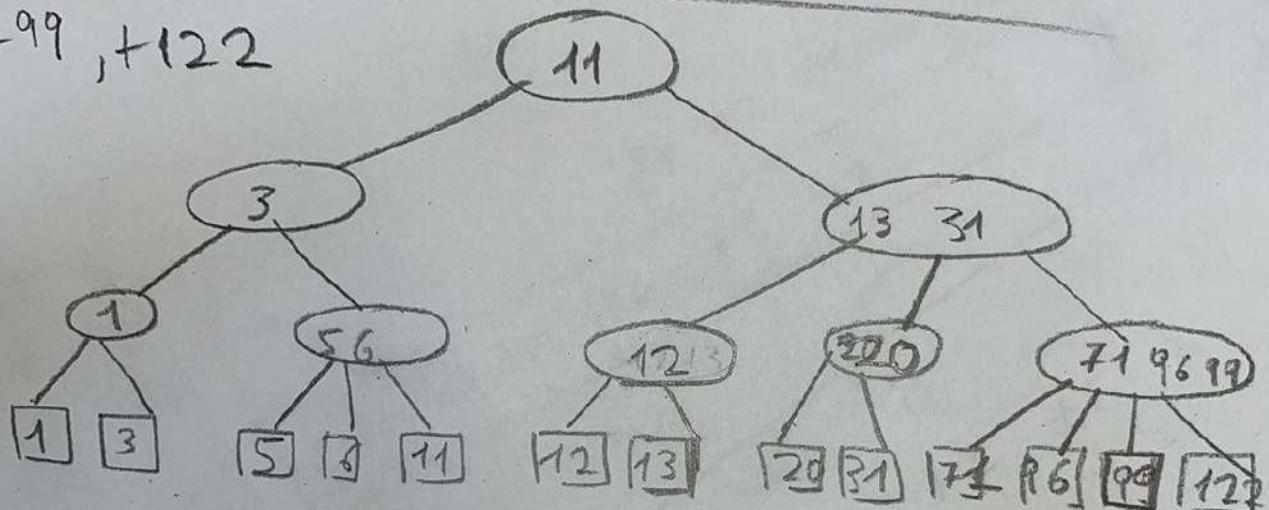


+20, +31

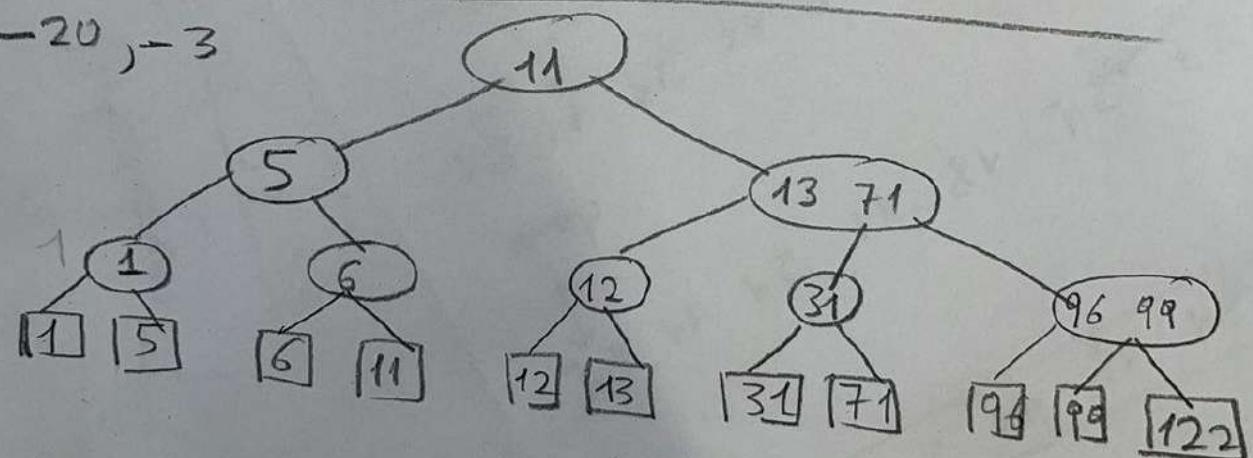
ΘΕΡΙΟ 6
συγχώνευση

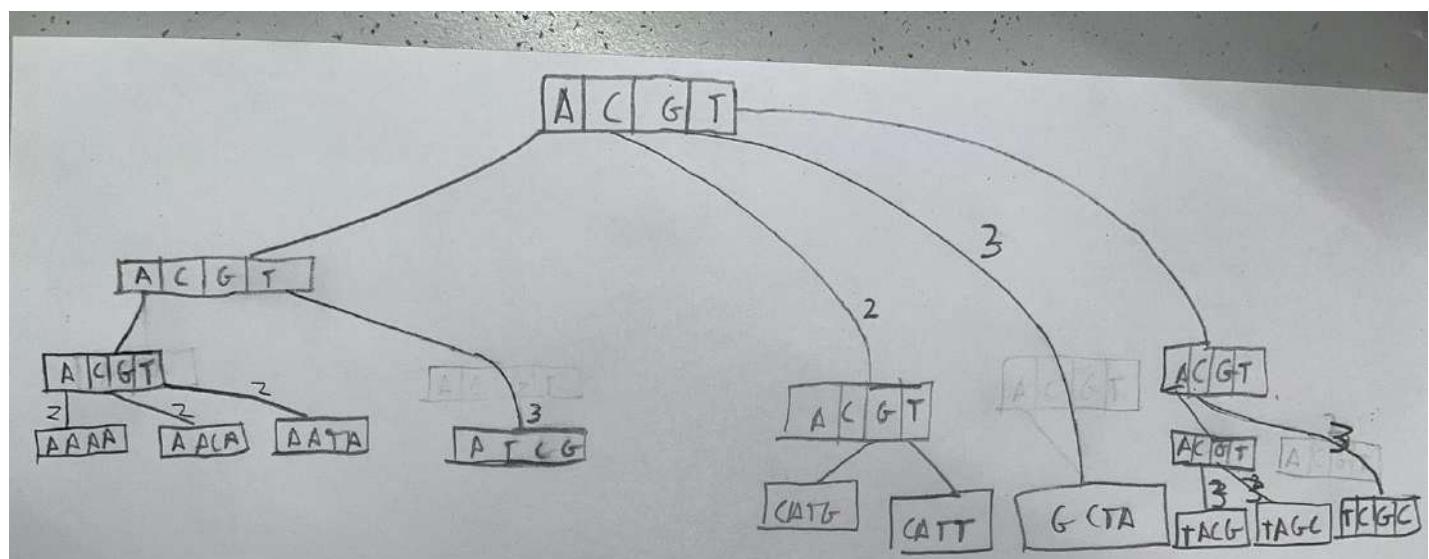
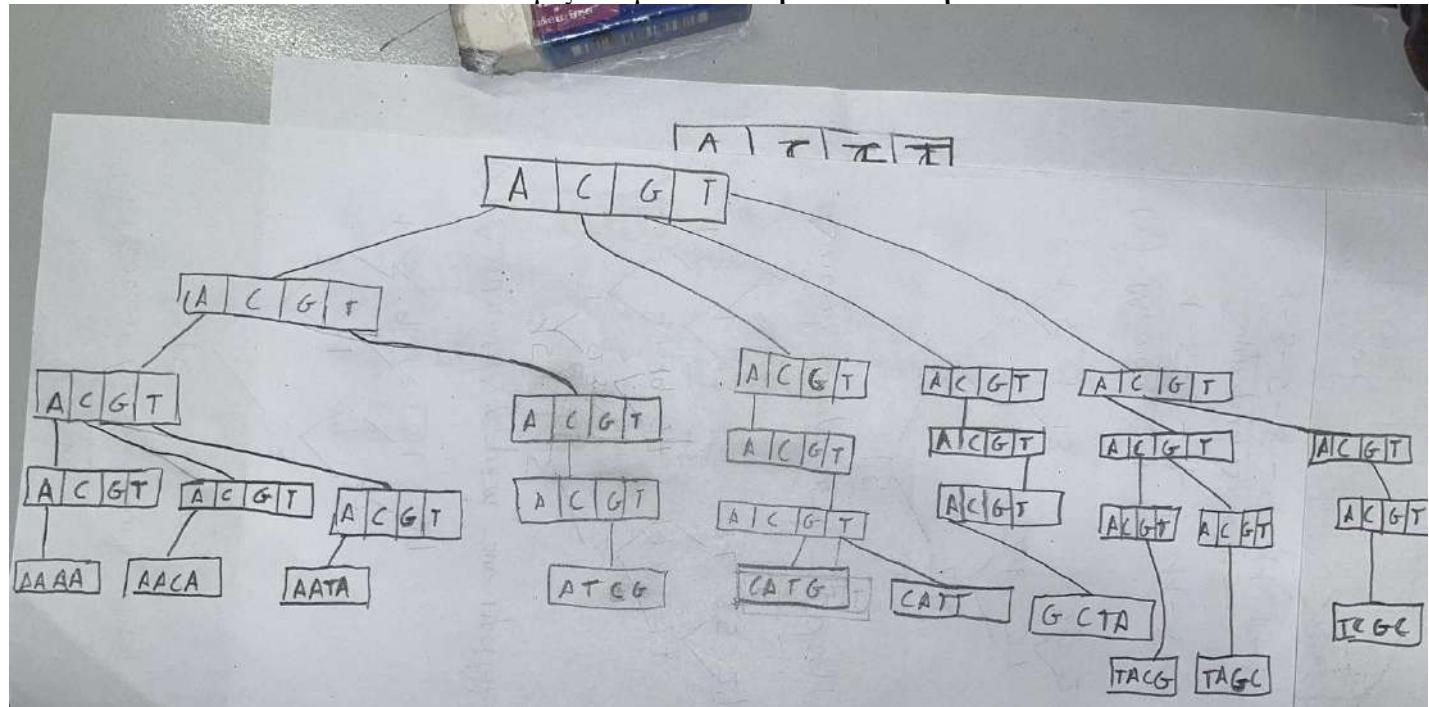


+99, +122



-20, -3





ΘΕΜΑ 1

Quicksort:

- Πρόβλημα:** Ο Quicksort είναι ένας αλγόριθμος ταξινόμησης που αντιμετωπίζει το πρόβλημα ταξινόμησης ενός πίνακα MH ταξινομημένων στοιχείων σε αύξουσα ή φθίνουσα σειρά.
- Λογική:** Ο Quicksort επιλέγει ένα στοιχείο ως "pivot" και ταξινομεί τα υπόλοιπα στοιχεία στον πίνακα έτσι ώστε τα στοιχεία μικρότερα από τον pivot να βρίσκονται αριστερά και τα στοιχεία μεγαλύτερα από τον pivot να βρίσκονται δεξιά. Στη συνέχεια, επαναλαμβάνει την ίδια διαδικασία για τον αριστερό και δεξιό υποπίνακα μέχρι όλος ο πίνακας να είναι ταξινομημένος.
- Ασυμπτωτική Συμπεριφορά:** Η χειρότερη περίπτωση είναι $O(n^2)$ και η μέση είναι $O(n \log n)$.

Αλγόριθμος Median (Select):

- Πρόβλημα:** Ο αλγόριθμος Median επιλέγει το k-όστο μικρότερο ή μεγαλύτερο στοιχείο από έναν πίνακα.
- Λογική:** Ο αλγόριθμος Select χρησιμοποιεί μια τροποποιημένη έκδοση του Quicksort για να επιλέξει το k-όστο στοιχείο.
- Ασυμπτωτική Συμπεριφορά:** Η ασυμπτωτική συμπεριφορά του αλγορίθμου Select εξαρτάται από την επιλογή του pivot. Σε μέση περίπτωση, η πολυπλοκότητά του είναι $O(n)$, αλλά η χειρότερη περίπτωση είναι $O(n^2)$.
- Όσον αφορά το ερώτημα εάν μπορούν να συνδυαστούν, ναι, οι αλγόριθμοι Quicksort και Select μπορούν να συνδυαστούν σε ορισμένες περιπτώσεις. Π.χ. χρησιμοποιούμε τον αλγόριθμο Select για να βρούμε το median ενός πίνακα, και στη συνέχεια χρησιμοποιούμε το median ως pivot για τον Quicksort. Αυτή η στρατηγική μπορεί να βελτιώσει την απόδοση του Quicksort.

ΘΕΜΑ 3

Heap	Pίζα Heap	Επόμενο	Έξοδος
TRAN	A	S	A
TRSN	N	F	AN
TRSF*	R	O	ANR
TO*SF*	S	R	ANRS
TO*R*F*	T	M	ANRST – 1^o μπλοκ Ταινία 1
M*O*R*F*	F*	E	F*
M*O*R*E	M*	R	F*M*
R*O*R*E	O*	A	F*M*O*
R*AR*E	R*	R	F*M*O*R*
R*AR*E	R*	C	F*M*O*R*R*
CAR*E	R*	H	F*M*O*R*R*R* - 2^o μπλοκ Ταινία 2
CAHE	A	I	A
CIHE	C	T	AC
TIHE	E	E	ACE
TIHE	E	C	ACEE
TIHC*	H	T	ACEEH
TITC*	I	U	ACEEHI
TUTC*	T	R	ACEEHIT
R*UTC*	T	E	ACEHITT
R*UE*C*	U	B	ACEEHITTU- 3^o μπλοκ Ταινία 3
R*B*E*C*	B*	E	B*
R*E*E*C*	C*	R	B*C*
R*E*E*R*	E*	T	B*C*E*- 4^o μπλοκ Ταινία 4
R*T*E*R*	-	-	E*R*R*T* - 5^o μπλοκ Ταινία 1

Τα μπλοκ που τοποθετούνται στις ταινίες είναι τα ακόλουθα:

Ταινία 1	ANRST	ERRT
Ταινία 2	FMORRR	
Ταινία 3	ACEEHITTU	
Ταινία 4	BCE	

Φάση Β-Συγχώνευση ταξινομημένων υπο-ακολουθιών. Συγχωνεύονται οι ταινίες που παρήχθησαν κατά τη Φάση A. Τα μπλοκ που παρήχθησαν κατά τη Φάση A συγχωνεύονται και το αποτέλεσμα γράφεται στις p ταινίες που έχουν απομείνει και οι οποίες ονομάζονται ταινίες εξόδου. Η συγχώνευση επαναλαμβάνεται με την παραγωγή στην έξοδο ενός και μόνο μπλοκ τι οποίο θα αποτελεί την ταξινομημένη έξοδο.

Ταινία 5	AABCCEEFFHMNORRRRSTTTU
Ταινία 6	ERRT
Ταινία 7	
Ταινία 8	

Φάση Β-Συγχώνευση στην Ταινία 1

3 ΕΞΕΤΑΣΤΙΚΗ ΣΕΠΤΕΜΒΡΙΟΥ 2021

ΘΕΜΑ 2 (3 μονάδες)

Δίνονται οι ακόλουθες διαπεράσεις ενός δυαδικού δένδρου:

ΠΡΟΔΙΑΤΑΞΗ: IJNOPKLMQR

ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ: ONPJILKQMR

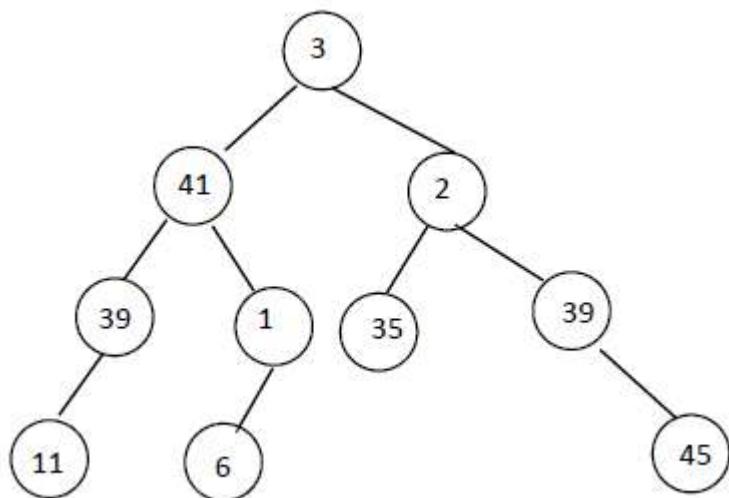
Ανασχηματίστε τη μορφή του συγκεκριμένου δυαδικού δένδρου.

ΘΕΜΑ 3 (3 μονάδες)

Έστω ότι θέλουμε να ενθέσουμε τα κλειδιά 5,14,20,1,5,11,91,6 σε ένα πίνακα μήκους $m=10$ με τη σειρά που δίνονται χρησιμοποιώντας hashing με αλυσίδες και hashing με open addressing. Και στις δύο μεθόδους η βασική hash συνάρτηση είναι $h1(k)=k \bmod m$. Στο hashing με open addressing έστω η $h2(k)=(2k+l) \bmod m$ (μέθοδος με double hashing ΟΠΟΥ 1 το τελευταίο ψηφίο του αριθμού μητρώου σας). Δώστε σχηματικά τους hashing πίνακες.

ΘΕΜΑ 4 (1 μονάδες)

Δείξτε το αποτέλεσμα της ενδοδιάταξης:



ΘΕΜΑ 2 (3 μονάδες)

Δίνονται οι ακόλουθες διαπεράσεις ενός δυαδικού δένδρου:

ΠΡΟΔΙΑΤΑΞΗ: MLCPUGHTJ

ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ: LPCUMGFTHJ

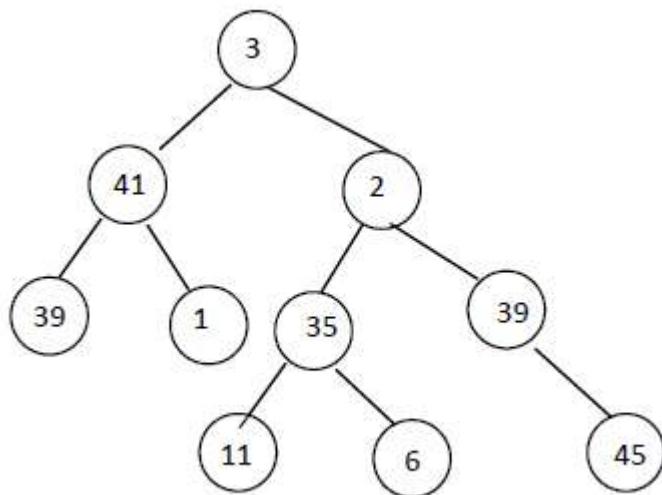
Ανασχηματίστε τη μορφή του συγκεκριμένου δυαδικού δένδρου.

ΘΕΜΑ 3 (3 μονάδες)

Έστω ότι θέλουμε να ενθέσουμε τα κλειδιά 2,1,20,15,5,11,9,6 σε ένα πίνακα μήκους $m=10$ με τη σειρά που δίνονται χρησιμοποιώντας hashing με αλυσίδες και hashing με open addressing. Και στις δύο μεθόδους η βασική hash συνάρτηση είναι $h1(k)=k \bmod m$. Στο hashing με open addressing έστω η $h2(k)=(k+l) \bmod m$ (μέθοδος με double hashing ΟΠΟΥ 1 το τελευταίο ψηφίο του αριθμού μητρώου σας). Δώστε σχηματικά τους hashing πίνακες,

ΘΕΜΑ 4 (1 μονάδες)

Δείξτε το αποτέλεσμα της ενδοδιάταξης:



4 MONTELO RAM KAI PM

4.1 Θέμα 1 Ιούνιος 2010

Ποια η διαφορά του μοντέλου **RAM** από το μοντέλο **PM**; Σε ποια βασική υπόθεση στηρίζεται η ανάλυση πολυπλοκότητας των αλγορίθμων που μελετάμε; Να κατηγοριοποιήσετε τις παρακάτω δομές δεδομένων/αλγορίθμους ανάλογα με το μοντέλο υπολογισμού (Pointer Machine ή RAM) στο οποίο υλοποιούνται. 1) IST δέντρο, 2) Bubble sort, 3) AVL δέντρο, 4) Hashing με αλυσίδες, 5) Trie, 6) Σωρός (Stack), 7) Insertion Sort, 8) Interpolation search, 9) union find, 10) (2-4) δέντρο 11) HeapSort 12) External 13) Splay tree

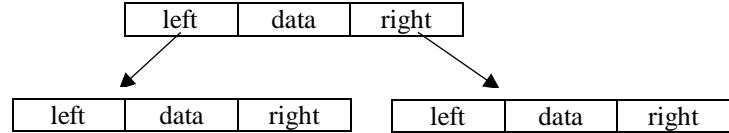
Απάντηση

- Η διαφορά του μοντέλου **RAM** από το μοντέλο **PM** είναι ότι το μοντέλο PM δεν επιτρέπει την προσπέλαση με υπολογισμό διεύθυνσης. Προσπέλαση σε μια συγκεκριμένη διεύθυνση μπορεί να γίνει μόνο αν υπάρχει δείκτης σε αυτή τη διεύθυνση. Το μοντέλο PM είναι ασθενέστερο της RAM
- Η βασική υπόθεση που στηρίζομαστε είναι ότι **δουλεύουμε σε RAM μοναδιαίου κόστους** δηλ. το περιεχόμενο κάθε μεταβλητής χωρά σε μια θέση μνήμης και κάθε πράξη χρειάζεται σταθερό χρόνο

1) IST δέντρο → PM

2) Bubble Sort → RAM

3) AVL δέντρο → PM



4) Hashing με Αλυσίδες → RAM

5) Trie → PM

6) Σωρός (Stack) → RAM

7) Insertion Sort → RAM

8) Interpolation Search → RAM

9) Union Find → RAM

10) (2-4) δέντρο → PM

11) HeapSort → PM

12) External → RAM

13) Splay tree → PM

4.2 ΠΙΝΑΚΕΣ

Να περιγραφεί η δομή του πίνακα.

Απάντηση

Πίνακας είναι μια δομή που περιέχει στοιχεία του ίδιου τύπου (δηλαδή ακέραιους, πραγματικούς κ.λ.π.). Η αναφορά των στοιχείων ενός πίνακα γίνεται με τη χρήση ενός συμβολικού ονόματος του πίνακα ακολουθούμενου από την τιμή ενός ή περισσότερων δεικτών σε παρένθεση ή αγκύλη. (π.χ. πίνακας[θέση], πίνακας[γραμμή, στήλη]). Το μέγεθος του πίνακα ορίζεται κατά τη διάρκεια εκτέλεσης του κώδικα (δυναμικά) είτε κατά τη διάρκεια μετάφρασης του κώδικα (στατικά) και οι θέσεις μνήμης που δεσμεύει ο πίνακας είναι πάντα διαδοχικές στη στατική δέσμευση μνήμης ενώ στη δυναμική δέσμευση μνήμης οι θέσεις που δεσμεύει ο πίνακας είναι τυχαίες.

Πλεονεκτήματα

- Προσπέλαση στοιχείου σε σταθερό χρόνο, $O(1)$
- Τυχαία προσπέλαση

Μειονεκτήματα

- Σταθερό μέγεθος (εκτός από τους δυναμικούς πίνακες).
- Δεν μπορούμε να προσθέσουμε στοιχεία στη μέση ενός πίνακα.
- Αναδιάταξη στοιχείων μόνο με αντιγραφή.

4.3 ΣΤΟΙΒΑ

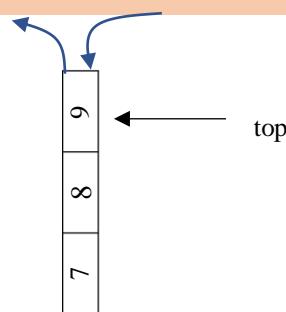
Να περιγραφεί η δομή της Στοίβας και οι βασικές λειτουργίες της

Απάντηση

Στοίβα (Stack) ονομάζεται μια δομή δεδομένων που μπορεί να υλοποιηθεί με τη βοήθεια ενός μονοδιάστατου πίνακα και επιτρέπει στα δεδομένα που βρίσκονται στην κορυφή της να εξάγονται πρώτα, ενώ σε αυτά που βρίσκονται στο βάθος της να εξάγονται τελευταία. Αυτή η μέθοδος επεξεργασίας στοιχείων ονομάζεται **LIFO** (Last-In- First-Out). Μια βοηθητική μεταβλητή που ονομάζεται **top** χρησιμοποιείται για να δείχνει το στοιχείο που τοποθετήθηκε τελευταίο στην κορυφή της στοίβας. Στη στοίβα υπάρχουν οι λειτουργίες ώθηση (push) που εισάγει στοιχείο στην κορυφή της στοίβας και η απώθηση (pop) που διαγράφει στοιχείο από την κορυφή της στοίβας. Η διαδικασία της ώθησης (push) πρέπει πρώτα να ελέγχει, αν η στοίβα είναι γεμάτη (αν γίνεται δηλαδή υπερχείλιση -overflow της στοίβας), ενώ η διαδικασία απώθησης (pop) πρέπει να ελέγχει αν η στοίβα είναι κενή (αν γίνεται δηλαδή υποχείλιση -underflow της στοίβας). Επίσης η στοίβα μπορεί να υλοποιηθεί με τη μορφή **διασυνδεδεμένης λίστας** όπου οι εισαγωγές και οι εξαγωγές γίνονται στην αρχή της λίστας.

Οι βασικές λειτουργίες σε μία στοίβα είναι οι ακόλουθες:

- **push(data)** → ώθηση στοιχείου στην κορυφή της στοίβας σε χρόνο **O(1)**
- **pop()** → Απώθηση στοιχείου από την κορυφή της στοίβας σε χρόνο **O(1)**
- **empty()** → έλεγχος αν η στοίβα είναι κενή **O(1)**
- **top()** → Επιστροφή του κορυφαίου στοιχείου της στοίβας χωρίς να το διαγράφει σε χρόνο **O(1)**



Πίνακας **s** που αντιπροσωπεύει τη στοίβα και δείκτης **top**, αρχικά ίσος με μηδέν, που δείχνει την τελευταία κατειλημμένη θέση της στοίβας

<pre> push(x) { if (top<N) //έλεγχος υπερχείλισης { top=top+1; s[top]=x; done=true; } else { γράψε "Στοίβα γεμάτη" done=false; } } </pre>	<pre> pop() { if (top=0) //έλεγχος υποχείλισης { γράψε "Η στοίβα είναι άδεια" done=false; } else { x=s[top] γράψε "Εξαγωγή στοιχείου", x top=top-1; done=true; } } </pre>
--	--

4.4 Πιθανό Θέμα με Στοίβα για σωστό ταίριασμα παρενθέσεων και αγκυλών

Να γραφεί αλγόριθμος ο οποίος με χρήση στοίβας ελέγχει το σωστό ταίριασμα Αγκυλών, Παρενθέσεων και Αγκίστρων σε μια παράσταση. [{ }], [()

Απάντηση

- Όσο υπάρχουν χαρακτήρες στην παράσταση εισόδου επανέλαβε
 - Διάβασε τον επόμενο χαρακτήρα εισόδου
 - Αν είναι (ή | ή { κάνε push το χαρακτήρα στη στοίβα
 - Αν είναι) σύγκρινε το χαρακτήρα αυτό με το χαρακτήρα στην κορυφή της στοίβας. Αν ο χαρακτήρας στην κορυφή της στοίβας είναι (τότε κάνε pop αλλιώς αγνόησε το χαρακτήρα)
 - Αν είναι] σύγκρινε το χαρακτήρα αυτό με το χαρακτήρα στην κορυφή της στοίβας. Αν ο χαρακτήρας στην κορυφή της στοίβας είναι [τότε κάνε pop αλλιώς αγνόησε το χαρακτήρα]
 - Αν είναι } σύγκρινε το χαρακτήρα αυτό με το χαρακτήρα στην κορυφή της στοίβας. Αν ο χαρακτήρας στην κορυφή της στοίβας είναι { τότε κάνε pop αλλιώς αγνόησε το χαρακτήρα }
- Αν στο τέλος της παράστασης εισόδου η στοίβα είναι κενή return 'επιτυχία'
- Αν στο τέλος της εισόδου η στοίβα ΔΕΝ είναι κενή return 'αποτυχία'

Δομές Δεδομένων –Computer Ανάλυση
Παράδειγμα Εφαρμογής του Προηγούμενου Αλγορίθμου σε παράσταση με σωστό ταίριασμα

Δίνεται η παράσταση **() { () }** . Να ελεγχθεί αν έχει σωστό ταίριασμα **αγκυλών, παρενθέσεων και αγκίστρων** με εφαρμογή του προηγούμενου αλγορίθμου.

Απάντηση

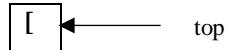
Βήμα 1: Πρώτα διαβάζεται ο χαρακτήρας **(** και γίνεται push στη στοίβα, συνεπώς:



Βήμα 2: Μετά διαβάζεται ο χαρακτήρας **)** και συγκρίνεται με το χαρακτήρα στην κορυφή της στοίβας που είναι **η** (και γίνεται pop από τη στοίβα, συνεπώς η στοίβα αδειάζει:



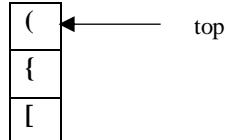
Βήμα 3: Μετά διαβάζεται ο χαρακτήρας **[** και γίνεται push στη στοίβα, συνεπώς:



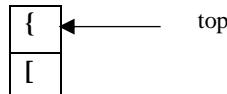
Βήμα 4: Μετά διαβάζεται ο χαρακτήρας **{** και γίνεται push στη στοίβα, συνεπώς:



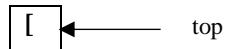
Βήμα 5: Μετά διαβάζεται ο χαρακτήρας **(** και γίνεται push στη στοίβα, συνεπώς:



Βήμα 6: Μετά διαβάζεται ο χαρακτήρας **)** και συγκρίνεται με το χαρακτήρα **(** στην κορυφή της στοίβας ο οποίος γίνεται pop από τη στοίβα, συνεπώς:



Βήμα 7: Μετά διαβάζεται ο χαρακτήρας **}** και συγκρίνεται με το χαρακτήρα **{** στην κορυφή της στοίβας ο οποίος γίνεται pop από τη στοίβα, συνεπώς:



Βήμα 8: Μετά διαβάζεται ο χαρακτήρας **]** και συγκρίνεται με το χαρακτήρα **[** στην κορυφή της στοίβας ο οποίος γίνεται pop από τη στοίβα, συνεπώς:



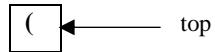
Επειδή στο τέλος της εισόδου η στοίβα είναι κενή, η συμβολοσειρά έχει σωστό ταίριασμα και επιστρέφεται το μήνυμα 'επιτυχία'

Δομές Δεδομένων –Computer Ανάλυση
Παράδειγμα Εφαρμογής του Προηγούμενου Αλγορίθμου σε παράσταση χωρίς σωστό ταίριασμα

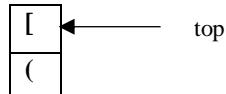
Δίνεται η παράσταση ([)]. Να ελεγχθεί αν έχει σωστό ταίριασμα αγκυλών, παρενθέσεων και αγκίστρων με εφαρμογή του προηγούμενου αλγορίθμου.

Απάντηση

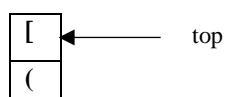
Βήμα 1: Πρώτα διαβάζεται ο χαρακτήρας (και γίνεται push στη στοίβα, συνεπώς:



Βήμα 2: Μετά διαβάζεται ο χαρακτήρας [και γίνεται push στη στοίβα, συνεπώς:



Βήμα 3: Μετά διαβάζεται ο χαρακτήρας) και συγκρίνεται με το χαρακτήρα [στην κορυφή της στοίβας ο οποίος ΔΕΝ γίνεται pop από τη στοίβα, συνεπώς:



Βήμα 4: Μετά διαβάζεται ο χαρακτήρας] και συγκρίνεται με το χαρακτήρα [στην κορυφή της στοίβας ο οποίος γίνεται pop από τη στοίβα, συνεπώς:



Επειδή στο τέλος της εισόδου η στοίβα ΔΕΝ είναι κενή, η συμβολοσειρά ΔΕΝ έχει σωστό ταίριασμα και επιστρέφεται το μήνυμα 'αποτυχία'

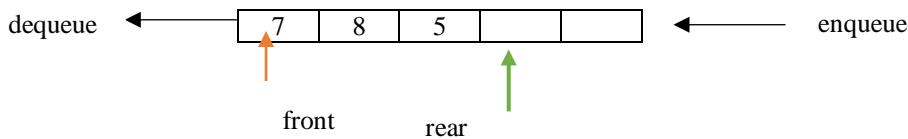
5 ΟΥΡΑ

Να περιγραφεί η δομή της Ουράς και οι βασικές λειτουργίες της.

Απάντηση

Ουρά (Queue) ονομάζεται μια δομή δεδομένων στην οποία τα νέα δεδομένα που εισέρχονται στην ουρά μπαίνουν στο τέλος της ενώ τα στοιχεία που είναι στην αρχή της ουράς θα εξαχθούν πρώτα. Αυτή η μέθοδος επεξεργασίας ονομάζεται **FIFO** (First-In-First-Out).

Μια Ουρά μπορεί να υλοποιηθεί εύκολα με τη βοήθεια ενός μονοδιάστατου πίνακα. Στην περίπτωση της ουράς απαιτούνται δύο δείκτες: ο δείκτης **front** ("εμπρός") που δείχνει την αρχή της ουράς και ο πίσω δείκτης **rear** ("πίσω") που δείχνει το τέλος της ουράς. Πρέπει να ελέγχεται αν υπάρχει ελεύθερος χώρος στην ουρά για την εισαγωγή και αν υπάρχει ένα τουλάχιστον στοιχείο στην ουρά όταν γίνεται εξαγωγή. **Επίσης η ουρά μπορεί να υλοποιηθεί με τη μορφή διασυνδεδεμένης λίστας** όπου οι εισαγωγές γίνονται στο τέλος της λίστας και οι διαγραφές μπορούν να γίνουν μόνο από την αρχή της λίστας.



Οι βασικές λειτουργίες που εκτελούνται σε μία ουρά είναι:

- **εισαγωγή (enqueue)** στοιχείου στο τέλος της ουράς **Χρόνος O(1)**
- **εξαγωγή (dequeue)** στοιχείου από την αρχή της ουράς **Χρόνος O(1)**
- **is_empty()** → επιστρέφει ΝΑΙ αν η ουρά δεν περιέχει κανένα στοιχείο, αλλιώς επιστρέφει ΟΧΙ. **Χρόνος O(1)**
- **first()** → επιστροφή του στοιχείου στην αρχή της ουράς χωρίς να το αφαιρεί. **Χρόνος O(1)**

Πίνακας όπου αντιπροσωπεύει την ουρά, έχει τους δείκτες front που δείχνει την αρχή της ουράς και rear που δείχνει το τέλος της ουράς

<pre> enqueue(x) { if (front=1 and rear=N) { γράψε "Η ουρά είναι γεμάτη" done=false; } else { if rear=0 then front=1 //αντό ισχύει μόνο την 1^η φορά που εισάγονται στοιχείο μέσα στην ουρά rear=rear+1; q[rear]=x; done=true; } } </pre>	<pre> dequeue() { if (front=0 and rear=0) { γράψε "Η ουρά είναι κενή" done=false; } else if (front>rear) αντό ισχύει όταν η ουρά αδειάσει { γράψε "Η ουρά είναι κενή" done=false; rear=0; front=0; } else { x=q[front] γράψε "Εξαγωγή στοιχείου", x front=front+1; done=true; } } </pre>
---	---

6 ΛΙΣΤΑ-ΟΧΙ

Να περιγράψετε τη Δομή της Λίστας και τις βασικές λειτουργίες της.

Απάντηση

Η λίστα χρησιμοποιείται για την αναπαράσταση ενός πίνακα, μιας στοίβας ή μιας ουράς. Η λίστα είναι ένα σύνολο διασυνδεδεμένων κόμβων όπου κάθε κόμβος αναπαριστάνεται ως εξής:

struct Node

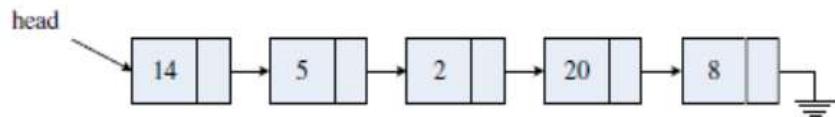
```
{
    int elem; //τιμή κόμβου
    struct Node *next; //δείκτης στον επόμενο κόμβο της λίστας
};
```

Υπάρχει ένας δείκτης head που δείχνει στην αρχή της λίστας.

Οι βασικές πράξεις σε μια λίστα με τις αντίστοιχες πολυπλοκότητες τους είναι οι εξής:

- **first(): O(1)**
- **last(): O(1)**
- **insert_after(pointer,data): O(1)** Η πολυπλοκότητα είναι O(1) με την προϋπόθεση ότι η λίστα είναι διπλή δηλ. έχει δείκτες και στις 2 κατευθύνσεις. Διαφορετικά η πολυπλοκότητα είναι:
last(),size(),catenate_with → O(n)
- **remove(pointer)**
- **find(data): O(n)**
- **size(): O(1)**
- **catenate_with(list): O(1)**

ΥΛΟΠΟΙΗΣΗ ΛΙΣΤΑΣ ΜΕ ΔΕΙΚΤΕΣ



7 ΔΕΝΤΡΑ

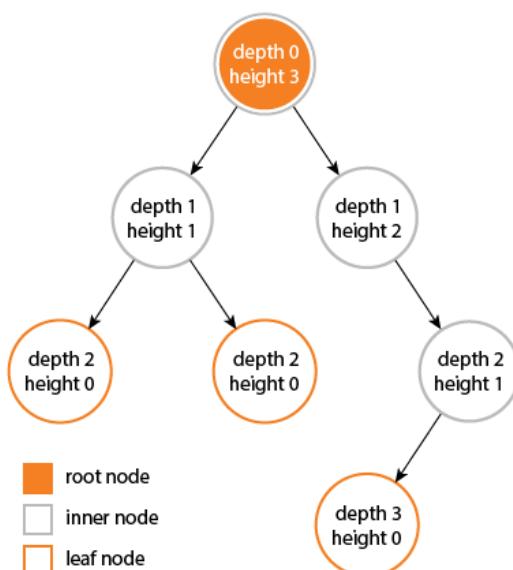
7.1 Ορισμός Δέντρου

Το δέντρο είναι ένα άκυκλο συνεκτικό και μη κατευθυνόμενο γράφημα.

- ✓ Για οποιουσδήποτε δύο κόμβους υπάρχει ένα μοναδικό μονοπάτι
- ✓ Πλήθος ακμών είναι ίσο με πλήθος κόμβων μείον ένα. $|E|=|V|-1$.

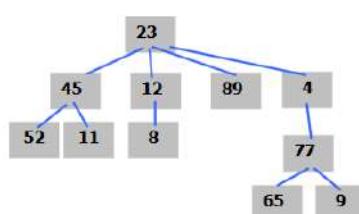
7.2 Χαρακτηριστικά Δυαδικού Δέντρου -ΟΧΙ

- Δυαδικό Δέντρο είναι αυτό που κάθε κόμβος έχει το πολύ 2 παιδιά
- Το βάθος (depth) ενός κόμβου είναι ο αριθμός ακμών από τον κόμβο προς τη ρίζα. Η ρίζα έχει βάθος 0
- Το ύψος (height) ενός κόμβου είναι ο αριθμός ακμών στο μεγαλύτερο μονοπάτι από τον κόμβο προς ένα φύλλο.
- Ένα φύλλο έχει ύψος 0
- Το ύψος ενός δέντρου είναι το ύψος της ρίζας ή το βάθος του κόμβου που βρίσκεται στο μεγαλύτερο βάθος



7.2.1 Πράξεις Δυαδικού Δέντρου=ΟΧΙ

- `create()`: δημιουργία κενού δυαδικού δέντρου
- `empty()`: ελέγχει αν το δέντρο είναι κενό
- `left()`: επιστρέφει το αριστερό παιδί του κόμβου
- `right()`: επιστρέφει το δεξιό παιδί του κόμβου



7.3 Πιθανό Θέμα με Ακμές και Κορυφές Δέντρου-ΟΧΙ

Ένα οποιοδήποτε δέντρο με $|V|=n$ κορυφές έχει $|E|=n-1$ ακμές

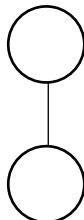
Απάντηση

Αποδεικνύουμε το θεώρημα αυτό με επαγωγή στο πλήθος n των κορυφών του γραφήματος.

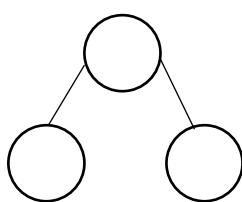
- **Βασικό Βήμα:** Η σχέση $|E|=n-1$ είναι προφανώς αληθής για $n=1, 2, 3$ όπως φαίνεται στα ακόλουθα σχήματα:



$$|E|=1-1=0$$



$$|E|=2-1=1$$



$$|E|=3-1=2$$

- **Επαγωγική Υπόθεση:** Υποθέτουμε ότι η σχέση $|E|=n-1$ είναι αληθής για όλα τα δέντρα με λιγότερες από n κορυφές
- **Επαγωγικό Βήμα:** Θα αποδείξουμε τη σχέση $|E|=n-1$ για ένα οποιοδήποτε δέντρο με n κορυφές. Έστω T δέντρο με n κορυφές και έστω e μια ακμή με κορυφές u και v . Το μοναδικό μονοπάτι μεταξύ των κορυφών u και v είναι η ακμή e . Η διαγραφή της ακμής e κάνει το T μη συνεκτικό γράφημα. Άρα τώρα το $T-e$ αποτελείται από ακριβώς δύο υπογραφήματα π.χ. T_1 και T_2 και επειδή δεν υπάρχουν κύκλοι στο αρχικό δέντρο T , δεν υπάρχουν κύκλοι ούτε και στα υπογραφήματα T_1 και T_2 , άρα κάθε υπογράφημα αποτελεί δέντρο. Έστω n_1 και n_2 ο αριθμός των κορυφών στα υπογραφήματα T_1 και T_2 αντίστοιχα. Ισχύει ότι $n_1+n_2=n$. Επίσης $n_1 < n$ και $n_2 < n$. Από την Επαγωγική Υπόθεση ο αριθμός των ακμών στα υποδέντρα T_1 και T_2 είναι n_1-1 και n_2-1 αντίστοιχα. Άρα ο συνολικός αριθμός ακμών στο T είναι $T = n_1-1 + n_2-1 + 1$ (η ακμή $u-v$) = $n_1+n_2-1=n-1$

7.4 Θέμα Σεπτέμβριος 2018-ΟΧΙ

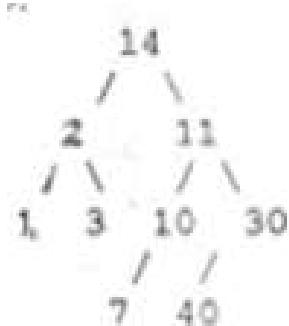
Θεωρείστε ένα δέντρο όπου κάθε εσωτερικός κόμβος έχει δύο ή παραπάνω παιδιά. Αποδείξτε ότι στο δέντρο αυτό το πλήθος των φύλλων είναι πάντα μεγαλύτερο ή ίσο από το πλήθος των εσωτερικών κόμβων +1

Απάντηση

7.5 Διασχίσεις Δέντρων

7.5.1.1 Θέμα 4 Ιούνιος 2008

Για το παρακάτω δυαδικό δέντρο δώστε τη σειρά επίσκεψης των κόμβων όταν εκτελούμε: α) Προδιάταξη ή Προδιαπέραση (Preorder)
β) Μεταδιάταξη ή Μεταδιαπέραση (Postorder) και γ) Συμμετρική διάταξη ή Ενδοδιαπέραση (Symmetric order ή Inorder)

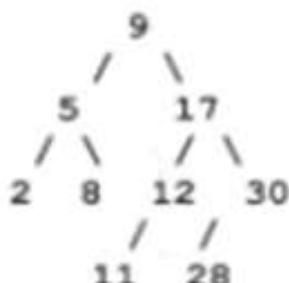


Απάντηση

- **Προδιάταξη (Preorder-ΠΑΔ)** → 14-2-1-3-11-10-7-30-40
- **Μεταδιάταξη (Postorder -ΑΔΠ)** → 1-3-2-7-10-40-30-11-14
- **Ενδοδιάταξη ή Συμμετρική Διάταξη (Symmetric ή Inorder)-ΑΠΔ** → 1-2-3-14-7-10-11-40-30

7.5.1.2 Θέμα 1 Ιούνιος 2009

Για το παρακάτω δυαδικό δέντρο δώστε τη σειρά επίσκεψης των κόμβων όταν εκτελούμε α) Προδιάταξη β) Μεταδιάταξη και γ)
Συμμετρική διάταξη



Απάντηση

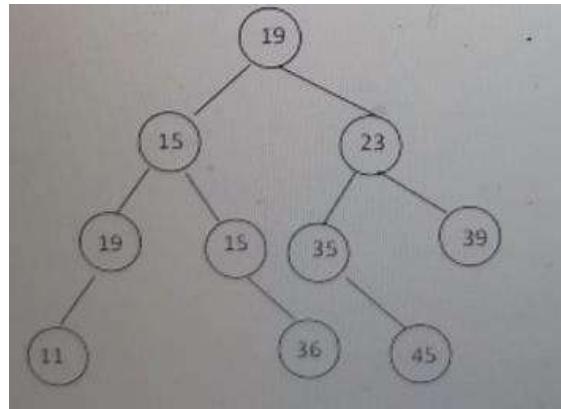
- **Προδιάταξη-Preorder (ΠΑΔ)** → 9-5-2-8-17-12-11-30-28
- **Μεταδιάταξη-Postorder (ΑΔΠ)** → 2-8-5-11-12-28-30-17-9
- **Ενδοδιάταξη ή Συμμετρική (Inorder- ΑΠΔ)** → 2-5-8-9-11-12-17-28-30

Παρατίρηση

Δυαδικό Δέντρο ονομάζεται αυτό που ο κάθε κόμβος έχει το πολύ 2 παιδιά.

7.6 Θέμα 2 Σεπτέμβριος 2020 – Ομάδα B

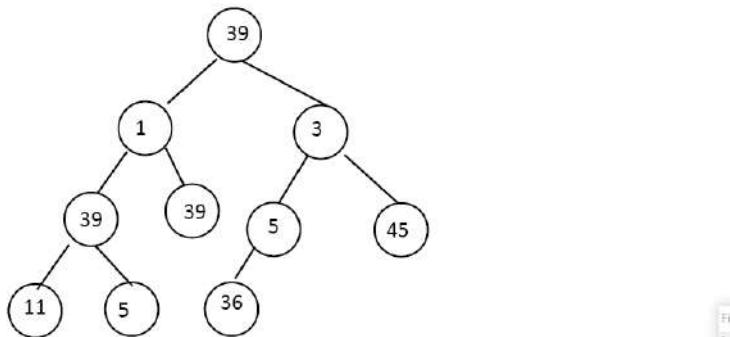
Σας δίνεται το ακόλουθο δέντρο. Δείξτε το αποτέλεσμα όλων των επισκέψεων

Απάντηση

- Η σειρά επίσκεψης των κόμβων με την **Προδιάταξη (Π Α Δ)** είναι: **19-15-19-11-15-36-23-35-45-39**
- Η σειρά επίσκεψης των κόμβων με την **Μεταδιάταξη (Α Δ Ρ)** είναι: **11-19-36-15-15-45-35-39-23-19**
- Η σειρά επίσκεψης των κόμβων με τη Συμμετρική διάταξη ή **Ενδοδιάταξη (Α Ρ Α)** είναι: **11-19-15-15-36-19-35-45-23-39**

7.6.1 Θέμα 4 Σεπτέμβριος 2020

Σας δίνεται το ακόλουθο δέντρο. Δείξτε το αποτέλεσμα της **προδιάταξης**.

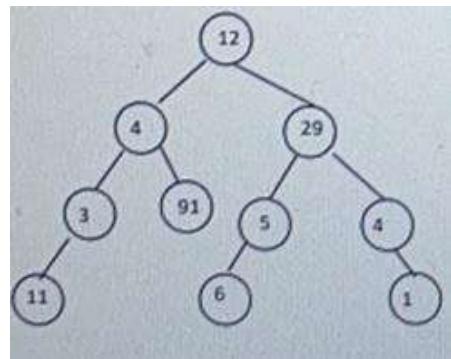
Απάντηση

Προδιάταξη Ρ Α Δ

39-1-39-11-5-39-3-5-36-45

7.6.2 θέμα 4 Φεβρουάριος 2021 - Ομάδα Α

Σας δίνεται το ακόλουθο δέντρο. Δείξτε το αποτέλεσμα όλων των Διαπεράσεων.



Απάντηση

Η σειρά επίσκεψης των κόμβων με τη Συμμετρική διάταξη (Α Ρ Δ) είναι:

11-3-4-91-12-6-5-29-4-1

Η σειρά επίσκεψης των κόμβων με την Προδιάταξη (Ρ Α Δ) είναι:

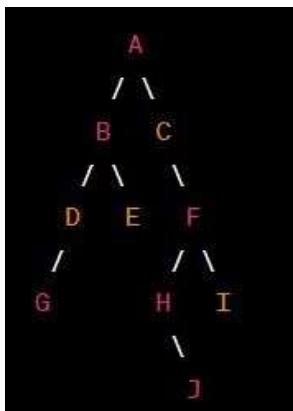
12-4-3-11-91-29-5-6-4-1

Η σειρά επίσκεψης των κόμβων με την Μεταδιάταξη (Α Δ Ρ) είναι:

11-3-91-4-6-5-1-4-29-12

7.6.3 Άσκηση με Διασχίσεις

Για το παρακάτω δυαδικό δέντρο δώστε τη σειρά επίσκεψης των κόμβων όταν εκτελούμε: α) Προδιάταξη ή Προδιαπέραση (Preorder)
 β) Μεταδιάταξη ή Μεταδιαπέραση (Postorder) και γ) Συμμετρική διάταξη ή Ενδοδιαπέραση (Symmetric order ή Inorder)



Απάντηση

α) Προδιάταξη ή Προδιαπέραση (Preorder)

A→B→D→G→E→C→F→H→J→I

β) Μεταδιάταξη ή Μεταδιαπέραση (Postorder)

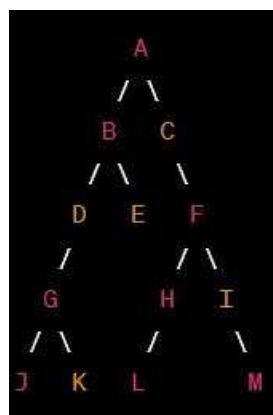
G→D→B→E→H→I→F→C→A

γ) Συμμετρική διάταξη ή Ενδοδιαπέραση (Symmetric order ή Inorder)

G→D→B→E→A→C→H→J→F→I

7.6.4 Άσκηση με Διασχίσεις

Για το παρακάτω δυαδικό δέντρο δώστε τη σειρά επίσκεψης των κόμβων όταν εκτελούμε: α) Προδιάταξη ή Προδιαπέραση (Preorder)
 β) Μεταδιάταξη ή Μεταδιαπέραση (Postorder) και γ) Συμμετρική διάταξη ή Ενδοδιαπέραση (Symmetric order ή Inorder)



Απάντηση

α) Προδιάταξη ή Προδιαπέραση (Preorder)

A→B→D→G→J→K→E→C→F→H→L→I→M

β) Μεταδιάταξη ή Μεταδιαπέραση (Postorder)

J→K→G→D→E→B→L→H→M→I→F→C→A

γ) Συμμετρική διάταξη ή Ενδοδιαπέραση (Symmetric order ή Inorder)

J→G→K→D→B→E→A→C→L→H→F→I→M

7.7 Ασκήσεις με Ανακατασκευή Δέντρου από Διασχίσεις

7.7.1 Πιθανό Θέμα 1 με Κατασκευή Δέντρου από τις Διατρέξεις του

Έχοντας ως δεδομένα τα αποτελέσματα της inorder και της preorder διάσχισης του δένδρου, περιγράψτε τη διαδικασία ανακατασκευής και να την επιδείξετε ανακατασκευάζοντας βήμα προς βήμα, ένα δυαδικό δένδρο στους κόμβους του οποίου είναι αποθηκευμένα γράμματα του λατινικού αλφαριθμητικού και για το οποίο ξέρετε τα εξής:

Inorder: d b g e a c f

Preorder: a b d e g c f

Απάντηση

Ζητάμε την ανακατασκευή του δέντρου και προχωράμε ως εξής:

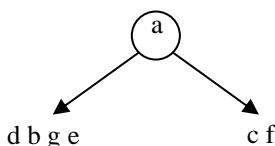
1) Από την Προδιάταξη ξέρουμε πως το a είναι η ρίζα (πάντα η διάσχιση αυτή ξεκινά από τη ρίζα).

Ενδοδιάταξη - Inorder: [d b g e] **a** [c f]

Προδιάταξη - Preorder: **a** b d e g c f

Πηγαίνουμε στην Ενδοδιάταξη - inorder και ότι υπάρχει αριστερά του a είναι στο αριστερό υποδένδρο με ρίζα το b ενώ ότι υπάρχει δεξιά του a είναι στο δεξιό υποδένδρο με ρίζα το e. Άρα [d b g e] είναι στο αριστερό υποδένδρο με ρίζα το a και [c f] είναι στο δεξιό υποδένδρο με ρίζα το e.

Υπενθύμιση ότι στην Ενδοδιάταξη έχουμε A-P-Δ. Άρα είμαστε σε θέση να ανακατασκευάσουμε το πρώτο επίπεδο του δένδρου

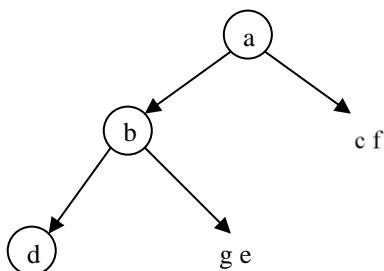


2) Συνεχίζουμε στην Προδιάταξη - preorder και ο επόμενος κόμβος είναι το b. Πηγαίνουμε στην Ενδοδιάταξη - inorder και ότι υπάρχει αριστερά του b είναι στο αριστερό υποδένδρο με ρίζα το b ενώ ότι υπάρχει δεξιά του b είναι στο δεξιό υποδένδρο με ρίζα το e. Άρα [d] είναι στο αριστερό υποδένδρο με ρίζα το b και [g e] είναι στο δεξιό υποδένδρο με ρίζα το e.

Ενδοδιάταξη - Inorder: [[d] **b** [g e]] **a** [c f]

Προδιάταξη - Preorder: **a** **b** d e g c f

Άρα είμαστε σε θέση να αρχίσουμε την ανακατασκευή του δεύτερου επιπέδου του δένδρου.



Το επόμενο βήμα μας πείθει πως το d είναι φύλλο και κατόπιν πάμε στο δεξιό υποδένδρο.

Ενδοδιάταξη - Inorder: [[**d**] **b** [g e]] **a** [c f]

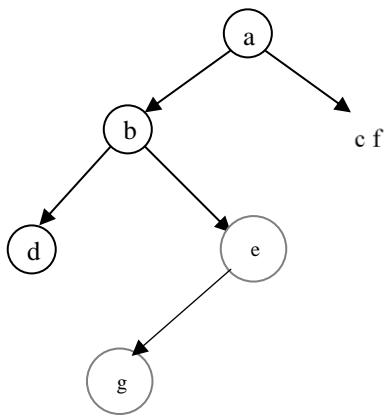
Προδιάταξη - Preorder: **a** **b** **d** e g c f

3) Το επόμενο βήμα είναι να πάμε στην προδιάταξη και να πάρουμε το e.

Ενδοδιάταξη - Inorder: [[**d**] **b** [[g] **e**]] **a** [c f]

Προδιάταξη - Preorder: **a** **b** **d** **e** g c f

Παρατηρούμε από την Ενδοδιάταξη ότι το g είναι αριστερό παιδί του e.

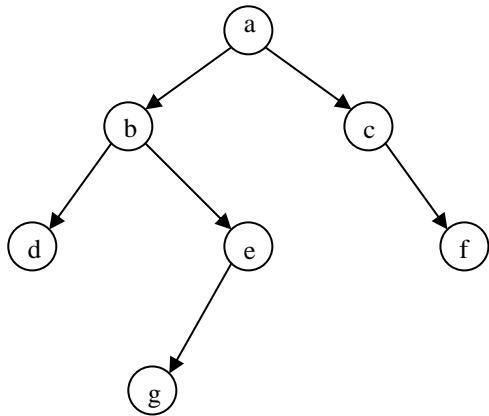


40Το επόμενο βήμα είναι να πάμε στην προδιάταξη και να πάρουμε το c. Το f είναι δεξί παιδί του c και το δέντρο ολοκληρώνεται.

Ενδοδιάταξη - Inorder: [[**d**] **b** [[**g**] **e**]] **a** [**c** [f]]

Προδιάταξη - Preorder: **a** **b** **d** **e** **g** **c** **f**

Το δέντρο είναι τελικά:

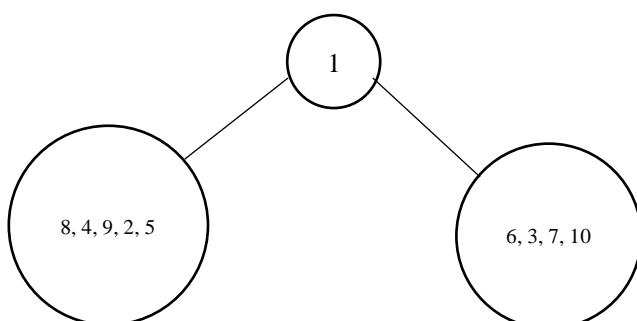


7.7.2 Ασκηση Κατασκευή Δέντρου από τις Διατρέξεις του

Ενδοδιάταξη (Inorder traversal: [8, 4, 9, 2, 5], **1**, [6, 3, 7, 10]

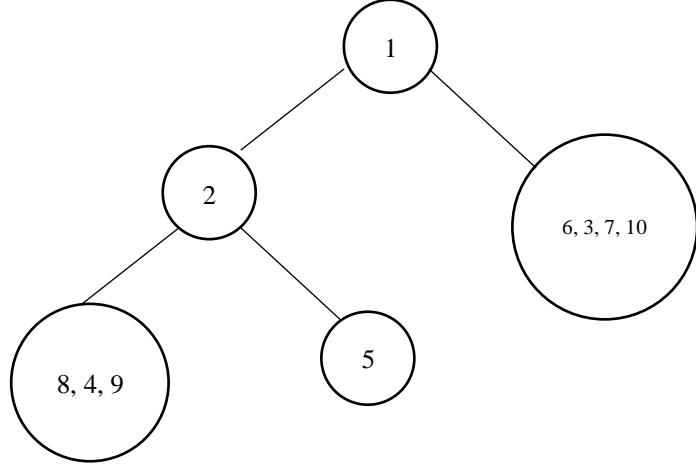
Προδιάταξη (Preorder traversal: **1**, 2, 4, 8, 9, 5, 3, 6, 7, 10

Απάντηση



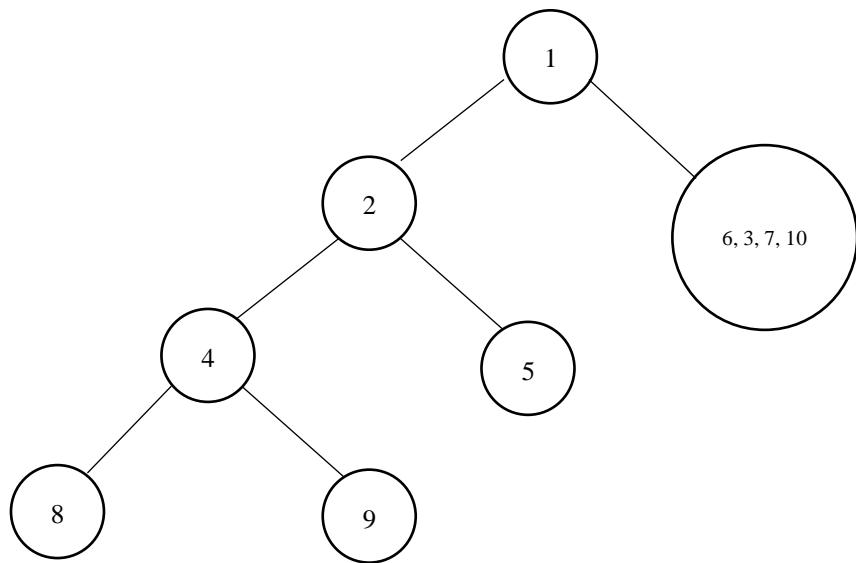
Ενδοδιάταξη (Inorder traversal: [8, 4, 9], **2**, [5], **1**, [6, 3, 7, 10]

Προδιάταξη (Preorder traversal: **1**, **2**, 4, 8, 9, 5, 3, 6, 7, 10



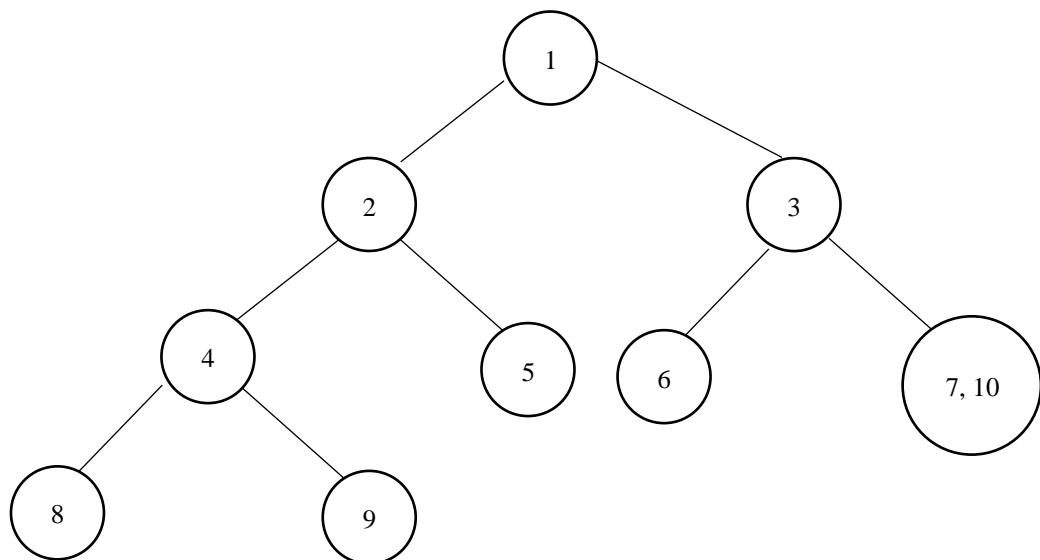
Ενδοδιάταξη (Inorder traversal: [8], **4**, [9], **2**, [5], **1**, [6, 3, 7, 10]

Προδιάταξη (Preorder traversal: **1**, **2**, **4**, 8, 9, 5, 3, 6, 7, 10



Ενδοδιάταξη (Inorder traversal: [8], **4**, [9], **2**, [5], **1**, [6], **3**, [7, 10]

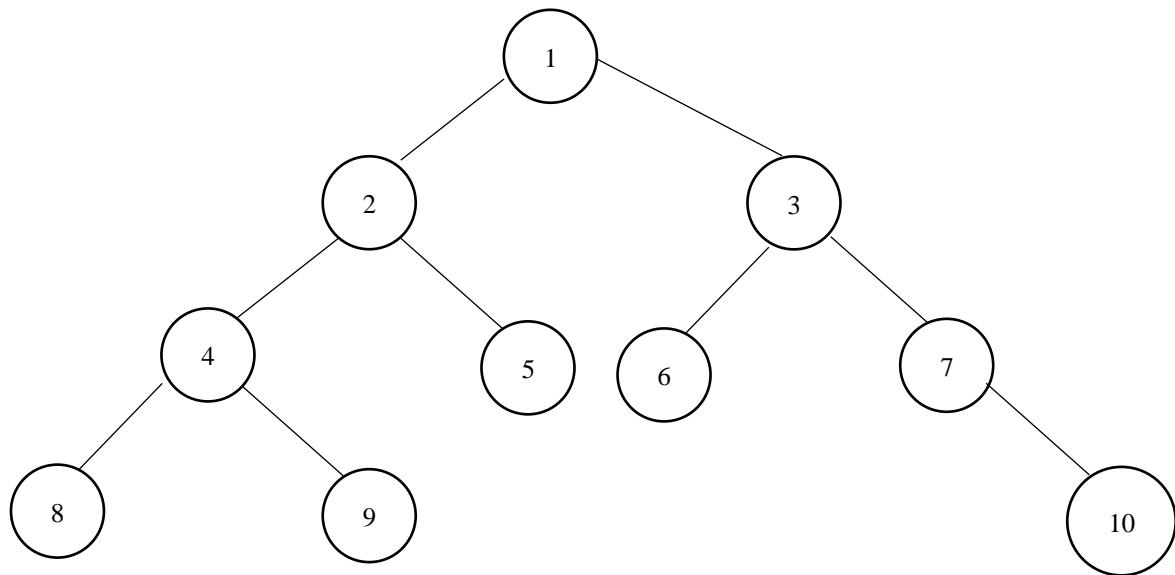
Προδιάταξη (Preorder traversal: **1**, **2**, **4**, **8**, **9**, **5**, **3**, 6, 7, 10



Δομές Δεδομένων –Computer Ανάλυση

Ενδοδιάταξη (Inorder traversal: [8], **4**, [9], **2**, [5], **1**, [6], **3**, **7**, [10])

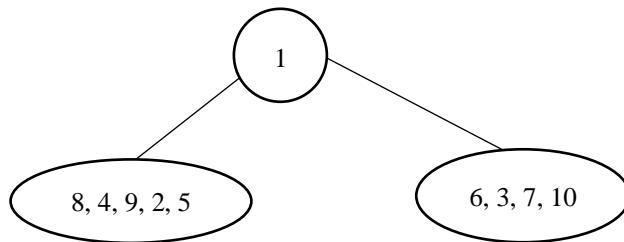
Προδιάταξη (Preorder traversal: **1**, **2**, **4**, **8**, **9**, **5**, **3**, **6**, **7**, 10)



7.7.3 Άσκηση Κατασκευή Δέντρου από τις Διατρέξεις του

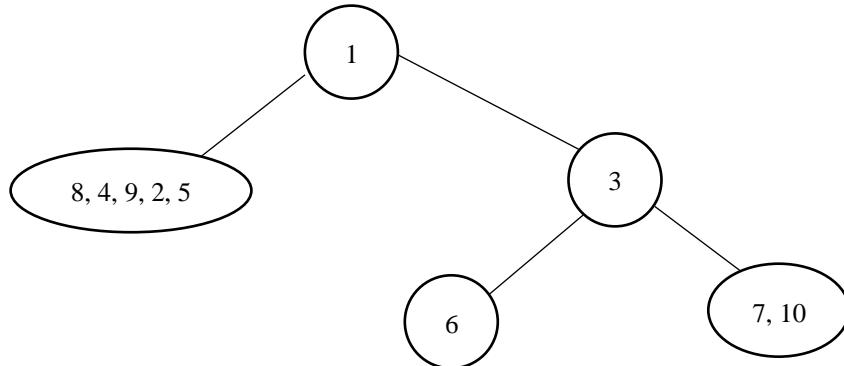
Μεταδιάταξη (Postorder traversal): 8, 9, 4, 5, 2, 6, 7, 10, 3, **1**

Συμμετρική Διάταξη (Symmetric Order): [8, 4, 9, 2, 5], **1**, [6, 3, 7, 10]



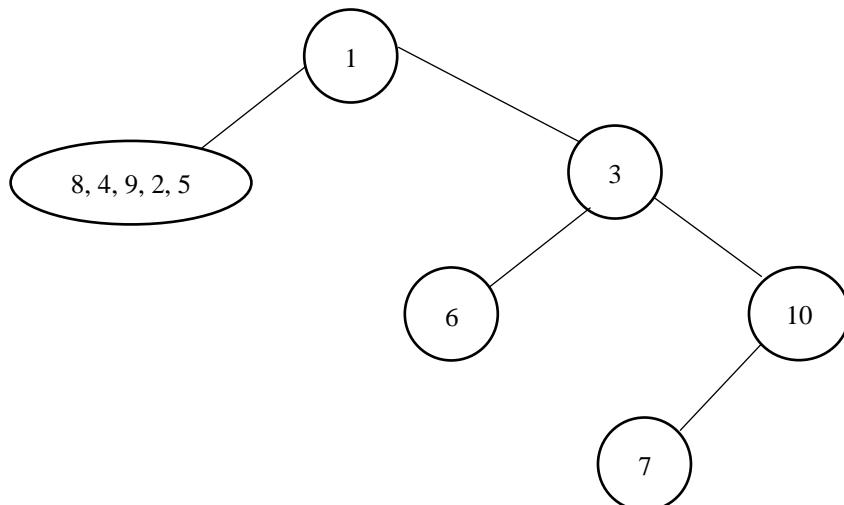
Μεταδιάταξη (Postorder traversal): 8, 9, 4, 5, 2, 6, 7, **10, 3, 1**

Συμμετρική Διάταξη (Symmetric Order): [8, 4, 9, 2, 5], **1**, [6], **3**, [7, 10]



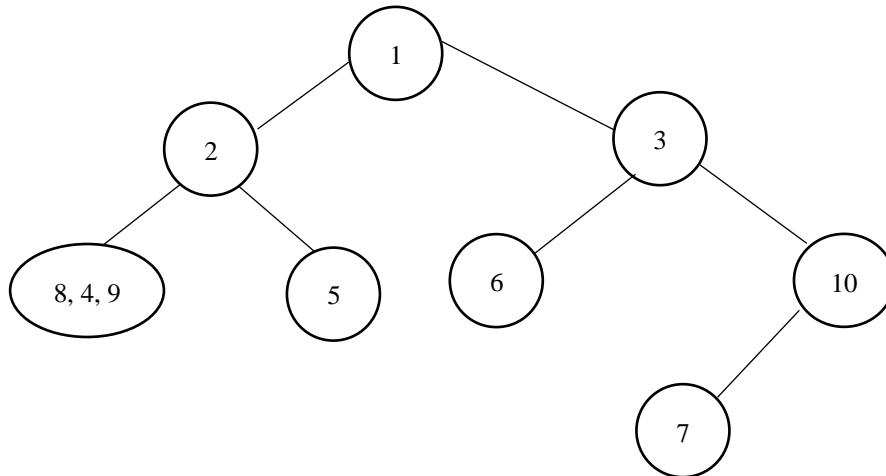
Μεταδιάταξη (Postorder traversal): 8, 9, 4, 5, 2, 6, 7, **10, 3, 1**

Συμμετρική Διάταξη (Symmetric Order): [8, 4, 9, 2, 5], **1**, [6], **3**, [7] **10**



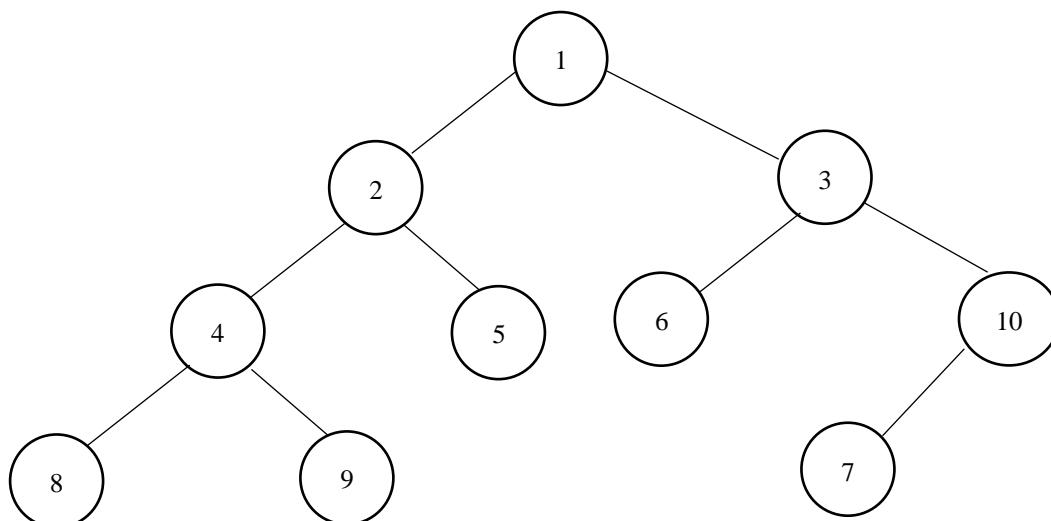
Μεταδιάταξη (Postorder traversal): 8, 9, 4, 5, **2, 6, 7, 10, 3, 1**

Συμμετρική Διάταξη (Symmetric Order): [8, 4, 9], **2, [5], 1, [6], 3, [7] 10**



Μεταδιάταξη (Postorder traversal): 8, 9, **4, 5, 2, 6, 7, 10, 3, 1**

Συμμετρική Διάταξη (Symmetric Order): [8], **4, [9], 2, [5], 1, [6], 3, [7] 10**



7.7.4 Πιθανό Θέμα 2 με Κατασκευή Δέντρου από τις Διατρέξεις του

Δίνεται η Προδιαπέραση (Preorder) 14-2-1-3-11-10-7-30-40. Δίνεται ακόμα η Μεταδιαπέραση (Postorder) 1-3-2-7-10-40-30-11-

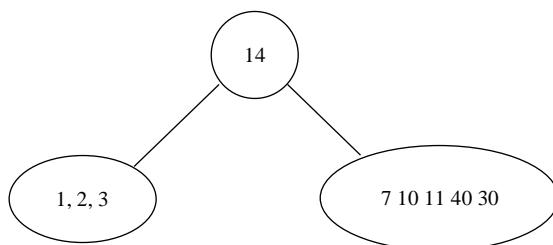
14. Δίνεται επίσης και η Ενδοδιαπέραση (Inorder) 1-2-3-14-7-10-11-40-30. Να κατασκευάσετε το αντίστοιχο δυναδικό δέντρο.

Απάντηση

- Στη Μεταδιάταξη - Μεταδιαπέραση ο τελευταίος κόμβος είναι η ρίζα. Η Μεταδιάταξη είναι Α-Δ-Ρ
- Στη Προδιάταξη - Προδιαπέραση ο αρχικός κόμβος που γράφεται είναι η ρίζα. Η Προδιάταξη είναι Ρ-Α-Δ
- Για την ανακατασκευή του δέντρου χρειάζονται μόνο η Προδιαπέραση (Preorder) και η Ενδοδιαπέραση (Inorder)
- Με βάση την Προδιάταξη και την Ενδοδιάταξη θα έχουμε τα εξής:

Ενδοδιάταξη -Inorder [1-2-3]-**14** [7 10 11 40 30]

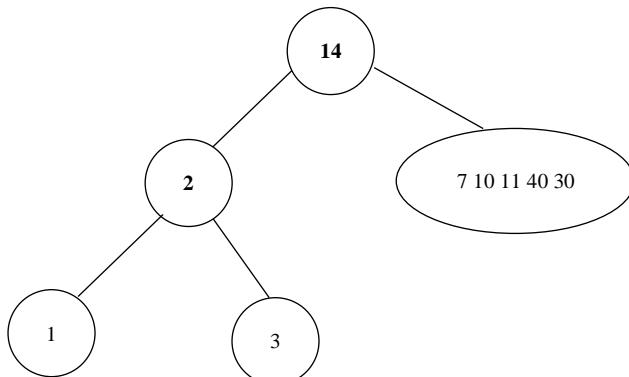
Προδιάταξη - Preorder: **14** 2 1 3 11 10 7 30 40



- Συνεχίζουμε στην προδιάταξη και επιλέγουμε τον επόμενο κόμβο δηλ. το 2

Ενδοδιάταξη -Inorder: [1] **2** [3]-**14** [7 10 11 40-30]

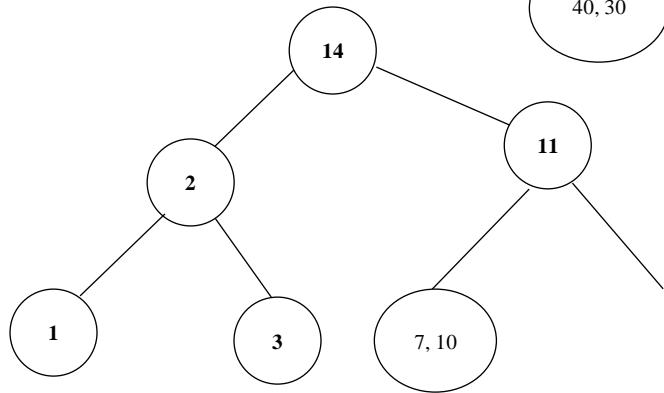
Προδιάταξη - Preorder: **14** **2** 1 3 11 10 7 30 40



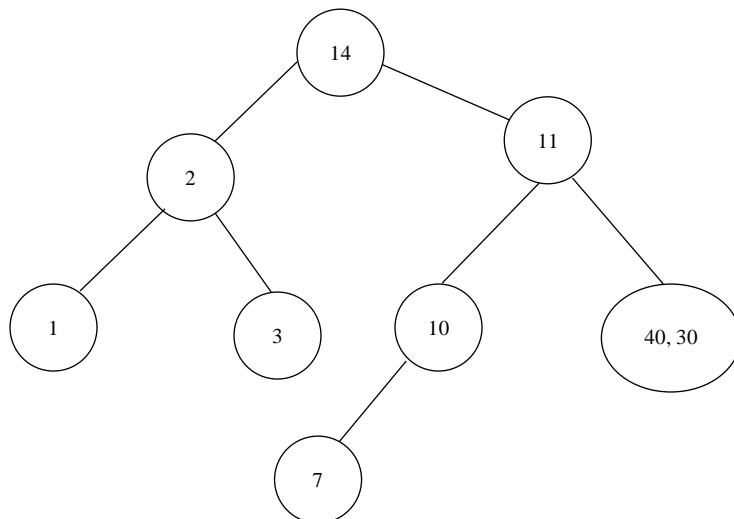
- Διαπιστώνουμε ότι οι κόμβοι 1 και το 3 έχουν τοποθετηθεί στο δέντρο. Άρα επιλέγουμε τον επόμενο κόμβο που είναι το 11

Ενδοδιάταξη -Inorder: [1] **2** [3]-**14** [7 10] **11** [40 30]

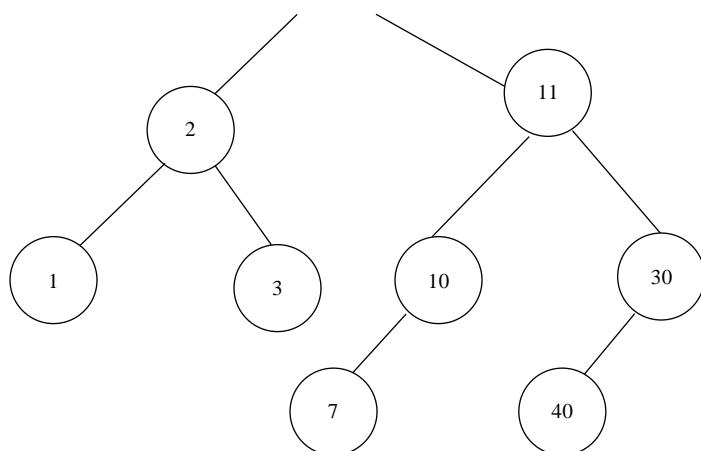
Προδιάταξη - Preorder: **14** **2** **1** **3** **11** 10 7 30 40



- Συνεχίζουμε στην προδιάταξη και επιλέγουμε τον επόμενο κόμβο δηλ. το 10
- Ενδοδιάταξη -Inorder:** [1] **2** [3]-**14** [7 **10**] **11** [40 30]
- Προδιάταξη - Preorder:** **14** **2** **1** **3** **11** **10** 7 30 40



- Συνεχίζουμε στην προδιάταξη και επιλέγουμε τον επόμενο κόμβο δηλ. το 30
- Ενδοδιάταξη -Inorder:** [1] **2** [3]-**14** [7] **1** **14** 40 **30**
- Προδιάταξη - Preorder:** **14** **2** **1** **3** **11** **10** 7 **30** 40
- Το δέντρο που σχηματίζεται σε κάθε περίπτωση είναι το ακόλουθο:



7.7.5 Θέμα 3 Ιούνιος 2018 με Κατασκευή Δέντρου από τις Διατρέξεις του

Δίνονται οι ακόλουθες διαπεράσεις ενός δυαδικού δέντρου

ΠΡΟΔΙΑΤΑΞΗ:

A	B	F	G	C	D	E	L	T
---	---	---	---	---	---	---	---	---

ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ:

F	<u>B</u>	G	<u>A</u>	D	<u>C</u>	L	E	T
---	----------	---	----------	---	----------	---	---	---

Ανασχηματίστε τη μορφή του συγκεκριμένου δυαδικού δέντρου

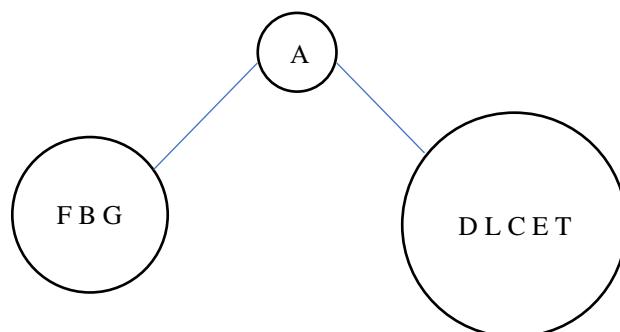
Απάντηση

ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ:

[F]	B	G]	<u>A</u>	[D]	C	L	E	T]
-----	---	----	----------	-----	---	---	---	----

ΠΡΟΔΙΑΤΑΞΗ:

<u>A</u>	B	F	G	C	D	E	L	T
----------	---	---	---	---	---	---	---	---

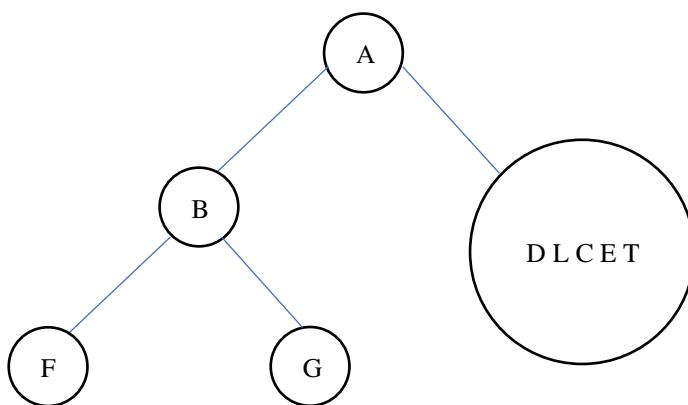


ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ:

[F]	<u>B</u>	[G]	<u>A</u>	[D]	C	L	E	T]
-----	----------	-----	----------	-----	---	---	---	----

ΠΡΟΔΙΑΤΑΞΗ:

<u>A</u>	<u>B</u>	F	G	C	D	E	L	T
----------	----------	---	---	---	---	---	---	---

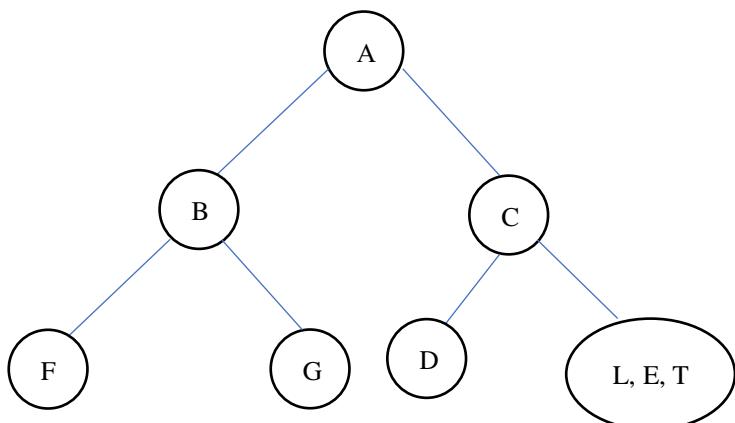


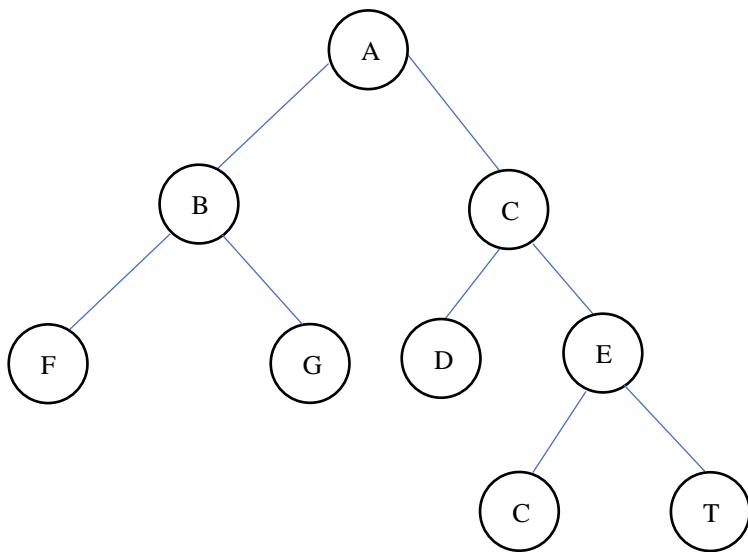
ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ:

[F]	<u>B</u>	[G]	<u>A</u>	[D]	<u>C</u>	[L]	E	T]
-----	----------	-----	----------	-----	----------	-----	---	----

ΠΡΟΔΙΑΤΑΞΗ:

<u>A</u>	<u>B</u>	F	G	<u>C</u>	D	E	L	T
----------	----------	---	---	----------	---	---	---	---



ΠΡΟΔΙΑΤΑΞΗ:**A B F G C D E L T****ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ:****[F] B [G] A [D] C [L] **E** L T**

7.7.6 Θέμα 3 Σεπτέμβριος 2018 με Κατασκευή Δέντρου από τις Διατρέξεις του

Δίνονται οι ακόλουθες διαπεράσεις ενός δυαδικού δέντρου:

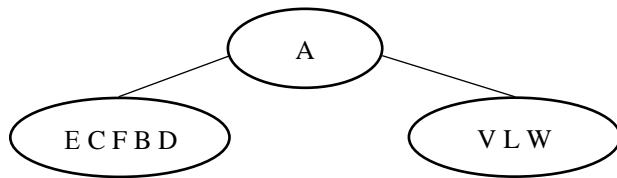
- **ΜΕΤΑΔΙΑΤΑΞΗ:** E F C D B V W L A
- **ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ:** E C F B D A V L W

Ανασχηματίστε τη μορφή του συγκεκριμένου δυαδικού δέντρου

Απάντηση

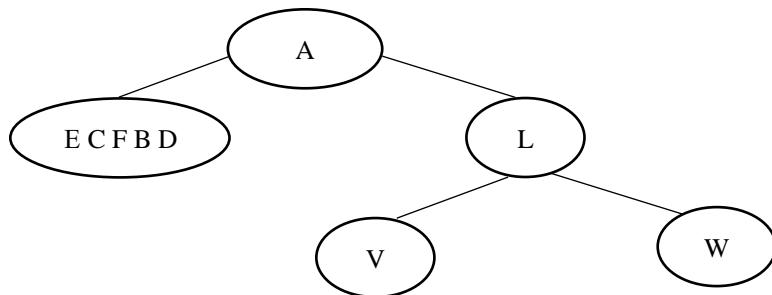
- Η ΜΕΤΑΔΙΑΤΑΞΗ έχει τη μορφή A-Δ-Ρ και η ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ ή αλλιώς ΕΝΔΟΔΙΑΤΑΞΗ έχει τη μορφή A-P-Δ
- Στη **Μεταδιάταξη** ο τελευταίος κόμβος είναι **ΠΑΝΤΑ η ρίζα του συνολικού δέντρου**
- Άρα αρχίζουμε από τη μεταδιάταξη και υπογραμμίζουμε τον τελευταίο κόμβο ως ρίζα όλου του δέντρου

ΜΕΤΑΔΙΑΤΑΞΗ:	E	F	C	D	B	V	W	L	A
ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ:	[E	C	F	B	D]	A	[V	L	W]



Συνεχίζουμε στη Μετάδιαταξη προς τα αριστερά και επιλέγουμε τον προτελευταίο κόμβο δηλ. το L και το υπογραμμίζουμε ως ρίζα υποδέντρου

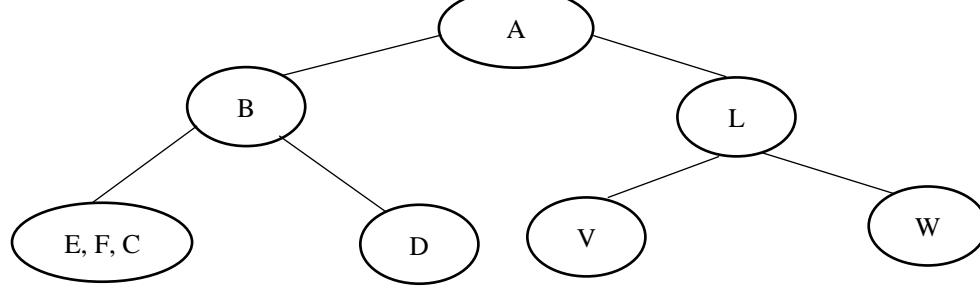
ΜΕΤΑΔΙΑΤΑΞΗ:	E	F	C	D	B	V	W	<u>L</u>	A
ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ:	[E	C	F	<u>B</u>	D]	<u>A</u>	[[V]]	<u>L</u>	[W]]



- Συνεχίζουμε στη Μετάδιαταξη προς τα αριστερά και επιλέγουμε το B

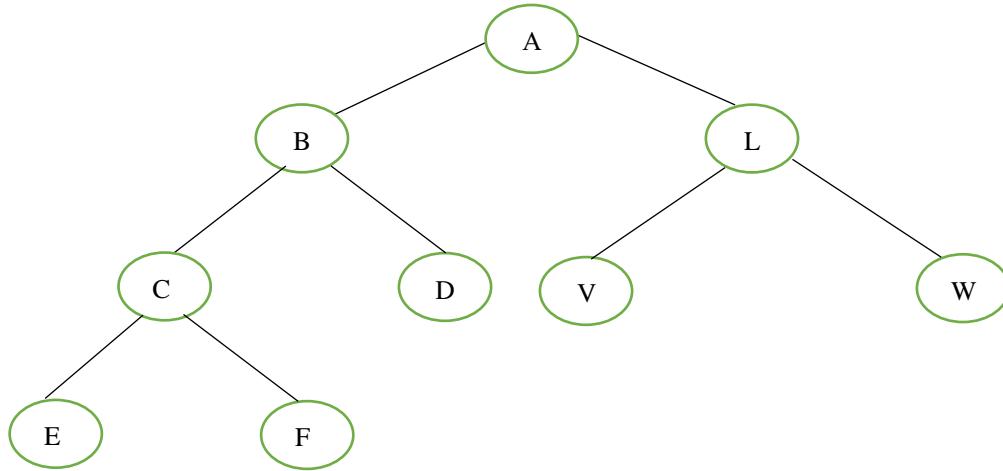
ΜΕΤΑΔΙΑΤΑΞΗ:	E	F	C	D	<u>B</u>	V	W	<u>L</u>	A
ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ:	[E	C	F]	<u>B</u>	[D]	<u>A</u>	[[V]]	<u>L</u>	[W]]

Δομές Δεδομένων –Computer Ανάλυση



- Συνεχίζουμε στη Μετάδιαταξη προς τα αριστερά και επιλέγουμε το C

ΜΕΤΑΔΙΑΤΑΞΗ:	E	F	C	D	B	V	W	L	A	
ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ:	[E]		C	[F]	B	D]	A	[V]	L	[W]



Αυτό είναι το τελικό δέντρο

7.7.7 Θέμα 2 Σεπτέμβριος 2020

Δίνονται οι ακόλουθες διαπεράσεις ενός δυαδικού δένδρου:

ΠΡΟΔΙΑΤΑΞΗ:

ACEMFGBDL

ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ:

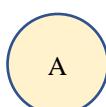
MECFGADBL

Ανασχηματίστε τη μορφή του συγκεκριμένου δυαδικού δένδρου.

Απάντηση

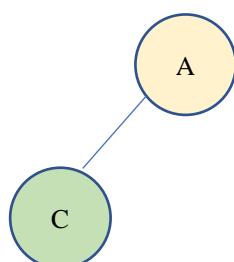
ΠΡΟΔΙΑΤΑΞΗ: A C E M F G B D L

ΣΥΜΜΕΤΡΙΚΗ: [M E] C F G] A [D B B L]



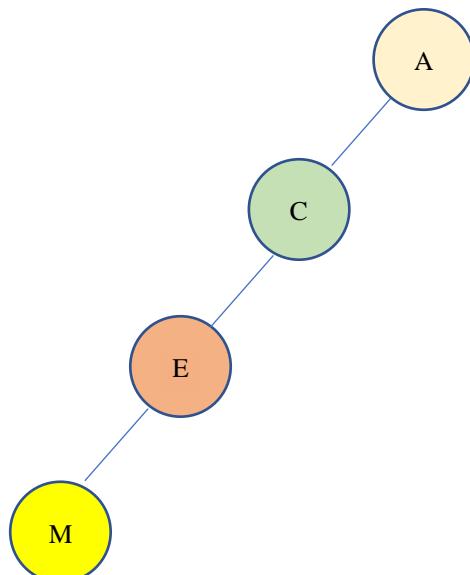
ΠΡΟΔΙΑΤΑΞΗ: A C E M F G B D L

ΣΥΜΜΕΤΡΙΚΗ: [M E] C [F G] A [D B B L]



ΠΡΟΔΙΑΤΑΞΗ: A C E M F G B D L

ΣΥΜΜΕΤΡΙΚΗ: [M E] C [F G] A [D B B L]



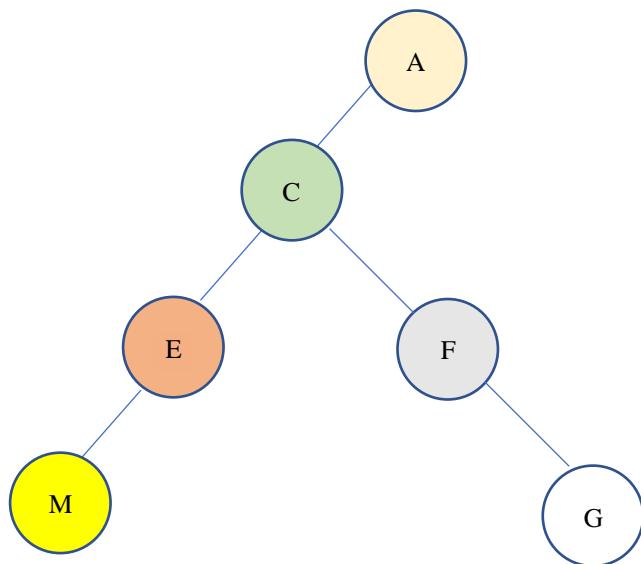
Δομές Δεδομένων –Computer Ανάλυση

ΠΡΟΔΙΑΤΑΞΗ:

A C E M F G B D L

ΣΥΜΜΕΤΡΙΚΗ:

[M] [E] [C] [F] [G] [A] [D] [B] [L]

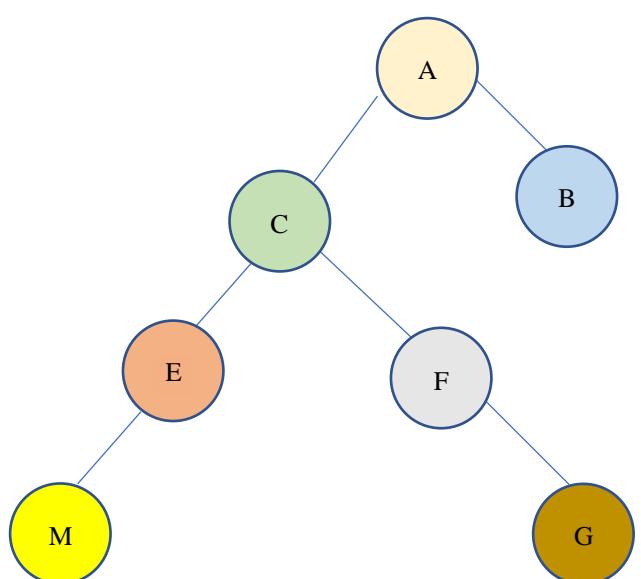


ΠΡΟΔΙΑΤΑΞΗ:

A C E M F G B D L

ΣΥΜΜΕΤΡΙΚΗ:

[M] [E] [C] [F] [G] [A] [D] [B] [L]



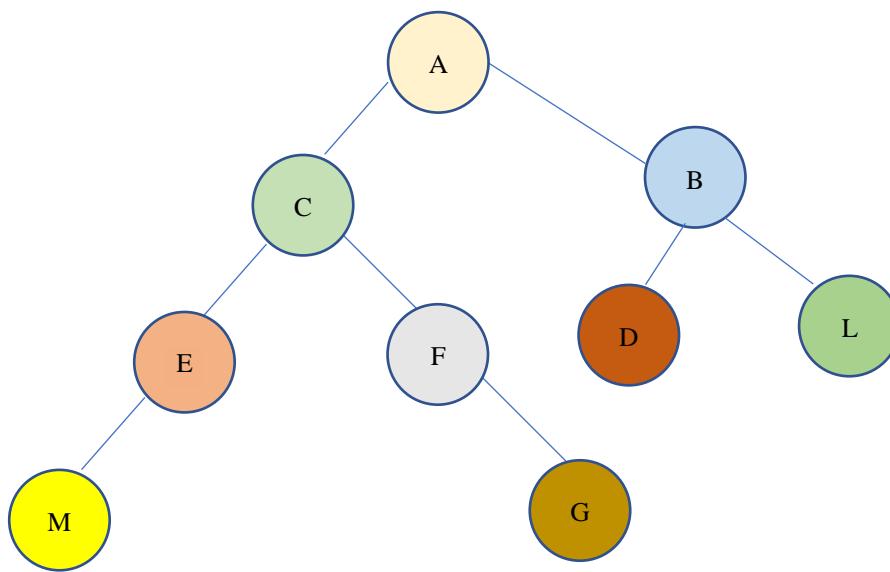
Δομές Δεδομένων –Computer Ανάλυση

ΠΡΟΔΙΑΤΑΞΗ:

A C E M F G B D L

ΣΥΜΜΕΤΡΙΚΗ:

[M] [E] [C] [F] [G] [A] [D] [B] [L]



7.7.8 θέμα 3 και 4 Φεβρουάριος 2021 – Ομάδα B

Δίνονται οι ακόλουθες διαπεράσεις ενός δυαδικού δένδρου:

ΜΕΤΑΔΙΑΤΑΞΗ: QLDEFGKHIJA

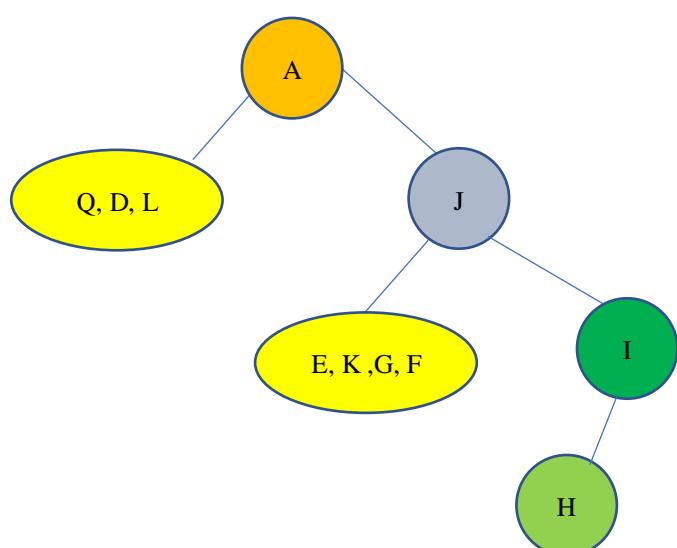
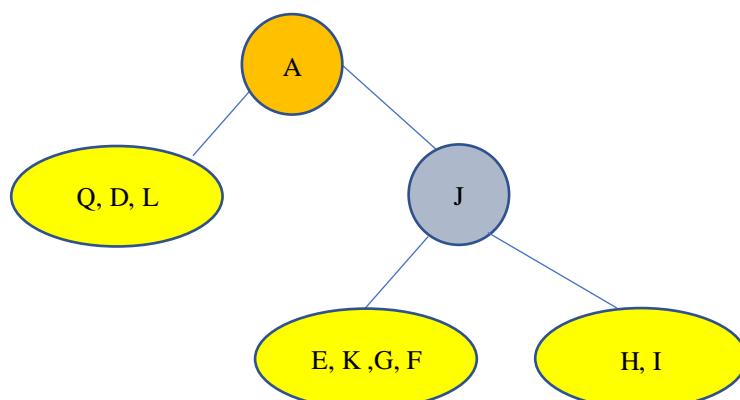
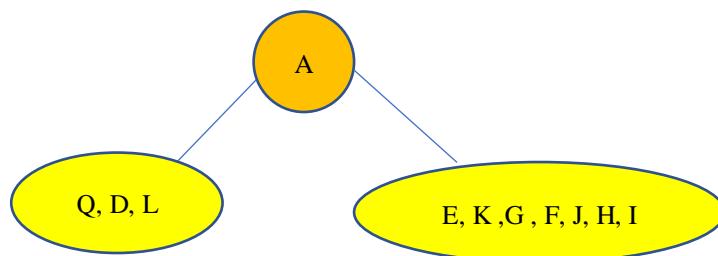
ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ: QDLAEKGFIJI

Ανασχηματίστε τη μορφή του συγκεκριμένου δυαδικού δένδρου.

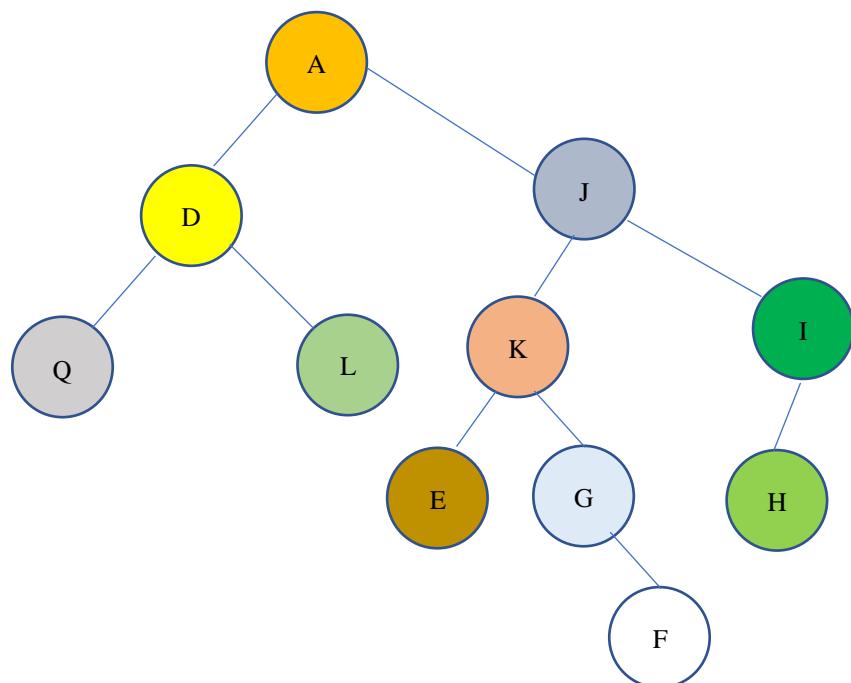
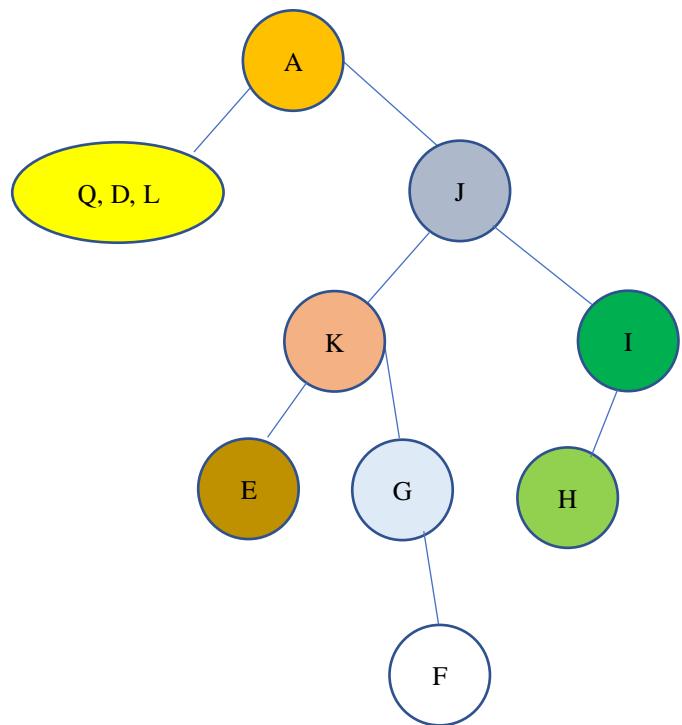
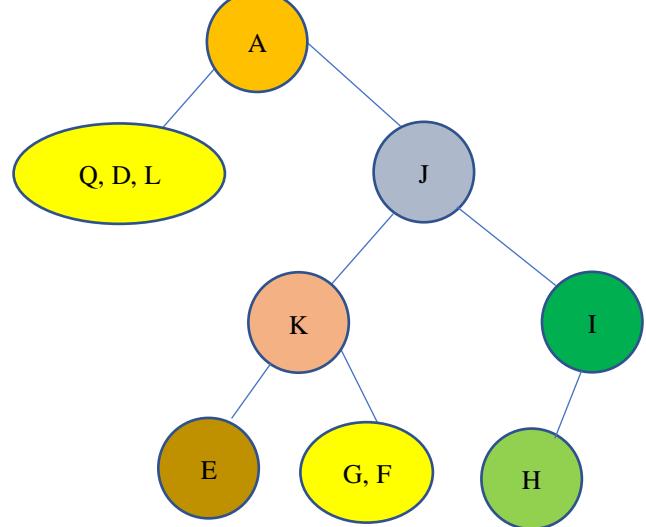
Απάντηση

ΜΕΤΑΔΙΑΤΑΞΗ: Q L D E F G K H I J A

ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ: [Q] **D** [L] **A** [E] [F] **K** **G** [H] **J** [I] [H] **I**



Δομές Δεδομένων –Computer Ανάλυση



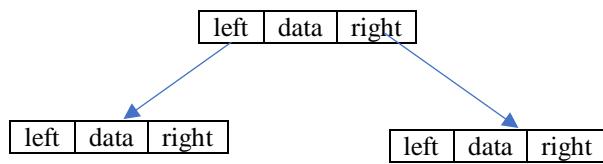
7.8 Πιθανό Θέμα με Κώδικα Διαπεράσεων Δέντρου

Να γραφούν σε γλώσσα C οι μέθοδοι διαπέρασης ενός δυαδικού δέντρου.

Απάντηση

Ο τύπος TreeNode περιγράφει ένα κόμβο του δέντρου και δηλώνεται ως εξής:

```
struct TreeNode
{
    int data;
    struct TreeNode *left;
    struct TreeNode *right;
};
```



Αρχικά ο δείκτης t με τον οποίο καλείται κάθε συνάρτηση δείχνει στη ρίζα του δέντρου

Προδιάταξη (ΠΑΔ)

void preorder(TreeNode *t) //Η συνάρτηση preorder υλοποιεί την προδιάταξη. Δέχεται ως όρισμα το δείκτη t που είναι δείκτες σε κόμβο και αρχικά δείχνει στη ρίζα

```
{
    if (t!=NULL)
    {
        printf("%d",t->data);
        preorder(t->left);
        preorder(t->right);
    }
}
```

Συμμετρική Διάταξη ή Ενδοδιάταξη (ΑΡΔ)

void inorder(TreeNode *t) //Η συνάρτηση inorder υλοποιεί την ενδοδιάταξη. Δέχεται ως όρισμα το δείκτη t που είναι δείκτες σε κόμβο και αρχικά δείχνει στη ρίζα

```
{
    if (t!=NULL)
    {
        inorder(t->left);
        printf("%d", t->data);
        inorder(t->right);
    }
}
```

Μεταδιάταξη (ΑΔΡ)

void postorder(TreeNode *t) //Η συνάρτηση postorder υλοποιεί τη μεταδιάταξη. Δέχεται ως όρισμα το δείκτη t που είναι δείκτες σε κόμβο και αρχικά δείχνει στη ρίζα

```
{
    if (t!=NULL)
    {
        postorder (t->left);
        postorder (t->right);
        printf("%d", t->data);
    }
}
```

7.9 Πιθανό θέμα Διαπεράσεων - OXI

Πόσες και ποιές διαπεράσεις απαιτούνται για να ανακατασκευάσουμε το αρχικό δυαδικό δένδρο;

Απάντηση

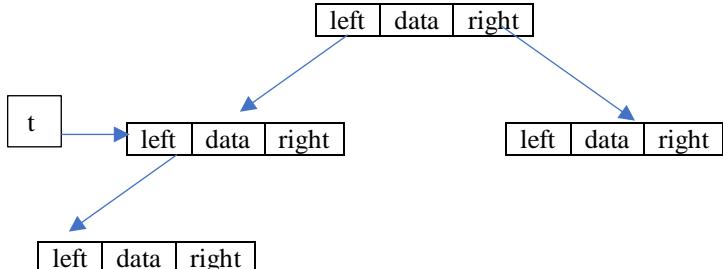
- Αν έχουμε μόνο μεταδιαπέραση (postorder) ή μόνο προδιαπέραση (preorder) δεν είναι εφικτός ο προσδιορισμός αριστερού και δεξιού υποδέντρου, ενώ αν έχουμε μόνο ενδοδιαπέραση (inorder) δεν μπορούμε να προσδιορίσουμε τη ρίζα. Συνεπώς μία διαπέραση δεν αρκεί
- Αν έχουμε δύο διαπεράσεις και συγκεκριμένα μεταδιαπέραση (postorder) και προδιαπέραση (preorder) και πάλι είναι αδύνατο να προσδιορίσουμε αριστερό και δεξιό υποδένδρο
- Αν έχουμε μεταδιαπέραση, προδιαπέραση ΚΑΙ ενδοδιαπέραση, τότε με την προδιαπέραση ή με τη μεταδιαπέραση μπορούμε να προσδιορίσουμε τη ρίζα και με την ενδοδιαπέραση να προσδιορίσουμε τα στοιχεία του αριστερού και δεξιού υποδέντρου. Αρα δεν χρειάζονται και η μεταδιαπέραση και η προδιαπέραση, μόνο μια από τις δύο, σε συνδυασμό με την ενδοδιαπέραση. ΣΥΝΕΠΙΩΣ ΔΥΟ ΔΙΑΠΕΡΑΣΕΙΣ είναι αρκετές αρκεί η μία από τις δύο να είναι η ενδοδιαπέραση και η δεύτερη η προδιαπέραση ή η μεταδιαπέραση

7.10 Πιθανό θέμα Εύρεσης Μέγιστου Βάθους ή Ύψους ενός δέντρου-OXI

Να γραφεί κώδικας C για την εύρεση του μέγιστου Βάθους ή Ύψους ενός δυαδικού δέντρου.

Απάντηση

```
struct TreeNode
{
    int data;
    struct TreeNode * left;
    struct TreeNode * right;
};
```



```
int maxDepth(struct TreeNode * t)
{
    if (t==NULL)
        return 0;
    else
    {
        /* compute the depth of each subtree */

        int lDepth = maxDepth(t->left); /*υπολογισμός ύψους αριστερού υποδέντρου*/
        int rDepth = maxDepth(t->right); /*υπολογισμός ύψους δεξιού υποδέντρου*/

        if (lDepth > rDepth)
            return lDepth+1; /*το +1 είναι για τη μέτρηση της ρίζας*/
        else
            return rDepth+1;
    }
}
```

7.11 Κατασκευή Φυλλοπροσανατολισμένου Δέντρου Εύρεσης -OXI

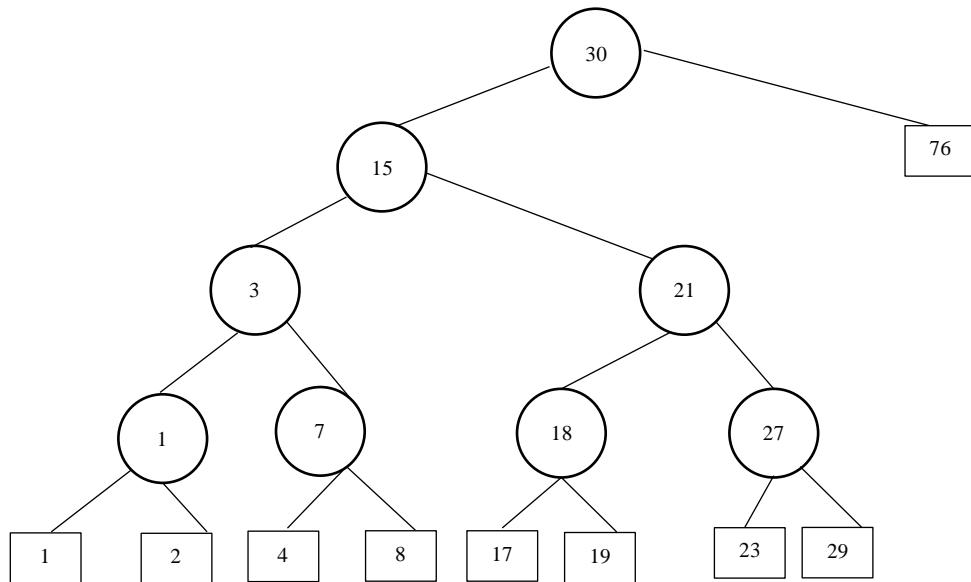
7.11.1 Θέμα 2 Ιούνιος 2009-OXI

Δίνεται το σύνολο $S=\{4, 29, 8, 17, 1, 19, 23, 2, 76\}$. Να κατασκευαστεί το Φυλλοπροσανατολισμένο Δέντρο Εύρεσης για το σύνολο S . Αναπαραστήστε το δέντρο που κατασκευάσατε με τη χρήση πίνακα.

Απάντηση

- **Φυλλοπροσανατολισμένο** είναι το δέντρο που όλες οι τιμές του συνόλου βρίσκονται στα φύλλα του
- **Δέντρο Εύρεσης (Αναζήτησης)** είναι το δέντρο που κάθε κόμβος του ικανοποιεί την ακόλουθη συνθήκη: $\text{ΑΠ} \leq \text{Ρίζα} < \Delta \Pi$
- Για να κατασκευάσουμε το δέντρο πρέπει πρώτα να ταξινομήσουμε τις τιμές του δηλ. $S=\{1, 2, 4, 8, 17, 19, 23, 29, 76\}$

Φυλλοπροσανατολισμένο Δέντρο Εύρεσης



Ο Πίνακας που κατασκευάζεται για το Φυλλοπροσανατολισμένο Δέντρο Εύρεσης είναι ο ακόλουθος:

Γραμμή	Περιεχόμενο	ΑΠ	ΔΠ	Σχόλιο
1	30	2	3	Ρίζα
2	15	4	5	—
3	76	—	—	Φύλλο
4	3	6	7	—
5	21	8	9	—
6	2	10	11	—
7	7	12	13	—
8	18	14	15	—
9	27	16	17	—
10	1	—	—	Φύλλο
11	2	—	—	Φύλλο
12	4	—	—	Φύλλο
13	8	—	—	Φύλλο
14	17	—	—	Φύλλο
15	19	—	—	Φύλλο
16	23	—	—	Φύλλο
17	29	—	—	Φύλλο

7.12 Δυαδικά Δέντρα Αναζήτησης - Εύρεσης (Binary Search Tree – BST)

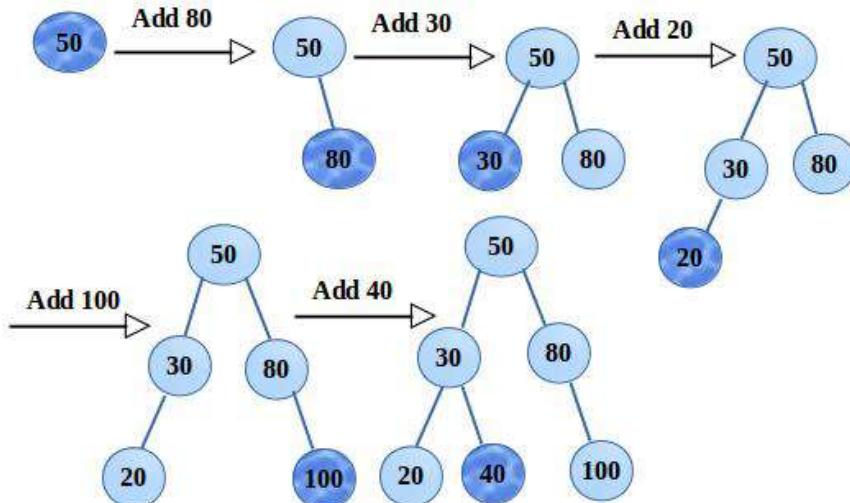
7.12.1 Παράδειγμα 1 με Κατασκευή Δυαδικού Δέντρου Αναζήτησης

Να κατασκευαστεί **Δυαδικό Δέντρο Αναζήτησης (Binary Search Tree – BST)** με τις ακόλουθες τιμές: **50, 80, 30, 20, 100, 40**

Απάντηση

Βασική Θεωρία: Σε ένα δυαδικό δέντρο αναζήτησης (Binary Search Tree – BST) ισχύει ο εξής βασικός κανόνας:

ΑΠ≤Ρίζα<ΔΠ



Παρατήρηση

Αρχίζουμε από τη ρίζα του δέντρου. Συγκρίνουμε κάθε νέο κόμβο που εισάγεται με τη ρίζα του δέντρου. Αν η τιμή του νέου κόμβου είναι μεγαλύτερη της ρίζας πηγαίνουμε δεξιά και αν είναι μικρότερη της ρίζας πηγαίνουμε αριστερά. Η διαδικασία συνεχίζεται μέχρι να τοποθετήσουμε το νέο κόμβο στη σωστή θέση του BST. Πρέπει να προσέχουμε κατά την τοποθέτηση κάθε νέου κόμβου ώστε να ικανοποιείται πάντα η σχέση $\text{ΑΠ} \leq \text{Ρίζα} < \Delta\text{Π}$

7.12.2 Παράδειγμα 2 με Κατασκευή Δυαδικού Δέντρου Αναζήτησης

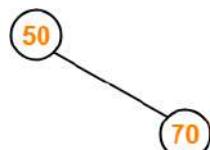
Να κατασκευαστεί **Δυαδικό Δέντρο Αναζήτησης (Binary Search Tree – BST)** με τις ακόλουθες τιμές **50, 70, 60, 20, 90, 10, 40, 100**.

Απάντηση

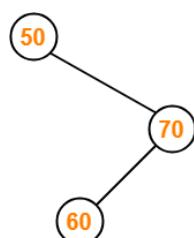
Insert 50



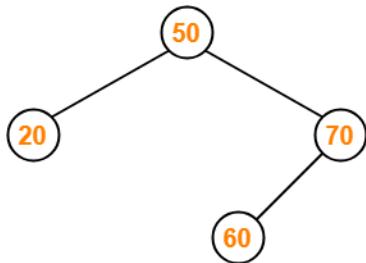
Insert 70



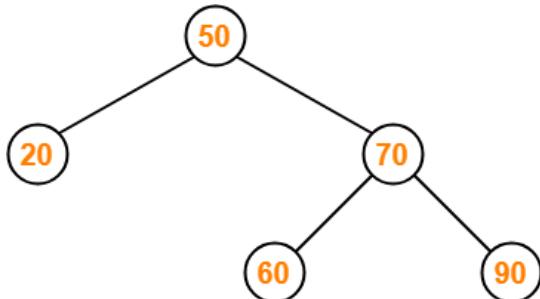
Insert 60



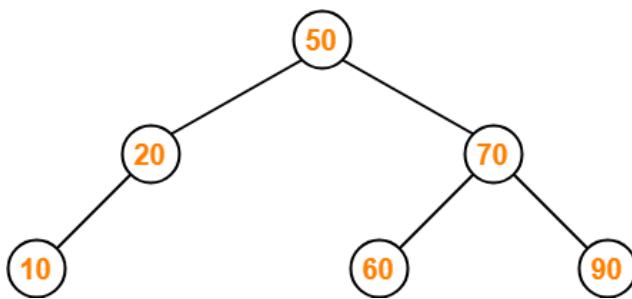
Insert 20



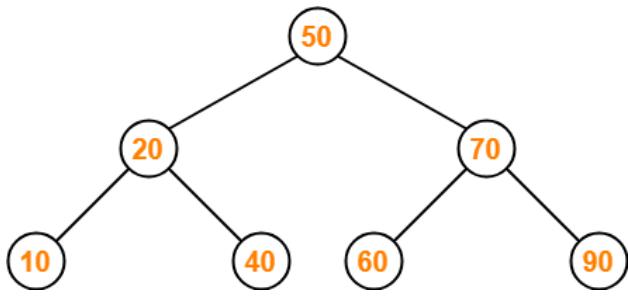
Insert 90



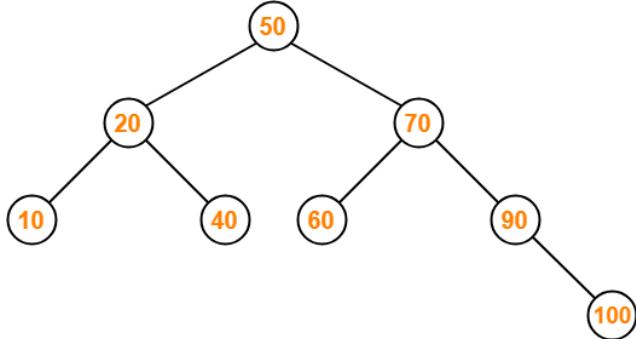
Insert 10



Insert 40

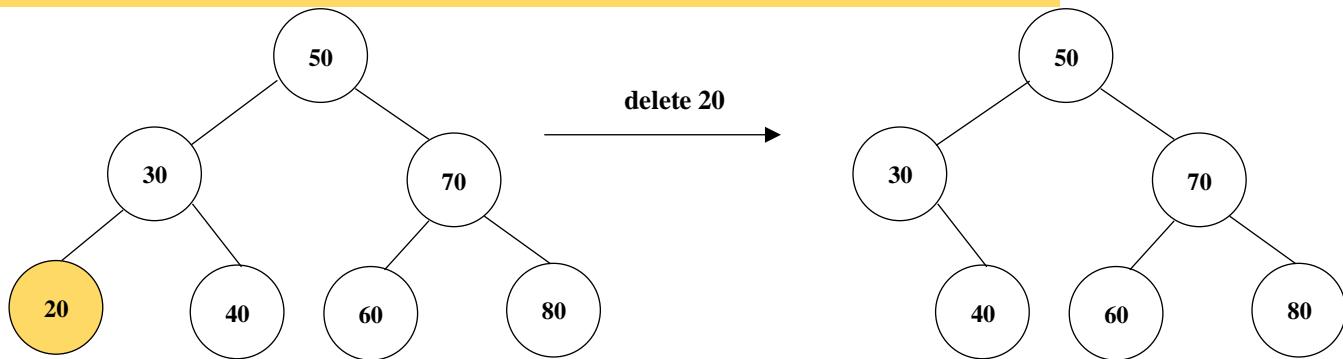


Insert 100

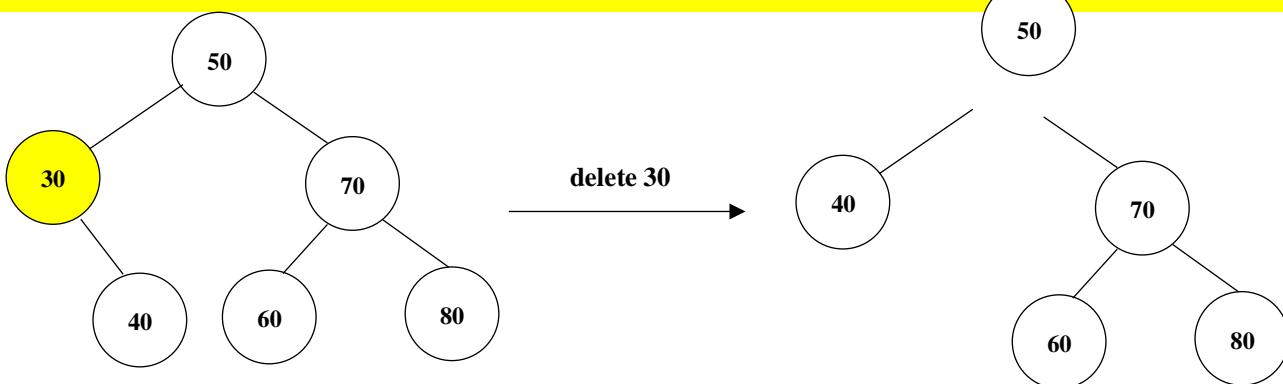


7.12.3 Περιπτώσεις Διαγραφών σε Δυαδικό Δέντρο Αναζήτησης

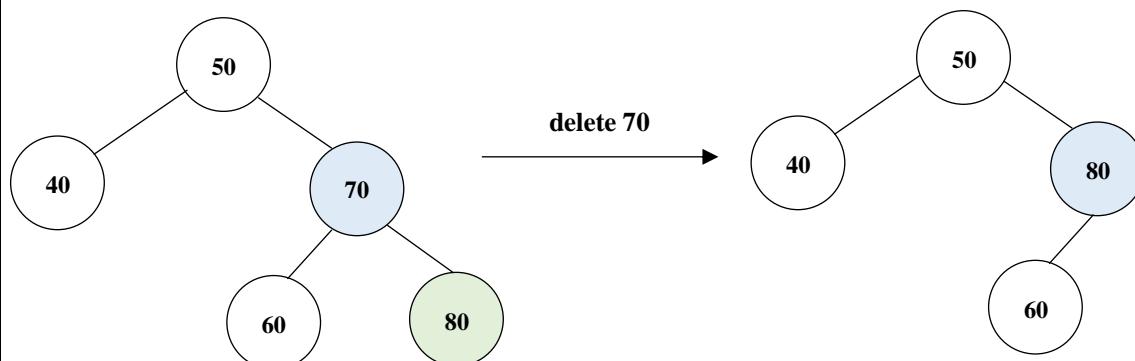
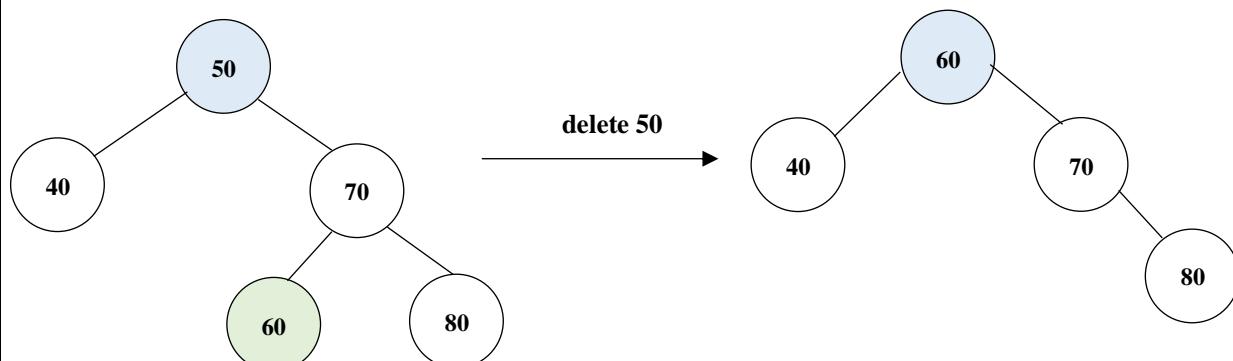
(1) Ο κόμβος που θα διαγραφεί είναι φύλλο. Στην περίπτωση αυτή αφαιρείται από το δέντρο.



2) Ο κόμβος που θα διαγραφεί έχει μόνο 1 παιδί: Στη θέση του κόμβου που διαγράφεται τοποθετείται το παιδί του.



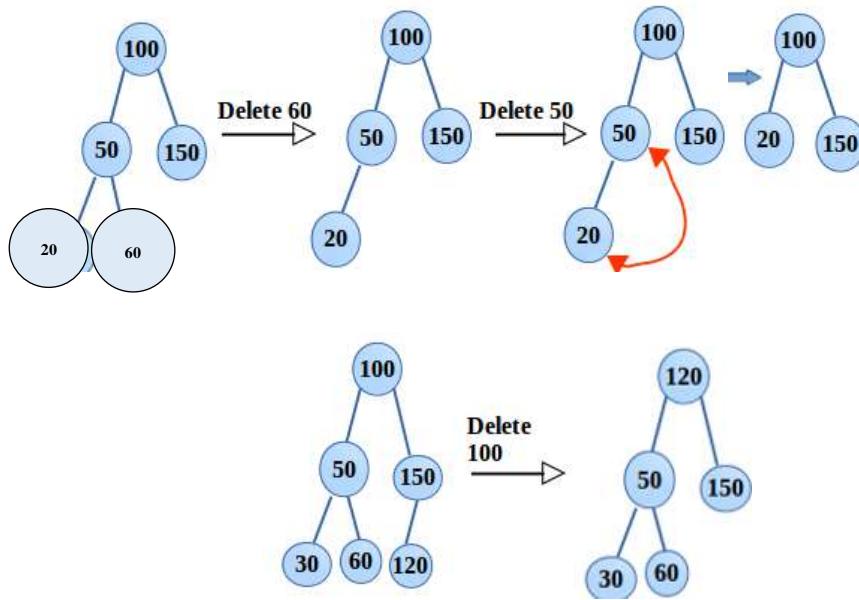
3) Ο κόμβος που θα διαγραφεί έχει 2 παιδιά: Βάζουμε στη θέση του κόμβου που διαγράφεται τη μικρότερη τυμή του δεξιού υποδέντρου του



7.12.4 Παράδειγμα με Διαγραφή στοιχείων από Δυαδικό Δέντρο Αναζήτησης

Να διαγραφούν τα στοιχεία **60, 50, 100** από το Δυαδικό Δέντρο Αναζήτησης (Binary Search Tree – BST)

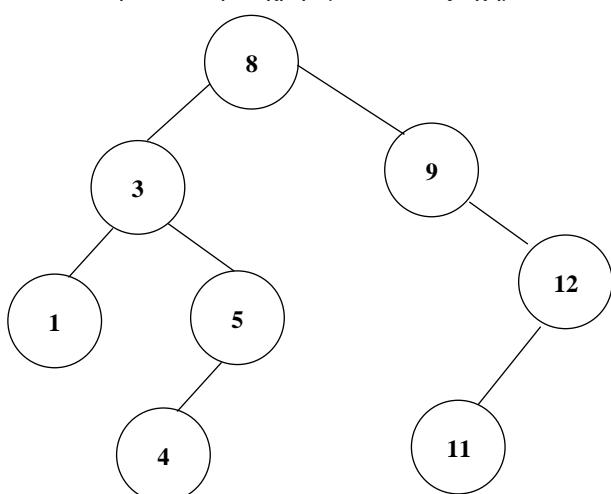
Απάντηση



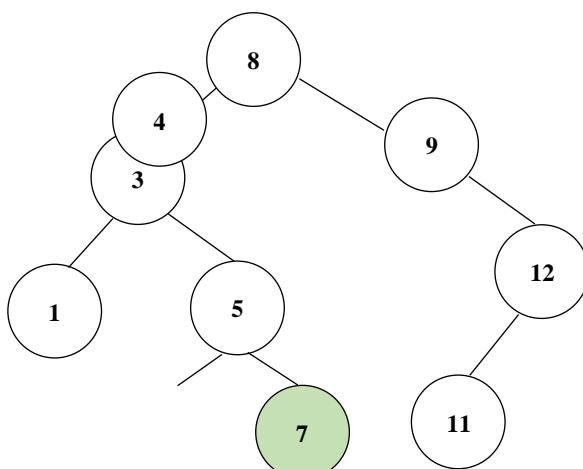
Βάζουμε στη θέση του 100 τη μικρότερη τιμή του δεξιού του υποδέντρου δηλ. το 120

7.12.5 Θέμα 4 Ιούνιος 2019

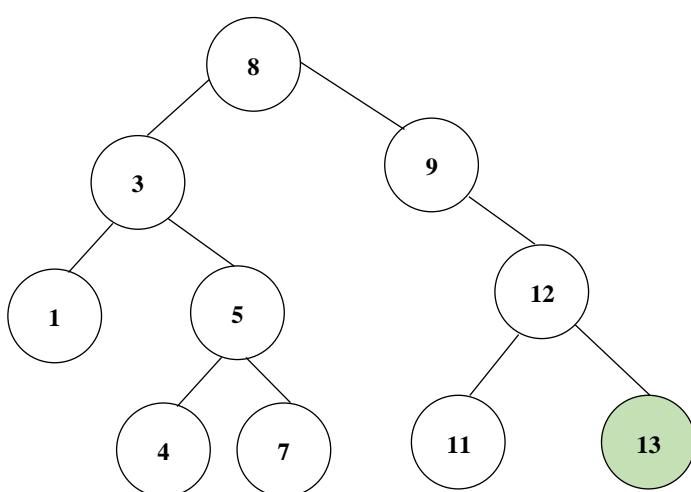
Στο παρακάτω δυαδικό δέντρο **να εισάγετε τα στοιχεία 7, 13, 15 και στη συνέχεια να διαγράψετε τα στοιχεία 3, 15 και 8.** Να δείξετε σε κάθε στάδιο την κατάσταση του δέντρου χρησιμοποιώντας σχήματα:

Απάντηση

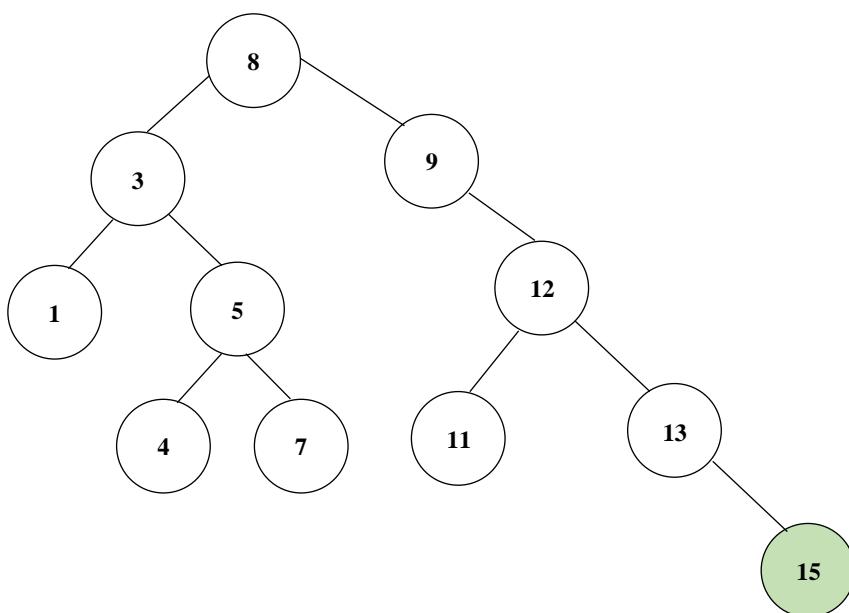
+7



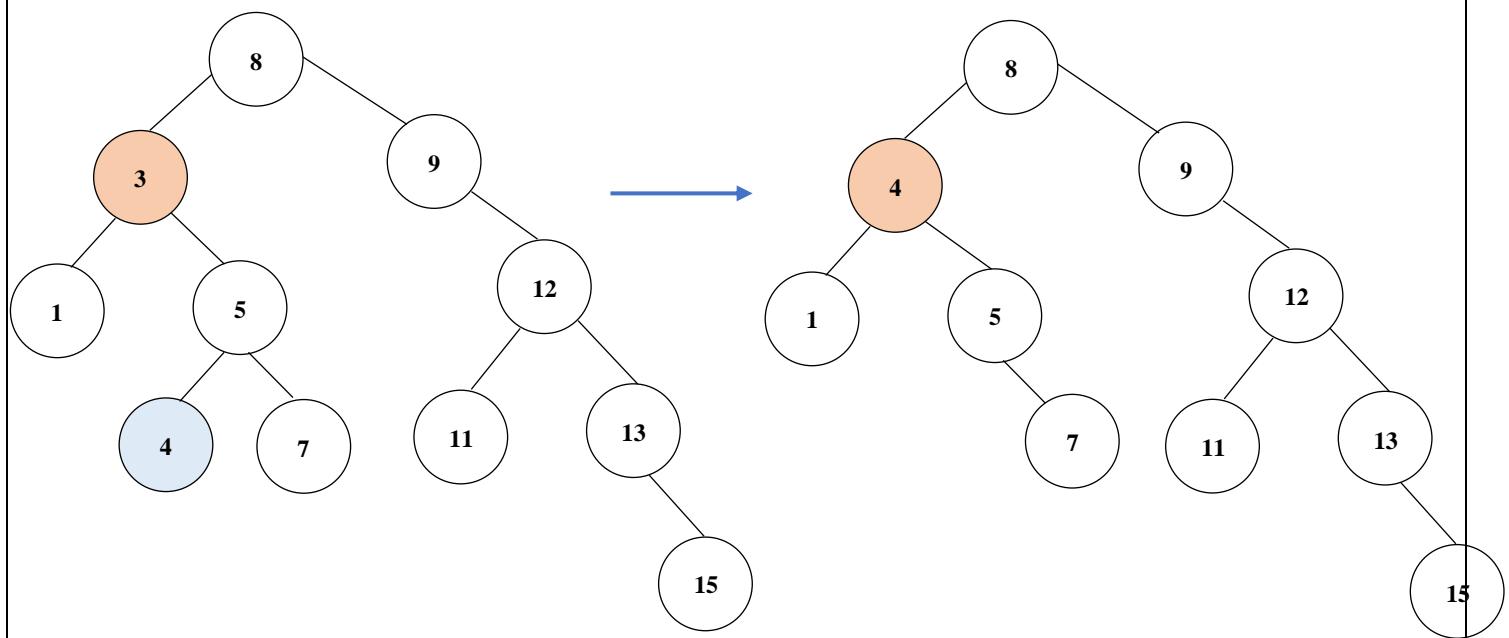
+13



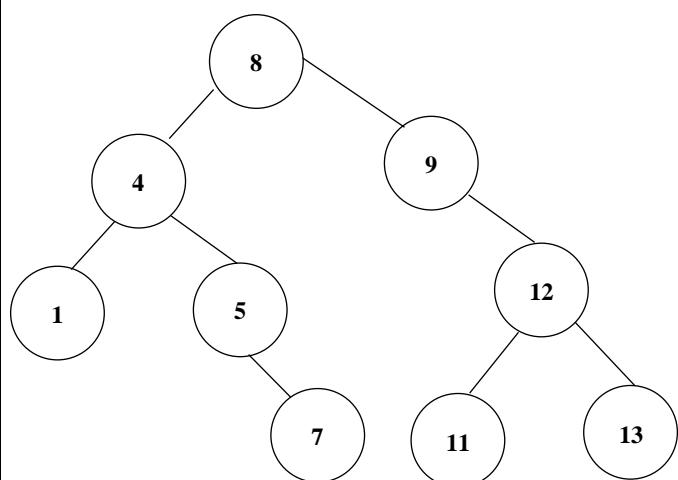
+15



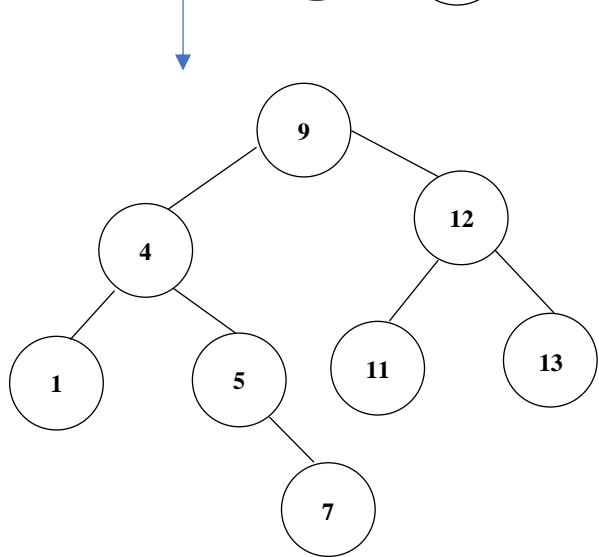
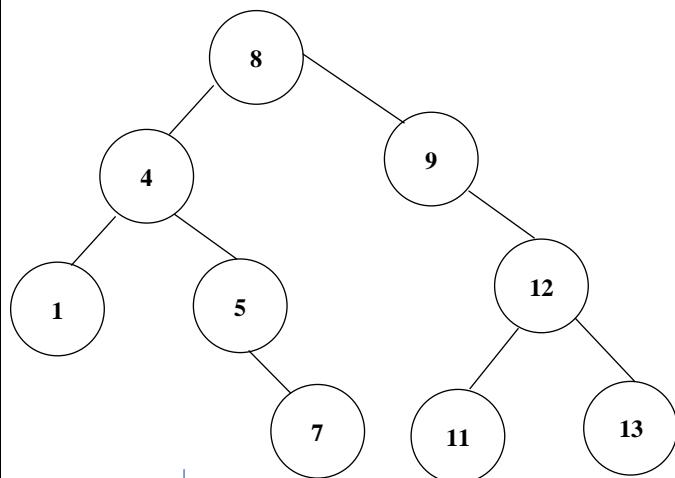
-3



-15



-8



7.12.6 Βασικές πράξεις σε Φυλλοπροσανατολισμένα Δέντρα Αναζήτησης-OXI

Να περιγράψετε τις βασικές πράξεις σε φυλλοπροσανατολισμένα δέντρα εύρεσης.

Απάντηση

Οι πράξεις που υλοποιούνται σε Δυαδικά Δέντρα Αναζήτησης είναι οι ακόλουθες:

α).

search(x) //Αναζήτηση στοιχείου x στο δέντρο. Το στοιχείο x εφόσον υπάρχει θα βρίσκεται σε φύλλο του δέντρου

```
{
    v←ρίζα //Η Αναζήτηση του στοιχείου x ξεκινά από τη ρίζα του δέντρου
```

```

while (v≠ φύλλο)
{
    εντόπισε το υποδέντρο i για το οποίο ισχύει ότι Hi-1(v)≤x<Hi(v)
    v←i-οστό παιδί του v
}
if πληροφορία(v)=x then 'βρέθηκε' else 'δεν βρέθηκε'
}
```

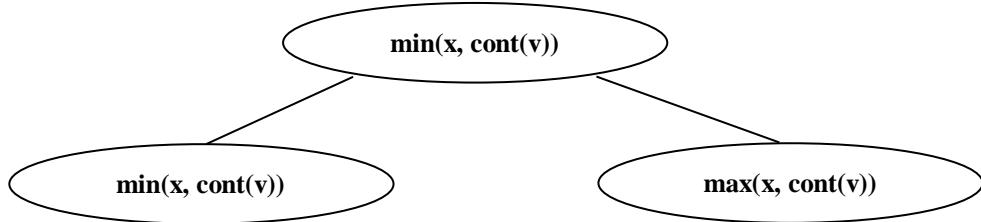
β)

insert(x) →Προσθήκη στοιχείου x στο δέντρο. Το στοιχείο x θα τοποθετηθεί σε φύλλο

```
{
    v←το φύλλο στο οποίο τερμάτισε η search(x)
```

Δομές Δεδομένων –Computer Ανάλυση

if πληροφορία(v)=x **then** 'μην κάνεις τίποτα' γιατί ένα δέντρο δεν έχει διπλότυπες τιμές **else** αντικατέστησε το v με το υποδέντρο:



γ)

```
delete(x) //Διαγραφή στοιχείου χ από το δέντρο
{
    v←το φύλλο στο οποίο τερμάτισε η search(x)
    if πληροφορία(v)≠x then μην κάνεις τίποτα γιατί το στοιχείο δεν υπάρχει στο δέντρο
    else
    {
        u←ο κοντινότερος πρόγονος του v με βαθμό ≥2
        σβήσε την i-οστή πλευρά του u και ένα από τα υποδέντρα H_{i-1}(u), H_i(u)
        if degree(u)=1 then αντικατέστησε το δείκτη από τον πατέρα του u προς το u με το δείκτη στο γιό του u
    }
}
```

Όλες προηγούμενες λειτουργίες υλοποιούνται σε φυλλοπροσανατολισμένο δέντρο εύρεσης και απαιτούν χρόνο **O(log |S|)** όπου |S| το πλήθος των φύλλων του δέντρου.

8 ΤΑΞΙΝΟΜΗΣΗ

8.1 Τι ονομάζεται ταξινόμηση. Σε ποιες κατηγορίες χωρίζονται οι αλγόριθμοι ταξινόμησης; Ποιοι αλγόριθμοι ανήκουν σε κάθε κατηγορία;

Απάντηση

- Ταξινόμηση ονομάζεται η εύρεση μιας αλγορίθμικής διαδικασίας που τοποθετεί τα στοιχεία ενός συνόλου S σε αύξουσα σειρά
- Οι κατηγορίες στις οποίες χωρίζονται οι αλγόριθμοι ταξινόμησης είναι οι ακόλουθες:

Βάσει συγκρίσεων

- Φυσαλίδας (Bubble Sort)
- Σωρού (Heap Sort)
- Με συμβολή (Merge Sort)
- Εισαγωγής (Insertion Sort)
- Γρήγορη Ταξινόμηση (Quick Sort)

Βάσει πληροφορίας εισόδου

- Ταξινόμηση με μέτρηση (Counting Sort)
- Radix Sort
- Bucket Sort

Βάσει χώρου αποθήκευσης

External Sorting

8.2 Ταξινόμηση Φυσαλίδας (Bubble Sort)

Ο αλγόριθμος φυσαλίδας συγκρίνει κάθε στοιχείο ενός πίνακα με το επόμενο του και το αντιμεταθέτει. Σε κάθε βήμα επανάληψης ταξινομείται ένα στοιχείο του πίνακα και απαιτούνται το πολύ N-1 βήματα για την ταξινόμηση όλου του πίνακα όπου N το μέγεθος του. Αν σε ένα βήμα δεν γίνει καμία εναλλαγή στοιχείων ο πίνακας είναι ταξινομημένος και ο αλγόριθμος μπορεί να τερματίσει. Στα επόμενα βήματα, τα στοιχεία που έχουν ταξινομηθεί δεν ξαναελέγχονται.

8.2.1.1 Κόδικας Bubble Sort

for (b=1; b<N; b++) //το for αυτό εκτελεί τα βήματα (τις επαναλήψεις). Σε κάθε βήμα ταξινομείται ένα στοιχείο του πίνακα
for (i=0; i<N-b; i++) //το for αυτό κάνει τις συγκρίσεις των στοιχείων στο κάθε βήμα

```
if (a[i]>a[i+1])
{
    temp=a[i];
    a[i] = a[i+1];
    a[i+1]=temp;
}
```

8.2.1.2 Βελτιωμένος Κόδικας Bubble Sort

for (b=1; b<N; i++) //το for αυτό εκτελεί τα βήματα (τις επαναλήψεις). Σε κάθε βήμα ταξινομείται ένα στοιχείο του πίνακα

{ swap=0; //υποθέτουμε ότι δεν θα χρειαστεί να κάνουμε εναλλαγές στοιχείων στο βήμα αυτό

for (i =0; i<N-b; i++) //το for αυτό κάνει τις συγκρίσεις των στοιχείων στο κάθε βήμα

```
if (a[i]>a[i+1])
{
    temp=a[i];
    a[i] = a[i+1];
    a[i+1]=temp;
    swap=1; //αν γίνει έστω και μια εναλλαγή το swap θα γίνει 1
}
```

```
if (swap==0)
    break;
}
```

8.2.2 Παράδειγμα Εφαρμογής BubbleSort

Έστω $S = \{15, 9, 6, 10\}$. Εφαρμόστε βήμα-βήμα τον αλγόριθμο φυσαλίδας και δείξτε πως ταξινομούνται τα στοιχεία του πίνακα.

Απάντηση

Παρατήρηση: Με την υπογράμμιση αναφερόμαστε στα στοιχεία που κάθε φορά εξετάζουμε και συγκρίνουμε

Βήμα 1

15, 9, 6, 10

9 15 6 10

9 6 15 10

9 6 10 15

Άρα στο Βήμα 1 ταξινομήθηκε το 15

Βήμα 2

9 6 10 15

6 9 10 15

6 9 10 15

Άρα στο Βήμα 2 ταξινομήθηκε το 10

Βήμα 3

6 9 10 15

6 9 10 15

Άρα στο Βήμα 3 ταξινομήθηκε το 9

Παρατηρήσεις

1. Αν ο πίνακας έχει N στοιχεία τότε στη χειρότερη περίπτωση απαιτούνται $N-1$ βήματα για την ταξινόμηση του
2. Αν εκτελεστεί ένα βήμα χωρίς εναλλαγή στοιχείων, ο πίνακας είναι ταξινομημένος και ο αλγόριθμος μπορεί να τερματίσει απευθείας
3. Η πολυπλοκότητα μέσης και χειρότερης περίπτωσης του αλγορίθμου BubbleSort είναι $O(N^2)$ όπου N το μέγεθος του πίνακα

8.3 Ταξινόμηση με Εισαγωγή (Insertion Sort)

Σε κάθε βήμα ταξινομείται ένα στοιχείο του πίνακα και απαιτούνται $N-1$ βήματα για την ταξινόμηση του πίνακα. Η ταξινόμηση με Εισαγωγή συγκρίνει κάθε στοιχείο με όλα τα στοιχεία που βρίσκονται πριν από αυτό και το ταξινομεί σε σχέση με αυτά τοποθετώντας το στη σωστή θέση.

Στη φάση i γίνονται τα ακόλουθα:

1. Υποθέτουμε πως ο πίνακας $A[1.., i-1]$ είναι ταξινομημένος
2. Εισάγουμε το στοιχείο $A[i]$ στην ακολουθία $A[1.., i-1]$ στη σωστή θέση ώστε να διατηρείται η ταξινόμηση της
3. Μετακινούμε όλα τα στοιχεία που είναι μεγαλύτερα του $A[i]$ μια θέση δεξιά

8.3.1.1 Κόδικας Insertion Sort

```
for (i=1; i<N; i++)
{
    pivot= a[i]; //στο pivot καταχωρούμε το στοιχείο που ταξινομείται στο βήμα αυτό
    for (j =i-1; j>=0 && pivot<a[j]; j--)
        a[j+1]= a[j]; //μεταφέρει όλα τα στοιχεία του πίνακα 1 θέση δεξιά
```

```
a[j+1]=pivot; //στη θέση που αδειάζει τοποθετείται το στοιχείο που ταξινομείται που είναι το pivot
}
```

8.3.1.2 Παράδειγμα Εφαρμογής InsertionSort

<i>Αρχική Τιμή</i>	34	8	64	51	32	21
Με i=2	8	34	64	51	32	21
Με i=3	8	34	64	51	32	21
Με i=4	8	34	51	64	32	21
Με i=5	8	32	34	51	64	21
Με i=6	8	21	32	34	51	64

Παρατήρηση

Η πολυπλοκότητα μέσης και χειρότερης περίπτωσης του αλγορίθμου InsertionSort είναι $O(N^2)$ όπου N το μέγεθος του πίνακα

8.4 Ταξινόμηση με Επιλογή (Selection Sort)

Σε κάθε βήμα ταξινομείται ένα στοιχείο του πίνακα και απαιτούνται $N-1$ βήματα για την ταξινόμηση του πίνακα. Η ταξινόμηση με επιλογή (SelectionSort) υλοποιείται σε ένα μονοδιάστατο πίνακα σε 3 κινήσεις:

1. Επιλογή του ελάχιστου στοιχείου
2. Εναλλαγή του ελάχιστου με το στοιχείο του βήματος
3. Επανάληψη των βημάτων 1 και 2 για τα υπόλοιπα στοιχεία του πίνακα

8.4.1.1 Κόδικας Selection Sort

int k, temp, i, j;

```
for (i=1; i<n; i++)
{
    pivot=i; //στο pivot αποθηκεύω τη θέση του στοιχείου που ταξινομείται

    for (j=i+1; j<=n; j++) //ελέγχω τα υπόλοιπα στοιχεία του πίνακα
        if (A[j]<A[pivot]) //βρίσκω τη θέση του μικρότερου στοιχείου από τα υπόλοιπα προ τα δεξιά
            pivot=j; //στη θέση pivot αποθηκεύω τη θέση του μικρότερου στοιχείου
    //εναλλάσσουμε το στοιχείο του βήματος i (δηλ. το στοιχείο που ταξινομείται) με το μικρότερο στοιχείο από όλα τα υπόλοιπα προς τα δεξιά
    temp=A[i];
    A[i]=A[pivot];
    A[pivot]=temp;
}
```

8.4.2 Παράδειγμα Εφαρμογής SelectionSort

Θέση	1	2	3	4	5	6
Αρχική Τιμή	34	8	64	51	32	33
Με i=1	8	34	64	51	32	33
Με i=2	8	32	64	51	34	33
Με i=3	8	32	33	51	34	64
Με i=4	8	32	33	34	51	64
Με i=5	8	32	33	34	51	64

Παρατήρηση

Η πολυπλοκότητα μέσης και χειρότερης περίπτωσης του αλγορίθμου SelectionSort είναι $O(N^2)$ όπου N το μέγεθος του πίνακα.

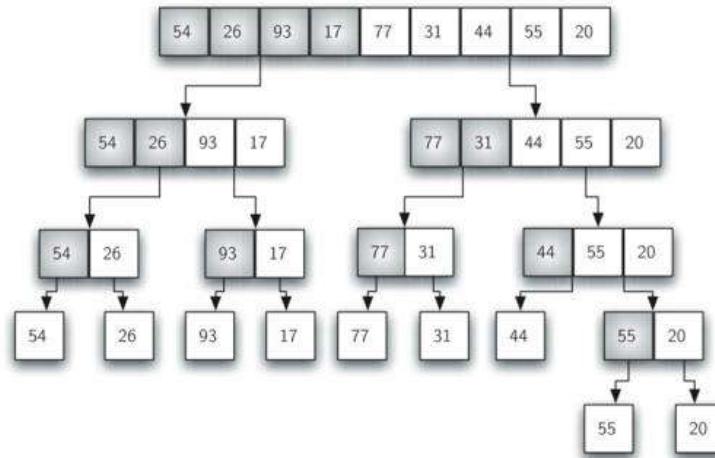
8.5 Ταξινόμηση με Συγχώνευση (MergeSort)

Η ταξινόμηση με συγχώνευση (MergeSort) αποτελεί ένα παράδειγμα εφαρμογής της τεχνικής «διαιρεί και βασίλευε». Η εφαρμογή του αλγορίθμου συνοψίζεται στα επόμενα βήματα:

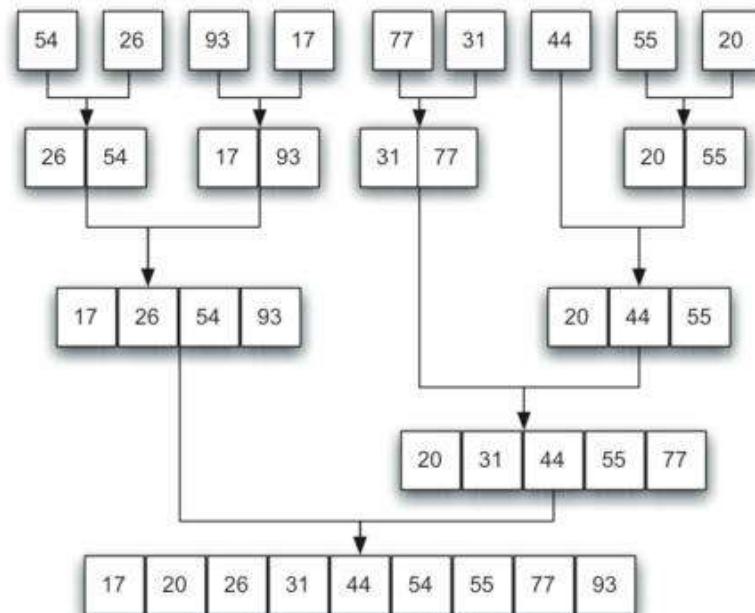
1. **Διαιρεί:** Διαιρεση ακολουθίας εισόδου μήκους n σε δύο υποακολουθίες μήκους $n/2$. Ο διαχωρισμός σταματά όταν φτάσουμε σε υποακολουθίες με μέγεθος 1 στοιχείο. Αν η κάθε υποακολουθία δεν διαιρείται ακριβώς στα δύο **η δεξιά υποακολουθία έχει μέγεθος $\lceil n/2 \rceil$ και η αριστερή υποακολουθία έχει μέγεθος $\lfloor n/2 \rfloor$**
2. **Βασίλευε:** Ταξινόμηση των δύο υποακολουθιών αναδρομικά
3. **Συνένωσε:** Συγχώνευση των δύο ταξινομημένων υποακολουθιών σε μία συνολική ταξινομημένη ακολουθία

8.5.1.1 Παράδειγμα 1 Εφαρμογής MergeSort

Βήμα 1. Διαιρεί



Βήμα 2 και 3: Βασίλευε και Συνένωσε



Παρατήρηση

Η πολυπλοκότητα **μέσης και χειρότερης περίπτωσης** του αλγορίθμου MergeSort **είναι $O(N \cdot \log N)$** όπου N το μέγεθος του πίνακα.

Η συνένωση των στοιχείων του πίνακα γίνεται με την ίδια σειρά που έγινε και η διάσπαση του

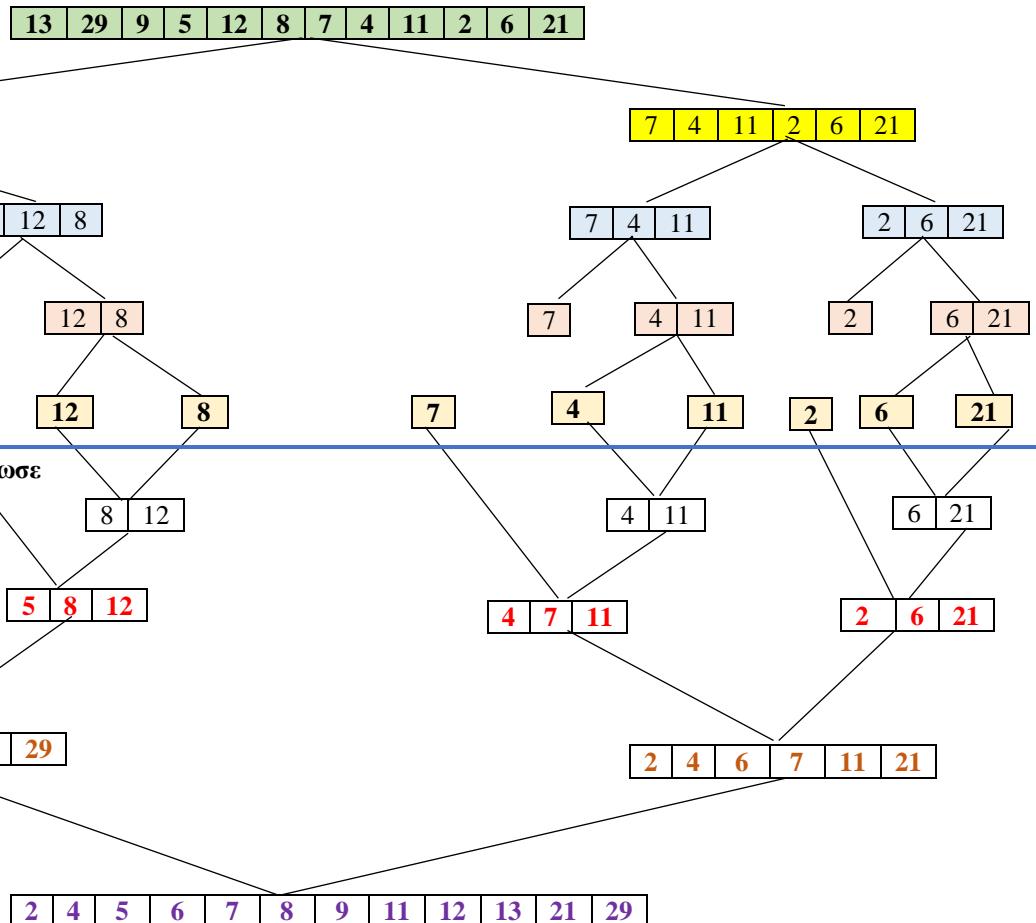
8.5.1.2 Θέμα 1 Ιούνιος 2019

Επιδείξτε την εφαρμογή του αλγορίθμου Merge Sort (**ταξινόμηση με συγχώνευση**) στον πίνακα:

$$A=\{13, 29, 9, 5, 12, 8, 7, 4, 11, 2, 6, 21\}$$

Απάντηση

Διαίρει



8.5.1.3 Κόδικας MergeSort

```
algorithm Merge_Sort(int A[], int left, int right)
begin
    if (left≥right) return;
    mid = (left + right)/2;
    Merge_Sort(A, left, mid); //αναδρομική εκτέλεση του Merge_Sort στον αριστερό υποπίνακα
    Merge_Sort(A, mid+1, right); //αναδρομική εκτέλεση του Merge_Sort στο δεξιό υποπίνακα
    Merge(A, left, mid, right); //συνένωση με ταξινόμηση των 2 πινάκων
end
```

```
void Merge(int arr[], int min, int mid, int max)
{
    int tmp[30], i, j, k, m;
    j=min;
    m=mid+1;

    for (i=min; j≤mid && m≤max ; i++)
    {
        if(arr[j]≤arr[m])
        {
            tmp[i]=arr[j];
```

```
        j++;
    }
    else
    {
        tmp[i]=arr[m];
        m++;
    }
}

if(j>mid)
{
    for(k=m; k<=max; k++)
    {
        tmp[i]=arr[k];
        i++;
    }
}
else
{
    for(k=j; k<=mid; k++)
    {
        tmp[i]=arr[k];
        i++;
    }
}
for(k=min; k<=max; k++)
    arr[k]=tmp[k];
}
```

8.6 Ταξινόμηση Σωρού (HeapSort)

Δυαδικός Σωρός ονομάζεται ένα πλήρες δυαδικό δέντρο με την εξής ιδιότητα: **Κάθε κόμβος έχει τιμή μεγαλύτερη γενικά από τα 2 παιδιά του.** Τα παιδιά του κόμβου στη θέση i , βρίσκονται στις θέσεις $2i$ (αριστερό παιδί) και $2i+1$ (δεξιό παιδί) όσον αφορά την αποθήκευση του σωρού σε πίνακα.

Ο Δυαδικός Σωρός υποστηρίζει τις ακόλουθες λειτουργίες -SOS:

- Εύρεση ελάχιστου ή μέγιστου στοιχείου σε σταθερό χρόνο $O(1)$
- Εισαγωγή στοιχείου στο σωρό σε χρόνο $O(\log N)$ όπου N το πλήθος των κόμβων
- Διαγραφή ελάχιστου ή μέγιστου στοιχείου από το σωρό σε χρόνο $O(\log N)$ όπου N το πλήθος των κόμβων

Η ταξινόμηση σωρού αποτελείται από 2 φάσεις:

Φάση 1- Φάση Δόμησης

Μετατρέπω τον πίνακα $S=\{s_1,s_2,\dots,s_n\}$ σε σωρό αρχίζοντας από τη θέση 1. Το μεγαλύτερο στοιχείο στο τέλος θα βρίσκεται στη ρίζα του σωρού.

Φάση 2- Φάση Διαλογής

Διαλέγουμε και απομακρύνουμε το μεγαλύτερο στοιχείο το οποίο είναι στη ρίζα του σωρού και το τοποθετούμε στην κατάλληλη θέση του πίνακα (αυτό το στοιχείο είναι πλέον ταξινομημένο). Στη θέση του στοιχείου που απομακρύναμε, τοποθετούμε ένα οποιοδήποτε φύλλο και επαναλαμβάνουμε τις δύο φάσεις για τα υπόλοιπα στοιχεία μέχρι να ταξινομηθεί όλος ο πίνακας.

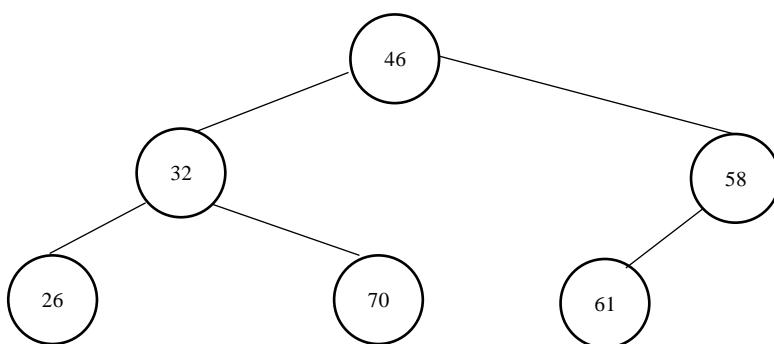
8.6.1 Παράδειγμα Εφαρμογής HeapSort

Δίνεται ο πίνακας $A=[46, 32, 58, 26, 70, 61]$. Να τον ταξινομήσετε με τον αλγόριθμο **Ταξινόμησης Σωρού (HeapSort)** δείχνοντας αναλυτικά τα βήματα:

Απάντηση

Παίρνουμε τις αρχικές τιμές και τις τοποθετούμε σε ένα δυαδικό δέντρο από πάνω προς τα κάτω και από αριστερά προς τα δεξιά.

Προκύπτει το ακόλουθο δέντρο:

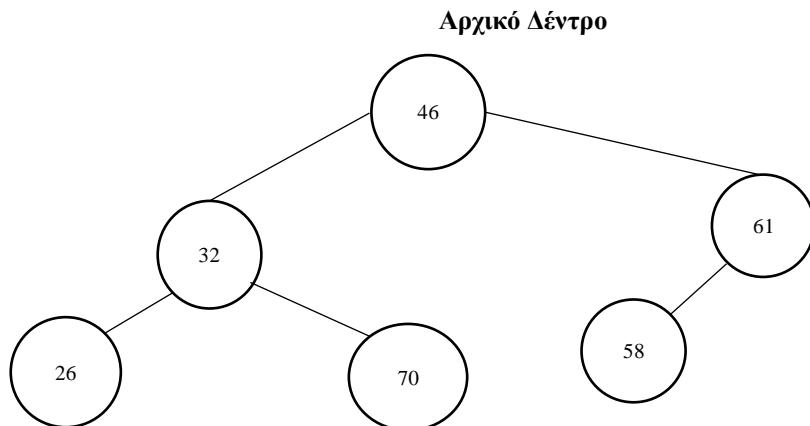


Δομές Δεδομένων –Computer Ανάλυση

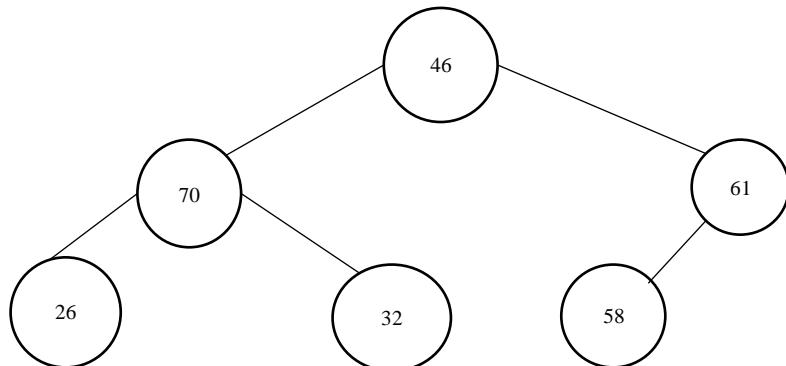
Τοποθετούμε τις τιμές του δέντρου σε ένα πίνακα:

A ₁	A ₂	A ₃	A ₄	A ₅	A ₆
46	32	58	26	70	61

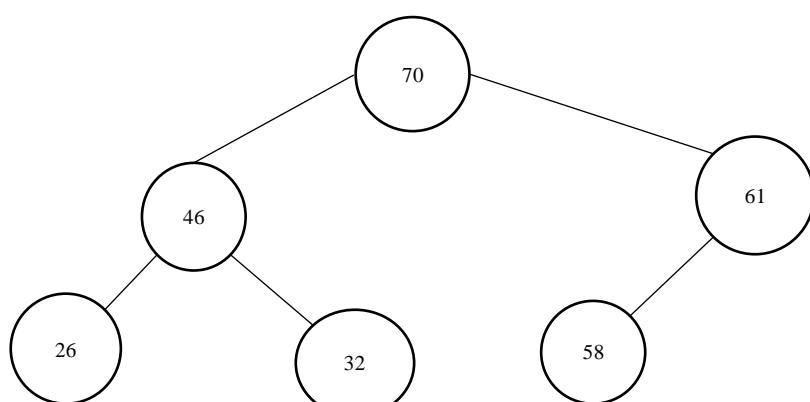
Φάση Δόμησης Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων από κάτω προς τα πάνω**



1^o Βήμα



2^o Βήμα



Οι αντίστοιχες εναλλαγές στοιχείων για την κατασκευή του σωρού φαίνονται στον ακόλουθο πίνακα:

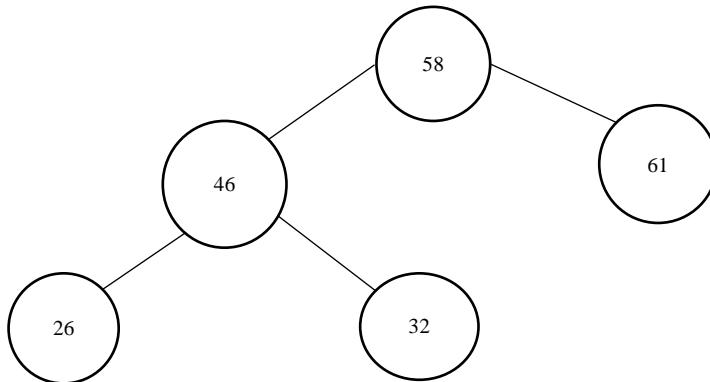
A ₁	A ₂	A ₃	A ₄	A ₅	A ₆	
46	32	58	26	70	61	Αρχικός πίνακας
70	46	61	26	32	58	Σωρός

Δομές Δεδομένων –Computer Ανάλυση

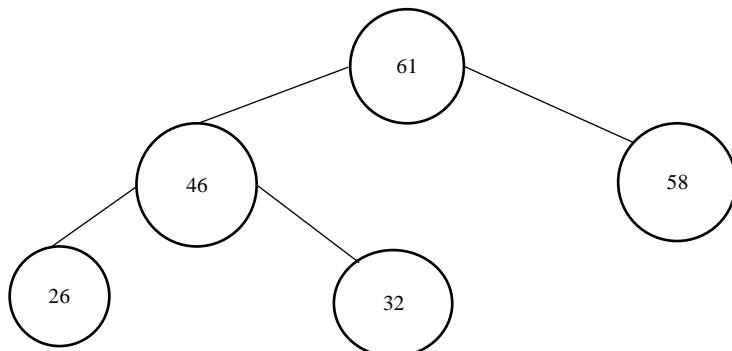
Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (που είναι στη ρίζα του σωρού και είναι το 70) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6
58	46	61	26	32	70

Στη θέση του 70 βάζουμε το 58 και δημιουργείται το ακόλουθο δέντρο:



Φάση Δόμησης Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων.**



Οι αντίστοιχες εναλλαγές στοιχείων για την κατασκευή του σωρού φαίνονται στον ακόλουθο πίνακα:

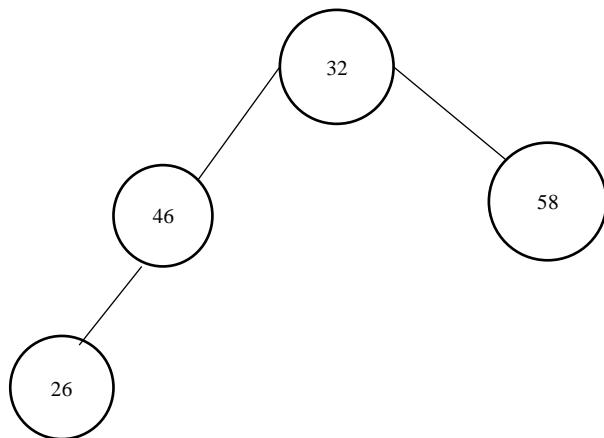
A1	A2	A3	A4	A5	A6	
61	46	58	26	32	70	Σωρός

Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (που είναι στη ρίζα του σωρού και είναι το 61) και το εναλλάσσουμε με το προτελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

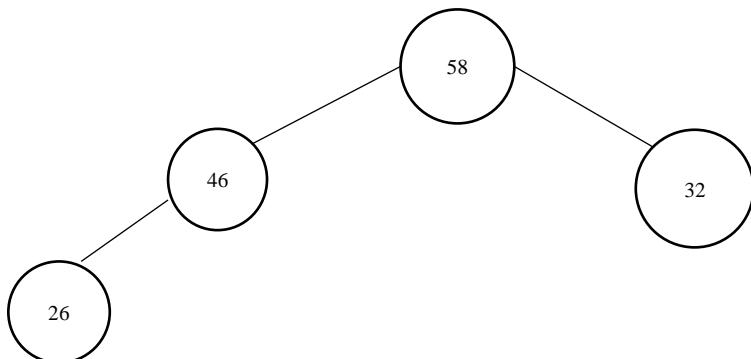
A1	A2	A3	A4	A5	A6
32	46	58	26	61	70

Δομές Δεδομένων –Computer Ανάλυση

Τις τιμές του πίνακα (εκτός των 61 και 70 που ταξινομήθηκαν) τις τοποθετούμε σε ένα νέο δέντρο από πάνω προς τα κάτω και από αριστερά προς τα δεξιά και δημιουργείται το ακόλουθο δέντρο:



Φάση Δόμησης Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων**.



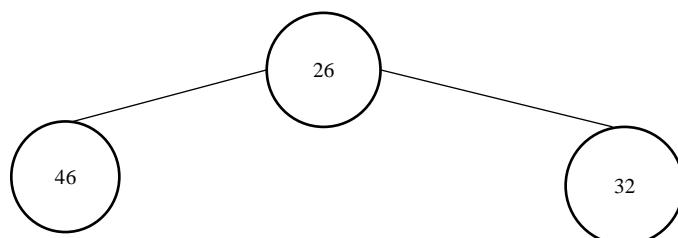
Οι αντίστοιχες εναλλαγές στοιχείων για την κατασκευή του σωρού φαίνονται στον ακόλουθο πίνακα:

A1	A2	A3	A4	A5	A6
58	46	32	26	61	70

Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι στη ρίζα του σωρού και είναι το 58**) και το εναλλάσσουμε με το στοιχείο που βρίσκεται στην 3^η θέση από το τέλος του πίνακα δηλ. με το 26 όπως φαίνεται ακολούθως:

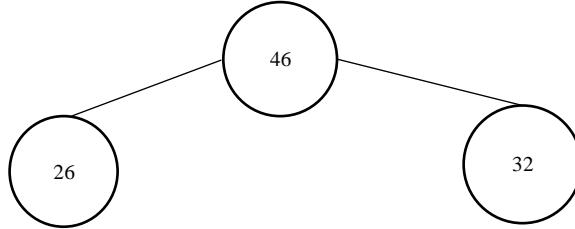
A1	A2	A3	A4	A5	A6
26	46	32	58	61	70

Τις τιμές του πίνακα (εκτός των 58, 61 και 70 που ταξινομήθηκαν) τις τοποθετούμε σε ένα νέο δέντρο από πάνω προς τα κάτω και από αριστερά προς τα δεξιά και δημιουργείται το ακόλουθο δέντρο



Φάση Δόμησης Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων**.

Δομές Δεδομένων –Computer Ανάλυση



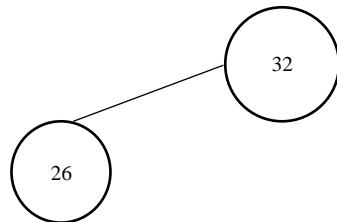
Οι αντίστοιχες εναλλαγές στοιχείων για την κατασκευή του σωρού φαίνονται στον ακόλουθο πίνακα:

A1	A2	A3	A4	A5	A6	
46	26	32	58	61	70	σωρός

Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι στη ρίζα του σωρού και είναι το 46**) και το εναλλάσσουμε με το στοιχείο που βρίσκεται στην 4^η θέση από το τέλος του πίνακα δηλαδή με το 32 όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6
32	26	46	58	61	70

Τις τιμές του πίνακα (εκτός των 46, 58, 61 και 70 που ταξινομήθηκαν) τις τοποθετούμε σε ένα νέο δέντρο από πάνω προς τα κάτω και από αριστερά προς τα δεξιά και δημιουργείται το ακόλουθο δέντρο



Φάση Δόμησης Μετατρέπουμε το νέο δέντρο πάλι σε **σωρό**. Παρατηρούμε όμως ότι το δέντρο είναι ήδη σωρός οπότε δεν χρειάζεται καμία εναλλαγή στοιχείων και ο πίνακας παραμένει αμετάβλητος.

A1	A2	A3	A4	A5	A6	
32	26	46	58	61	70	σωρός

Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι στη ρίζα του σωρού και είναι το 32**) και το εναλλάσσουμε με το στοιχείο που βρίσκεται στην 5^η θέση από το τέλος του πίνακα δηλαδή με το 26 όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6
26	32	46	58	61	70

Το τελευταίο στοιχείο που απέμεινε είναι το 26 και είναι ήδη ταξινομημένο, άρα ο αλγόριθμος HeapSort ολοκληρώθηκε και ο πίνακας ταξινομήθηκε.

Παρατίρηση

Η πολυπλοκότητα **μέσης** και **χειρότερης** περίπτωσης του αλγορίθμου HeapSort είναι **O(N·logN)** όπου N το μέγεθος του πίνακα.

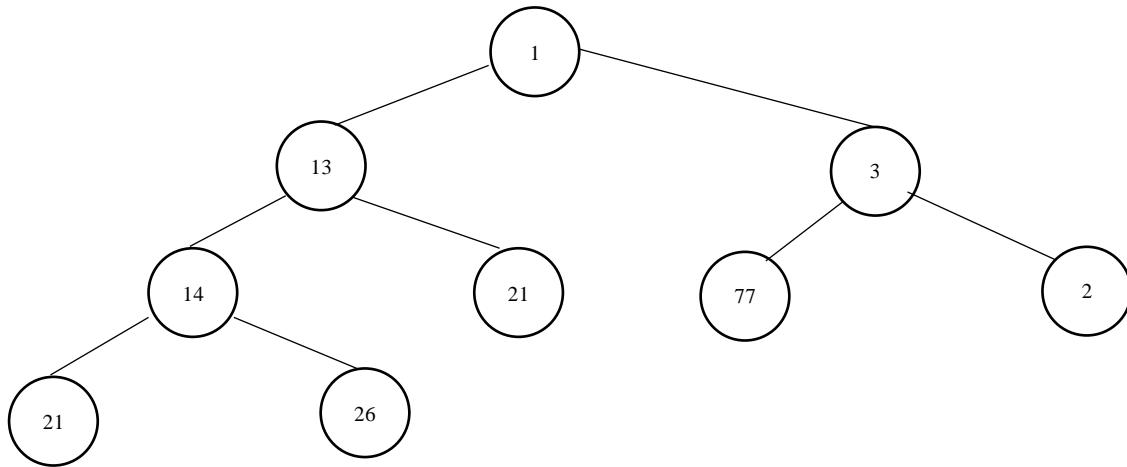
8.6.2 Θέμα 1 Σεπτέμβριος 2020-HW

Ταξινομήστε τον πίνακα ακεραίων: {1, 13, 3, 14, 21, 77, 2, 21, 26} με χρήση του HeapSort. Αναφέρετε την πολυπλοκότητα καλύτερης περίπτωσης.

Απάντηση

Παίρνουμε τις αρχικές τιμές και τις τοποθετούμε σε ένα δυαδικό δέντρο από πάνω προς τα κάτω και από αριστερά προς τα δεξιά.

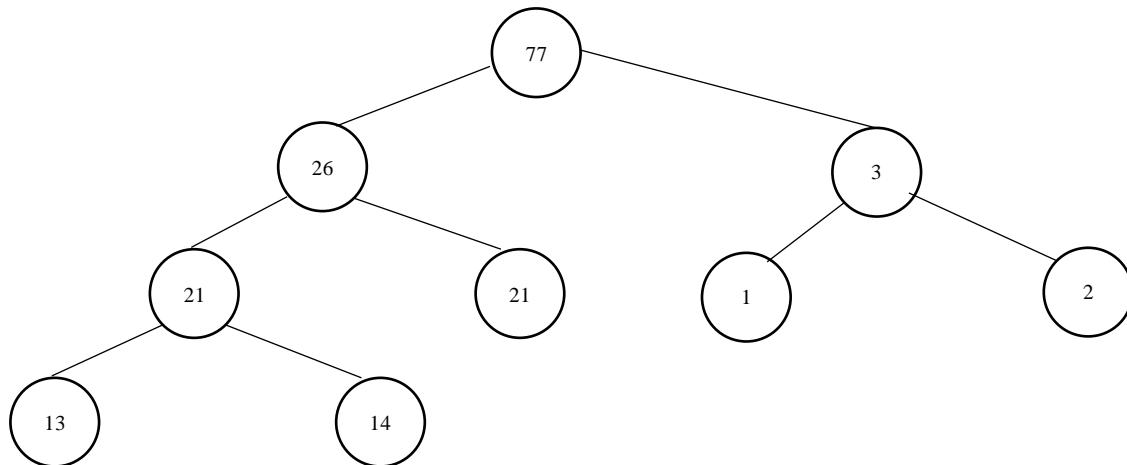
Προκύπτει το ακόλουθο δέντρο



Τοποθετούμε τις τιμές του δέντρου σε ένα πίνακα:

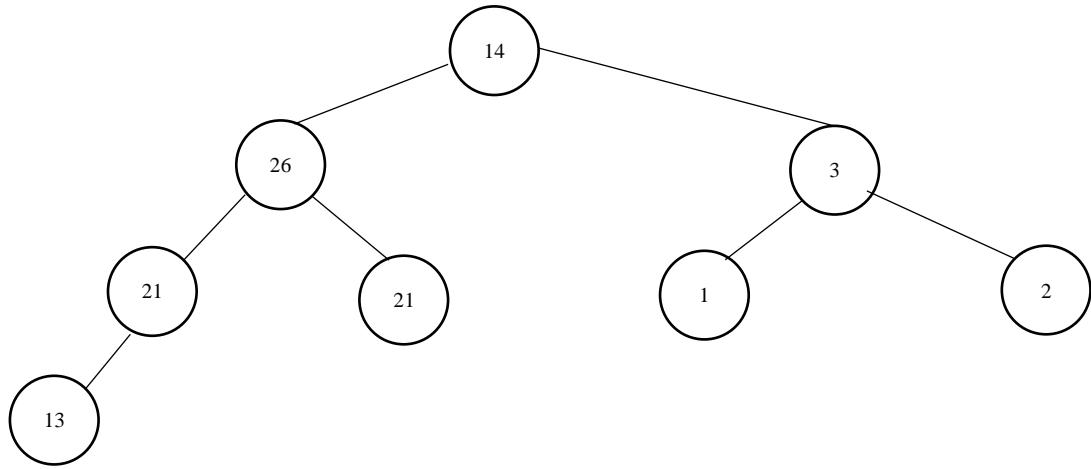
A1	A2	A3	A4	A5	A6	A7	A8	A9
1	13	3	14	21	77	2	21	26

Φάση Δόμησης Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων από κάτω προς τα πάνω**

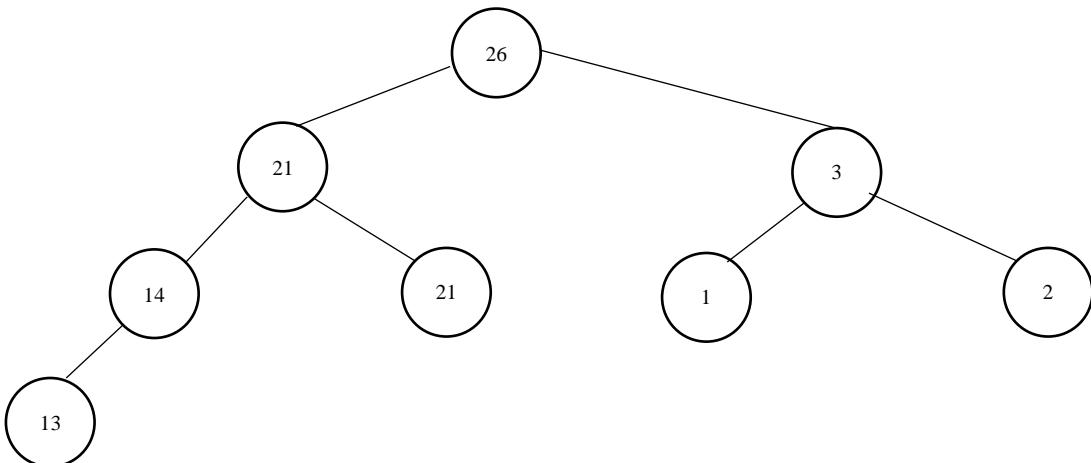


Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (που είναι στη ρίζα του σωρού και είναι το 77) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A7	A8	A9	
77	26	3	21	21	1	2	13	14	Σωρός
14	26	3	21	21	1	2	13	77	Ταξινόμηση 77

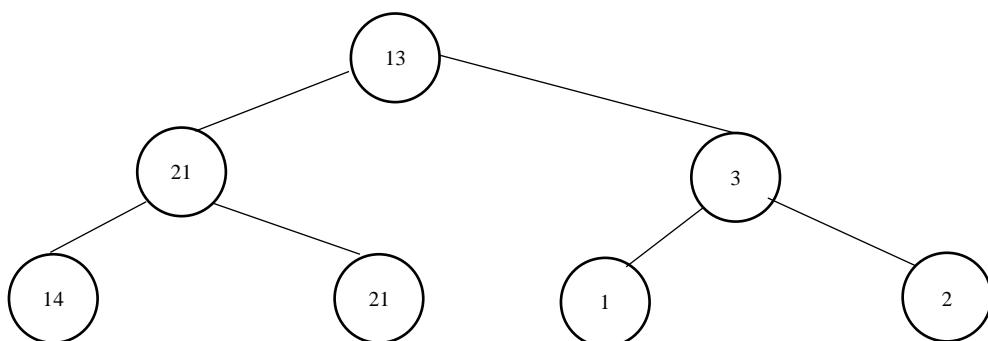


Φάση Δόμησης: Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων**

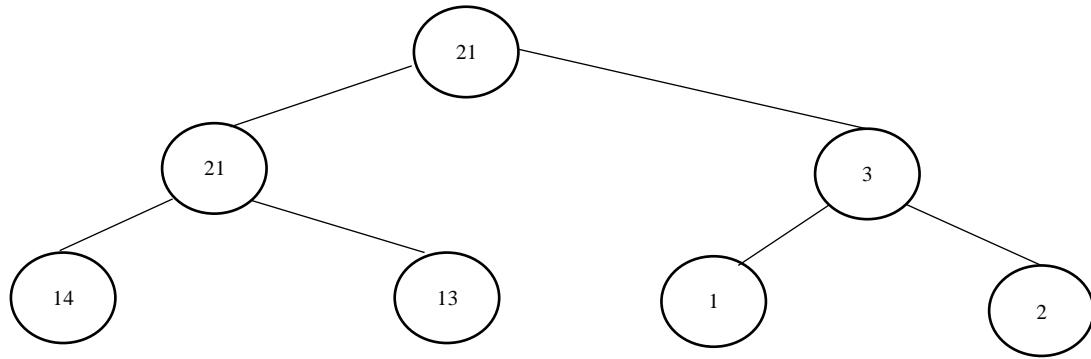


Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι στη ρίζα του σωρού και είναι το 26**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A7	A8	A9	
26	21	3	14	21	1	2	13	77	Σωρός
13	21	3	21	14	1	2	26	77	Ταξινόμηση 26

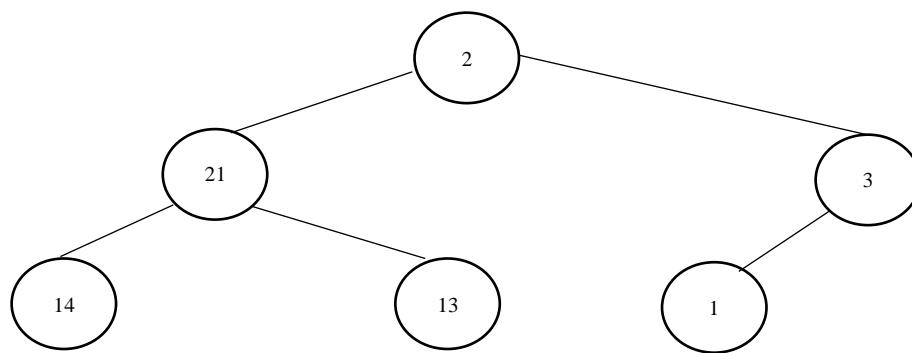


Φάση Δόμησης: Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων.**

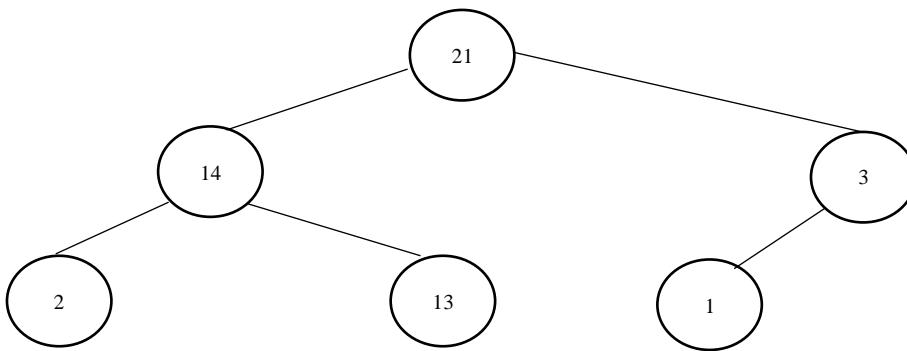


Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι στη ρίζα του σωρού και είναι το 21**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A7	A8	A9	
21	21	3	14	13	1	2	26	77	Σωρός
2	21	3	13	14	1	21	26	77	Ταξινόμηση 21

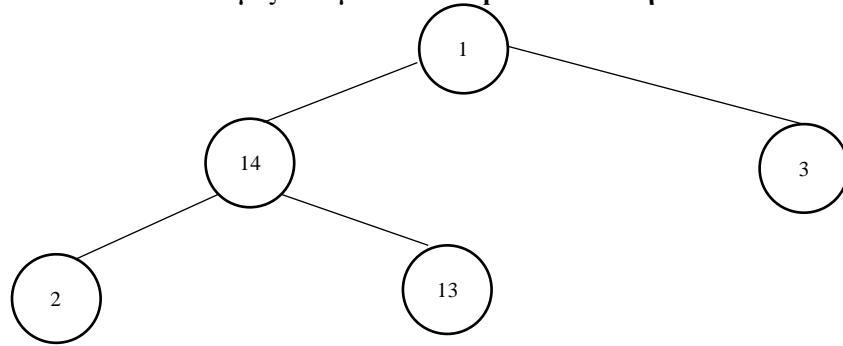


Φάση Δόμησης: Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων**

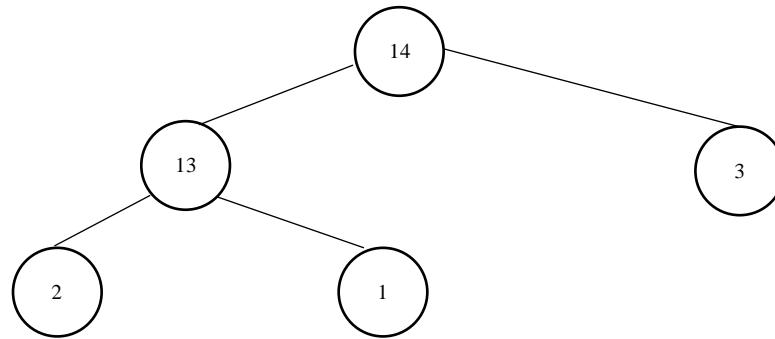


A1	A2	A3	A4	A5	A6	A7	A8	A9	
21	14	3	2	13	1	21	26	77	Σωρός
1	14	3	2	13	21	21	26	44	Ταξινόμηση 21

Δομές Δεδομένων –Computer Ανάλυση

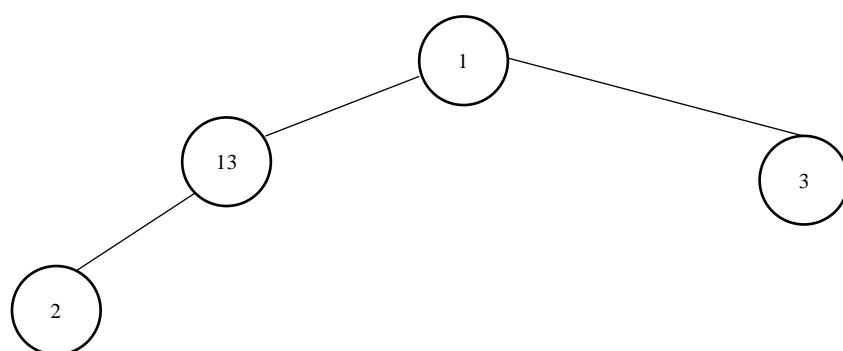


Φάση Δόμησης: Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων**

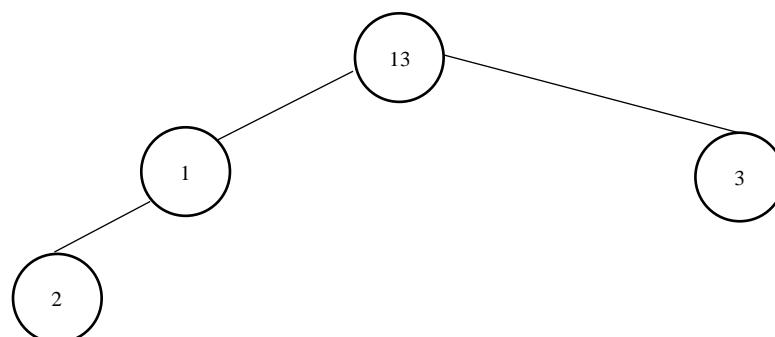


Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι στη ρίζα του σωρού και είναι το 14**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A7	A8	A9	
14	13	3	2	1	21	21	26	44	Σωρός
1	13	3	2	14	21	21	26	44	

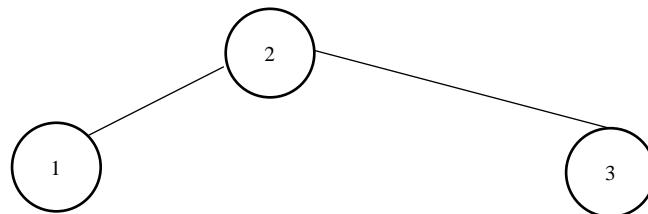


Φάση Δόμησης: Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων.**

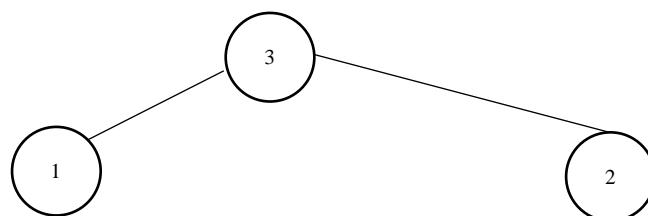


Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι στη ρίζα του σωρού και είναι το 13**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A7	A8	A9	
13	1	3	2	14	21	21	26	44	Σωρός
2	1	3	13	14	21	21	26	44	

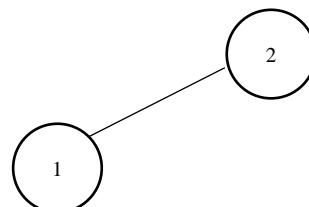


Φάση Δόμησης Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων**.

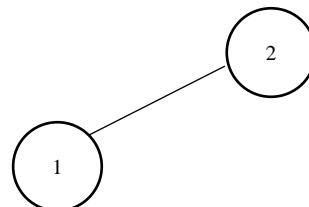


Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι στη ρίζα του σωρού και είναι το 3**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A7	A8	A9	
3	1	2	13	14	21	21	26	77	Σωρός
2	1	3	13	14	21	21	26	77	Ταξινόμηση 3



Φάση Δόμησης Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων**.



Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι στη ρίζα του σωρού και είναι το 2**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

Δομές Δεδομένων –Computer Ανάλυση

A1	A2	A3	A4	A5	A6	A7	A8	A9	
2	1	3	13	14	21	21	26	77	Σωρός
1	2	3	13	14	21	21	26	77	Ταξινόμηση 2

Άρα ο πίνακας ταξινομήθηκε.

Πολυπλοκότητες HeapSort

- **Η πολυπλοκότητα Καλύτερης Περίπτωσης είναι $O(N)$** όπου N το πλήθος τιμών που ταξινομούμε
- **Η πολυπλοκότητα Χειρότερης Περίπτωσης είναι $O(N \cdot log N)$** όπου N το πλήθος τιμών που ταξινομούμε
- **Η πολυπλοκότητα Μέσης Περίπτωσης είναι $O(N \cdot log N)$** όπου N το πλήθος τιμών που ταξινομούμε

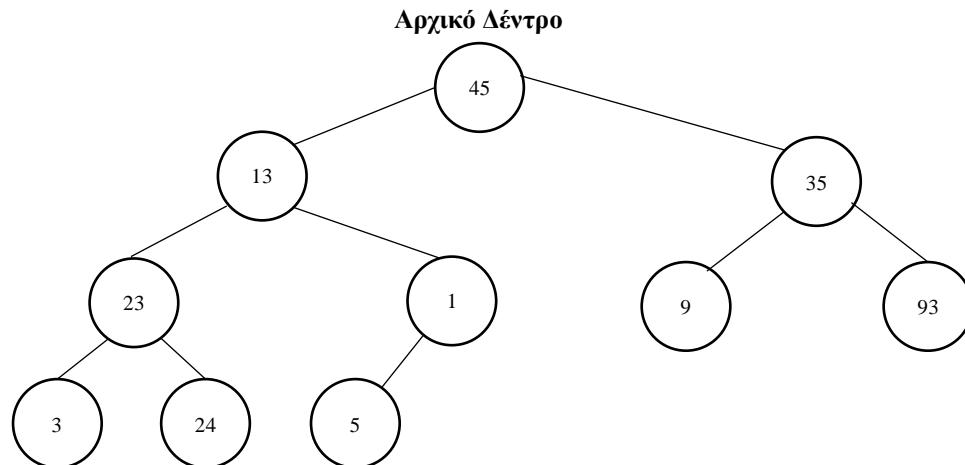
8.6.3 Θέμα 1 Φεβρουάριος 2021-HW

Ταξινομήστε τον πίνακα ακεραίων: {45, 13, 35, 23, 1, 9, 93, 3, 24, 5} με χρήση του Heap Sort. Αναφέρετε την πολυπλοκότητα χειρότερης περίπτωσης.

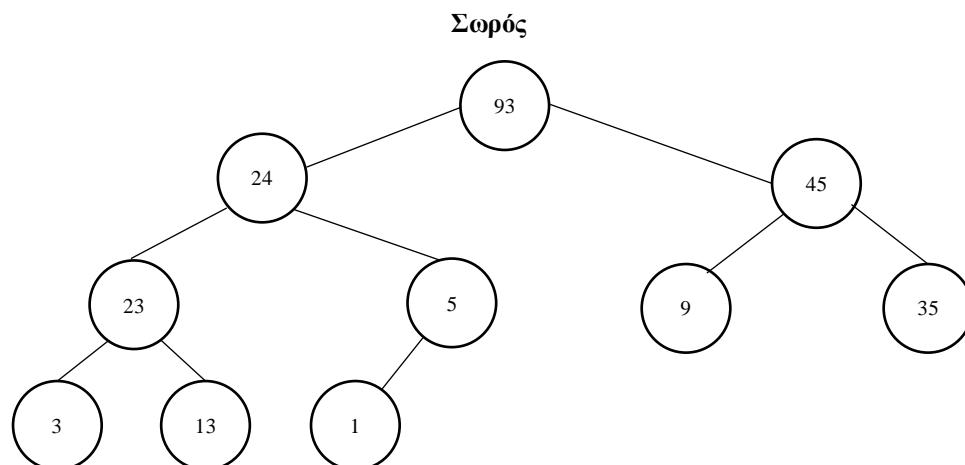
Απάντηση

Παίρνουμε τις αρχικές τιμές και τις τοποθετούμε σε ένα δυαδικό δέντρο από πάνω προς τα κάτω και από αριστερά προς τα δεξιά.

Προκύπτει το ακόλουθο δέντρο:



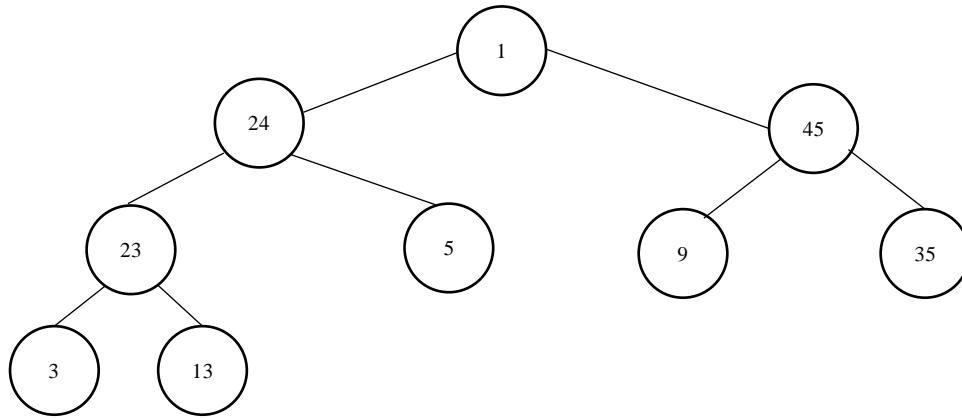
Φάση Δόμησης Μετατρέπουμε το νέο δέντρο σε **σωρό** κάνοντας τις κατάλληλες εναλλαγές στοιχείων από κάτω προς τα πάνω



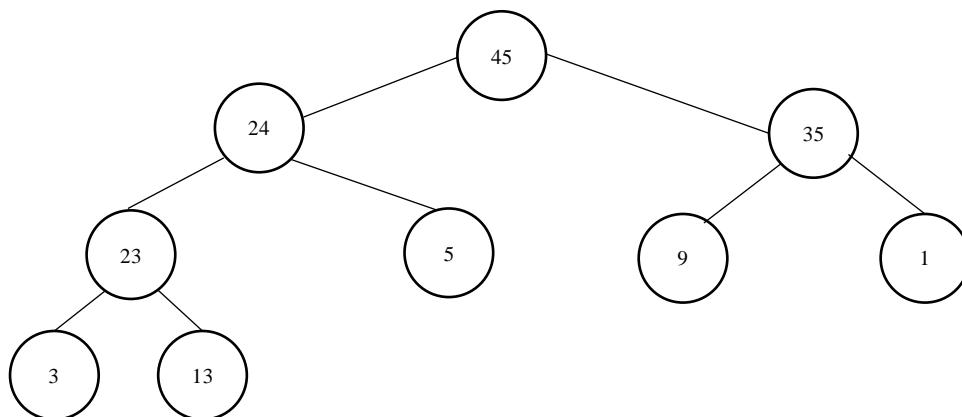
A1	A2	A3	A4	A5	A6	A6	A7	A8	A9	
93	24	45	23	5	9	35	3	13	1	Αρχικός πίνακας

Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι το 93 στη ρίζα του σωρού**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A6	A7	A8	A9
1	24	45	23	5	9	35	3	13	93

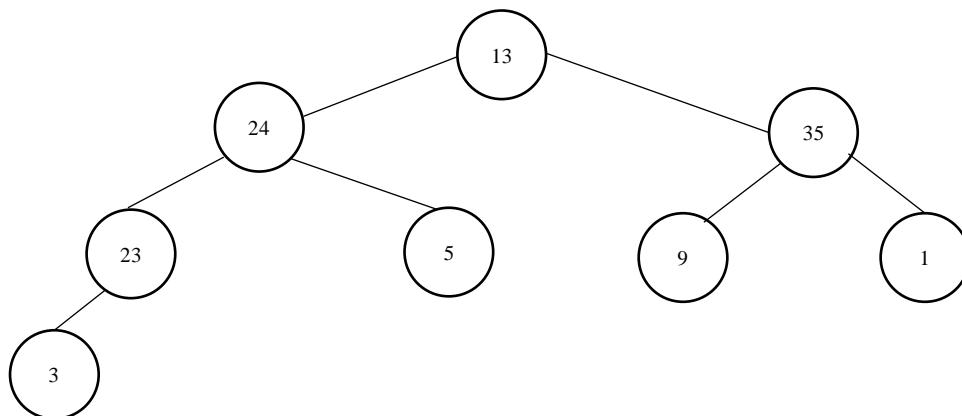


Φάση Δόμησης: Μετατρέπουμε το νέο δέντρο σε **σωρό** κάνοντας τις κατάλληλες εναλλαγές στοιχείων από κάτω προς τα πάνω
Σωρός

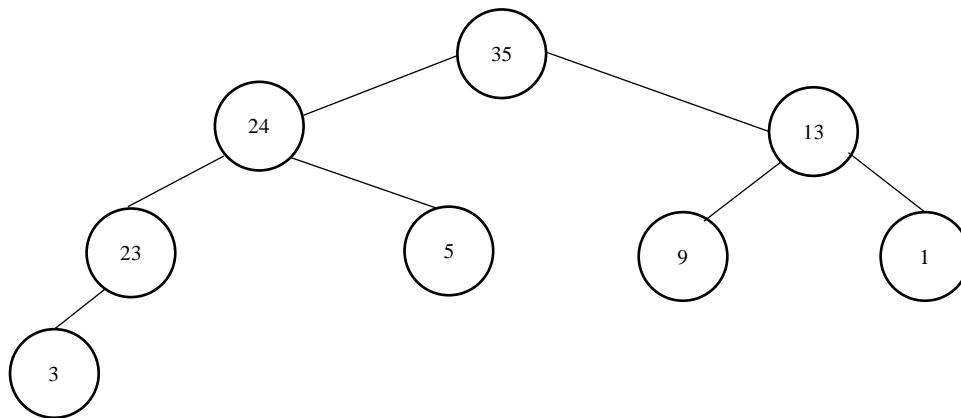


Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι το 45 στη ρίζα του σωρού**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A6	A7	A8	A9
45	24	35	23	5	9	1	3	13	93
13	24	35	23	5	9	1	3	45	93

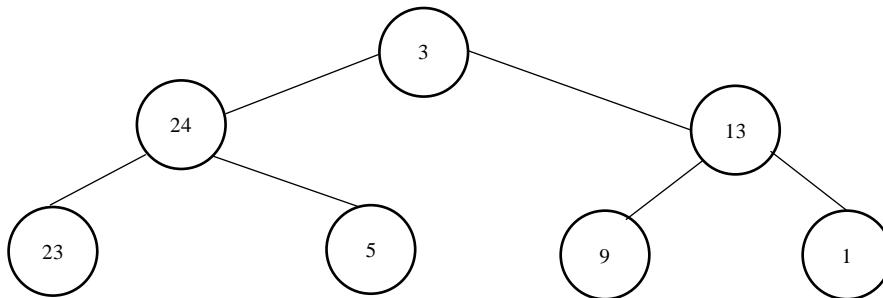


Φάση Δόμησης: Μετατρέπουμε το νέο δέντρο σε **σωρό** κάνοντας τις κατάλληλες εναλλαγές στοιχείων από κάτω προς τα πάνω



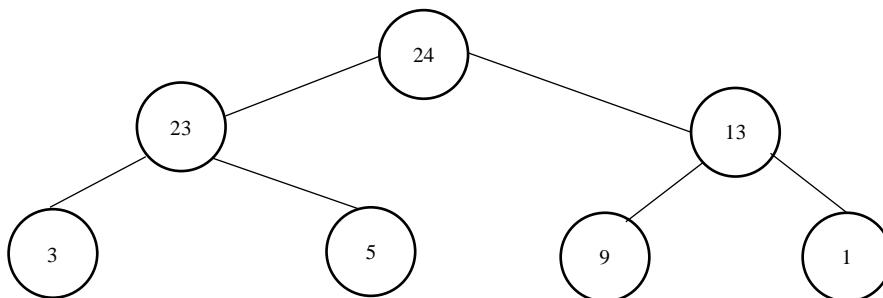
Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι το 35 στη ρίζα του σωρού**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A6	A7	A8	A9
35	24	13	23	5	9	1	3	45	93
3	24	13	23	5	9	1	35	45	93



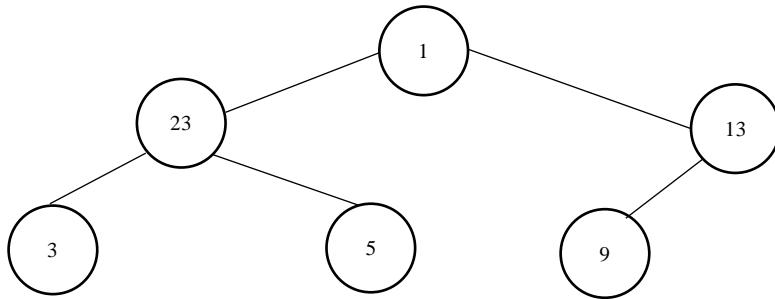
Φάση Δόμησης Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων από κάτω προς τα πάνω**

Σωρός

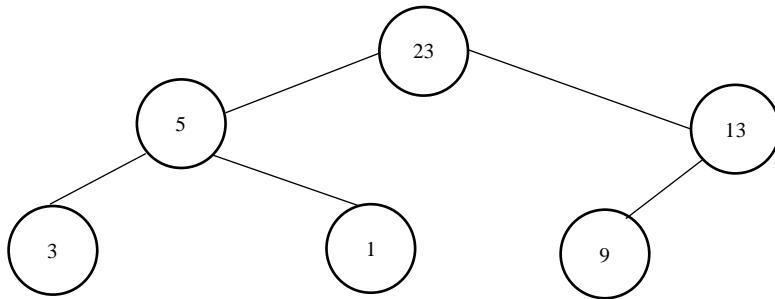


Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι το 24 στη ρίζα του σωρού**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A6	A7	A8	A9
24	23	13	3	5	9	1	35	45	93
1	23	13	3	5	9	24	35	45	93

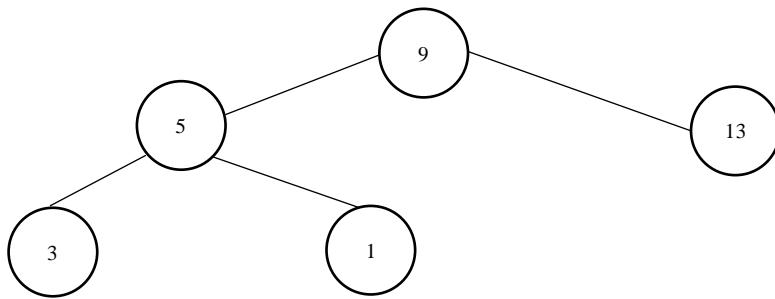


Φάση Δόμησης: Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων από κάτω προς τα πάνω** Σωρός

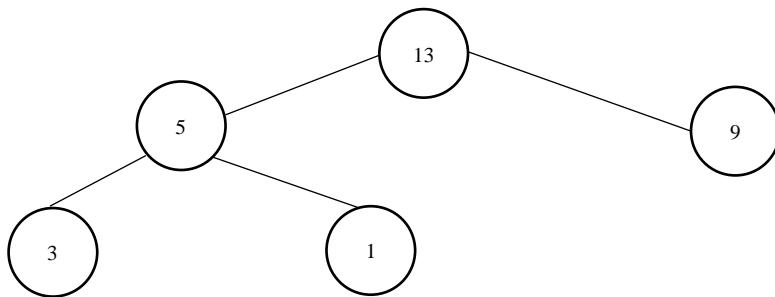


Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι το 23 στη ρίζα του σωρού**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A7	A8	A9
23	5	13	3	1	9	24	35	45
9	5	13	3	1	23	24	35	45



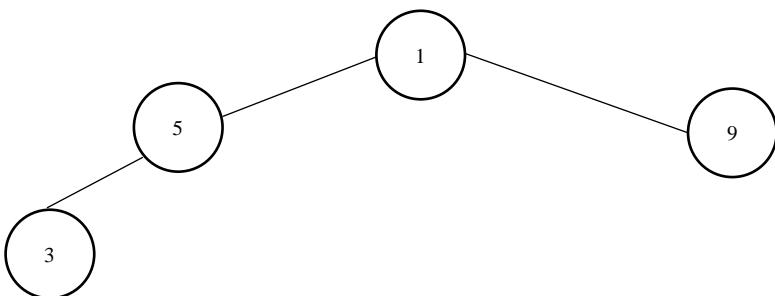
Φάση Δόμησης: Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων από κάτω προς τα πάνω** Σωρός



Δομές Δεδομένων –Computer Ανάλυση

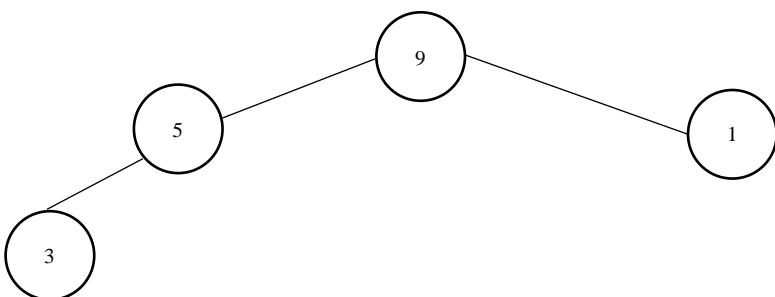
Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι το 13 στη ρίζα του σωρού**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A7	A8	A9
13	5	9	3	1	23	24	35	45
1	5	9	3	13	23	24	35	45



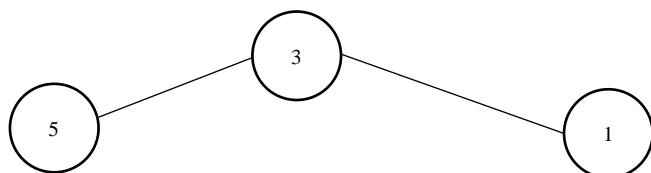
Φάση Δόμησης Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων από κάτω προς τα πάνω**

Σωρός



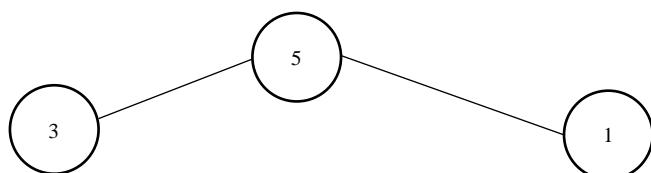
Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι το 9 στη ρίζα του σωρού**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A7	A8	A9
9	5	1	3	13	23	24	35	45
3	5	1	9	13	23	24	35	45



Φάση Δόμησης Μετατρέπουμε το νέο δέντρο σε **σωρό κάνοντας τις κατάλληλες εναλλαγές στοιχείων από κάτω προς τα πάνω**

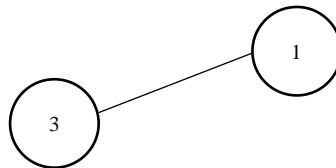
Σωρός



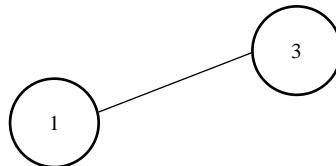
Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι το 5 στη ρίζα του σωρού**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

Δομές Δεδομένων –Computer Ανάλυση

A1	A2	A3	A4	A5	A6	A6	A7	A8	A9
5	3	1	9	13	23	24	35	45	93
1	3	5	19	13	23	24	35	45	93



Φάση Δόμησης: Μετατρέπουμε το νέο δέντρο σε **σωρό** κάνοντας τις κατάλληλες εναλλαγές στοιχείων από κάτω προς τα πάνω **Σωρός**



Φάση Διαλογής: Απομακρύνουμε το μεγαλύτερο στοιχείο (**που είναι το 3 στη ρίζα του σωρού**) και το εναλλάσσουμε με το τελευταίο στοιχείο του πίνακα όπως φαίνεται ακολούθως:

A1	A2	A3	A4	A5	A6	A6	A7	A8	A9
3	1	5	19	13	23	24	35	45	93
1	3	5	19	13	23	24	35	45	93

Όταν απομένει μόνο 1 στοιχείο ο πίνακας έχει ταξινομηθεί

8.6.4 Ψευδοκώδικας για τον αλγόριθμο HeapSort

```
void heapify(int arr[], int n, int i) //Η συνάρτηση heapify μετατρέπει το δέντρο σε σωρό
{
```

```
    int largest = i; // Αρχικοποίηση largest ως ρίζα
```

```
    int l = 2*i; // left = 2*i. Αυτή είναι η θέση του αριστερού παιδιού
```

```
    int r = 2*i+1; // right = 2*i +1. Αυτή είναι η θέση του δεξιού παιδιού
```

```
    if (l < n && arr[l] > arr[largest]) //Αν το αριστερό παιδί που είναι το arr[l] είναι μεγαλύτερο από τη ρίζα που είναι το arr[largest]
```

```
        largest = l; //στη θέση largest βάζουμε το αριστερό παιδί (εναλλάσσουμε τη ρίζα με το αριστερό παιδί)
```

```
    if (r < n && arr[r] > arr[largest]) //Αν το δεξιό παιδί που είναι το arr[r] είναι μεγαλύτερο από τη ρίζα που είναι το arr[largest]
```

```
        largest = r; //στη θέση largest βάζουμε το δεξιό παιδί (εναλλάσσουμε τη ρίζα με το δεξιό παιδί)
```

```
    if (largest != i) //Αν η μεγαλύτερη τιμή δεν είναι η ρίζα
```

```
    {
        swap(arr[i], arr[largest]); //εναλλαγή ρίζας με το μεγαλύτερο παιδί της
```

```
        heapify(arr, n, largest); // εκτέλεσε αναδρομικά τη heapify στο επηρεαζόμενο υπο-δέντρο
```

```
}
```

```
//Κύρια συνάρτηση για την εκτέλεση του heapsort
void heapSort(int arr[], int n)
{
    //Κατασκευή Σωρού (αναδιάταξη του πίνακα) Build heap (rearrange array)
    for (int i = n/2-1; i >= 0; i--)
        heapify(arr, n, i);
    for (int i=n-1; i>=0; i--)
    {
        swap(arr[0], arr[i]); //Μετακίνηση τρέχουνσας ρίζας στο τέλος. Εναλλάσσουμε τη ρίζα το σωρού που είναι το μεγαλύτερο με το
        //στοιχείο του πίνακα στη θέση i. Έτσι μεταφέρουμε το μεγαλύτερο στοιχείο στο τέλος του πίνακα arr και στη θέση το βάζουμε το στοιχείο
        //στη θέση i.
        heapify(arr, i, 0); //Κλήση της συνάρτησης max heapify για να μετατρέψουμε το δέντρο ξανά σε σωρό
    }
}
```

8.7 Γρήγορος Αλγόριθμος (Quicksort)

Αν ο πίνακας περιέχει 0 ή 1 στοιχεία δεν κάνουμε τίποτα, διαφορετικά αναδρομικά:

1. διαλέγουμε ένα στοιχείο p , το οποίο ονομάζουμε στοιχείο οδηγός (pivot) και το αφαιρούμε από τα δεδομένα εισόδου
 2. χωρίζουμε τον πίνακα σε δύο μέρη $S1$ και $S2$ όπου $S1$ περιέχει τα στοιχεία του πίνακα που είναι μικρότερα από το p και $S2$ περιέχει τα στοιχεία που είναι μεγαλύτερα από το p , άρα ταξινομείται το pivot
 3. Αναδρομικά επαναλαμβάνουμε την ίδια διαδικασία στον αριστερό και στο δεξιό υποπίνακα αντίστοιχα
 4. Επιστρέφουμε τον πίνακα $T1$, p , $T2$

8.7.1 Θέμα 3 Ιούνιος 2017 και Σεπτέμβριος 2018

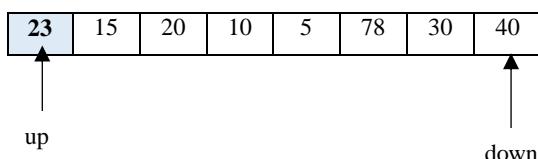
Ταξινομήστε τον πίνακα ακεραίων [23, 15, 20, 10, 5, 78, 30, 40] με χρήση του **Quicksort**. Αναφέρετε **τις πολυπλοκότητες μέσης και χειρότερης περίπτωσης**.

Απάντηση

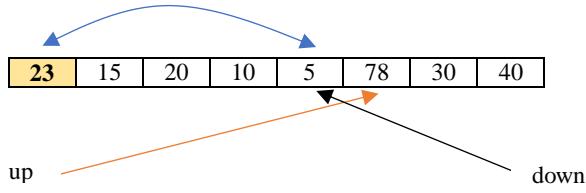
Μεθοδολογία Quicksort

- Σε κάθε βήμα επιλέγουμε το $1^{\text{ο}}$ στοιχείο του εκάστοτε πίνακα ως pivot και το ταξινομούμε
 - Βάζουμε ένα δείκτη up στην αρχή του πίνακα και ένα δείκτη down στο τέλος του πίνακα
 - Για το δείκτη up ψάχνουμε **την $1^{\text{η}}$ θέση προς τα δεξιά που το στοιχείο της είναι $>\text{pivot}$** . Ο δείκτης up μετακινείται μόνο προς τα δεξιά
 - Για το δείκτη down ψάχνουμε **την $1^{\text{η}}$ θέση προς τα αριστερά που το στοιχείο της είναι $\leq\text{pivot}$** . Ο δείκτης down μετακινείται μόνο προς τα αριστερά
 - Αν διασταυρώνονται (τέμνονται) οι δείκτες up και down, εναλλάσσουμε το στοιχείο pivot με το στοιχείο που δείχνει ο δείκτης down και το στοιχείο pivot ταξινομείται δηλ. $A[\text{pivot}] \leftrightarrow A[\text{down}]$ και ταξινομείται το pivot στοιχείο
 - Αν οι δείκτες up και down δεν μπορούν να προχωρήσουν άλλο εναλλάσσουμε τα στοιχεία που δείγνουν

Pivot 23



- Για το δείκτη up ψάχνουμε το 1^o στοιχείο προς τα δεξιά που είναι > pivot δηλ. του 23
 - Για το δείκτη down ψάχνουμε την 1^η θέση προς τα αριστερά που το στοιχείο της είναι ≤ pivot δηλ. του 23
 - Άρα οι θέσεις που σταματάνε οι δείκτες up και down είναι οι εξής:



- Επειδή διασταυρώνονται οι δείκτες up και down, εναλλάσσουμε **το pivot (23)** με το στοιχείο του δείκτη down (**5**) οπότε ο πίνακας παίρνει τη μορφή:

5	15	20	10	23	78	30	40
---	----	----	----	-----------	----	----	----

Άρα ταξινομείται το pivot στοιχείο δηλ. το 23.

Επαναλαμβάνουμε αναδρομικά την ίδια διαδικασία για τον αριστερό και το δεξιό υποπίνακα δηλ:

Αριστερός Υποπίνακας

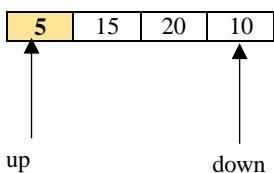
5	15	20	10
---	----	----	----

Δεξιός Υποπίνακας

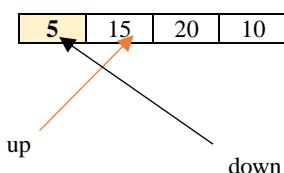
78	30	40
----	----	----

Πρώτα εκτελούμε τον quicksort στον αριστερό υποπίνακα

Pivot 5



- Ο δείκτης up τοποθετείται στο στοιχείο 5 και ψάχνουμε την 1^η θέση προς τα δεξιά που το στοιχείο της είναι > pivot δηλ. του 5
- Ο δείκτης down τοποθετείται στο στοιχείο 10 και ψάχνουμε την 1^η θέση προς τα αριστερά που το στοιχείο της είναι ≤ pivot (5)
- Άρα οι νέες θέσεις των δεικτών up και down είναι οι ακόλουθες:



- Επειδή διασταυρώνονται οι δείκτες up και down, εναλλάσσουμε **το pivot στοιχείο (5)** με το στοιχείο του δείκτη down (**δηλαδή 5**), συνεπώς:

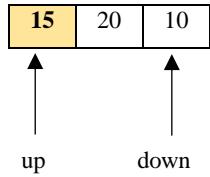
5	15	20	10
----------	----	----	----

Άρα ταξινομείται το pivot στοιχείο που είναι το 5

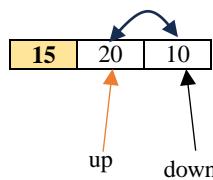
Εκτελούμε τον quicksort στον δεξιό υποπίνακα που είναι ο ακόλουθος:

15	20	10
----	----	----

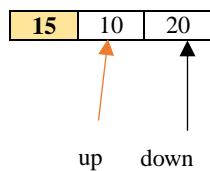
Pivot 15



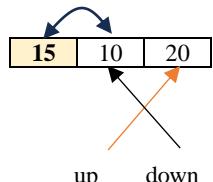
- Ο δείκτης up τοποθετείται στο στοιχείο 15 και ψάχνουμε την 1^η θέση προς τα δεξιά που το στοιχείο της είναι > pivot (15)
- Ο δείκτης down τοποθετείται στο στοιχείο 10 και ψάχνουμε την 1^η θέση προς τα αριστερά που το στοιχείο της είναι ≤ pivot (15)
- Άρα οι θέσεις των δεικτών up και down είναι οι ακόλουθες:



- Όταν οι δείκτες up και down δεν μπορούν να προχωρήσουν περαιτέρω και έχουν διασταυρωθεί, εναλλάσσουμε τα στοιχεία που δείγνουν** και συνεχίζουμε την προηγούμενη διαδικασία. Συνεπώς:



- Για το δείκτη up ψάχνουμε την 1^η θέση προς τα δεξιά που το στοιχείο της είναι > pivot (15), άρα ο δείκτης up πηγαίνει στο 20
- Για το δείκτη down ψάχνουμε την 1^η θέση προς τα αριστερά που το στοιχείο της είναι ≤ pivot (15), άρα ο down πηγαίνει στο 10
- Άρα οι θέσεις των δεικτών των up και down είναι οι ακόλουθες:



Επειδή διασταυρώνονται οι δείκτες up και down, εναλλάσσουμε το pivot (15) με το στοιχείο του δείκτη down, συνεπώς:

10	15	20
----	----	----

Άρα ταξινομείται το pivot στοιχείο που είναι το 15

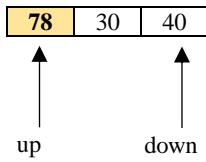
Επειδή ο αριστερός και ο δεξιός υποπίνακας περιλαμβάνουν μόνο 1 στοιχείο ο καθένας, η διαδικασία ολοκληρώνεται.

Άρα ο ταξινομημένος πίνακας μέχρι εδώ είναι:

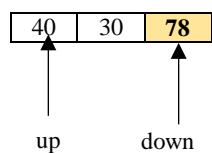
5	10	15	20
---	----	----	----

Μετά εκτελούμε τον quicksort στον αρχικό δεξιό υποπίνακα

78	30	40
----	----	----

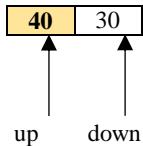
Pivot 78

- Για το δείκτη up ψάχνουμε την 1^η θέση προς τα δεξιά που το στοιχείο της είναι > pivot δηλ. του 78
- Για το δείκτη down ψάχνουμε την 1^η θέση προς τα αριστερά που το στοιχείο της είναι ≤ pivot δηλ. του 78
- Ο δείκτης up παραμένει στη θέση του γιατί δεν υπάρχει στοιχείο προς τα δεξιά > pivot=78
- Ο δείκτης down παραμένει στη θέση του γιατί το στοιχείο 40 που δείχνει είναι μικρότερο του pivot=78
- Άρα εναλλάσσουμε τα στοιχεία που δείχνουν οι δείκτες up και down και συνεχίζεται η ίδια διαδικασία. Άρα:

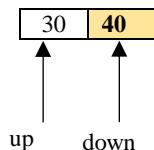


Επειδή οι δείκτες up και down δεν μπορούν να προχωρήσουν άλλο ακόμα και μετά την εναλλαγή το pivot 78 ταξινομήθηκε

Εφαρμόζουμε τον Quicksort στον αριστερό υποπίνακα:

Pivot 40

Ο δείκτης up δεν μπορεί να προχωρήσει άλλο ούτε και ο down γιατί βρίσκεται στο πρώτο στοιχείο που είναι <= του pivot



Επειδή οι δείκτες up και down δεν μπορούν να προχωρήσουν άλλο ούτε μετά την εναλλαγή το pivot 40 ταξινομήθηκε.

Το στοιχείο 30 που είναι αριστερά είναι ταξινομημένο άρα **ταξινομήθηκε και ο δεξιός υποπίνακας του αρχικού pivot=23**

30	40	78
----	----	----

Επειδή αριστερά και δεξιά του pivot υπάρχει μόνο ένα στοιχείο είναι και αυτά ταξινομημένα. Άρα έτσι ταξινομείται όλος ο πίνακας.

Ο τελικός ταξινομημένος πίνακας είναι:

5	10	15	20	23	30	40	78
---	----	----	----	----	----	----	----

Πολυπλοκότητες Quicksort

Η πολυπλοκότητα χειρότερης περίπτωσης για τον Quicksort είναι $O(n^2)$ και μέσης περίπτωσης είναι $O(n \log n)$

8.7.2 Θέμα 1 Ιούνιος 2023

Ο παρακάτω ψευδοκώδικας αντιστοιχεί στον αναδρομικό αλγόριθμο γρήγορης ταξινόμησης (quick sort) ενός πίνακα A μεγέθους n=right-left+1. Αρχικά left=0 και right=n-1. Συμπληρώστε τα κομμάτια κώδικα που λείπουν:

```
void Quicksort(int list[], int left, int right)
{
    int pivot, leftArrow, rightArrow;
    leftArrow=left;
    rightArrow=right;
    pivot=list[(left+right)/2];
    do
    {
        while (list[rightArrow]>pivot)
            .....
        while (.....)
            ++leftArrow;
        if (leftArrow<=rightArrow)
        {
            Swap_Data(list[leftArrow], list[rightArrow])
            .....
            .....
        }
    } while (rightArrow>= leftArrow);
    if (left< rightArrow)
        .....
    if (leftArrow< right)
        .....
}
```

Απάντηση

```
void Quicksort(int list[], int left, int right)
{
    int pivot, leftArrow, rightArrow;
    leftArrow=left; //στη μεταβλητή leftArrow καταχωρείται η 1η θέση του πίνακα. Το leftArrow αντιστοιχεί στο δείκτη up
    rightArrow=right; //στη μεταβλητή leftArrow καταχωρείται η τελευταία θέση του πίνακα. Το rightArrow αντιστοιχεί στο δείκτη down
    pivot=list[(left+right)/2]; // στη μεταβλητή pivot καταχωρείται το μεσαίο στοιχείο του πίνακα ως pivot
    do
    {
        while (list[rightArrow]>pivot) //όσο το στοιχείο list[rightArrow]>pivot εκτελείται η επανάληψη
            rightArrow- -; //μειώνουμε το δείκτη rightArrow κατά 1 δηλ. τον προχωράμε κατά 1 θέση αριστερά ώστε η επανάληψη να
        σταματήσει στο 1o στοιχείο που είναι ≤pivot
        while (list[leftArrow] <=pivot) // όσο το στοιχείο list[leftArrow]≤pivot εκτελείται η επανάληψη
```

Δομές Δεδομένων –Computer Ανάλυση

++leftArrow; // ανξάνουμε το δείκτη leftArrow κατά 1 δηλ. τον προχωράμε κατά 1 θέση δεξιά ώστε η επανάληψη να σταματήσει στο 1^o στοιχείο που είναι >pivot

```
if(leftArrow<=rightArrow) //αν οι δείκτες leftArrow και rightArrow δεν διαστανόνται τότε εναλλάσσουμε τα στοιχεία τους
{
    Swap_Data(list[leftArrow], list[rightArrow]); //αντιμεταθέτουμε τα στοιχεία των δεικτών leftArrow (up) και rightArrow (down)
    leftArrow++; //ανξάνουμε το δείκτη leftArrow (up) για να μετακινηθεί στο επόμενο δεξιά στοιχείο
    rightArrow - -; //μειώνουμε το δείκτη rightArrow (down) για να μετακινηθεί στο προηγούμενο αριστερά στοιχείο
}
} while (rightArrow>=leftArrow); //η επανάληψη εκτελείται όσο βρισκόμαστε εντός ορίων πίνακα

if (left<rightArrow) //αν ισχύει η συνθήκη υπάρχει τμήμα του πίνακα μη ταξινομημένο αριστερά του pivot
    QuickSort(list, leftArrow, pivot-1); //εκτελούμε αναδρομικά τον αλγόριθμο quicksort στον αριστερό υποπίνακα για να ταξινομήσουμε τα στοιχεία αριστερά του pivot
if (leftArrow< right) αν ισχύει η συνθήκη υπάρχει τμήμα του πίνακα μη ταξινομημένο δεξιά του pivot
    QuickSort(list, pivot+1, rightArrow); //εκτελούμε τον αλγόριθμο quicksort αναδρομικά στο δεξιό υποπίνακα για να ταξινομήσουμε τα στοιχεία δεξιά του pivot
}
```

8.7.3 Θέμα 3 Σεπτέμβριος 2016 και Φεβρουάριος 2019

Δύο πολύ δημοφιλείς αλγόριθμοι ταξινόμησης είναι ο Quicksort και ο MergeSort. Στην πράξη ο Quicksort χρησιμοποιείται συχνά για την ταξινόμηση των δεδομένων στην κύρια μνήμη αντί του MergeSort. Δώστε ένα λόγο για τον οποίο ο Quicksort είναι πιθανό να είναι ο καταλληλότερος αλγόριθμος ταξινόμησης για αυτή την εφαρμογή. Σε ποια λογική βασίζονται αυτοί οι δύο αλγόριθμοι; Περιγράψτε αναλυτικά τον MergeSort και δώστε τις πολυπλοκότητες μέσης και χειρότερης περίπτωσης.

Απάντηση

Ο αλγόριθμος QuickSort χρησιμοποιείται συχνά για την ταξινόμηση των δεδομένων στην κύρια μνήμη (δηλ. για ταξινόμηση πινάκων) και είναι καλύτερος του Mergesort διότι ο **Mergesort απαιτεί O(n) επιπλέον μνήμη όπου το n συμβολίζει το μέγεθος του πίνακα και αυτός ο χώρος είναι σημαντικά μεγάλος**. Η δέσμευση και η αποδέσμευση του επιπλέον χώρου που απαιτείται από τον Mergesort αυξάνει το χρόνο εκτέλεσης του Mergesort.

Συγκρίνοντας τη μέση πολυπλοκότητα και των δύο αλγορίθμων διαπιστώνουμε ότι και οι δύο αλγόριθμοι (Quicksort και MergeSort) **έχουν ίδια μέση πολυπλοκότητα O(nlogn)** αλλά διαφέρουν στις σταθερές τους. Συνεπώς στην ταξινόμηση των δεδομένων στην κύρια μνήμη ο Mergesort υστερεί και εξαιτίας του επιπλέον χώρου O(n) που χρειάζεται και εξαιτίας του επιπλέον χρόνου για δέσμευση και αποδέσμευση αυτού του χώρου.

Η λογική που βασίζονται οι αλγόριθμοι Quicksort και ο MergeSort είναι η μέθοδος Διαίρει και Βασίλευε.

Αναλυτική Περιγραφή Mergesort

Αυτός ο αλγόριθμος ταξινόμησης αποτελεί ένα πολύ καλό παράδειγμα εφαρμογής της τεχνικής «διαίρει και βασίλευε». Η εφαρμογή του αλγορίθμου συνοψίζεται ως εξής:

1. **Διαίρει:** Διαχωρισμός των στοιχείων στην μέση ώστε να προκύψουν δύο πίνακες με τα μισά περίπου στοιχεία ο καθένας. Ο Διαχωρισμός αυτός σταματά όταν φτάσουμε σε πίνακες με 1 στοιχείο
2. **Βασίλευε:** Ταξινόμηση κάθε υποπίνακα στοιχείων (καλώντας την ίδια μέθοδο αναδρομικά)
3. **Συνένωσε:** Ένωση των ταξινομημένων υποπινάκων σε ένα νέο ταξινομημένο πίνακα

- **Πολυπλοκότητα μέσης και χειρότερης περίπτωσης για Mergesort είναι O(nlogn)**
- **Πολυπλοκότητα μέσης περίπτωσης για Quicksort είναι O(nlogn) και χειρότερης είναι O(n²)**

algorithm Merge_Sort(A, left, right)

begin

if (left≥right) return;

mid = (left + right)/2;

Merge_Sort (A, left, mid); //αναδρομική εκτέλεση του Merge_Sort στον αριστερό υποπίνακα

Merge_Sort (A, mid+1, right); //αναδρομική εκτέλεση του Merge_Sort στο δεξιό υποπίνακα

Merge(A, left, mid, right); //συνένωση με ταξινόμηση των 2 πινάκων

end

Παρατήρηση

Αν τα δεδομένα είναι πολύ μεγάλου μεγέθους και αποθηκεύονται σε εξωτερικές συσκευές όπως ο σκληρός δίσκος, τότε ο Mergesort είναι καλύτερος από άποψη ταχύτητας. Ελαχιστοποιεί το κόστος ανάγνωσης από την εξωτερική συσκευή και επιτρέπει παράλληλο υπολογισμό. Σε αυτή την περίπτωση μόνο υπερτερεί ο Mergesort έναντι του Quicksort.

8.8 Ταξινόμηση με Μέτρηση (Counting sort)

Ο αλγόριθμος Counting sort είναι ένας απλός αλγόριθμος για ταξινόμηση ακεραίων. Η ιδέα του αλγορίθμου είναι να **μετρά πόσοι αριθμοί είναι μικρότεροι από δοσμένο ακέραιο έστω x, ώστε να προσδιορίσει τη θέση του x στον ταξινομημένο πίνακα.** Π.χ. αν υπάρχουν i ακέραιοι μικρότεροι του x, το x θα τοποθετηθεί στην i-οστή θέση του πίνακα.

Για τη διατύπωση του αλγορίθμου θα υποθέσουμε ότι τα στοιχεία που ταξινομούμε είναι ακέραιοι αριθμοί στο [1, k] και μας δίνεται ένας πίνακας A[n]. Θεωρήσουμε ότι έχουμε στη διάθεση μας 2 βοηθητικούς πίνακες B[1..n] για την παραγωγή της εξόδου και C[1..k] για τη μέτρηση των τιμών. Ο αλγόριθμος διαιρείται σε 3 φάσεις και έχει ως εξής:

CountingSort(A, B, n, k) //Ταξινόμηση του πίνακα A με n στοιχεία στο διάστημα [1..k]

1. **for** i=1 **to** k **do**
 2. C[i]=0; //αρχικοποίηση πίνακα C
 3. **end for**

 4. **for** j=1 **to** n **do**
 5. C[A[j]]=C[A[j]]+1; //To C[i] περιέχει το πλήθος των στοιχείων A[j] όπου A[j]=i
 6. **end for**

 7. **for** i=2 **to** k **do**
 8. C[i]=C[i] + C[i-1]; //To C[i] περιέχει το πλήθος των στοιχείων A[j] όπου A[j]≤i
 9. **end for**

 10. **for** j=n **downto** 1 **do**
 11. B[C[A[j]]]=A[j]
 12. C[A[j]]= C[A[j]] -1
 13. **end for**

 14. **end**
- Η πολυπλοκότητα χειρότερης περίπτωσης είναι O(n+k) όπου n το πλήθος των στοιχείων που ταξινομούνται και k το εύρος των τιμών τους
 - Αν k=O(n) δηλ. αν k≤c·n τότε ο counting sort εκτελείται σε χρόνο O(n) άρα είναι ασυμπτωτικά καλύτερος από τους αλγορίθμους που βασίζονται σε συγκρίσεις
 - Ο αλγόριθμος Counting Sort είναι γρήγορος για μεγάλα εύρη τιμών (k) και σχετικά μικρό πλήθος τιμών (n)

8.8.1 Παράδειγμα Ταξινόμησης με Counting Sort

Δίνεται ο πίνακας **A=[5, 1, 1, 2, 3, 6, 3, 3]**. Να ταξινομηθεί βήμα προς βήμα με τον αλγόριθμο **Counting Sort**

Απάντηση

Ο πίνακας που δίνεται είναι ο ακόλουθος:

A

1	2	3	4	5	6	7	8
5	1	1	2	3	6	3	3

Ο πίνακας A έχει n=8 στοιχεία στο διάστημα [1..6] άρα **n=8 και k=6**

Εφαρμόζουμε τον αλγόριθμο Counting Sort

(γραμμές 1-3)

C

1	2	3	4	5	6
0	0	0	0	0	0

(γραμμές 4-6)

for j=1 to 8 do

j=1, A[1]=5, C[5]=C[5]+1=0+1=1 //μετρήσαμε 1 φορά το στοιχείο 5

j=2, A[2]=1, C[1]=C[1]+1=0+1=1 //μετρήσαμε 1 φορά το στοιχείο 1

j=3 A[3]=1, C[1]=C[1]+1=1+1=2 //μετρήσαμε 2 φορές το στοιχείο 1

j=4, A[4]=2, C[2]=C[2]+1=0+1=1 //μετρήσαμε 1 φορά το στοιχείο 2

j=5, A[5]=3, C[3]=C[3]+1=0+1=1 //μετρήσαμε 1 φορά το στοιχείο 3

j=6, A[6]=6, C[6]=C[6]+1=0+1=1 //μετρήσαμε 1 φορά το στοιχείο 6

j=7, A[7]=3, C[3]=C[3]+1=1+1=2 //μετρήσαμε 2 φορές το στοιχείο 3

j=8 ,A[8]=3, C[3]=C[3]+2=1+1=3 //μετρήσαμε 3 φορές το στοιχείο 3

Ο πίνακας C στο τέλος αυτής της επανάληψης είναι ο ακόλουθος:

C

1	2	3	4	5	6
2	1	3	0	1	1

(γραμμές 7-9)

for i=2 to 6 do

i=2, C[2]=C[2] + C[1]=2+1=3

i=3, C[3]=C[3] + C[2]=3+3=6

i=4, C[4]=C[4] + C[3]=0+6=6

i=5, C[5]=C[5] + C[4]=1+6=7

i=6, C[6]=C[6] + C[5]=1+7=8

Δομές Δεδομένων –Computer Ανάλυση

Ο πίνακας C στο τέλος αυτής της επανάληψης είναι ο ακόλουθος:

C

1	2	3	4	5	6
2	3	6	6	7	8

(γραμμές 10-13)

for j=8 downto 1 do

j=8, A[8]=3, C[3]=6, B[6]=A[8]=3⇒B[6]=3

j=8, A[8]=3, C[3]=C[3]-1=5⇒C[3]=5

j=7, A[7]=3, C[3]=5, B[5]=A[7]=3⇒B[5]=3

j=7, A[7]=3, C[3]=C[3]-1=5-1=4⇒C[3]=4

j=6 A[6]=6, C[6]=8, B[8]=A[6]⇒B[8]=6

j=6, A[6]=6, C[6]=C[6]-1=8-1=7⇒C[6]=7

j=5, A[5]=3, C[3]=4, B[4]=A[5]=3⇒B[4]=3

j=5, A[5]=3, C[3]=C[3]-1=4-1=3⇒C[3]=3

j=4, A[4]=2, C[2]=3, B[3]=A[4]=2⇒B[3]=2

j=4, A[4]=2, C[2]=C[2]-1=3-1=2⇒C[2]=2

j=3, A[3]=1, C[1]=2, B[2]=A[3]=1⇒B[2]=1

j=3, A[3]=1, C[1]=C[1]-1=2-1=1⇒C[1]=1

j=2, A[2]=1, C[1]=1, B[1]=A[2]=1⇒B[1]=1

j=2, A[2]=1, C[1]=C[1]-1=1-1=0⇒C[1]=0

j=1, A[1]=5, C[5]=7, B[7]=A[1]=5⇒B[7]=5

j=1, A[1]=5, C[5]=C[5]-1=7-1=6⇒C[5]=6

B

1	2	3	4	5	6	7	8
1	1	2	3	3	3	5	6

Αυτός είναι ο τελικός ταξινομημένος πίνακας με τον αλγόριθμο Counting Sort

8.8.2 Θέμα 7 με Counting Sort – Ιούνιος 2023

Ταξινομήστε τον πίνακα ακεραίων {7, 5, 2, 2, 4, 3, 3, 6, 4, 5} με χρήση ταξινόμησης **με μέτρηση** (Counting Sort). Αναφέρετε την **πολυπλοκότητα χειρότερης περίπτωσης**

Απάντηση

Ο πίνακας που δίνεται είναι ο ακόλουθος:

A

1	2	3	4	5	6	7	8	9	10
7	5	2	2	4	3	3	6	4	5

Ο πίνακας A έχει 10 στοιχεία στο διάστημα [1..7] άρα **n=10 και k=7**

Εφαρμόζουμε τον αλγόριθμο Counting Sort

(γραμμές 1-3)

C

1	2	3	4	5	6	7
0	0	0	0	0	0	0

(γραμμές 4-6)

for j=1 to 10 do

$$j=1 \ C[7]=C[7]+1=0+1=1$$

$$j=2 \ C[5]=C[5]+1=0+1=1$$

$$j=3 \ C[2]=C[2]+1=0+1=1$$

$$j=4 \ C[2]=C[2]+1=1+1=2$$

$$j=5 \ C[4]=C[4]+1=0+1=1$$

$$j=6 \ C[3]=C[3]+1=0+1=1$$

$$j=7 \ C[3]=C[3]+1=1+1=2$$

$$j=8 \ C[6]=C[6]+1=0+1=1$$

$$j=9 \ C[4]=C[4]+1=1+1=2$$

$$j=10 \ C[5]=C[5]+1=1+1=2$$

Ο πίνακας C στο τέλος αυτής της επανάληψης είναι ο ακόλουθος:

C

1	2	3	4	5	6	7
0	2	2	2	2	1	1

(γραμμές 7-9)

for i=2 to 7 do

i=2, C[2]=C[2]+C[1]=2+0=2

i=3, C[3]=C[3] + C[2]=2+2=4

i=4, C[4]=C[4] + C[3]=2+4=6

i=5, C[5]=C[5] + C[4]=2+6=8

i=6, C[6]=C[6] + C[5]=1+8=9

i=7, C[7]=C[7] + C[6]=1+9=10

Ο πίνακας C στο τέλος αυτής της επανάληψης είναι ο ακόλουθος:

1	2	3	4	5	6	7
0	2	4	6	8	9	10

for j=10 downto 1 do

j=10 A[10]=5, C[5]=8, B[8]=A[10]=5

j=10 A[10]=5, C[5]= C[5] -1=7

j=9 A[9]=4, C[4]=6, B[6]=A[9]=4

j=9 A[9]=4 C[4]= C[4]-1=5

j=8 A[8]=6, C[6]=9, B[9]=A[8]=6

j=8 A[8]=6 C[6]=C[6]-1=8

j=7 A[7]=3, C[3]=4, B[4]=A[7]=3

j=7 A[7]=3, 6 C[3]=C[3]-1=3

j=6, A[6]=3, C[3]=3, B[3]=A[6]=3

j=6, A[6]=3, C[3]=C[3]-1=2

j=5, A[5]=4, C[4]=5, B[5]=A[5]=4

j=5, A[5]=4, C[4]=C[4]-1=4

j=4, A[4]=2, C[2]=2, B[2]=A[4]=2

j=4, A[4]=2, C[2]=C[2]-1=1

j=3, A[3]=2, C[2]=1, B[1]=A[3]=2

j=3, A[3]=2, C[2]=C[2]-1=0

j=2, A[2]=5, C[5]=7, B[7]=A[2]=5

j=2, A[2]=5, C[5]=C[5]-1=6

j=1, A[1]=7, C[7]=10, B[10]=A[1]=7

j=1, A[1]=7, C[7]=C[7]-1=9

1	2	3	4	5	6	7	8	9	10
2	2	3	3	4	4	5	5	6	7

Αυτός είναι ο τελικός ταξινομημένος πίνακας με τον αλγόριθμο Counting Sort

Πολυπλοκότητα Counting Sort

- **Η πολυπλοκότητα χειρότερης περίπτωσης για τον αλγόριθμο Counting Sort είναι $O(n+k)=O(\max(k, n))$ όπου n το πλήθος των στοιχείων που ταξινομούνται και k το εύρος των τιμών τους**
- Αν $k=O(n)$ δηλ. αν $k \leq c \cdot n$ τότε ο counting sort εκτελείται σε χρόνο $O(n)$ άρα είναι ασυμπτωτικά καλύτερος από τους αλγορίθμους που βασίζονται σε συγκρίσεις
- **Ο αλγόριθμος Counting Sort είναι γρήγορος για μεγάλα εύρη τιμών (k) και σχετικά μικρό πλήθος τιμών (n)**

8.9 SOS-Πολυπλοκότητες Αλγορίθμων Ταξινόμησης

Algorithm	Time Complexity	
	Average	Worst
Quicksort	$O(n \log(n))$	$O(n^2)$
Mergesort	$O(n \log(n))$	$O(n \log(n))$
Heapsort	$O(n \log(n))$	$O(n \log(n))$
Bubble Sort	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n^2)$	$O(n^2)$
Selection Sort	$O(n^2)$	$O(n^2)$

9 ANAZHTHSEH

Βασική Ιδέα: Δίνεται ένα στοιχείο αναζήτησης και θέλουμε να εντοπίσουμε τη θέση του σε ένα πίνακα. Αν το στοιχείο υπάρχει στον πίνακα εντοπίζουμε τη θέση του, ενώ αν δεν υπάρχει επιστρέφεται μια τιμή που δεν αποτελεί θέση του πίνακα ως ένδειξη ότι το στοιχείο δεν εντοπίστηκε. Σε κάθε αλγόριθμο αναζήτησης ισχύουν τα εξής:

- Το next σε κάθε αλγόριθμο αναζήτησης δηλώνει την θέση αναζήτησης του στοιχείου στον πίνακα
- Το key είναι το στοιχείο αναζήτησης
- To left ή low είναι το αριστερό άκρο του πίνακα ενώ το right ή high είναι το δεξιό άκρο του πίνακα

Αλγόριθμοι Αναζήτησης

- Η γραμμική (ή σειριακή) αναζήτηση εφαρμόζεται σε ΜΗ ταξινομημένο πίνακα. Αρχικά $next \leftarrow left$ (αριστερό άκρο του πίνακα) και $next \leftarrow next + 1$ (σε κάθε επανάληψη)
- Η δυαδική (ή δυνική) αναζήτηση (binary search) εφαρμόζεται σε ταξινομημένο πίνακα. Το $next \leftarrow \lfloor (left + right) / 2 \rfloor$
- Η αναζήτηση παρεμβολής (interpolation search) εφαρμόζεται σε ταξινομημένο πίνακα. Το next δίνεται από τον τύπο:

$$next \leftarrow \left\lfloor (high - low) \cdot \frac{key - A[low]}{A[high] - A[low]} \right\rfloor + low$$

- Η δυαδική (ή δυνική) αναζήτηση παρεμβολής (binary interpolation search) εφαρμόζεται σε ταξινομημένο πίνακα. Το next δίνεται από τον τύπο:

$$next \leftarrow \left\lfloor (left - right) \cdot \frac{key - table[left]}{table[right] - table[left]} \right\rfloor + 1$$

9.1 Σειριακή (Γραμμική) Αναζήτηση

algorithm linear_search(key) //To key είναι το στοιχείο αναζήτησης και table είναι ο πίνακας στον οποίο γίνεται η αναζήτηση
begin

```

left=1; //αρχική θέση του πίνακα
right=n; //τελευταία θέση του πίνακα
next=left; //θέση στοιχείου αναζήτησης
while (next<=right) do //όσο δεν έχουμε φτάσει στο τέλος του πίνακα
    if (key=table[next]) //αν το στοιχείο αναζήτησης είναι ίσο με το στοιχείο του πίνακα στη θέση next
        return next; //ο αλγόριθμος επιστρέφει τη θέση next, που είναι η θέση που εντοπίστηκε το στοιχείο στον πίνακα
    else
        next=next+1; //θέτουμε στο next την επόμενη θέση του πίνακα και συνεχίζουμε την αναζήτηση
    od //To od δηλώνει το τέλος του do
    return -1 //η τιμή αυτή επιστρέφεται ως ένδειξη ότι το στοιχείο δεν εντοπίστηκε στον πίνακα
end;

```

Η πολυπλοκότητα χειρότερης περίπτωσης και για τη Σειριακή Αναζήτηση είναι η συγκρίσεις άρα O(n).

Η πολυπλοκότητα μέσης περίπτωσης για τη Σειριακή Αναζήτηση είναι n/2 συγκρίσεις άρα O(n).

9.2 Δυαδική Αναζήτηση (Binary Search)

algorithm binary_search(key) //Το key είναι το στοιχείο αναζήτησης και table είναι ο πίνακας στον οποίο γίνεται η αναζήτηση.

Προϋπόθεση για να εφαρμοστεί είναι ο πίνακας να είναι ταξινομημένος

begin

left=1; //αρχική θέση του πίνακα

right=n; //τελευταία θέση του πίνακα

► **while** (left<=right) **do** //όσο βρισκόμαστε εντός των ορίων του πίνακα

 next ← ⌊(left + right) / 2⌋; //το next είναι η θέση αναζήτησης και είναι πάντα η μεσαία θέση του πίνακα

if (key=table[next]) //αν το στοιχείο αναζήτησης είναι ίσο με το στοιχείο του πίνακα στη θέση next

return next; //ο αλγόριθμος επιστρέφει τη θέση next, που είναι η θέση που εντοπίστηκε το στοιχείο στον πίνακα

else

if (key>table[next]) //αν το στοιχείο αναζήτησης είναι μεγαλύτερο από το στοιχείο του πίνακα στη θέση next, η αναζήτηση συνεχίζεται στο δεξιό μισό του πίνακα λόγω της ταξινόμησης

 left=next+1; //πηγαίνουμε στο δεξιό μισό του πίνακα

else //αν το στοιχείο αναζήτησης είναι μικρότερο από το στοιχείο του πίνακα στη θέση next, η αναζήτηση συνεχίζεται στο αριστερό μισό του πίνακα λόγω της ταξινόμησης

 right=next-1; //πηγαίνουμε στο αριστερό μισό του πίνακα

► **od** //Το od δηλώνει το τέλος των do

return -1 //η τιμή αυτή επιστρέφεται ως ένδειξη ότι το στοιχείο δεν εντοπίστηκε στον πίνακα

end;

Η πολυπλοκότητα μέσης και χειρότερης περίπτωσης για τη Δυαδική Αναζήτηση είναι O(logn).

Παράδειγμα 1

Αναζήτηση του στοιχείου 10 με δυαδική αναζήτηση στον επόμενο πίνακα ο οποίος είναι ταξινομημένος

1	2	3	4	5
6	8	10	15	20

left=1; //αρχική θέση του πίνακα

right=5; //τελευταία θέση του πίνακα

while (left<=right=5) **do**

 next ← ⌊(1+5) / 2⌋ = 3

if (key(=10)=table[3](=10)) // το στοιχείο αναζήτησης 10 εντοπίζεται σε ένα βήμα

return next; //επιστρέφεται η θέση 3 που είναι η θέση του στοιχείου στον πίνακα

Παράδειγμα 2

Αναζήτηση του στοιχείου 20 με δυαδική αναζήτηση στον επόμενο πίνακα ο οποίος είναι ταξινομημένος

1	2	3	4	5
6	8	10	15	20

left=1; //αρχική θέση του πίνακα

right=5; //τελευταία θέση του πίνακα

while (left(=1)<=right(=5)) **do**

 next ← ⌊(1+5) / 2⌋ = 3 //θέση του στοιχείου αναζήτησης

if (key(=20)=table[3](=10)) //δεν ισχύει η ισότητα

if (key=20>table[3] (=10)) //ισχύει αυτή η συνθήκη

Δομές Δεδομένων –Computer Ανάλυση

left=next+1=4 //θα ψάξουμε στο δεξιό μισό του πίνακα

while (left(=4)<=right(=5)) **do**

if (key(=20)=table[4](=15)) //δεν ισχύει η ισότητα

if (key(=20)>table[4]=15) //ισχύει αυτή η συνθήκη

left=next+1=5//θα ψάξουμε πάλι στο δεξιό μισό του πίνακα

while (left (=5)<=right (=5)) **do**

if (key(=20)=table[5] (=20)) // ισχύει η ισότητα

return next; //επιστρέφεται η θέση 5 που είναι η θέση του στοιχείου στον πίνακα. Άρα το στοιχείο εντοπίστηκε

9.3 Αναζήτηση Παρεμβολής (Interpolation Search)

algorithm interpolation_search(key) //To key είναι το στοιχείο αναζήτησης και A είναι ο πίνακας στον οποίο γίνεται η αναζήτηση.

Προϋπόθεση ο πίνακας να είναι ταξινομημένος

begin

 low=1; //αρχική θέση πίνακα

 high=n; //τελευταία θέση πίνακα

while (key>A[low] **and** key<=A[high] **do** //όσο το στοιχείο αναζήτησης βρίσκεται εντός των ορίων του πίνακα γίνεται επανάληψη

$$\text{next} \leftarrow \left\lfloor \left(\frac{\text{key} - \text{A}[\text{low}]}{\text{A}[\text{high}] - \text{A}[\text{low}]} \right) \cdot (\text{high} - \text{low}) \right\rfloor + \text{low} //\text{to next είναι η επόμενη θέση αναζήτησης του στοιχείου key. To}$$

A[low] είναι το στοιχείο του πίνακα A στη θέση low και A[high] είναι το στοιχείο του πίνακα A στη θέση high

if (key>A[next]) //αν το στοιχείο αναζήτησης είναι μεγαλύτερο από το στοιχείο του πίνακα A στη θέση next, η αναζήτηση συνεχίζεται στην επόμενη θέση του πίνακα προς τα δεξιά λόγω ταξινόμησης δηλ.

 low=next+1;

else

if (key<A[next]) //αν το στοιχείο αναζήτησης είναι μικρότερο από το στοιχείο του πίνακα A στη θέση next, η αναζήτηση συνεχίζεται στην προηγούμενη θέση του πίνακα προς τα αριστερά λόγω της ταξινόμησης δηλ.

 high=next-1;

else

 low=next; //το στοιχείο εντοπίστηκε στη θέση next

od

if (key=A[low]) //αν το στοιχείο αναζήτησης εντοπιστεί στη θέση low του πίνακα A

return low; // επιστρέφεται ως αποτέλεσμα από τον αλγόριθμο η θέση low που το στοιχείο εντοπίστηκε στον πίνακα A

else

return -1 // επιστρέφεται ως αποτέλεσμα από τον αλγόριθμο η τιμή -1 ως ένδειξη ότι το στοιχείο δεν εντοπίστηκε στον πίνακα A

end;

Ο χρόνος χειρότερης περίπτωσης για την αναζήτηση παρεμβολής είναι O(n) ενώ ο χρόνος μέσης περίπτωσης είναι O(loglogn).

9.3.1 Θέμα 1 Ιούνιος 2018 με Αναζήτηση Παρεμβολής -SOS

Σας δίνεται ο ταξινομημένος πίνακας ακεραίων [4, 9, 11, 12, 13, 16, 17, 20, 21, 30, 40, 92]. Εκτελέστε το **Interpolation Search** για τους αριθμούς 13, 92 και 25. Για κάθε αναζήτηση καταγράψτε ποιο είναι το στοιχείο που ελέγχεται σε κάθε βήμα και πόσα βήματα εκτελούνται. Συγκρίνετε τις πολυπλοκότητες μέσης και χειρότερης περίπτωσης του αλγορίθμου αυτού με αυτές του Binary Search.

Απάντηση

Αναζήτηση στοιχείου key=13

Αρχικά

low=1; //αρχική θέση πίνακα

high=12; //τελενταία θέση πίνακα

Επανάληψη 1

►while (key=13>A[1]=4 and key=13<=A[12] =92 do. Η συνθήκη είναι αληθής

$$\text{next} \leftarrow \left\lfloor \left(\frac{\text{key} - A[\text{low}]}{A[\text{high}] - A[\text{low}]} \right) \cdot \frac{A[\text{high}] - A[\text{low}]}{A[\text{high}] - A[\text{low}]} + \text{low} \right\rfloor = \left\lfloor \left(12 - 1 \right) \cdot \frac{13 - 4}{92 - 4} + 1 \right\rfloor = \left\lfloor 11 \cdot \frac{9}{88} + 1 \right\rfloor = \left\lfloor \frac{99}{88} + 1 \right\rfloor = \left\lfloor 1,125 \right\rfloor + 1 = 2$$

Αν key=13>A[2]=9 Η σύγκριση είναι αληθής άρα low=next+1=2+1=3

►od

Επανάληψη 2

while (key=13>A[3]=11 and key=13<=table[12] =92 do. Η συνθήκη είναι αληθής

$$\text{next} \leftarrow \left\lfloor \left(\frac{\text{key} - A[\text{low}]}{A[\text{high}] - A[\text{low}]} \right) \cdot \frac{A[\text{high}] - A[\text{low}]}{A[\text{high}] - A[\text{low}]} + \text{low} \right\rfloor = \left\lfloor \left(12 - 3 \right) \cdot \frac{13 - 11}{92 - 11} + 3 \right\rfloor = \left\lfloor \frac{18}{81} + 3 \right\rfloor = \left\lfloor 0,22 + 3 \right\rfloor + 3 = 0 + 3 = 3$$

Έλεγχος key=13>A[3]=11 → NAI

Η σύγκριση είναι αληθής άρα low=next+1=3+1=4

►od

Επανάληψη 3

while (key=13>A[4]=12 and key=13<=table[12] =92 do. Η συνθήκη είναι αληθής

$$\text{next} \leftarrow \left\lfloor \left(\frac{\text{key} - A[\text{low}]}{A[\text{high}] - A[\text{low}]} \right) \cdot \frac{A[\text{high}] - A[\text{low}]}{A[\text{high}] - A[\text{low}]} + \text{low} \right\rfloor = \left\lfloor \left(12 - 4 \right) \cdot \frac{13 - 12}{92 - 12} + 4 \right\rfloor = \left\lfloor \frac{8}{80} + 4 \right\rfloor = \left\lfloor 0,1 + 4 \right\rfloor + 4 = 0 + 4 = 4$$

Έλεγχος key=13>A[4]=12 → NAI

Η σύγκριση είναι αληθής άρα low=next+1=4+1=5

►od

Επανάληψη 4

while (key=13>A[5]=13 and key=13<=A[12]=92 do. Η συνθήκη είναι ψευδής και το while τερματίζει

if key=13=A[5]=13. Η συνθήκη είναι αληθής, άρα το στοιχείο 13 εντοπίζεται στη θέση 5 μετά από 3 επαναλήψεις και επιστρέφεται από τον αλγόριθμο η τιμή 5 που είναι η θέση του στοιχείου 13 στον πίνακα A

Αναζήτηση στοιχείου 92**Αρχικά**

```
low=1; //αρχική θέση πίνακα
high=12; //τελευταία θέση πίνακα
```

Επανάληψη 1

► **while (key=92>A[1]=4 and key=92<=A[12]=92 do.** Η συνθήκη είναι αληθής.

$$\text{nnext} \leftarrow \left\lfloor \left(\text{high} - \text{low} \right) \cdot \frac{\text{key} - \text{A}[\text{low}]}{\text{A}[\text{high}] - \text{A}[\text{low}]} \right\rfloor + \text{low} = \left\lfloor \left(12 - 1 \right) \cdot \frac{92 - 4}{92 - 4} \right\rfloor + 1 = \lfloor 11 \rfloor + 1 = 12$$

Έλεγχος **key=92>A[12]=92.** Η σύγκριση είναι ψευδής

Έλεγχος **key=92<A[12]=92.** Η σύγκριση είναι ψευδής

Έλεγχος **key=92=table[12]=12.** Η σύγκριση είναι αληθής άρα **low=next=12**

► **od**

Επανάληψη 2

while (key=92>A[12]=92 and key=92<=A[12]=92 do. Η συνθήκη είναι ψευδής και το while τερματίζεται

if key=92=A[12]=92. Η συνθήκη είναι αληθής. Άρα το στοιχείο 92 εντοπίζεται στη θέση 12 του πίνακα A μετά από 1 επανάληψη και επιστρέφεται από τον αλγόριθμο η θέση 12 που είναι η θέση του στοιχείου 92 στον πίνακα A

Αναζήτηση στοιχείου 25**Αρχικά**

```
low=1; //αρχική θέση πίνακα
high=12; //τελευταία θέση πίνακα
```

Επανάληψη 1

while (key=25>A[1]=4 and key=25<=A[12]=92 do. Η συνθήκη είναι αληθής

$$\text{next} \leftarrow \left\lfloor \left(\text{high} - \text{low} \right) \cdot \frac{\text{key} - \text{A}[\text{low}]}{\text{A}[\text{high}] - \text{A}[\text{low}]} \right\rfloor + \text{low} = \left\lfloor \left(12 - 1 \right) \cdot \frac{25 - 4}{92 - 4} \right\rfloor + 1 = \left\lfloor \frac{231}{88} \right\rfloor + 1 = \lfloor 2,625 \rfloor + 1 = 2 + 1 = 3$$

Έλεγχος **key=25>A[3]=11** Η σύγκριση είναι αληθής άρα **low=next+1=4**

od

Επανάληψη 2

$$\text{next} \leftarrow \left\lfloor (\text{high}-\text{low}) \cdot \frac{\text{key}-\text{A}[\text{low}]}{\text{A}[\text{high}]-\text{A}[\text{low}]} \right\rfloor + \text{low} = \left\lfloor (12-4) \cdot \frac{25-12}{92-12} \right\rfloor + 4 = \lfloor 1,3 \rfloor + 4 = 5$$

Έλεγχος $\text{key}=25 > \text{A}[5]=13$ Η σύγκριση είναι αληθής άρα $\text{low}=\text{next}+1=5+1=6$

od

Επανάληψη 3

$$\text{next} \leftarrow \left\lfloor (\text{high}-\text{low}) \cdot \frac{\text{key}-\text{A}[\text{low}]}{\text{A}[\text{high}]-\text{A}[\text{low}]} \right\rfloor + \text{low} = \left\lfloor (12-6) \cdot \frac{25-16}{92-16} \right\rfloor + 6 = \lfloor 0,94 \rfloor + 6 = 0 + 6 = 6$$

Έλεγχος $\text{key}=25 > \text{A}[6]=16$ Η σύγκριση είναι αληθής άρα $\text{low}=\text{next}+1=6+1=7$

od

Επανάληψη 4

$$\text{next} \leftarrow \left\lfloor (\text{high}-\text{low}) \cdot \frac{\text{key}-\text{A}[\text{low}]}{\text{A}[\text{high}]-\text{A}[\text{low}]} \right\rfloor + \text{low} = \left\lfloor (12-7) \cdot \frac{25-17}{92-17} \right\rfloor + 7 = \lfloor 0,533 \rfloor + 7 = 0 + 7 = 7$$

Έλεγχος $\text{key}=25 > \text{A}[7]=17$ Η σύγκριση είναι αληθής άρα $\text{low}=\text{next}+1=7+1=8$

od

Επανάληψη 5

$$\text{next} \leftarrow \left\lfloor (\text{high}-\text{low}) \cdot \frac{\text{key}-\text{A}[\text{low}]}{\text{A}[\text{high}]-\text{A}[\text{low}]} \right\rfloor + \text{low} = \left\lfloor (12-8) \cdot \frac{25-20}{92-20} \right\rfloor + 8 = \lfloor 0,277 \rfloor + 8 = 0 + 8 = 8$$

Έλεγχος $\text{key}=25 > \text{A}[8]=20$ Η σύγκριση είναι αληθής άρα $\text{low}=\text{next}+1=8+1=9$

od

Επανάληψη 6

$$\text{next} \leftarrow \left\lfloor (\text{high}-\text{low}) \cdot \frac{\text{key}-\text{A}[\text{low}]}{\text{A}[\text{high}]-\text{A}[\text{low}]} \right\rfloor + \text{low} = \left\lfloor (12-9) \cdot \frac{25-21}{92-21} \right\rfloor + 9 = \lfloor 0,169 \rfloor + 9 = 0 + 9 = 9$$

Έλεγχος $\text{key}=25 > \text{table}[9]=21$ Η σύγκριση είναι αληθής άρα $\text{low}=\text{next}+1=10$

od

Επανάληψη 7

Όταν ξαναγυρίσει στο while η συνθήκη είναι ψευδής, και η επανάληψη τερματίζεται.

while ($\text{key}=25 > \text{A}[10]=30$ **and** $\text{key}=25 \leq \text{A}[\text{high}]=92$) **do**

if $\text{key}=25=\text{A}[10]=30$ δεν ικανοποιείται άρα ο αλγόριθμος τερματίζει χωρίς να εντοπίσει το στοιχείο μετά από 6 βήματα και επιστρέφεται η τιμή -1 ως ένδειξη ότι το στοιχείο 25 δεν εντοπίστηκε στον πίνακα A

Δομές Δεδομένων –Computer Ανάλυση
Σύγκριση Δυαδικής Αναζήτησης και Αναζήτησης Παρεμβολής ως προς την πολυπλοκότητα

- Ο αλγόριθμος **Binary Search** (δυνική αναζήτηση) έχει χρόνο $O(\log n)$ στη μέση και χειρότερη περίπτωση
- Ο αλγόριθμος **interpolation search** (αναζήτηση παρεμβολής) έχει χρόνο χειρότερης περίπτωσης $O(n)$ και χρόνο μέσης περίπτωσης είναι $O(\log \log n)$

- Στη μέση περίπτωση η αναζήτηση παρεμβολής με χρόνο **$O(\log \log n)$** υπερισχύει του **Binary Search** που θέλει χρόνο **$O(\log n)$** και αυτό είναι λογικό αφού τα προς εξέταση διαστήματα δεν υποδιπλασιάζονται σε κάθε βήμα αλλά μπορούν να γίνουν πολύ μικρότερα
- Στη χειρότερη περίπτωση ο αλγόριθμος **Binary Search** (δυνική αναζήτηση) με χρόνο $O(\log n)$ έχει καλύτερη απόδοση από την αναζήτηση παρεμβολής που έχει χρόνο $O(n)$

9.3.2 Θέμα 5 Σεπτέμβριος 2018 με Αναζήτηση Παρεμβολής-SOS-HW

Σας δίνεται ο ταξινομημένος πίνακας ακεραίων [4, 10, 11, 12, 15, 16, 18, 20, 21, 32, 40, 92]. Εκτελέστε το Interpolation Search για τους αριθμούς **13, 92 και 25**. Για κάθε αναζήτηση καταγράψτε ποιο είναι το στοιχείο που ελέγχεται σε κάθε βήμα και πόσα βήματα εκτελούνται. Συγκρίνετε τις πολυπλοκότητες μέσης και χειρότερης περίπτωσης του αλγορίθμου αυτού με αυτές του Binary Search.

Απάντηση**Αναζήτηση στοιχείου 13****Αρχικά**

low=1;

high=12;

Επανάληψη 1

while (key=13>table [1] =4 **and** key=13<=table [12] =92 **do**. Η συνθήκη είναι αληθής

$$\text{next} \leftarrow \left\lfloor (\text{high}-\text{low}) \cdot \frac{\text{key} - A[\text{low}]}{A[\text{high}]-A[\text{low}]} \right\rfloor + \text{low} = \left\lfloor (12-1) \cdot \frac{13-4}{92-4} \right\rfloor + 1 = \left\lfloor \frac{99}{88} = 1,125 \right\rfloor + 1 = 2$$

Έλεγχος key=13>table[2]=9 Η σύγκριση είναι αληθής άρα low=next+1=2+1=3

Επανάληψη 2

while (key=13>table [3] =11 **and** key=13<=table [12] =92 **do**. Η συνθήκη είναι αληθής

$$\text{next} \leftarrow \left\lfloor (\text{high}-\text{low}) \cdot \frac{\text{key} - A[\text{low}]}{A[\text{high}]-A[\text{low}]} \right\rfloor + \text{low} = \left\lfloor (12-3) \cdot \frac{13-11}{92-11} \right\rfloor + 3 = \left\lfloor \frac{18}{81} = 0,22 \right\rfloor + 3 = 3$$

Έλεγχος key=13>table[3]=11. Η σύγκριση είναι αληθής άρα low=next+1=3+1=4

Επανάληψη 3

while (key=13>table [4] =12 **and** key=13<=table [12] =92 **do**. Η συνθήκη είναι αληθής

Θέση αναζήτησης: $\text{next} \leftarrow \left\lfloor (\text{high}-\text{low}) \cdot \frac{\text{key} - A[\text{low}]}{A[\text{high}]-A[\text{low}]} \right\rfloor + \text{low} = \left\lfloor (12-4) \cdot \frac{13-12}{92-12} \right\rfloor + 4 = \left\lfloor \frac{8}{88} = 0,1 \right\rfloor + 4 = 4$

Έλεγχος key=13>table[4]=12. Η σύγκριση είναι αληθής άρα low=next+1=4+1=5

while (key=13>table [5] =13 **and** key=13<=table [12] =92 **do**. Η συνθήκη είναι ψευδής και η επανάληψη τερματίζεται

Έλεγχος key=13=table[5]=13. Η συνθήκη είναι αληθής. Άρα το στοιχείο 13 εντοπίζεται στη θέση 5 **μετά από 3 επαναλήψεις**

Αρχικά

low=1;

high=12;

while (key=92>table [1] =4 **and** key=92<=table [12] =92 **do.** Η συνθήκη είναι αληθής.

Επανάληψη 1

$$next \leftarrow \left\lfloor (high - low) \cdot \frac{key - A[low]}{A[high] - A[low]} \right\rfloor + low = \left\lfloor (12 - 1) \cdot \frac{92 - 4}{92 - 4} \right\rfloor + 1 = \lfloor 11 \rfloor + 1 = 12$$

Έλεγχος key=92>table[12]=92. Η σύγκριση είναι ψευδής

Μετά έλεγχος key=92<table[12]=92. Η σύγκριση είναι ψευδής

Άρα low=next=12

while (key=92>table [12] =92 **and** key=92<=table [12] =92 **do.** Η συνθήκη είναι ψευδής και η επανάληψη τερματίζεται

Έλεγχος key=92=table[12]=92. Η συνθήκη είναι αληθής. Άρα το στοιχείο 92 εντοπίζεται στη θέση 12 **μετά από 1 επανάληψη**

Αναζήτηση στοιχείου 25

Αρχικά

low=1;

high=12;

while (key=25>table [1] =4 **and** key=25<=table [12] =92 **do.** Η συνθήκη είναι αληθής

Επανάληψη 1

$$next \leftarrow \left\lfloor (high - low) \cdot \frac{key - A[low]}{A[high] - A[low]} \right\rfloor + low = \left\lfloor (12 - 1) \cdot \frac{25 - 4}{92 - 4} \right\rfloor + 1 = \lfloor 2,625 \rfloor + 1 = 3$$

Έλεγχος key=25>table[3]=11 Η σύγκριση είναι αληθής άρα low=next+1=4

Επανάληψη 2

$$next \leftarrow \left\lfloor (high - low) \cdot \frac{key - A[low]}{A[high] - A[low]} \right\rfloor + low = \left\lfloor (12 - 4) \cdot \frac{25 - 12}{92 - 12} \right\rfloor + 4 = \lfloor 1,3 \rfloor + 4 = 5$$

Έλεγχος key=25>table[5]=13 Η σύγκριση είναι αληθής άρα low=next+1=6

Επανάληψη 3

$$next \leftarrow \left\lfloor (high - low) \cdot \frac{key - A[low]}{A[high] - A[low]} \right\rfloor + low = \left\lfloor (12 - 6) \cdot \frac{25 - 16}{92 - 16} \right\rfloor + 6 = \lfloor 0,94 \rfloor + 6 = 6$$

Έλεγχος key=25>table[6]=16 Η σύγκριση είναι αληθής άρα low=next+1=7

Επανάληψη 4

$$next \leftarrow \left\lfloor (high - low) \cdot \frac{key - A[low]}{A[high] - A[low]} \right\rfloor + low = \left\lfloor (12 - 7) \cdot \frac{25 - 17}{92 - 17} \right\rfloor + 7 = \lfloor 0,533 \rfloor + 7 = 7$$

Έλεγχος key=25>table[7]=17 Η σύγκριση είναι αληθής άρα low=next+1=8

Επανάληψη 5

$$next \leftarrow \left\lfloor (high - low) \cdot \frac{key - A[low]}{A[high] - A[low]} \right\rfloor + low = \left\lfloor (12 - 8) \cdot \frac{25 - 20}{92 - 20} \right\rfloor + 8 = \lfloor 0,277 \rfloor + 8 = 8$$

Έλεγχος key=25>table[8]=20 Η σύγκριση είναι αληθής άρα low=next+1=9

Επανάληψη 6

$$next \leftarrow \left\lfloor (high - low) \cdot \frac{key - A[low]}{A[high] - A[low]} \right\rfloor + low = \left\lfloor (12 - 9) \cdot \frac{25 - 21}{92 - 21} \right\rfloor + 9 = \lfloor 0,169 \rfloor + 9 = 9$$

Έλεγχος key=25>table[9]=21 Η σύγκριση είναι αληθής άρα low=next+1=10

Όταν ξαναγυρίσει στο while η συνθήκη είναι ψευδής, και η επανάληψη τερματίζεται. Η συνθήκη key=25=A[10]=30 δεν ισχύει άρα ο αλγόριθμος τερματίζει **χωρίς να εντοπίσει το στοιχείο μετά από 6 βήματα**

Σύγκριση Δυαδικής Αναζήτησης με Αναζήτηση Παρεμβολής ως προς την πολυπλοκότητα

- Ο αλγόριθμος Binary Search (δυική αναζήτηση) χρειάζεται χρόνο **O(logn)** στη μέση και χειρότερη περίπτωση
- Ο χρόνος χειρότερης περίπτωσης για την αναζήτηση παρεμβολής (interpolation search) **είναι O(n)** ενώ ο χρόνος μέσης περίπτωσης **είναι O(loglogn)**
- Στη μέση περίπτωση η αναζήτηση παρεμβολής που θέλει χρόνο **O(loglogn)**, υπερισχύει του Binary Search που θέλει χρόνο **O(logn)** και αυτό είναι λογικό αφού τα προς εξέταση διαστήματα δεν υποδιπλασιάζονται σε κάθε βήμα αλλά μπορούν να γίνουν πολύ μικρότερα
- Στη χειρότερη περίπτωση ο αλγόριθμος Binary Search (δυική αναζήτηση) που θέλει χρόνο **O(logn)** έχει καλύτερη απόδοση από την αναζήτηση παρεμβολής που έχει χρόνο **O(n)**

9.4 Δυνική Αναζήτηση Παρεμβολής (Binary Interpolation Search - BIS)-OXI

Access(x)	
1.	left $\leftarrow 1$
2	right $\leftarrow n$
3.	size $\leftarrow right - left + 1$
4	next $\leftarrow \left\lceil size \cdot \frac{x - s[left]}{s[right] - s[left]} \right\rceil + 1$
4a	<i>if</i> $x > s[n]$ <i>return</i> "Αποτυχία"
5	<i>while</i> ($x \neq s[next]$) <i>do</i>
6	i $\leftarrow 0$
7	size $\leftarrow right - left + 1$
8	<i>if</i> (size ≤ 3) 'Απευθείας Αναζήτηση'
9	<i>if</i> ($x \geq s[next]$)
10	<i>while</i> ($x > S[next + i \sqrt{size} - 1]$) <i>do</i>
11	i $\leftarrow i + 1$
12	<i>od</i>
13	right $\leftarrow next + i \sqrt{size}$
14	left $\leftarrow next + (i - 1) \sqrt{size}$
14a	<i>if</i> (right $> n$) <i>then return</i> 'Αποτυχία'
15	<i>fi</i>
16	<i>elseif</i> ($x < s[next]$)
17	<i>while</i> ($x < S[next - i \sqrt{size} + 1]$) <i>do</i>
18	i $\leftarrow i + 1$
19	<i>od</i>
20	right $\leftarrow next - (i - 1) \sqrt{size}$
21	left $\leftarrow next - i \sqrt{size}$
21a	<i>if</i> (left < 1) <i>then return</i> 'Αποτυχία'
22	<i>fi</i>
23	next $\leftarrow left + \left\lceil (right - left + 1) \cdot \frac{x - s[left]}{s[right] - s[left]} \right\rceil - 1$

Δομές Δεδομένων –Computer Ανάλυση

24	<i>od</i> (*while*)
25	<i>if</i> (x=s[next]
26	<i>write</i> 'Επιτυχία: Βρέθηκε στη θέση ',next
27	<i>else</i>
28	<i>write</i> 'Αποτυχία: Δεν Βρέθηκε το στοιχείο'
29	<i>end</i>

9.4.1 Θέμα 3 Ιούνιος 2019 και Ιούνιος 2023 με Δυαδική Αναζήτηση Παρεμβολής και Σωρό-SOS

α. Περιγράψτε το Binary Interpolation Search. Δώστε πολυπλοκότητες μέσης και χειρότερης περίπτωσης. Περιγράψτε αναλυτικά πως θα γίνει ο χειρότερος χρόνος ψαξίματος ίσος με $O(\log n)$.

β. Σχεδιάστε το δέντρο σωρό στο οποίο η μεταδιατεταγμένη διαπέραση να επισκέπτεται τους κόμβους με τη σειρά: 8, 10, 6, 5, 3, 4, 7, 2, 1. Γράψτε τον πίνακα της συνεχόμενης αναπαράστασης που αντιστοιχεί στο συγκεκριμένο δέντρο σωρό και αναφέρετε αν πρόκειται για σωρό μεγίστων ή σωρό ελαχίστων

Απάντηση

α.

Βασικά Βήματα (Περιγραφή) Binary Interpolation Search

Βήμα 1

Αναζητούμε ένα στοιχείο x σε ένα ταξινομημένο πίνακα S μεγέθους size

Βήμα 2

Επιλέγουμε τη θέση του πίνακα στην οποία θα αναζητήσουμε το στοιχείο x από τον τύπο $\text{next} \leftarrow \left\lfloor \text{size} \cdot \frac{x - S[\text{left}]}{S[\text{right}] - S[\text{left}]} \right\rfloor + 1$

Αν το στοιχείο βρεθεί στη θέση next ο αλγόριθμος τερματίζει και επιστρέφει το μήνυμα 'Επιτυχία'

Βήμα 3

Αν το στοιχείο δεν βρεθεί στη θέση next ψάχνουμε γραμμικά στον πίνακα κάνοντας άλματα μεγέθους $\sqrt{\text{size}}$ από τη θέση next είτε προς τα δεξιά αν $x > S[\text{next}]$ είτε προς τα αριστερά αν $x < S[\text{next}]$ για να προσδιορίσουμε το υποδιάστημα που περιέχει το στοιχείο αναζήτησης. Αν το μέγεθος του πίνακα γίνει πολύ μικρό π.χ. $\text{size} \leq 3$ εκτελούμε γραμμική (σειριακή) αναζήτηση στον πίνακα

Βήμα 4

Η διαδικασία αναζήτησης επαναλαμβάνεται είτε μέχρι να εντοπίσουμε τα στοιχείο είτε μέχρι να βρεθούμε εκτός ορίων πίνακα οπότε ο αλγόριθμος επιστρέφει το μήνυμα 'Αποτυχία'

Πολυπλοκότητες Μέσης και Χειρότερης περίπτωσης για τη Δυαδική Αναζήτηση Παρεμβολής (BIS)

- Η **Πολυπλοκότητα Χειρότερης περίπτωσης για το BIS** είναι $O(\sqrt{n})$
- Η **Πολυπλοκότητα Μέσης περίπτωσης για το BIS** είναι $O(\log \log n)$

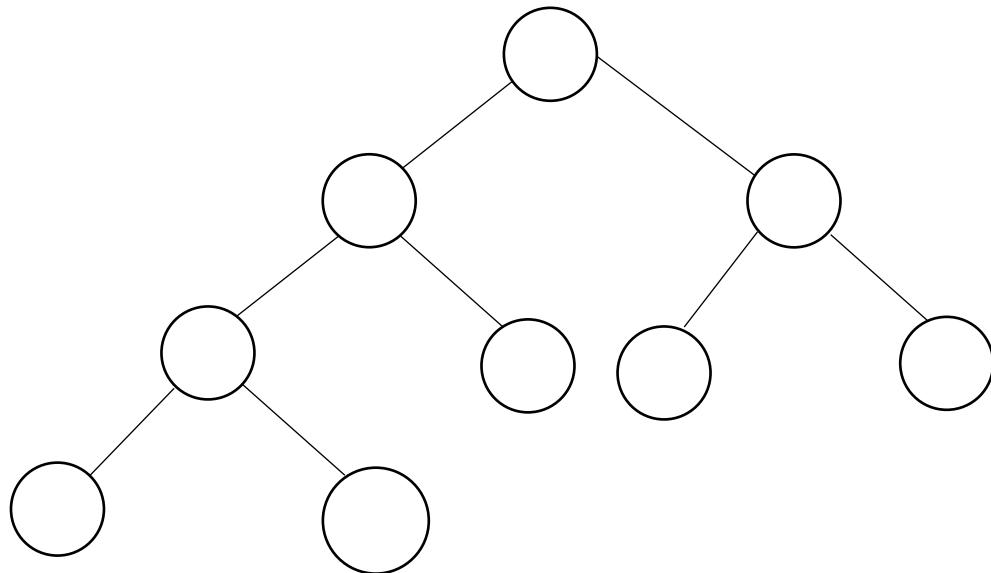
Βελτίωση του Χρόνου Χειρότερης Περίπτωσης από $O(\sqrt{n})$ σε $O(\log n)$

Ο χρόνος χειρότερης περίπτωσης του BIS μπορεί να βελτιωθεί (μειωθεί) από $O(\sqrt{n})$ σε $O(\log n)$ χωρίς να χειροτερεύει ο χρόνος μέσης περίπτωσης. Αυτό επιτυγχάνεται αν μέσα στο while αντί το i να αυξάνεται γραμμικά (δηλαδή $i \leftarrow i+1$) να αυξάνεται εκθετικά (δηλαδή $i \leftarrow i^2$). Ο χρόνος χειρότερης περίπτωσης γίνεται τώρα:

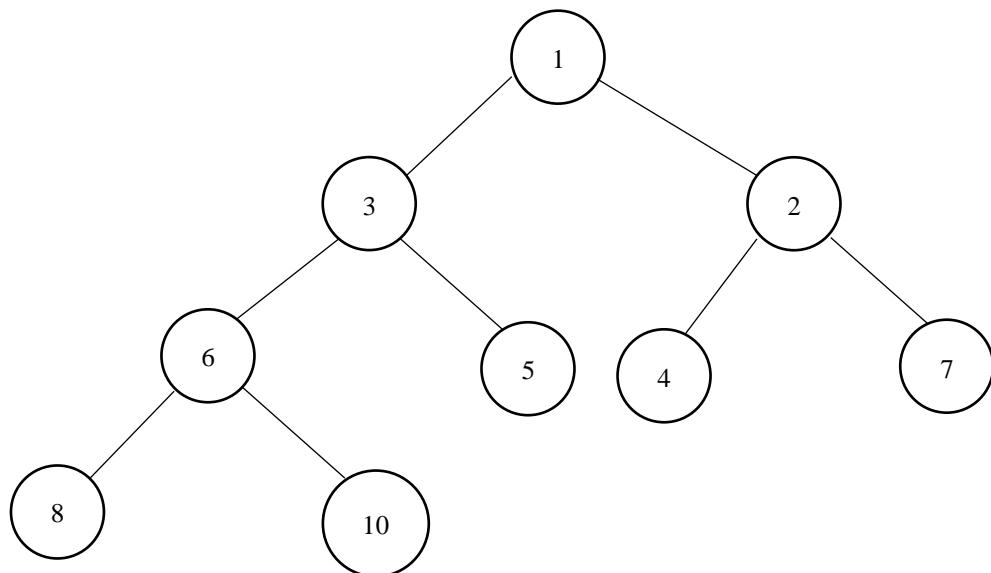
$$\log \sqrt{n} + \log \sqrt{\sqrt{n}} + \log \sqrt{\sqrt{\sqrt{n}}} + \dots = O(\log n)$$

b. Μια μεταδιατεταγμένη διαπέραση (postorder traversal) επισκέπτεται τους κόμβους 8, 10, 6, 5, 3, 4, 7, 2, 1 με τη σειρά Α-Δ-Ρ

Αρχικά κατασκευάζουμε ένα δυαδικό δέντρο 9 κόμβων από πάνω προς τα κάτω και από αριστερά προς τα δεξιά



Μετά τοποθετούμε τις τιμές στο δέντρο σύμφωνα με τη μεταδιάταξη που δίνεται:



Παρατηρούμε ότι από την εφαρμογή της μεταδιάταξης προκύπτει ένα δέντρο σωρός όπου κάθε κόμβος έχει τιμή < και από τα 2 παιδιά του άρα προκύπτει σωρός ελαχίστων.

Ο πίνακας της συνεχόμενης αναπαράστασης που αντιστοιχεί στο συγκεκριμένο δέντρο είναι ο ακόλουθος: (τοποθετούμε τα στοιχεία του σωρού-ελαχίστων στον πίνακα από πάνω προς τα κάτω και από αριστερά προς τα δεξιά).

A ₁	A ₂	A ₃	A ₄	A ₅	A ₆	A ₇	A ₈	A ₉
1	3	2	6	5	4	7	8	10

Παρατήρηση

Οι τιμές του σωρού τοποθετούνται στον πίνακα από πάνω προς τα κάτω και από αριστερά προς τα δεξιά

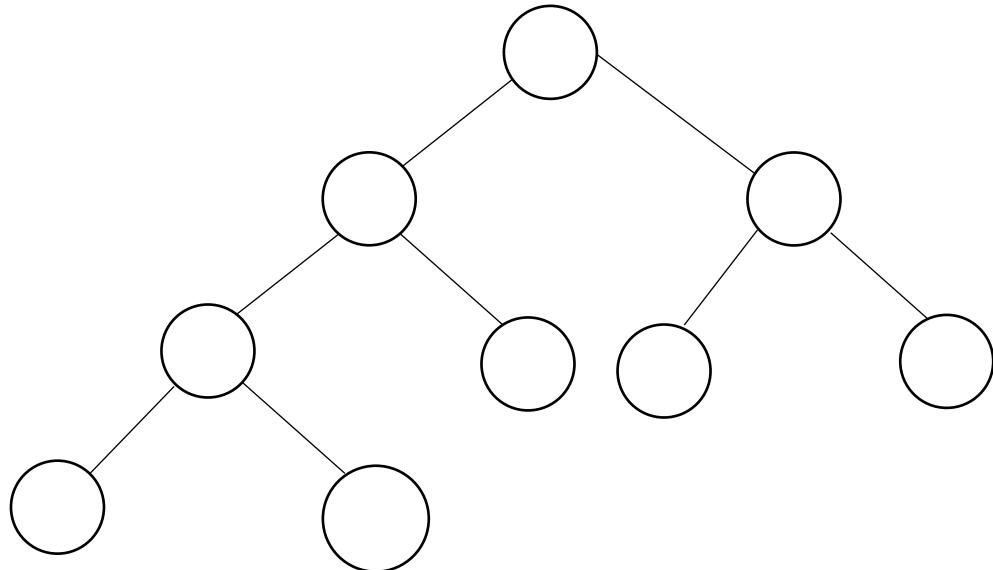
Πιθανό Θέμα: Σχεδιάστε το δέντρο σωρό στο οποίο η προ-διατεταγμένη διαπέραση (preorder traversal) επισκέπτεται τους κόμβους

1, 3, 6, 8, 10, 5, 2, 4, 7. Γράψτε τον πίνακα της συνεχόμενης αναπαράστασης που αντιστοιχεί στο συγκεκριμένο δέντρο σωρό και αναφέρετε αν πρόκειται για σωρό μεγίστων ή σωρό ελαχίστων

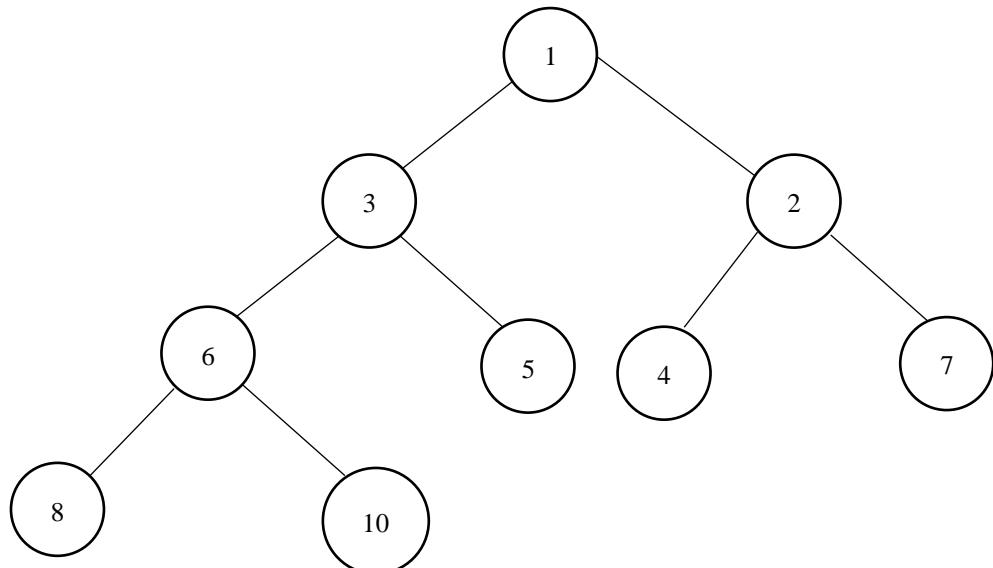
Απάντηση

Μια προ-διατεταγμένη διαπέραση (preorder traversal) επισκέπτεται τους κόμβους **1, 3, 6, 8, 10, 5, 2, 4, 7** με τη σειρά Ρ-Α-Δ

Αρχικά κατασκευάζουμε ένα δυαδικό δέντρο 9 κόμβων από πάνω προς τα κάτω και από αριστερά προς τα δεξιά



Μετά τοποθετούμε τις τιμές στο δέντρο σύμφωνα με τη προδιάταξη που δίνεται:



Παρατηρούμε ότι από την εφαρμογή της προδιάταξης προκύπτει ένα δέντρο σωρός όπου κάθε κόμβος έχει τιμή < και από τα 2 παιδιά του άρα προκύπτει σωρός ελαχίστων.

Ο πίνακας της συνεχόμενης αναπαράστασης που αντιστοιχεί στο συγκεκριμένο δέντρο είναι ο ακόλουθος: (τοποθετούμε τα στοιχεία του σωρού-ελαχίστων στον πίνακα από πάνω προς τα κάτω και από αριστερά προς τα δεξιά).

A ₁	A ₂	A ₃	A ₄	A ₅	A ₆	A ₇	A ₈	A ₉
1	3	2	6	5	4	7	8	10

9.4.2 Θέμα 7 Ιούνιος 2009 με Δυαδική Αναζήτηση Παρεμβολής -OXI

Να διατυπώσετε το Binary Interpolation Search. Δώστε τις πολυπλοκότητες μέσης και χειρότερης περίπτωσης. Περιγράψτε αναλυτικά πως θα γίνει ο χειρότερος χρόνος ψαζίματος ίσος με $O(\log n)$. Δείξτε τα βήματα του αλγορίθμου για το αρχείο [1 2 5 9 12 18 20 25 27 29 32 37 40 51 60 71 85 94 101 114 243 301 400 411 412 419 532 679 934 1123] καθώς αναζητείται το στοιχείο 556

Απάντηση

Α ερώτημα

Βασικά Βήματα (Περιγραφή) Binary Interpolation Search

Βήμα 1

Αναζητούμε ένα στοιχείο x σε ένα ταξινομημένο πίνακα S

Βήμα 2

Επιλέγουμε τη θέση του πίνακα στην οποία θα αναζητήσουμε το στοιχείο από τον τύπο $next \leftarrow \left\lceil size \cdot \frac{x - S[left]}{S[right] - S[left]} \right\rceil + 1$

Βήμα 3

Ψάχνουμε γραμμικά στον πίνακα κάνοντας άλματα μεγέθους \sqrt{size} για να προσδιορίσουμε το υποδιάστημα που περιέχει το στοιχείο αναζήτησης x . Αν το μέγεθος του πίνακα είναι μικρό π.χ. $size \leq 5$ τότε μπορούμε να εκτελέσουμε γραμμική αναζήτηση

Βήμα 4

Η διαδικασία επαναλαμβάνεται είτε μέχρι να εντοπίσουμε τα στοιχείο είτε μέχρι να βρεθούμε εκτός ορίων πίνακα οπότε επιστρέφουμε το μήνυμα Αποτυχία

Πολυπλοκότητες Μέσης και Χειρότερης περίπτωσης για το BIS

- Η Πολυπλοκότητα Χειρότερης περίπτωσης είναι $O(\sqrt{n})$
- Η Πολυπλοκότητα Μέσης περίπτωσης είναι $O(\log \log n)$

Βελτίωση του Χρόνου Χειρότερης Περίπτωσης από $O(\sqrt{n})$ σε $O(\log n)$

Ο χρόνος χειρότερης περίπτωσης μπορεί να βελτιωθεί από $O(\sqrt{n})$ σε $O(\log n)$ χωρίς να χειροτερεύει ο χρόνος μέσης περίπτωσης. Αυτό επιτυγχάνεται αν μέσα στο while loop αντί το i να αυξάνεται γραμμικά ($i \leftarrow i+1$) το i να αυξάνεται εκθετικά ($i \leftarrow i^2$). Με αυτόν τον τρόπο γίνονται ολοένα μεγαλύτερα άλματα κρατώντας το αριστερό άκρο σταθερό, με αποτέλεσμα το τελευταίο υποδιάστημα να έχει μέγεθος πολύ μεγαλύτερο από \sqrt{n} . Μόλις βρεθεί το διάστημα αυτό εφαρμόζουμε δυική αναζήτηση στα στοιχεία του που απέχουν κατά \sqrt{n} και έτσι προκύπτει το ζητούμενο υποδιάστημα μεγέθους \sqrt{n} . Ο χρόνος χειρότερης περίπτωσης είναι τώρα:

$$\log \sqrt{n} + \log \sqrt{\sqrt{n}} + \log \sqrt{\sqrt{\sqrt{n}}} + \dots = \log n^{\frac{1}{2}} + \log n^{\frac{1}{4}} + \log n^{\frac{1}{8}} + \dots = O(\log n)$$

Β ερώτημα**Εφαρμογή του αλγορίθμου Binary Interpolation Search για την αναζήτηση του στοιχείου 556**1.left \leftarrow 12.right \leftarrow 30 // (right \leftarrow n) πλήθος τιμών πίνακα = 303.size \leftarrow 30-1+1=30 // size \leftarrow right-left+1

$$4.\text{next} \leftarrow \left\lceil 30 \cdot \frac{556-1}{1123-1} \right\rceil + 1 = 16 \quad // \left(\left\lceil \text{size} \cdot \frac{x - s[\text{left}]}{s[\text{right}] - s[\text{left}]} \right\rceil + 1 \right)$$

4a. **if** (x=556>s[30]=1123 **return** "Αποτυχία" // (if x>s[n] return "Αποτυχία"). Η συνθήκη δεν ισχύει και συνεχίζει στο βήμα 5)5.**while** (x=556≠s[16]=71) do (ισχύει η συνθήκη και μπαίνει στο βρόχο)6.i \leftarrow 07. size \leftarrow 30-1+1 (right \leftarrow right-left+1)8. **if** (size≤3) απευθείας αναζήτηση (δεν ισχύει και ο αλγόριθμος συνεχίζει)9.**if** (x=556≥s[16]=71) // (if x≥s[next])10. **while** (x=556>s[16+0-1]=s[15]=60) // while($x > s[\text{next} + i\sqrt{\text{size}} - 1]$)11. i \leftarrow 1 (i \leftarrow i+1)12.**od**10. **while** (x=556>s[20]=s[16+ $\sqrt{30}$ -1]=114) / while($x > s[\text{next} + i\sqrt{\text{size}} - 1]$)11. i \leftarrow 2 (i \leftarrow i+1)12.**od**10. **while** (x=556>s[25]=s[15+2 $\sqrt{30}$ -1]=412)11.i \leftarrow 3 (i \leftarrow i+1)12.**od**10. **while** (x=556>s[31]=s[25+3 $\sqrt{30}$ -1]) δεν ισχύει η συνθήκη γιατί είμαστε εκτός ορίων πίνακα13. right \leftarrow 16+3 $\sqrt{30}$ =32 // right \leftarrow next+ i $\sqrt{\text{size}}$ 14. left \leftarrow 16+2 $\sqrt{30}$ =26 // left \leftarrow next+(i-1) $\sqrt{\text{size}}$ 14a. **if** (right=32>n=30) **then return** 'Αποτυχία' // **if** (right>n) **then return** 'Αποτυχία'

Η συνθήκη ισχύει και ο αλγόριθμος τερματίζει επιστρέφοντας το μήνυμα 'Αποτυχία'. Αυτό σημαίνει ότι το στοιχείο 556 δεν εντοπίστηκε στον πίνακα, κάτι που είναι σωστό αφού η τιμή αυτή όντως δεν υπάρχει στον πίνακα

9.4.3 Θέμα 12 Ιούνιος 2008 με Δυαδική Αναζήτηση Παρεμβολής-OXI

Να διατυπώσετε το Binary Interpolation Search. Δώστε τις πολυπλοκότητες μέσης και χειρότερης περίπτωσης. Περιγράψτε αναλυτικά πως θα γίνει ο χειρότερος χρόνος ψαζίματος ίσος με $O(\log n)$. Δείξτε τα βήματα του αλγορίθμου για το αρχείο [1 2 5 9 12 18 20 25 27 29 32 37 40 51 60 71 85 94 101 114 243 301 400 411 412 419 532 679 934 1123] καθώς αναζητείται το στοιχείο 410.

Απάντηση**Α ερώτημα****Βασικά Βήματα (Περιγραφή) Binary Interpolation Search****Βήμα 1**

Αναζητούμε ένα στοιχείο x σε ένα ταξινομημένο πίνακα S

Βήμα 2

Επιλέγουμε τη θέση του πίνακα στην οποία θα αναζητήσουμε το στοιχείο από τον τύπο $next \leftarrow \left\lceil size \cdot \frac{x - S[left]}{S[right] - S[left]} \right\rceil + 1$

Βήμα 3

Ψάχνουμε γραμμικά στον πίνακα κάνοντας άλματα μεγέθους \sqrt{size} για να προσδιορίσουμε το υποδιάστημα που περιέχει το στοιχείο αναζήτησης x . Αν το μέγεθος του πίνακα είναι μικρό π.χ. $size \leq 5$ τότε μπορούμε να εκτελέσουμε γραμμική αναζήτηση

Βήμα 4

Η διαδικασία επαναλαμβάνεται είτε μέχρι να εντοπίσουμε τα στοιχείο είτε μέχρι να βρεθούμε εκτός ορίων πίνακα οπότε επιστρέφουμε το μήνυμα Αποτυχία

Πολυπλοκότητες Μέσης και Χειρότερης περίπτωσης για το BIS

- Η Πολυπλοκότητα Χειρότερης περίπτωσης είναι $O(\sqrt{n})$
- Η Πολυπλοκότητα Μέσης περίπτωσης είναι $O(\log \log n)$

Βελτίωση του Χρόνου Χειρότερης Περίπτωσης από $O(\sqrt{n})$ σε $O(\log n)$

Ο χρόνος χειρότερης περίπτωσης μπορεί να βελτιωθεί από $O(\sqrt{n})$ σε $O(\log n)$ χωρίς να χειροτερεύει ο χρόνος μέσης περίπτωσης.

Αυτό επιτυγχάνεται αν μέσα στο while loop αντί το i να αντικαθίστανται γραμμικά ($i \leftarrow i+1$) το i να αντικαθίστανται εκθετικά ($i \leftarrow i^2$). Με αυτόν τον τρόπο γίνονται ολοένα μεγαλύτερα άλματα κρατώντας το αριστερό άκρο σταθερό, με αποτέλεσμα το τελευταίο υποδιάστημα να έχει μέγεθος πολύ μεγαλύτερο από \sqrt{n} . Μόλις βρεθεί το διάστημα αυτό εφαρμόζουμε δυνική αναζήτηση στα στοιχεία του που απέχουν κατά \sqrt{n} και έτσι προκύπτει το ζητούμενο υποδιάστημα μεγέθους \sqrt{n} . Ο χρόνος χειρότερης περίπτωσης είναι τώρα:

$$\log \sqrt{n} + \log \sqrt{\sqrt{n}} + \log \sqrt{\sqrt{\sqrt{n}}} + \dots = \log n^{\frac{1}{2}} + \log n^{\frac{1}{4}} + \log n^{\frac{1}{8}} + \dots = O(\log n)$$

Β ερώτημα-OXI**Εφαρμογή του αλγορίθμου Binary Interpolation Search για την αναζήτηση του στοιχείου 410**1. left \leftarrow 12.right \leftarrow 303.size \leftarrow 30-1+1=30

$$4.\text{next} \leftarrow \left\lceil 30 \cdot \frac{410-1}{500-1} \right\rceil + 1 = 26$$

4a. if ($x=410 > s[30]=500$) (δεν ισχύει και ο αλγόριθμος συνεχίζει)5.while ($x=410 \neq s[26]=400$) do (ισχύει η συνθήκη και μπαίνει στο βρόχο)6.i \leftarrow 07. size \leftarrow 30-1+1=308. if (size ≤ 3) όχι (δεν ισχύει και ο αλγόριθμος συνεχίζει)9.if ($x==410 \geq s[26]=400$)10. while ($x=410 \geq s[26+0\sqrt{30-1}]=s[25]=300$) (ισχύει η συνθήκη και μπαίνει στο βρόχο)11. i \leftarrow i+1 (i=1)

12.od

10. while ($x=410 \geq s[26+1\sqrt{30-1}]=s[30]=500$) (δεν ισχύει η συνθήκη και ο βρόχος τερματίζεται)13. right $\leftarrow 26+1\sqrt{30}=31$ 14. left $\leftarrow 26+0\sqrt{30}=26$

14a. if (right=31>n=30) then return Αποτυχία.

Η συνθήκη ισχύει και ο αλγόριθμος τερματίζει επιστρέφοντας το μήνυμα Αποτυχία. Αυτό σημαίνει ότι το στοιχείο 410 δεν εντοπίστηκε στον πίνακα, κάτι που είναι σωστό αφού η τιμή αυτή όντως δεν υπάρχει στον πίνακα

10 ΑΝΑΠΑΡΑΣΤΑΣΗ ΓΡΑΦΩΝ ΜΕ ΠΙΝΑΚΑ ΚΑΙ ΛΙΣΤΑ ΓΕΙΤΝΙΑΣΗΣ-ΟΧΙ

Ένας γράφος λέγεται αραιός αν ο αριθμός των ακμών του είναι τάξης $O(n)$, όπου n είναι ο αριθμός κορυφών του διαφορετικά λέγεται πυκνός. Συχνά συσχετίζουμε κάθε ακμή ενός γράφου με κάποιο βάρος (weight). Τότε ο γράφος ονομάζεται γράφος με βάρη (weighted graph).

10.1 Αναπαράσταση με Πίνακες

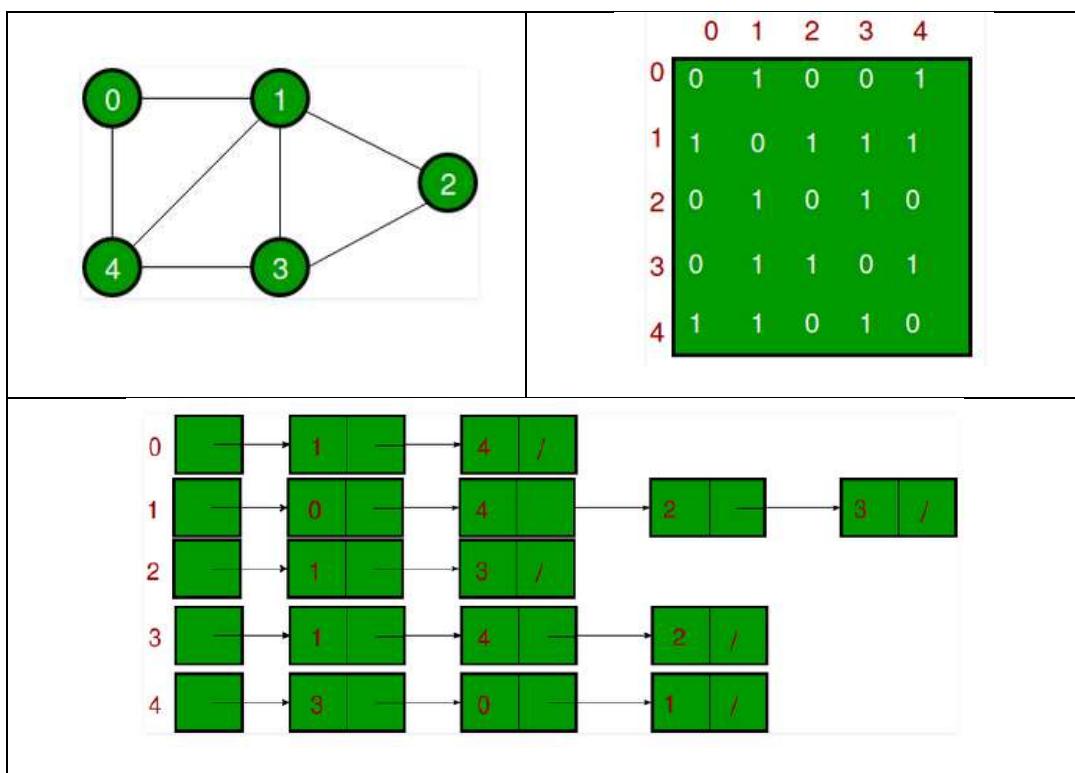
- Ένας γράφος $G=(V, E)$ με n κορυφές μπορεί να αναπαρασταθεί ως ένας $n \times n$ πίνακας που περιέχει τις τιμές 0 και 1 όπου:
 - αν η (i, j) είναι ακμή τότε $A[i, j]=1$, διαφορετικά $A[i, j]=0$
- Αν ο γράφος περιέχει βάρη και το βάρος κάθε ακμής είναι t , τότε για την αναπαράσταση του γράφου μπορεί να χρησιμοποιηθεί πίνακας με
 - $A[i, j] = \text{βάρος } (i, j)$ αν υπάρχει ακμή (i, j)
 - $A[(i, j)] = \infty$ αν δεν υπάρχει ακμή (i, j)
- **Η αναπαράσταση του γράφου με πίνακα απαιτεί χώρο $O(n^2)$, όπου $n=|V|$ δηλ. το πλήθος των κορυφών του**
- **Αν ο γράφος είναι αραιός η μέθοδος οδηγεί σε σπάταλη χώρου**

10.2 Αναπαράσταση με Λίστες Γειτνίασης

- Ένας γράφος $G=(V, E)$ αναπαρίσταται ως ένας μονοδιάστατος πίνακας A
- Για κάθε κορυφή v , $A[v]$ είναι ένας δείκτης σε μια συνδεδεμένη λίστα στην οποία αποθηκεύονται οι κορυφές που γειτνιάζουν με την v
- **Η μέθοδος αυτή απαιτεί χώρο $O(|V|+|E|)=\max(|V|, |E|)$ δηλ. ο χώρος είναι ο μεγαλύτερος από τα 2 μεγέθη όπου $|V|$ πλήθος κορυφών και $|E|$ πλήθος ακμών**
- **Επιτυγχάνεται εξοικονόμηση χώρου για αραιούς γράφους**
- **Στην περίπτωση γράφων με βάρη στη λίστα γειτνίασης αποθηκεύονται επίσης το βάρος κάθε ακμής**

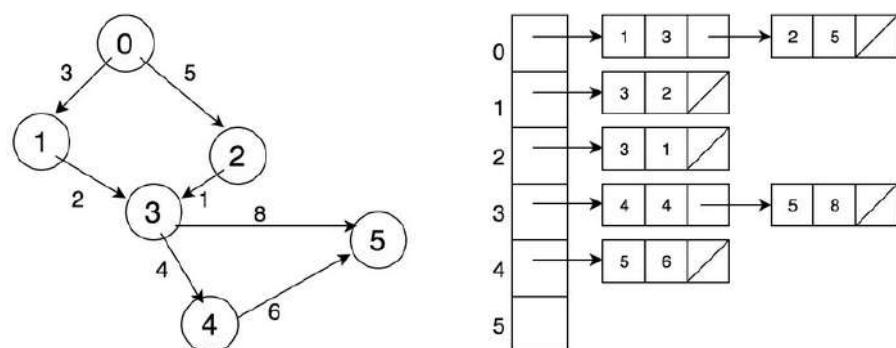
10.3 Παραδείγματα Αναπαράστασης Γράφων με Λίστες Γειτνίασης και Πίνακες

Παράδειγμα 1- Μη Κατευθυνόμενος Γράφος χωρίς Βάρη



Παράδειγμα 2 – Κατευθυνόμενος Γράφος Με Βάρη

Λίστα Γειτνίασης



Αναπαράσταση Γράφου με Πίνακα

	0	1	2	3	4	5
0	∞	3	5	∞	∞	∞
1	∞	∞	∞	2	∞	∞
2	∞	∞	∞	1	∞	∞
3	∞	∞	∞	∞	4	8
4	∞	∞	∞	∞	∞	6
5	∞	∞	∞	∞	∞	∞

11 ΚΑΤΑΚΕΡΜΑΤΙΣΜΟΣ

11.1 Θέμα 8 Ιούνιος 2023

Έστω ότι θέλουμε να ενθέσουμε τα κλειδιά 8, 20, 29, 2, 13, 26, 15, 86 σε ένα hash πίνακα μήκους $m=10$ με τη σειρά που δίνονται χρησιμοποιώντας **hashing με αλυσίδες** και **hashing με open addressing**. Και στις δύο μεθόδους η βασική hash συνάρτηση είναι η $h_1(k)=k \bmod m$. Στο hashing με open addressing έστω η $h_2(k)=(p+7-k) \bmod m$ (**μέθοδος με double hashing**) όπου p το τελευταίο ψηφίο του AM σας. Δώστε σχηματικά τους hash πίνακες και αναλυτικά τις πράξεις.

Απάντηση

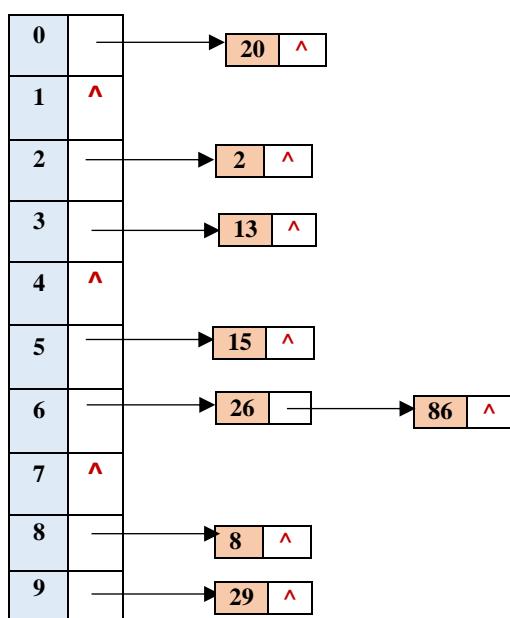
Hashing με Αλυσίδες

Οι θέσεις του πίνακα κατακερματισμού (hash table) αριθμούνται γενικά από $i=0 \dots m-1$ όπου m το μέγεθος του πίνακα κατακερματισμού. Από κάθε θέση του πίνακα κατακερματισμού αρχίζει μια λίστα με όλα τα κλειδιά που η hash function τοποθετεί στη λίστα αυτή.

Η συνάρτηση κατακερματισμού είναι $h_1(k)=k \bmod m$ και στην περίπτωση μας για $m=10$ έχει τη μορφή: **$h_1(k)=k \bmod 10$** . Βάζουμε στο k το κάθε κλειδί (δηλ. την κάθε τιμή που δίνεται στην εκφώνηση) προκειμένου να βρούμε σε ποια λίστα του πίνακα κατακερματισμού θα τοποθετηθεί. Η εκάστοτε λίστα περιέχει τα κλειδιά που η hash function τοποθετεί στη θέση αυτή.

Πιο συγκεκριμένα:

- $h_1(8)=8 \bmod 10=8$
- $h_1(20)=20 \bmod 10=0$
- $h_1(29)=29 \bmod 10=9$
- $h_1(2)=2 \bmod 10=2$
- $h_1(13)=13 \bmod 10=3$
- $h_1(26)=26 \bmod 10=6$
- $h_1(15)=15 \bmod 10=5$
- $h_1(86)=86 \bmod 10=6$



- Σε αντίθεση με τον κατακερματισμό με αλυσίδες, στον κατακερματισμό με Open Addressing τα κλειδιά τοποθετούνται απευθείας στο πίνακα κατακερματισμού (hash πίνακας) και ο hash πίνακας δεν επεκτείνεται. Αυτό σημαίνει ότι αν τα κλειδιά είναι περισσότερα από τις θέσεις του πίνακα κατακερματισμού, όσα κλειδιά περισσεύουν δηλ. δεν χωράνε στον πίνακα κατακερματισμού δεν τοποθετούνται σε αυτόν
- Η γενική συνάρτηση κατακερματισμού στο **double Hashing** είναι:

$$h(k, i) = [h_1(k) + i \cdot h_2(k)] \bmod m$$
 για $i=0, 1, \dots, m-1$ όπου m το μέγεθος του πίνακα κατακερματισμού
- Το k είναι το κλειδί που εισάγουμε κάθε φορά στον πίνακα κατακερματισμού
- Το i είναι η προσπάθεια στην οποία τοποθετούμε το κλειδί στον πίνακα κατακερματισμού
- **Σε κάθε νέο κλειδί η προσπάθεια δηλ. το i αρχίζει από το 0**
- Στη συγκεκριμένη άσκηση η μορφή της συνάρτησης κατακερματισμού στο **double hashing** είναι:

$$h(k, i) = [k \bmod m + i \cdot (p+7-k \bmod 7) \bmod m] \bmod m$$
 για $i=0, 1, 2, \dots, m-1$
Το p είναι το τελευταίο ψηφίο του ΑΜ και ενδεικτικά θέτουμε όπου $p=6$

$$h(k, i) = [k \bmod 10 + i \cdot (6+7-k \bmod 7) \bmod 10] \bmod 10$$
 για $i=0, 1, 2, \dots, 9$

Τοποθέτηση Κλειδιών με τη μέθοδο Κατακερματισμού Hashing με Open Addressing και Double Hashing

$$h(8, 0) = [8 \bmod 10 + 0 \times (6+7-8 \bmod 7) \bmod 10] \bmod 10 = 8 \bmod 10 = 8$$

$$h(20, 0) = [20 \bmod 10 + 0 \times (6+7-20 \bmod 7) \bmod 10] \bmod 10 = 0 \bmod 10 = 0$$

$$h(29, 0) = [29 \bmod 10 + 0 \times (6+7-29 \bmod 7) \bmod 10] \bmod 10 = 9 \bmod 10 = 9$$

$$h(2, 0) = [2 \bmod 10 + 0 \times (6+7-2 \bmod 7) \bmod 10] \bmod 10 = 2 \bmod 10 = 2$$

$$h(13, 0) = [13 \bmod 10 + 0 \times (6+7-13 \bmod 7) \bmod 10] \bmod 10 = 3 \bmod 10 = 3$$

$$h(26, 0) = [26 \bmod 10 + 0 \times (6+7-26 \bmod 7) \bmod 10] \bmod 10 = 6 \bmod 10 = 6$$

$$h(15, 0) = [15 \bmod 10 + 0 \times (6+7-15 \bmod 7) \bmod 10] \bmod 10 = 5 \bmod 10 = 5$$

h(86, 0) = [86 mod 10 + 0 × (6+7-85 mod 7) mod 10] mod 10 = 6. Προκύπτει σύγκρουση διότι η θέση 6 στον πίνακα κατακερματισμού είναι κατειλημμένη από το κλειδί 26. Δεν μπορούμε να τοποθετήσουμε 2 κλειδιά στην ίδια θέση του πίνακα κατακερματισμού. Δοκιμάζουμε για $i=1$

$$h(86, 1) = [86 mod 10 + 1 × (6+7-86 mod 7) mod 10] mod 10 = [6+(6+7-2)] mod 10 = 17 mod 10 = 7$$

Παρατήρηση: Ο τελεστής mod έχει μεγαλύτερη προτεραιότητα από την πρόσθεση και την αφαίρεση

Δομές Δεδομένων –Computer Ανάλυση
Πίνακας Κατακερματισμού

0	20
1	
2	2
3	13
4	
5	15
6	26
7	86
8	8
9	29

11.2 Θέμα 6 Σεπτέμβριος 2016 και Ιούνιος 2017

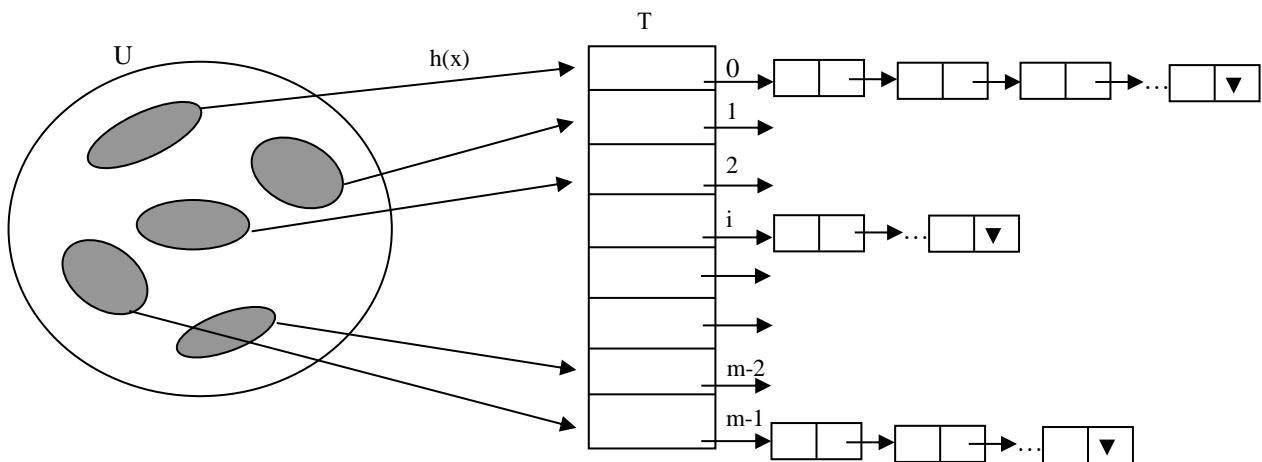
Έστω ότι θέλουμε να ενθέσουμε τα κλειδιά 42, 15, 27, 56, 34, 73, 4, 64 σε ένα hash πίνακα μήκους $m=8$ με τη σειρά που δίνονται χρησιμοποιώντας **hashing με αλυσίδες** και **hashing με open addressing**. Και στις δύο μεθόδους η βασική hash συνάρτηση είναι η $h_1(k)=k \bmod m$. Στο hashing με open addressing έστω η $h_2(k)=R=(key \bmod R)$ όπου το $R=7$ (μέθοδος με **double hashing**). Δώστε σχηματικά τους hash πίνακες και αναλυτικά τις πράξεις.

Απάντηση**Θεωρία**

Hashing (κατακερματισμός) είναι μια μεθοδολογία **οργάνωσης τιμών με σκοπό το γρήγορο και εύκολο εντοπισμό τους**

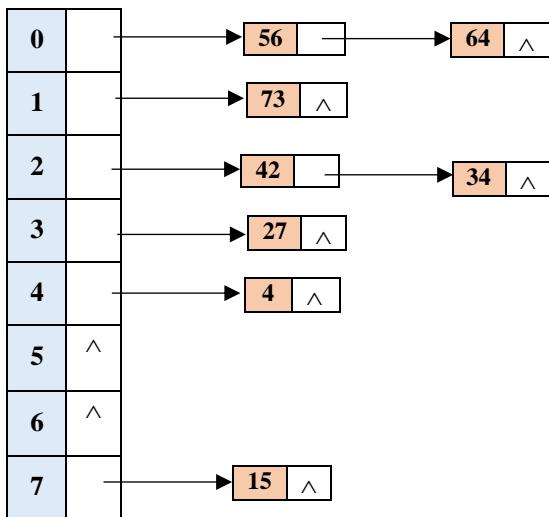
Θεωρία για Hashing με Αλυσίδες

- Στο hashing (κατακερματισμό) με αλυσίδες σε κάθε θέση του πίνακα κατακερματισμού αντιστοιχίζεται μία λίστα (η αντιστοίχιση γίνεται με δείκτη στη λίστα) έτσι ώστε να αποθηκεύει όσα στοιχεία χρειάζεται να εισαχθούν σε εκείνη τη θέση όπως στο σχήμα)

**Hashing με Αλυσίδες**

Η συνάρτηση κατακερματισμού είναι: $h_1(k)=k \bmod m$ και στην περίπτωση μας το $m=8$, άρα η συνάρτηση κατακερματισμού είναι: $h_1(k)=k \bmod 8$. Βάζουμε στο k το κάθε κλειδί (δηλ. την κάθε τιμή που δίνεται στην εκφώνηση) προκειμένου να βρούμε σε ποια θέση του πίνακα κατακερματισμού θα τοποθετηθεί.

- $h_1(42)=42 \bmod 8=2$
- $h_1(15)=15 \bmod 8=7$
- $h_1(27)=27 \bmod 8=3$
- $h_1(56)=56 \bmod 8=0$
- $h_1(34)=34 \bmod 8=2$
- $h_1(73)=73 \bmod 8=1$
- $h_1(4)=4 \bmod 8=4$
- $h_1(64)=64 \bmod 8=0$



Θεωρία για Hashing με Open Addressing (Double Hashing)

Η γενική συνάρτηση κατακερματισμού στο **double Hashing** είναι: $h(k, i) = [h_1(k) + i \times h_2(k)] \bmod m$

- $i=0,1,\dots,m-1$ όπου m το μέγεθος του πίνακα κατακερματισμού και i είναι η προσπάθεια στην οποία τοποθετούμε το κλειδί στον πίνακα. Σε κάθε νέο κλειδί το i αρχίζει από το 0
 - Το k είναι το κλειδί που εισάγουμε κάθε φορά στον πίνακα κατακερματισμού
 - Σε αντίθεση με το hashing με αλυσίδες, στον κατακερματισμό με Open Addressing τα κλειδιά τοποθετούνται απευθείας στο hash πίνακα κατακερματισμού
-
- Ο hash πίνακας δεν επεκτείνεται. Αυτό σημαίνει ότι αν τα κλειδιά είναι περισσότερα από τις θέσεις του πίνακα κατακερματισμού τότε όσα κλειδιά δεν χωράνε στον πίνακα κατακερματισμού απλά δεν τοποθετούνται σε αυτόν
-
- Στη συγκεκριμένη άσκηση η μορφή της συνάρτησης κατακερματισμού είναι: $h(k, i) = [k \bmod 8 + i \times (k \bmod 7)] \bmod 8$ για $i=0\dots 7$

Τοποθέτηση Κλειδιών με τη μέθοδο Κατακερματισμού Hashing με Open Addressing και Double Hashing

$$h(42, 0) = [42 \bmod 8 + 0 \times (42 \bmod 7)] \bmod 8 = [2 + 0] \bmod 8 = 2 \bmod 8 = 2$$

$$h(15, 0) = [15 \bmod 8 + 0 \times (15 \bmod 7)] \bmod 8 = [15 + 0] \bmod 8 = 7 \bmod 8 = 7$$

$$h(27, 0) = [27 \bmod 8 + 0 \times (27 \bmod 7)] \bmod 8 = [27 + 0] \bmod 8 = 3 \bmod 8 = 3$$

$$h(56, 0) = [56 \bmod 8 + 0 \times (56 \bmod 7)] \bmod 8 = [56 + 0] \bmod 8 = 0 \bmod 8 = 0$$

$$h(34, 0) = [34 \bmod 8 + 0 \times (34 \bmod 7)] \bmod 8 = [34 + 0] \bmod 8 = 2 \bmod 8 = 2.$$

Προκύπτει σύγκρουση διότι η θέση με αριθμό 2 είναι κατειλημμένη από το 42. Δοκιμάζουμε για $i=1$

$h(34, 1) = [34 \text{ mod } 8 + 1 \times (34 \text{ mod } 7)] \text{ mod } 8 = [34+6] \text{ mod } 8 = 40 \text{ mod } 8 = 0$. Προκύπτει πάλι σύγκρουση διότι η θέση με αριθμό 0 είναι κατειλημμένη από το 56. Δοκιμάζουμε για $i=2$

$h(34, 2) = [34 \text{ mod } 8 + 2 \times (34 \text{ mod } 7)] \text{ mod } 8 = [34+12] \text{ mod } 8 = 46 \text{ mod } 8 = 6$

$h(73, 0) = [73 \text{ mod } 8 + 0 \times (73 \text{ mod } 7)] \text{ mod } 8 = [73+0] \text{ mod } 8 = 1 \text{ mod } 8 = 1$

$h(4, 0) = [4 \text{ mod } 8 + 0 \times (4 \text{ mod } 7)] \text{ mod } 8 = [4+0] \text{ mod } 8 = 4 \text{ mod } 8 = 4$

$h(64, 0) = [64 \text{ mod } 8 + 0 \times (64 \text{ mod } 7)] \text{ mod } 8 = [0+0] \text{ mod } 8 = 0 \text{ mod } 8 = 0$. Προκύπτει σύγκρουση διότι η θέση με αριθμό 0 είναι κατειλημμένη από το 56. Δοκιμάζουμε για $i=1$

$h(64, 1) = [64 \text{ mod } 8 + 1 \times (64 \text{ mod } 7)] \text{ mod } 8 = [0+1] \text{ mod } 8 = 1 \text{ mod } 8 = 1$. Προκύπτει σύγκρουση διότι η θέση με αριθμό 1 είναι κατειλημμένη από το 73. Δοκιμάζουμε για $i=2$

$h(64, 2) = [64 \text{ mod } 8 + 2 \times (64 \text{ mod } 7)] \text{ mod } 8 = [0+2] \text{ mod } 8 = 2 \text{ mod } 8 = 2$. Προκύπτει πάλι σύγκρουση διότι η θέση με αριθμό 2 είναι κατειλημμένη από το 42. Δοκιμάζουμε για $i=3$

$h(64, 3) = [64 \text{ mod } 8 + 3 \times (64 \text{ mod } 7)] \text{ mod } 8 = [0+3] \text{ mod } 8 = 3 \text{ mod } 8 = 3$. Προκύπτει πάλι σύγκρουση διότι η θέση με αριθμό 3 είναι κατειλημμένη από το 27. Δοκιμάζουμε για $i=4$

$h(64, 4) = [64 \text{ mod } 8 + 4 \times (64 \text{ mod } 7)] \text{ mod } 8 = [0+4] \text{ mod } 8 = 4 \text{ mod } 8 = 4$. Προκύπτει πάλι σύγκρουση διότι η θέση με αριθμό 4 είναι κατειλημμένη από το 4. Δοκιμάζουμε για $i=5$

$h(64, 5) = [64 \text{ mod } 8 + 5 \times (64 \text{ mod } 7)] \text{ mod } 8 = [0+5] \text{ mod } 8 = 5 \text{ mod } 8 = 5$.

Παρατήρηση: Το i μετρά τις προσπάθειες τοποθέτησης κάθε νέου κλειδιού και παίρνει τις τιμές από 0 έως $m-1$ όπου m το μέγεθος του πίνακα κατακερματισμού. Εδώ το $i=0..7$. Αν συμπληρωθούν ΟΛΕΣ οι προσπάθειεις για το i και ΔΕΝ βρεθεί KENH θέση στον πίνακα κατακερματισμού, τότε το νέο κλειδί ΔΕΝ τοποθετείται στον πίνακα κατακερματισμού

Τελικός Πίνακας Κατακερματισμού

0	56
1	73
2	42
3	27
4	4
5	64
6	34
7	15

11.3 Θέμα 3-Ιούνιος 2011

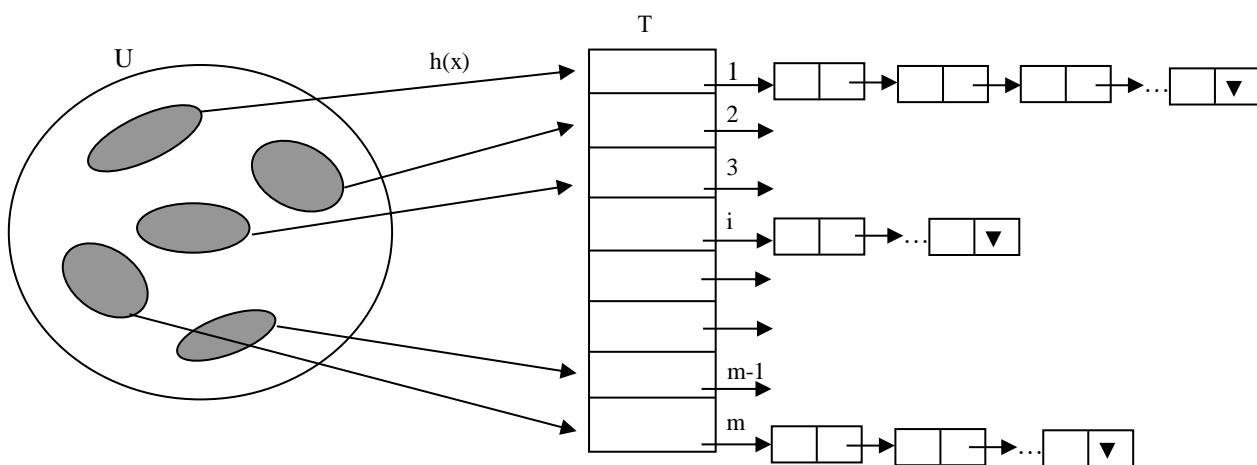
3. Τι είναι hashing με αλυσίδες και τι hashing με open addressing; Ποιες είναι οι χαρακτηριστικές τους πολυπλοκότητες; Με ποιες βασικές υποθέσεις γίνεται η ανάλυση; Έστω ότι θέλουμε να ενθέσουμε τα κλειδιά {22,10,13,41,51,8,17,58,53} έναν πίνακα μήκους $m=11$ με τη σειρά που δίνονται χρησιμοποιώντας hashing με αλυσίδες και hashing με open addressing. Και για τις δύο μεθόδους η βασική hash συνάρτηση είναι η $h_1(k)=k \bmod m$. Στο hashing με open addressing έστω ότι $h_2(k)=1+(k \bmod (m-1))$ (ακολουθείτε η μέθοδος double hashing). Δώστε σχηματικά τους hash πίνακες.

Απάντηση

Τι είναι το Hashing με Αλυσίδες;

Απάντηση

- Στο hashing (κατακερματισμό) με αλυσίδες σε κάθε θέση του πίνακα κατακερματισμού αντιστοιχίζεται μία λίστα (η αντιστοιχίση γίνεται με δείκτη στη λίστα) έτσι ώστε να αποθηκεύονται όσα στοιχεία χρειάζεται να εισαχθούν σε εκείνη τη θέση όπως στο σχήμα)



Ποιες οι χαρακτηριστικές πολυπλοκότητες του Κατακερματισμού (hashing) με αλυσίδες;

- Η χρονική πολυπλοκότητα χειρότερης περίπτωσης για την εισαγωγή είναι $O(1)$ ενώ για την εύρεση και την διαγραφή είναι $O(|S|)$ όπου S είναι το σύνολο των κλειδιών

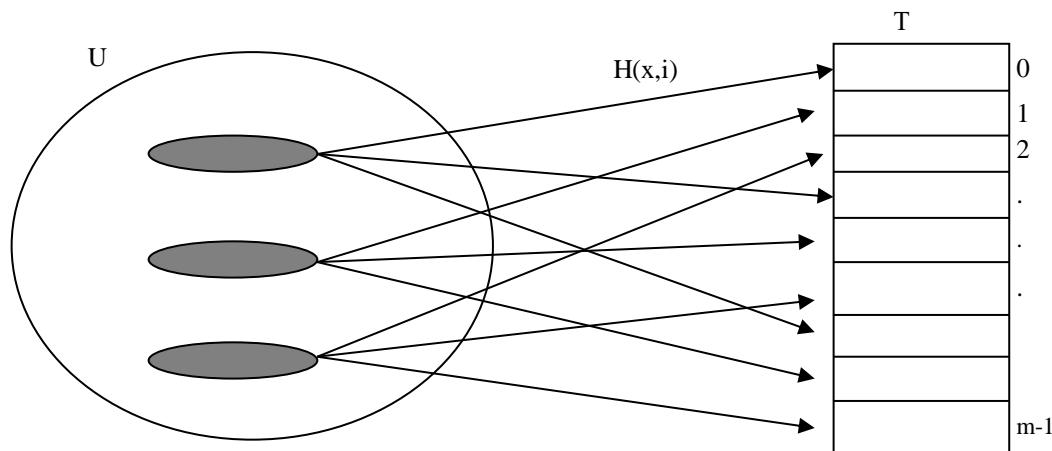
Με ποιες βασικές υποθέσεις γίνεται η ανάλυση του Κατακερματισμού (hashing) με αλυσίδες;

- Η συνάρτηση κατακερματισμού $h(x)$, $h: U \rightarrow [0 \dots m-1]$ κατανέμει το U ομοιόμορφα στις θέσεις του πίνακα T . Η αλλιώς σε κάθε θέση του πίνακα T αντιστοιχίζεται ίσος αριθμός στοιχείων του U .
 - Όλα τα στοιχεία του U είναι το ίδιο πιθανά να χρησιμοποιηθούν σε κάποια από τις K πράξεις. Για παράδειγμα στην κοστή πράξη κάθε στοιχείο $x, x \in U$ μπορεί να επιλεγεί με πιθανότητα $\frac{1}{|U|}$.
 - Με βάση τις παραπάνω υποθέσεις, μια ακολουθία από K πράξεις χρειάζεται χρόνο $O\left(K \left(1 + \frac{b}{2}\right)\right)$ με $b = \frac{k}{m}$ (Παράγοντας Κατανομής)

Τι γνωρίζετε για τον **Κατακερματισμό με open addressing (Hashing με open addressing)**;

Απάντηση

Στον κατακερματισμό με ανοικτή διεύθυνση (open addressing) γίνεται προσπάθεια οι συγκρούσεις να αντιμετωπισθούν στα όρια του πίνακα κατακερματισμού και όχι χρησιμοποιώντας επιπρόσθετο χώρο όπως στον κατακερματισμό με αλυσίδες. Κάθε στοιχείο x τοποθετείται στη θέση του πίνακα κατακερματισμού που προκύπτει ύστερα από μια ακολουθία δοκιμών. Κάθε πράξη με όρισμα το στοιχείο x σαρώνει την ακολουθία των θέσεων που μέχρι να βρεθεί μια κατάλληλη θέση.



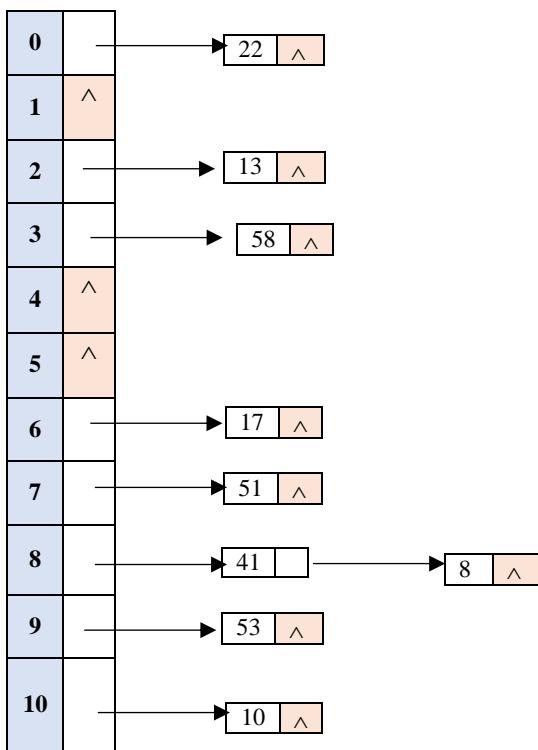
Hashing με Αλυσίδες

Η συνάρτηση κατακερματισμού είναι $h_1(k) = k \bmod m$

- $h_1(22) = 22 \bmod 11 = 0$
- $h_1(10) = 10 \bmod 11 = 10$
- $h_1(13) = 13 \bmod 11 = 2$
- $h_1(41) = 41 \bmod 11 = 8$
- $h_1(51) = 51 \bmod 11 = 7$
- $h_1(8) = 8 \bmod 11 = 8$
- $h_1(17) = 17 \bmod 11 = 6$
- $h_1(58) = 58 \bmod 11 = 3$
- $h_1(53) = 53 \bmod 11 = 9$

O hash πίνακας έχει την τελική μορφή:

Τοποθέτηση Κλειδιών στον Κατακερματισμό με Αλυσίδες



Θεωρία για Hashing με Open Addressing (Double Hashing)

- Η γενική συνάρτηση κατακερματισμού στο **double Hashing** είναι: $h(k, i) = [h_1(k) + i \times h_2(k)] \bmod m$
- $i=0,1,\dots,m-1$ όπου m το μέγεθος του πίνακα κατακερματισμού
- Το k είναι το κλειδί που εισάγουμε κάθε φορά στον πίνακα κατακερματισμού
- Το i είναι η προσπάθεια στην οποία τοποθετούμε το κλειδί στον πίνακα. Σε κάθε νέο κλειδί το i αρχίζει από το 0
- Σε αντίθεση με το hashing με αλυσίδες, στον κατακερματισμό με Open Addressing τα κλειδιά τοποθετούνται απευθείας μέσα στο hash πίνακα**
- Ο hash πίνακας δεν επεκτείνεται. Αυτό σημαίνει ότι αν τα κλειδιά είναι περισσότερα από τις θέσεις του πίνακα κατακερματισμού τότε όσα κλειδιά δεν χωράνε στον πίνακα κατακερματισμού απλά δεν τοποθετούνται σε αυτόν
- Στη συγκεκριμένη άσκηση η μορφή της συνάρτησης κατακερματισμού είναι:

$$h(k, i) = [k \bmod m + i \times (1+k \bmod (m-1))] \bmod m = [k \bmod 11 + i \times (1+k \bmod 10)] \bmod 11 \text{ για } i=0\dots 10$$

Η εισαγωγή των κλειδιών στον hash πίνακα γίνεται ως εξής:

$$h(22, 0) = [22 \bmod 11 + 0] \bmod 11 = 0$$

$$h(10, 0) = [10 \bmod 11 + 0] \bmod 11 = 10$$

$$h(13, 0) = [13 \bmod 11 + 0] \bmod 11 = 2$$

$$h(41, 0) = [41 \bmod 11 + 0] \bmod 11 = 8$$

$$h(51, 0) = [51 \bmod 11 + 0] \bmod 11 = 7$$

$$h(8, 0) = [8 \bmod 11 + 0] \bmod 11 = 8$$

Παρατηρούμε ότι έχουμε **σύγκρουση** καθώς στη θέση 8 έχουμε βάλει ήδη το 41. Θα κάνουμε λοιπόν μια δεύτερη προσπάθεια

$$h(8, 1) = [8 \bmod 11 + 1 * 9] \bmod 11 = 6$$

$$h(17, 0) = [17 \bmod 11 + 0] \bmod 11 = 6 \text{ σύγκρουση}$$

$$h(17, 1) = [17 \bmod 11 + 1 * 8] \bmod 11 = 3$$

$$h(58, 0) = [58 \bmod 11 + 0] \bmod 11 = 3 \text{ σύγκρουση}$$

$$h(58, 1) = [58 \bmod 11 + 9] \bmod 11 = 1$$

$$h(53, 0) = [53 \bmod 11 + 0] \bmod 11 = 9$$

0	22
1	58
2	13
3	17
4	
5	
6	8
7	51
8	41
9	53
10	10

Αυτός είναι ο τελικός πίνακας κατακερματισμού με τα κλειδιά που τοποθετήθηκαν σε αυτόν.

11.4 Θέμα 9-Ιούνιος 2008

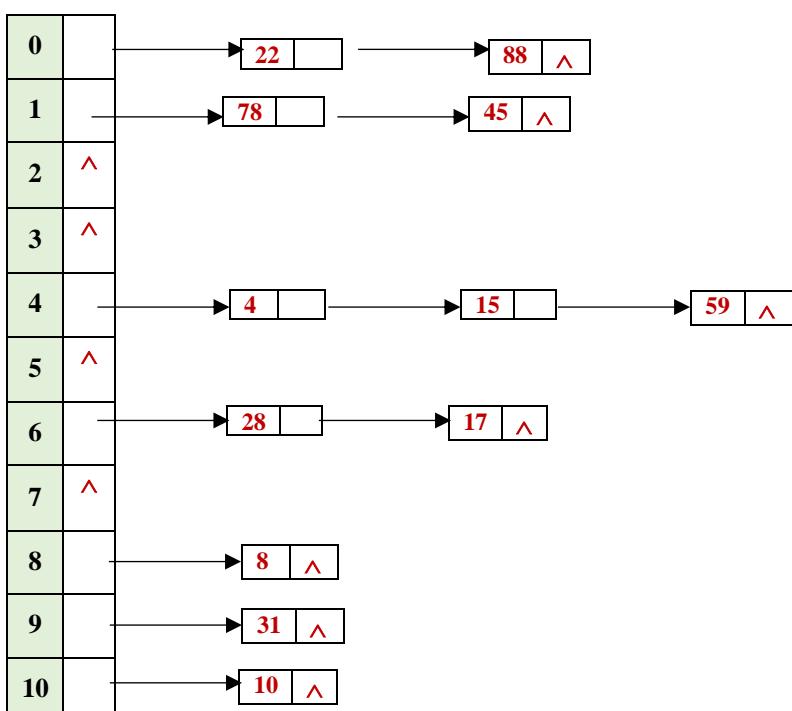
Τι είναι hashing με αλυσίδες και τι hashing με open addressing. Έστω ότι θέλουμε να ενθέσουμε τα κλειδιά **10, 22, 31, 4, 15, 28, 17, 88, 59, 78, 45, 8** σε ένα hash πίνακα μήκους **m=11** με τη σειρά που δίνονται χρησιμοποιώντας hashing με αλυσίδες και hashing με open addressing. Και στις δύο μεθόδους η βασική hash συνάρτηση είναι η **$h_1(k)=k \bmod m$** . Στο hashing με open addressing δίνεται ότι η **$h_2(k)=5+(k \bmod (m-1))$** (μέθοδος double hashing). Δώστε σχηματικά τους hash πίνακες.

Απάντηση**Α ερώτημα-Hashing με Αλυσίδες**

$$h_1(k)=k \bmod 11$$

- $h_1(10) = 10 \bmod 11 = 10$
- $h_1(22) = 22 \bmod 11 = 0$
- $h_1(31) = 31 \bmod 11 = 9$
- $h_1(4) = 4 \bmod 11 = 4$
- $h_1(15) = 15 \bmod 11 = 4$
- $h_1(28) = 28 \bmod 11 = 6$
- $h_1(17) = 17 \bmod 11 = 6$
- $h_1(88) = 88 \bmod 11 = 0$
- $h_1(59) = 59 \bmod 11 = 4$
- $h_1(78) = 78 \bmod 11 = 1$
- $h_1(45) = 45 \bmod 11 = 1$
- $h_1(8) = 8 \bmod 11 = 8$

Ο hash πίνακας έχει την τελική μορφή:



Δομές Δεδομένων –Computer Ανάλυση
Β ερότημα-Hashing με Open Addressing (Double Hashing)

- Η γενική συνάρτηση κατακερματισμού στο double Hashing είναι:

$$h(k, i) = [h_1(k) + i \cdot h_2(k)] \bmod m \quad \text{για } i=0, 1, \dots, m-1 \quad \text{όπου } m \text{ το μέγεθος του πίνακα κατακερματισμού}$$

- Το **k** είναι το κλειδί που εισάγουμε κάθε φορά στον πίνακα κατακερματισμού
- Το **i** είναι η προσπάθεια στην οποία τοποθετούμε το κλειδί στον πίνακα κατακερματισμού. Σε κάθε νέο κλειδί το i αρχίζει από το 0
- **Σε αντίθεση με το hashing με αλυσίδες, στον κατακερματισμό με Open Addressing τα κλειδιά τοποθετούνται απευθείας στο hash πίνακα και ο hash πίνακας δεν επεκτείνεται.** Αυτό σημαίνει ότι αν τα κλειδιά είναι περισσότερα από τις θέσεις του πίνακα κατακερματισμού τότε όσα κλειδιά δεν χωράνε στον πίνακα κατακερματισμού απλά δεν τοποθετούνται σε αυτόν
- Στη συγκεκριμένη άσκηση η μορφή της συνάρτησης κατακερματισμού στο double hashing είναι:

$$h(k, i) = [k \bmod 11 + i \times (5 + k \bmod 10)] \bmod 11 \quad \text{για } i=0, 1, 2, \dots, 10$$

Τοποθέτηση Κλειδιών με τη μέθοδο Κατακερματισμού Hashing με Open Addressing και Double Hashing

$$h(10,0) = [10 \bmod 11] \bmod 11 = 10$$

$$h(22,0) = [22 \bmod 11] \bmod 11 = 0$$

$$h(31,0) = [31 \bmod 11] \bmod 11 = 9$$

$$h(4,0) = [4 \bmod 11] \bmod 11 = 4$$

$$h(15,0) = [15 \bmod 11] \bmod 11 = 4 \text{ σύγκρουση. Δοκιμάζουμε για } i=1$$

$$h(15,1) = [15 \bmod 11 + 1(5 + 15 \bmod 10)] \bmod 11 = 3$$

$$h(28,0) = [28 \bmod 11] \bmod 11 = 6$$

$$h(17,0) = [17 \bmod 11] \bmod 11 = 6 \text{ σύγκρουση. Δοκιμάζουμε για } i=1$$

$$h(17,1) = [17 \bmod 11 + 1(5 + 17 \bmod 10)] \bmod 11 = 7$$

$$h(88,0) = [88 \bmod 11] \bmod 11 = 0 \text{ σύγκρουση. Δοκιμάζουμε για } i=1$$

$$h(88,1) = [88 \bmod 11 + 1(5 + 88 \bmod 10)] \bmod 11 = 2$$

$$h(59,0) = [59 \bmod 11] \bmod 11 = 4 \text{ σύγκρουση. Δοκιμάζουμε για } i=1$$

$$h(59,1) = [59 \bmod 11 + 1(5 + 59 \bmod 10)] \bmod 11 = 7 \text{ σύγκρουση. Δοκιμάζουμε για } i=2$$

$$h(59,2) = [59 \bmod 11 + 2(5 + 59 \bmod 10)] \bmod 11 = 10 \text{ σύγκρουση. Δοκιμάζουμε για } i=3$$

$$h(59,3) = [59 \bmod 11 + 3(5 + 59 \bmod 10)] \bmod 11 = 2 \text{ σύγκρουση. Δοκιμάζουμε για } i=4$$

$h(59,4) = [59 \text{ mod } 11 + 3(5+50 \text{ mod } 10)] \text{ mod } 11 = 5$

$h(78,0) = [78 \text{ mod } 11] \text{ mod } 11 = 1$

$h(45,0) = [45 \text{ mod } 1] \text{ mod } 11 = 4$ σύγκρουση. Δοκιμάζουμε για $i=1$

.....

$h(8,0) = [8 \text{ mod } 11] \text{ mod } 11 = 8$. Το κλειδί αυτό δεν μπαίνει διότι ο πίνακας έχει 10 θέσεις και αυτό είναι το 11^ο κλειδί

Ο τελικός hash πίνακας με τις τιμές είναι ο ακόλουθος:

0	22
1	78
2	88
3	15
4	4
5	59
6	28
7	17
8	45
9	31
10	10

Το κλειδί 8 δεν θα μπει στο hash πίνακα γιατί είναι η 12^η τιμή και δεν χωρά σε πίνακα με 11 θέσεις

Παρατήρηση

Υπάρχουν 3 περιπτώσεις στο double hashing

1) Ο αριθμός κλειδιών = αριθμός θέσεων του hash πίνακα

Στη περίπτωση αυτή ΟΛΑ τα κλειδιά θα τοποθετηθούν στο hash πίνακα όσες συγκρούσεις και αν συμβούν

2) Ο αριθμός κλειδιών < αριθμός θέσεων του hash πίνακα

Στη περίπτωση αυτή ΟΛΑ τα κλειδιά θα τοποθετηθούν στο hash πίνακα και θα μείνουν κενές θέσεις στον πίνακα κατακερματισμού

3) Ο αριθμός κλειδιών > αριθμός θέσεων του hash πίνακα

Στη περίπτωση αυτή ΔΕΝ ΘΑ ΜΠΟΥΝ ΟΛΑ τα κλειδιά στο hash πίνακα, κάποια θα μείνουν εκτός (αυτά που δεν χωρούν στον πίνακα κατακερματισμού)

11.5 Θέμα 3 Σεπτέμβριος 2020

Έστω ότι θέλουμε να ενθέσουμε τα κλειδιά 5, 16, 21, 31, 17, 8, 50 σε ένα πίνακα μήκους $m=10$ με τη σειρά που δίνονται χρησιμοποιώντας **hashing με αλυσίδες** και **hashing με open addressing**. Και στις δύο μεθόδους η βασική hash συνάρτηση είναι $h_1(k)=k \bmod m$. Στο hashing με open addressing έστω ότι $h_2(k)=(k+4) \bmod m$ (**μέθοδος με double hashing**). Δώστε σχηματικά τους hashing πίνακες.

Απάντηση**Hashing με Αλυσίδες**

$$h_1(k)=k \bmod m=k \bmod 10$$

$$h_1(5)=5 \bmod 10=5$$

$$h_1(16)=16 \bmod 10=6$$

$$h_1(24)=24 \bmod 10=4$$

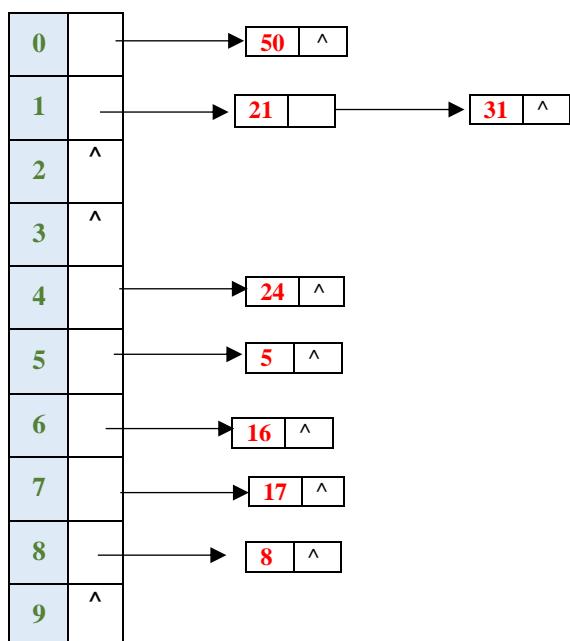
$$h_1(21)=21 \bmod 10=1$$

$$h_1(31)=31 \bmod 10=1$$

$$h_1(17)=17 \bmod 10=7$$

$$h_1(8)=8 \bmod 10=8$$

$$h_1(50)=50 \bmod 10=0$$



$h(k, i) = [h_1(k) + i \cdot h_2(k, i)] \bmod m$ για $i=0,1,\dots,m-1$ όπου m το μέγεθος πίνακα κατακερματισμού

$h(k, i) = [k \bmod 10 + i \cdot ((k+4) \bmod 10)] \bmod 10$ για $i=0,1,\dots,9$

$h(5, 0) = [5 \bmod 10 + 0 \times ((5+4) \bmod 10)] \bmod 10 = 5 \bmod 10 = 5$

$h(16, 0) = [16 \bmod 10 + 0 \times ((16+4) \bmod 10)] \bmod 10 = 6 \bmod 10 = 6$

$h(24, 0) = 24 \bmod 10 + 0 \times ((24+4) \bmod 10)] \bmod 10 = 4 \bmod 10 = 4$

$h(21, 0) = [21 \bmod 10 + 0 \times ((21+4) \bmod 10)] \bmod 10 = 1 \bmod 10 = 1$

$h(31, 0) = [31 \bmod 10 + 0 \times ((31+4) \bmod 10)] \bmod 10 = 1 \bmod 10 = 1$ σύγκρουση διότι η θέση 1 είναι κατειλημμένη με το κλειδί 21.

Άρα ανξάνουμε το i σε 1

$h(31, 1) = [31 \bmod 10 + 1 \times ((31+4) \bmod 10)] \bmod 10 = [1+5] \bmod 10 = 6$ σύγκρουση διότι η θέση 6 είναι κατειλημμένη με το κλειδί

16. Άρα ανξάνουμε το i σε 2

$h(31, 2) = [31 \bmod 10 + 2 \times ((31+4) \bmod 10)] \bmod 10 = [1+10] \bmod 10 = 1$ σύγκρουση διότι η θέση 1 είναι κατειλημμένη με το κλειδί

21. Άρα ανξάνουμε το i σε 3

$h(31, 3) = [31 \bmod 10 + 3 \times ((31+4) \bmod 10)] \bmod 10 = [1+15] \bmod 10 = 6$ σύγκρουση διότι η θέση 6 είναι κατειλημμένη με το κλειδί

16. Άρα ανξάνουμε το i σε 4

$h(31, 4) = [31 \bmod 10 + 4 \times ((31+4) \bmod 10)] \bmod 10 = [1+20] \bmod 10 = 1$ σύγκρουση διότι η θέση 1 είναι κατειλημμένη με το κλειδί

21. Άρα ανξάνουμε το i σε 5

$h(31, 5) = [31 \bmod 10 + 5 \times ((31+4) \bmod 10)] \bmod 10 = [1+25] \bmod 10 = 6$ σύγκρουση διότι η θέση 6 είναι κατειλημμένη με το κλειδί

16. Άρα ανξάνουμε το i σε 6

$h(31, 6) = [31 \bmod 10 + 6 \times ((31+4) \bmod 10)] \bmod 10 = [1+30] \bmod 10 = 1$ σύγκρουση διότι η θέση 1 είναι κατειλημμένη με το κλειδί

21. Άρα ανξάνουμε το i σε 7

$h(31, 7) = [31 \bmod 10 + 7 \times ((31+4) \bmod 10)] \bmod 10 = [1+35] \bmod 10 = 6$ σύγκρουση διότι η θέση 6 είναι κατειλημμένη με το κλειδί

16. Άρα ανξάνουμε το i σε 8

$h(31, 8) = [31 \bmod 10 + 8 \times ((31+4) \bmod 10)] \bmod 10 = [1+40] \bmod 10 = 1$ σύγκρουση διότι η θέση 1 είναι κατειλημμένη με το κλειδί

21. Άρα ανξάνουμε το i σε 9

$h(31, 9) = [31 \bmod 10 + 9 \times ((31+4) \bmod 10)] \bmod 10 = [1+45] \bmod 10 = 6$ σύγκρουση διότι η θέση 6 είναι κατειλημμένη με το κλειδί 16

Ολοκληρώνονται όλες οι προσπάθειες από $i=0\dots 9$ και το στοιχείο 31ΔΕΝ μπαίνει στον πίνακα κατακερματισμού διότι προκύπτουν διαρκείς συγκρούσεις. Άρα το 31 μένει εκτός πίνακα

$$h(17, 0) = [17 \bmod 10 + 0 \times ((17+4) \bmod 10)] \bmod 10 = 7 \bmod 10 = 7$$

$$h(8, 0) = [8 \bmod 10 + 0 \times ((8+4) \bmod 10)] \bmod 10 = 8 \bmod 10 = 8$$

$$h(50, 0) = [50 \bmod 10 + 0 \times ((50+4) \bmod 10)] \bmod 10 = 0 \bmod 10 = 0$$

Ο πίνακας Κατακερματισμού είναι ο ακόλουθος:

0	50
1	21
2	
3	
4	24
5	5
6	16
7	17
8	8
9	

Παρατήρηση

Ο πίνακας κατακερματισμού έχει 10 θέσεις και δίνονται 8 κλειδιά. Άρα εξορισμού 2 θέσεις θα μείνουν άδειες. Επιπλέον το 31 δεν τοποθετήθηκε στον πίνακα κατακερματισμού διότι όλες οι προσπάθειες κατέληξαν σε σύγκρουση οπότε υπάρχει ακόμα 1 κενή θέση.

11.6 Θέμα 3 Σεπτέμβριος 2020 – Ομάδα Β

Έστω ότι θέλουμε να ενθέσουμε τα κλειδιά 1,12, 23, 15, 35, 8, 57 σε ένα πίνακα μήκους $m=10$ με τη σειρά που δίνονται χρησιμοποιώντας **hashing με αλυσίδες** και **hashing με open addressing**. Και στις δύο μεθόδους η βασική hash συνάρτηση είναι $h_1(k)=k \bmod m$. Στο hashing με open addressing έστω η $h_2(k)=(k+3) \bmod m$ (**μέθοδος με double hashing**). Δώστε σχηματικά τους hashing πίνακες,

Απάντηση

Hashing με Αλυσίδες

$$h_1(k)=k \bmod m=k \bmod 10$$

$$h_1(1)=1 \bmod 10=1$$

$$h_1(12)=12 \bmod 10=2$$

$$h_1(23)=23 \bmod 10=3$$

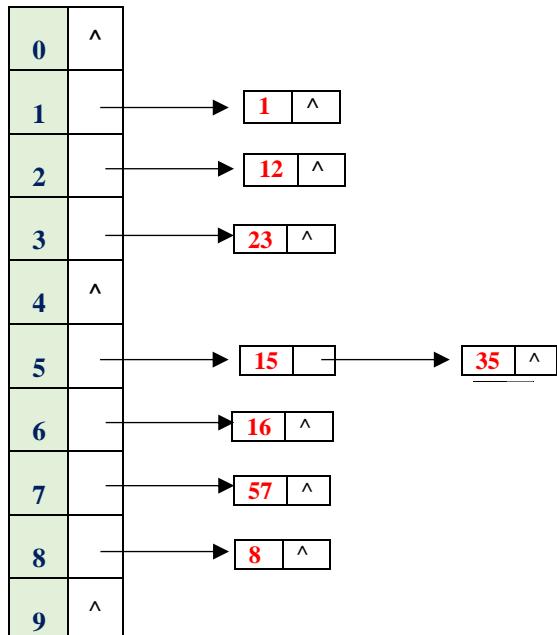
$$h_1(15)=15 \bmod 10=5$$

$$h_1(35)=35 \bmod 10=5$$

$$h_1(15)=15 \bmod 10=5$$

$$h_1(8)=8 \bmod 10=8$$

$$h_1(57)=57 \bmod 10=7$$



Δομές Δεδομένων –Computer Ανάλυση
Hashing με open addressing

$h(k, i) = [h_1(k) + i \cdot h_2(k)] \bmod m$ για $i=0,1,\dots,m-1$ όπου m το μέγεθος πίνακα κατακερματισμού

$$h(k, i) = [k \bmod m + i \cdot ((k+3) \bmod 10)] \bmod m$$

$$h(k, i) = [k \bmod 10 + i \cdot ((k+3) \bmod 10)] \bmod 10 \text{ για } i=0,1,\dots,9$$

$$h(1, 0) = [1 \bmod 10] \bmod 10 = 1$$

$$h(12, 0) = [12 \bmod 10] \bmod 10 = 2$$

$$h(23, 0) = [23 \bmod 10] \bmod 10 = 3$$

$$h(15, 0) = [15 \bmod 10] \bmod 10 = 5$$

$$h(35, 0) = [35 \bmod 10] \bmod 10 = 5 \text{ σύγκρουση}$$

$$h(35, 1) = [35 \bmod 10 + 1 \cdot ((35+3) \bmod 10)] \bmod 10 = (5+1 \cdot 8) \bmod 10 = 3 \text{ σύγκρουση}$$

$$h(35, 2) = [35 \bmod 10 + 2 \cdot ((35+3) \bmod 10)] \bmod 10 = (5+2 \cdot 8) \bmod 10 = 1 \text{ σύγκρουση}$$

$$h(35, 3) = [35 \bmod 10 + 3 \cdot ((35+3) \bmod 10)] \bmod 10 = (5+3 \cdot 8) \bmod 10 = 9$$

$$h(8, 0) = [8 \bmod 10] \bmod 10 = 8$$

$$h(57, 0) = [57 \bmod 10] \bmod 10 = 7$$

To 31 προκαλεί μόνο συγκρούσεις και γιαντό δεν τοποθετείται στον πίνακα κατακερματισμού.

0	
1	1
2	12
3	23
4	—
5	15
6	—
7	57
8	8
9	35

11.7 Θέμα 7 Σεπτέμβριος 2018 σε Extensible Hashing

Σας δίνεται το σύνολο $S = \{2, 3, 5, 11, 17, 19, 23, 29, 31\}$. Δώστε τη σχηματική οργάνωση του πίνακα hash βήμα προς βήμα χρησιμοποιώντας **extensible hashing με hash function $h(x) = x \bmod 8$ και μέγεθος bucket=3**. Πως διαμορφώνεται ο hash πίνακας με την εκτέλεση των ακόλουθων πράξεων:

a) delete 11

β) delete 31

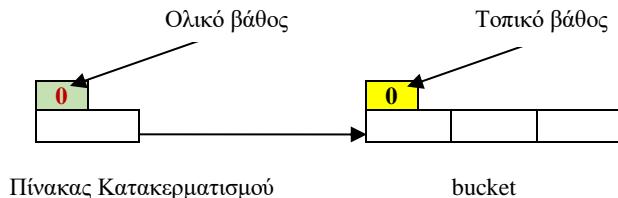
γ) insert 1

δ) insert 15

Πότε χρησιμοποιούμε τη **μέθοδο Extensible hashing; Τι είναι το ολικό και το τοπικό βάθος**

Απάντηση

Αρχικά



Εισαγωγή 2, 3, 5



Εισαγωγή 11

Το bucket στο οποίο πρέπει να μπει το κλειδί 11 είναι γεμάτο και δεν υπάρχει χώρος σε αυτό.

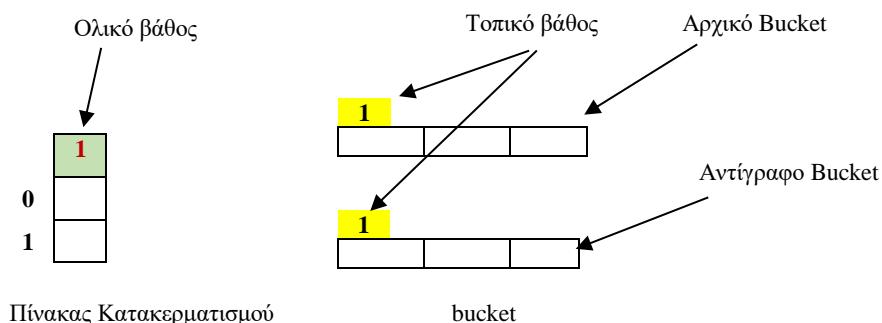
Γιαυτό εκτελούμε κατά σειρά τις ακόλουθες ενέργειες:

α) διπλασιάζεται ο hash πίνακας και αυξάνεται το ολικό βάθος του κατά 1

β) διπλασιάζεται το bucket που ήταν γεμάτο και το τοπικό βάθος του αυξάνεται κατά 1

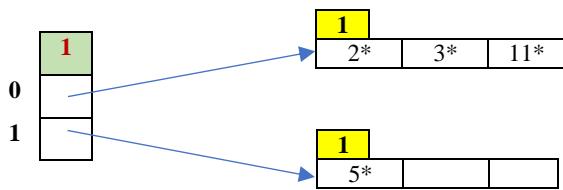
γ) Το βάθος του αντίγραφου bucket που προκύπτει από το διπλασιασμό είναι ίδιο με το βάθος του bucket που διπλασιάστηκε

δ) Τα κλειδιά επανατοποθετούνται στα buckets από την αρχή



Τα κλειδιά επανατοποθετούνται στα bucket από την αρχή εφαρμόζοντας τη συνάρτηση κατακερματισμού που δίνεται

- Εφαρμόζουμε τώρα τη hash function $h(x)=x \bmod 8$ που δίνεται για να δούμε σε ποιο bucket θα μπει κάθε κλειδί
- Βάζουμε τα κλειδιά στο κατάλληλο bucket **σύμφωνα με το 1^o bit από αριστερά της δυαδικής αναπαράστασης** του υπολοίπου
- $h(2)=2 \bmod 8=2=\underline{0}10$
- $h(3)=3 \bmod 8=3=\underline{0}11$
- $h(5)=5 \bmod 8=5=\underline{1}01$
- $h(11)=11 \bmod 8=3=\underline{0}11$



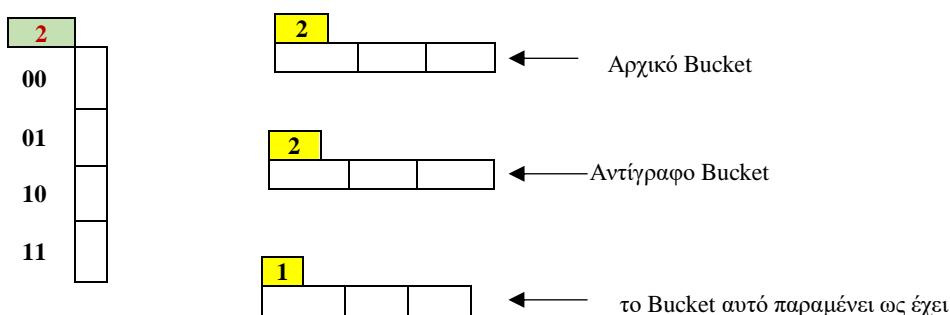
Εισαγωγή 17

- $h(17)=17 \bmod 8=1=\underline{0}01$

Το κλειδί 17 θα πρέπει να μπει στο bucket στο οποίο δείχνει η θέση 0 του hash πίνακα. Το bucket αυτό είναι γεμάτο και δεν χωράει.

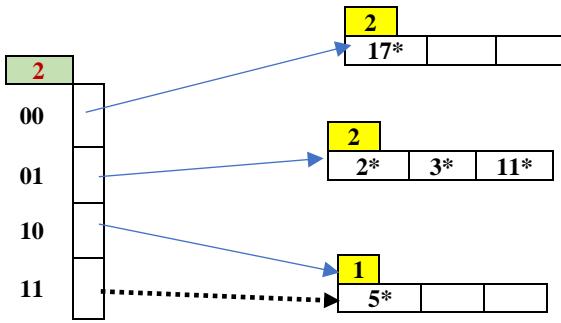
Γιαυτό εκτελούμε κατά σειρά τις ακόλουθες ενέργειες:

- α) διπλασιάζεται ο hash πίνακας και αυξάνεται το ολικό βάθος του κατά 1
- β) διπλασιάζεται το bucket που ήταν γεμάτο και το τοπικό βάθος του αυξάνεται κατά 1
- γ) Το βάθος του αντίγραφου bucket που προκύπτει από το διπλασιασμό είναι ίδιο με το βάθος του bucket που διπλασιάστηκε
- δ) Τα κλειδιά επανατοποθετούνται στα buckets από την αρχή



- $h(2)=2 \bmod 8=2=\underline{0}10$
- $h(3)=3 \bmod 8=3=\underline{0}11$
- $h(5)=5 \bmod 8=5=\underline{1}01$
- $h(11)=11 \bmod 8=3=\underline{0}11$

- $h(17) = 17 \bmod 8 = 1 = \underline{001}$



Παρατίρηση

Η θέση 11 είναι "φιλαράκι" με τη θέση 10 διότι έχει 1 κοινό bit από αριστερά («φιλαράκια» είναι οι θέσεις του hash πίνακα που έχουν τα περισσότερα κοινά bits από αριστερά). Βάζουμε το δείκτη της θέσης 11 να δείχνει στο ίδιο bucket με το δείκτη της θέσης 10 γιατί **δεν πρέπει να υπάρχουν κενές θέσεις στο hash πίνακα.**

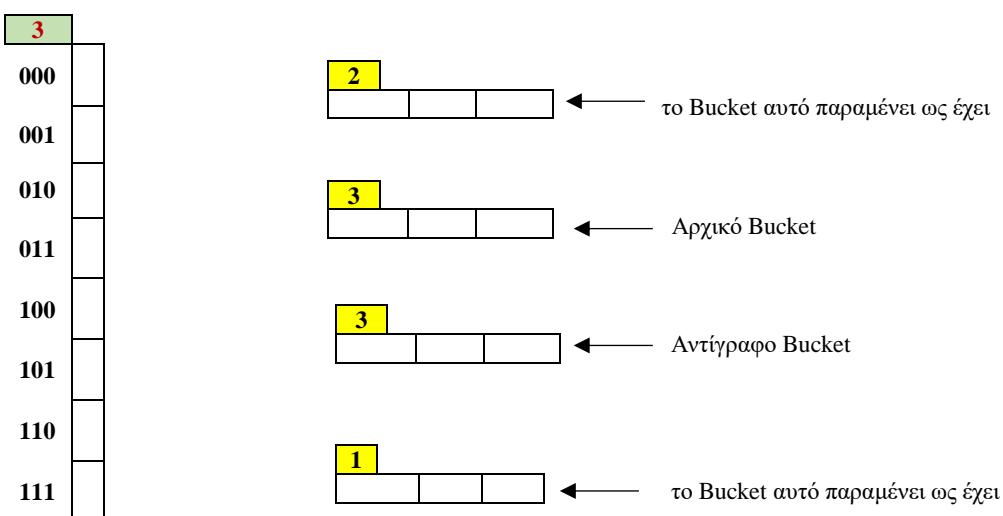
Εισαγωγή 19

- $h(19) = 19 \bmod 8 = 3 = \underline{011}$

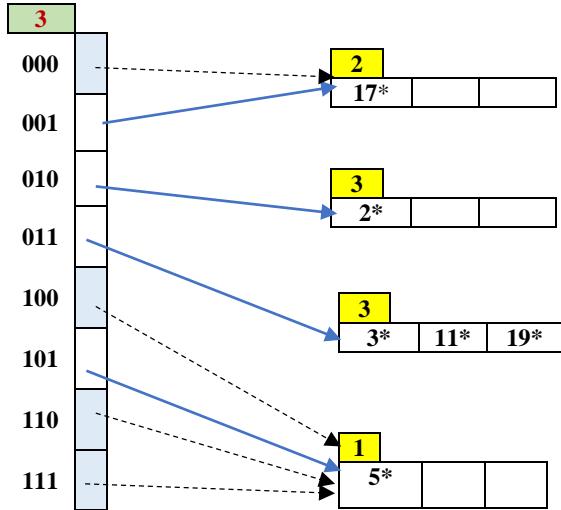
Το κλειδί 17 θα πρέπει να μπει στο bucket στο οποίο δείχνει η θέση 01 του hash πίνακα. Το bucket αυτό είναι γεμάτο και δεν χωράει.

Γιαυτό εκτελούμε κατά σειρά τις ακόλουθες ενέργειες:

- α) διπλασιάζεται ο hash πίνακας και αυξάνεται το ολικό βάθος του κατά 1
- β) διπλασιάζεται το bucket που ήταν γεμάτο και το τοπικό βάθος του αυξάνεται κατά 1
- γ) Το βάθος του αντίγραφου bucket που προκύπτει από το διπλασιασμό είναι ίδιο με το βάθος του bucket που διπλασιάστηκε
- δ) Τα κλειδιά επανατοποθετούνται στα buckets από την αρχή



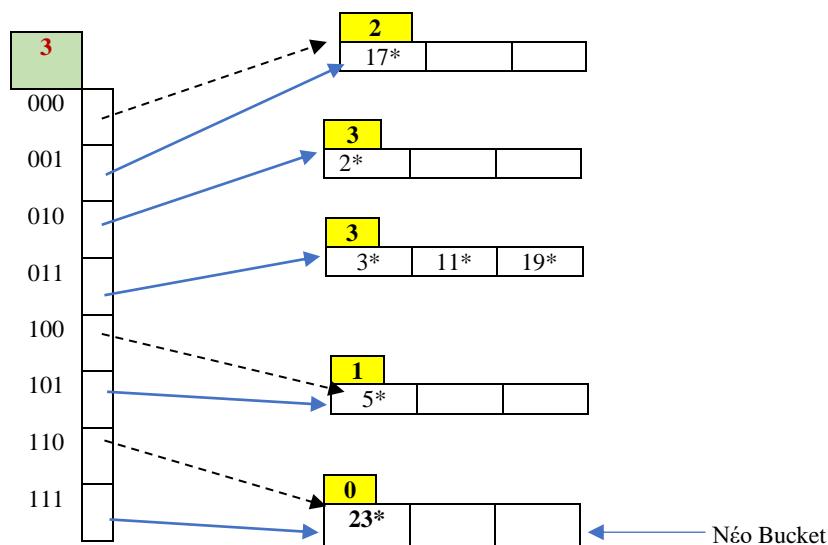
- $h(2) = 2 \bmod 8 = 2 = \underline{\text{010}}$
- $h(3) = 3 \bmod 8 = 3 = \underline{\text{011}}$
- $h(5) = 5 \bmod 8 = 5 = \underline{\text{101}}$
- $h(11) = 11 \bmod 8 = 3 = \underline{\text{011}}$
- $h(17) = 17 \bmod 8 = 1 = \underline{\text{001}}$
- $h(19) = 19 \bmod 8 = 3 = \underline{\text{011}}$



Εισαγωγή 23

- $h(23) = 23 \bmod 8 = 7 = \underline{\text{111}}$

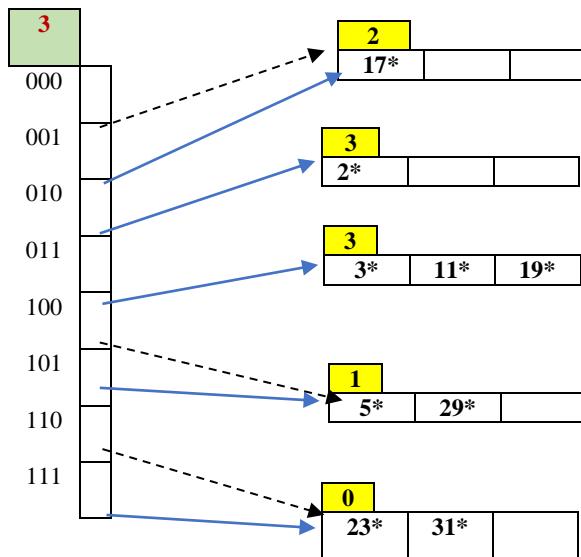
Παρατήρηση: Η τιμή 23 αφού εφαρμόσουμε τη hash function δίνει υπόλοιπο 7 που στη δυαδική αναπαράσταση είναι 111. Το κλειδί δεν θα μπει σε γεμάτο bucket άρα ο hash πίνακας δεν θα διπλασιαστεί. Αντίθετα θα δημιουργηθεί νέο bucket για την τοποθέτηση του 23 και θα βάλουμε το δείκτη 111 του hash πίνακα να δείχνει στο νέο bucket.



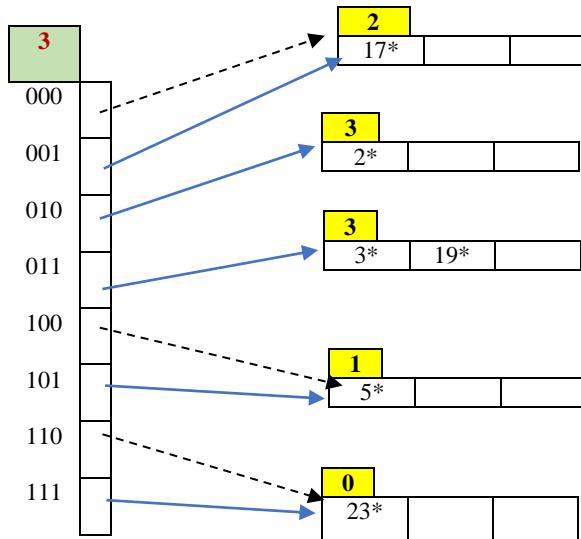
Εισαγωγή 29 και 31

$h(29) = 29 \bmod 8 = 5 = \textcolor{red}{101}$

$h(31) = 31 \bmod 8 = 7 = \textcolor{red}{111}$



Διαγραφή 11 και 31



Παρατήρηση

- Η συνθήκη για να υποδιπλασιάσουμε το hash πίνακα είναι: **Max(τοπικό βάθος όλων των bucket) < Ολικό Βάθος**
- Αν κάνουμε υποδιπλασιασμό τότε το ολικό βάθος μειώνεται κατά 1

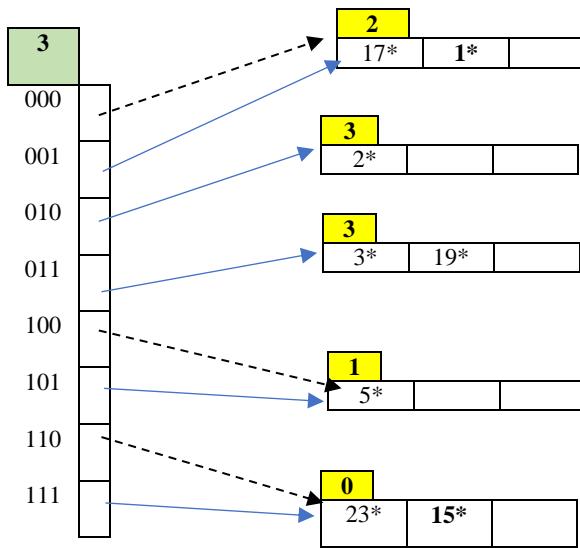
Παρατήρηση

Αν διαγράψουμε το 23 τότε το bucket που το περιέχει διαγράφεται και ο δείκτης 111 δείχνει στο φιλαράκι του που είναι το 110

Εισαγωγή 1 και 15

$h(1) = 1 \bmod 8 = 1 = \underline{\text{001}}$

$h(15) = 15 \bmod 8 = 7 = \underline{\text{111}}$



Η μέθοδος extensible hashing ενδείκνυται για την αποθήκευση στοιχείων σε δίσκο. Οι δίσκοι είναι αργές μονάδες αποθήκευσης.

Τα δεδομένα οργανώνονται σε σελίδες (pages), δηλαδή αποθηκεύονται σε ομάδες συνεχόμενων θέσεων αποθήκευσης, τα περιεχόμενα των οποίων μεταφέρονται στην κύρια μνήμη όταν ζητηθεί η ανάκτηση ή η τροποποίηση κάποιων εξ αυτών.

Ολικό βάθος $D(S)$ ενός συνόλου S καλείται το μικρότερο μήκος προθέματος των τιμών κατακερματισμού των στοιχείων του συνόλου S , που απαιτείται για να χωρισθούν τα στοιχεία αυτά σε ομάδες με b στοιχεία το πολύ σε κάθε μια, όπου όλα τα στοιχεία μιας ομάδας έχουν κοινό πρόθεμα μήκους $D(S)$ bits.

Για κάθε κάδο, το τοπικό βάθος είναι ο αριθμός των δυαδικών ψηφίων στα οποία βασίζεται η χρήση του κάδου

12 UNION FIND

12.1 Θεωρία για weighted-union rule

Στη μέθοδο weighted-union rule ενώνουμε 2 δέντρα σε ένα και τοποθετούμε το μικρότερο σε ύψος δέντρο ως παιδί του μεγαλύτερου σε ύψος δέντρο ώστε να αποφύγουμε τα μεγάλα βάθη δέντρων. Η ένωση των δύο δέντρων σε ένα γίνεται σύμφωνα με το ακόλουθο σχήμα:



Μια 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 $O(m + \log n)$

12.2 Θέμα 8 Σεπτέμβριος 2016 και Ιούνιος 2017

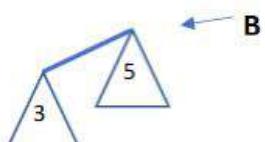
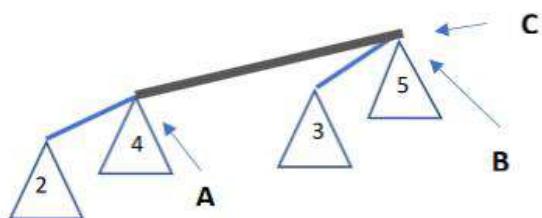
Εφαρμόστε το weighted union rule για τα παρακάτω **Unions** από 1 έως 7. Οι αριθμοί δηλώνουν τον αριθμό των στοιχείων (weight) κάθε συνόλου που λαμβάνει μέρος στο Union. Τα κεφαλαία γράμματα είναι ονόματα συνόλων.

1. $U(4,2,A)$
2. $U(3,5,B)$
3. $U(A,B,C)$
4. $U(6,9,D)$
5. $U(2,6,E)$
6. $U(D,C,F)$
7. $U(F,E,G)$

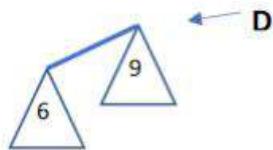
ΛΥΣΗ

Στο μάθημα θεωρήσαμε (**χωρίς να δουμε τι λεει η εκφώνηση**) ότι κάθε αριθμός είναι όνομα συνόλου και η λύση δόθηκε με την υπόθεση αυτή.

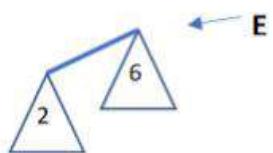
Αν θεωρήσουμε (**όπως λέει η εκφώνηση**) ότι κάθε αριθμός είναι ΠΛΗΘΟΣ ΣΤΟΙΧΕΙΩΝ ΤΟΤΕ Η ΣΩΣΤΗ ΛΥΣΗ ΕΙΝΑΙ Η ΑΚΟΛΟΥΘΗ (κάθε τρίγωνο αναπαριστά δεντρο-συνολο και ο αριθμός ειναι το πλήθος στοιχείων σε αυτό):

ΕΝΩΣΗ 1**ΕΝΩΣΗ 2****ΕΝΩΣΗ 3**

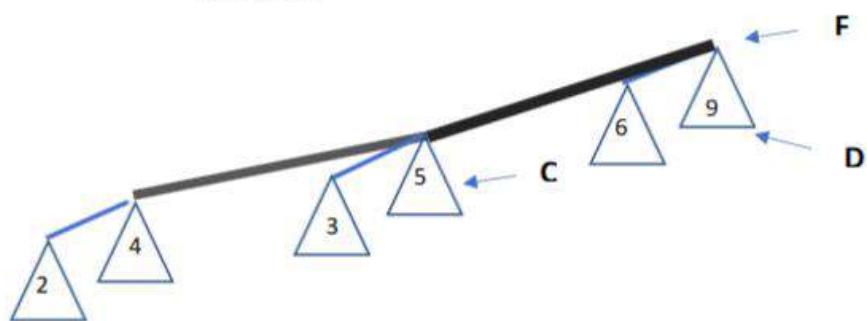
ΕΝΩΣΗ 4



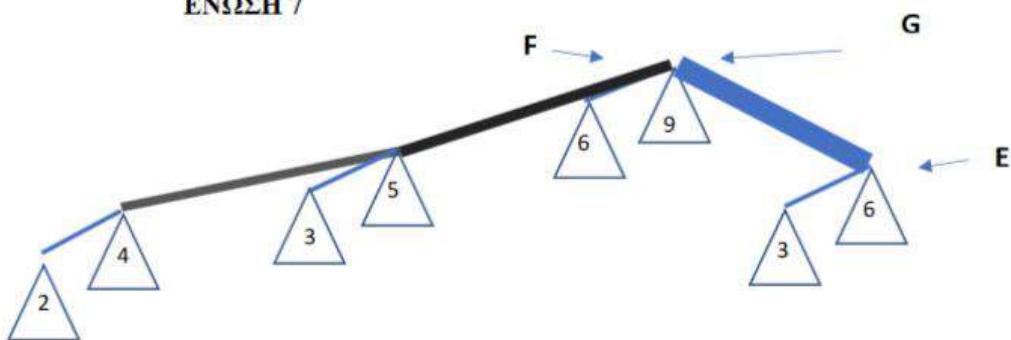
ΕΝΩΣΗ 5



ΕΝΩΣΗ 6



ΕΝΩΣΗ 7



12.3 Θέμα 8 Ιούνιος 2008, Θέμα 8 Ιούνιος 2009, Θέμα 3 Ιούνιος 2010, Θέμα 5 Ιούνιος 2011 και Θέμα 5 Ιούνιος 2015

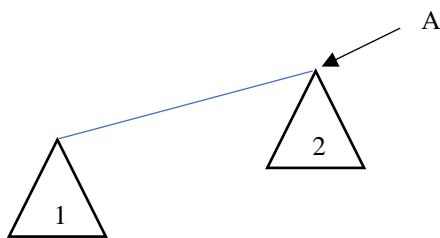
Εφαρμόστε το weighted union rule για τα παρακάτω Unions από 1 έως 7. Οι αριθμοί δηλώνουν τον αριθμό των στοιχείων (weight) κάθε συνόλου που λαμβάνει μέρος στο Union. Τα κεφαλαία γράμματα είναι ονόματα συνόλων.

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. $U(D, E, F)$
7. $U(C, F, G)$

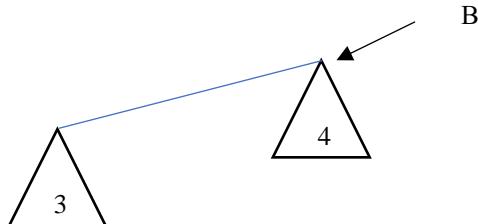
Απάντηση

Στη μέθοδο weighted-union rule βάζουμε το μικρότερο σε ύψος δέντρο ως παιδί του μεγαλύτερου σε ύψος δέντρο ώστε να αποφύγουμε τα μεγάλα ύψη

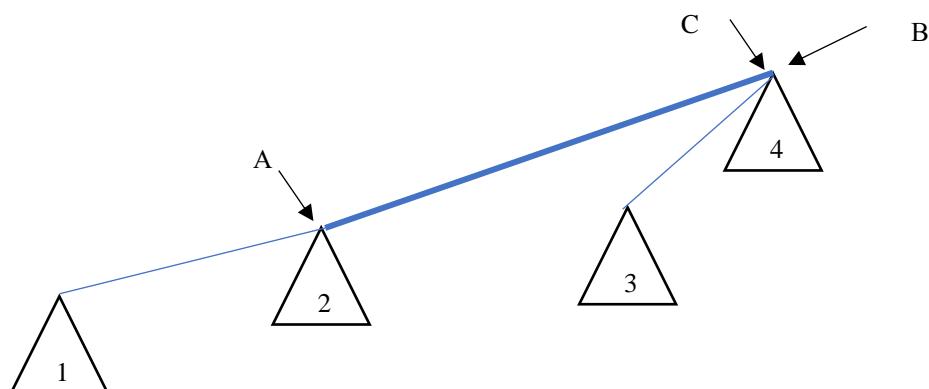
1. $U(1, 2, A)$



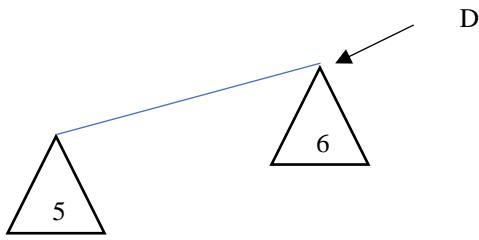
2. $U(3, 4, B)$



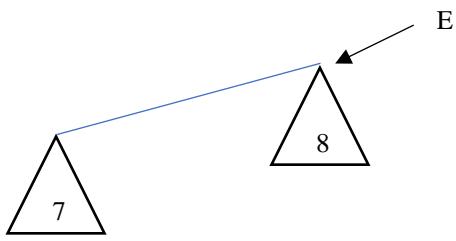
3. $U(A, B, C)$



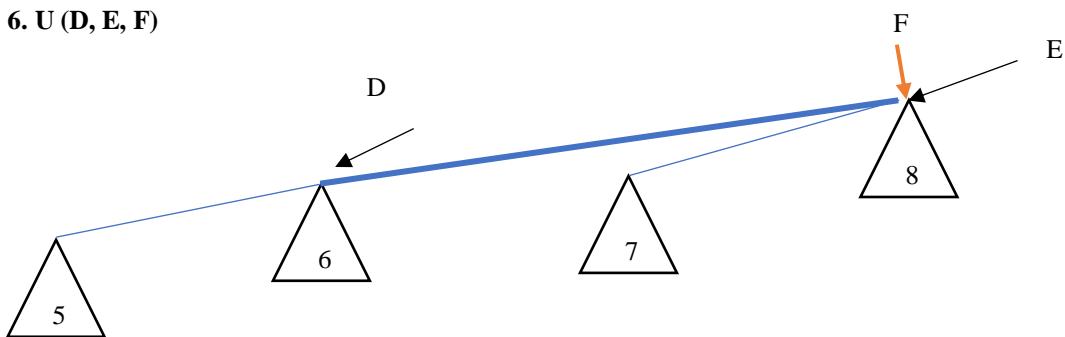
4. U (5, 6, D)



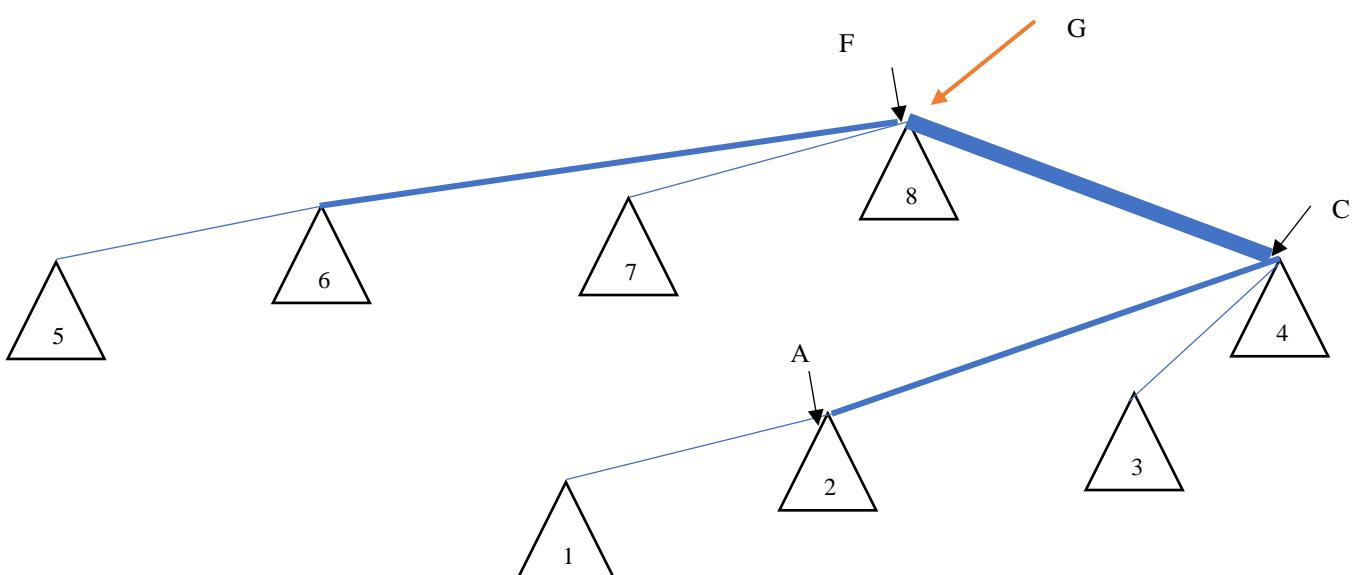
5. U (7, 8, E)



6. U (D, E, F)



7. U (C, F, G)



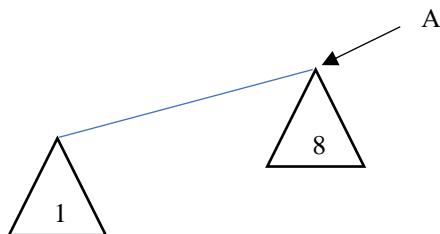
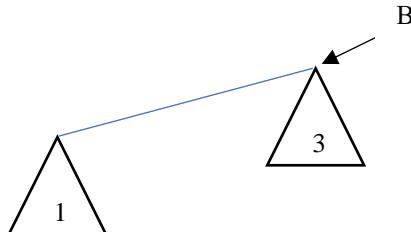
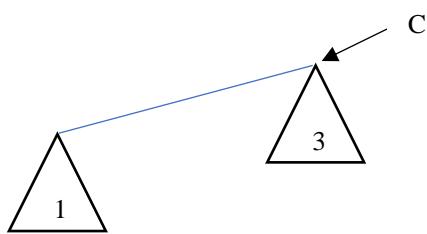
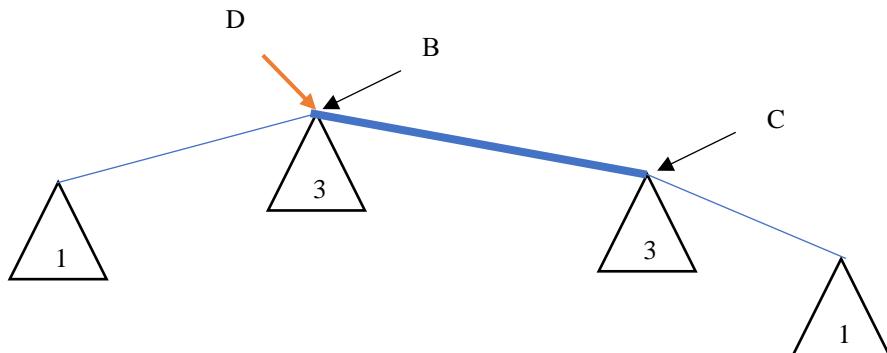
12.4 Θέμα 3 Σεπτέμβριος 2020

Εφαρμόστε το weighted union rule για τα παρακάτω Unions από 1 έως 7. Οι αριθμοί δηλώνουν το πλήθος των στοιχείων (weight) κάθε συνόλου που λαμβάνει μέρος στο Union. Τα κεφαλαία γράμματα είναι ονόματα συνόλων.

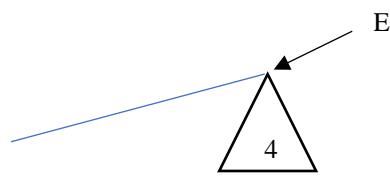
1. $U(1, 8, A)$
2. $U(3, 1, B)$
3. $U(1, 3, C)$
4. $U(B, C, D)$
5. $U(2, 4, E)$
6. $U(A, E, F)$
7. $U(D, F, G)$

Απάντηση

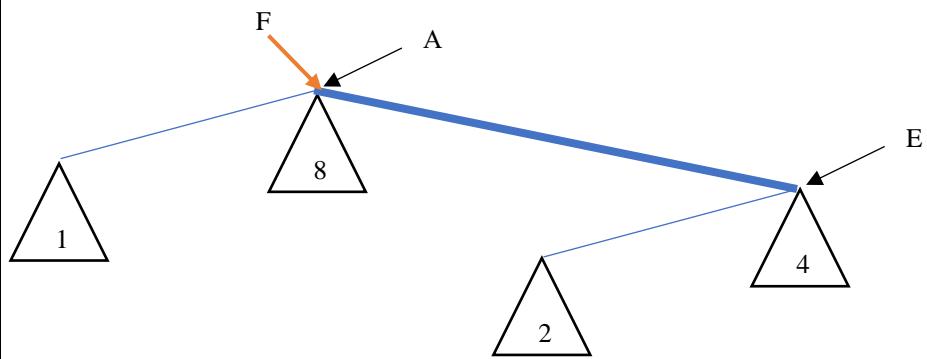
Στη μέθοδο weighted-union rule βάζουμε το μικρότερο σε ύψος δέντρο ως παιδί του μεγαλύτερου σε ύψος δέντρο ώστε να αποφύγουμε τα μεγάλα ύψη

1. $U(1, 8, A)$ **2. $U(3, 1, B)$** **3. $U(1, 3, C)$** **4. $U(B, C, D)$** 

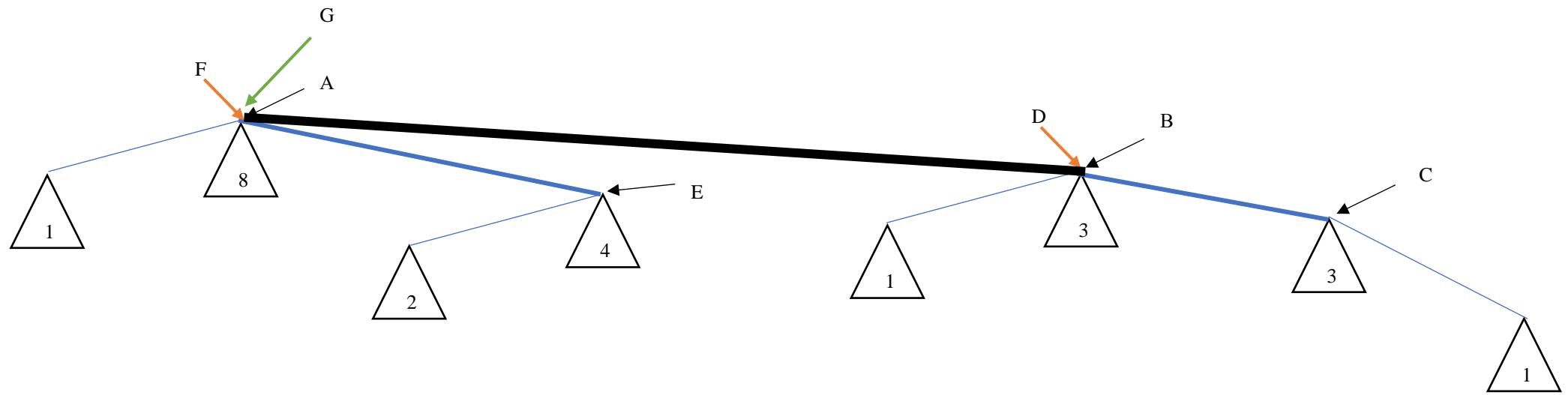
5. U(2, 4, E)



6. U(A, E, F)



7. U(D, F, G)



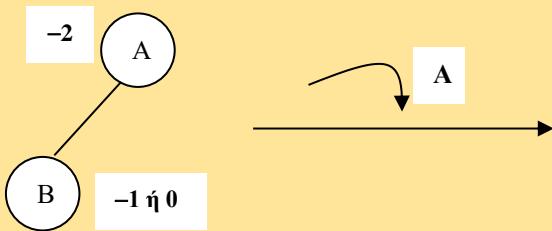
13 AVL ΔΕΝΤΡΑ

13.1 Χαρακτηριστικά AVL Δέντρου

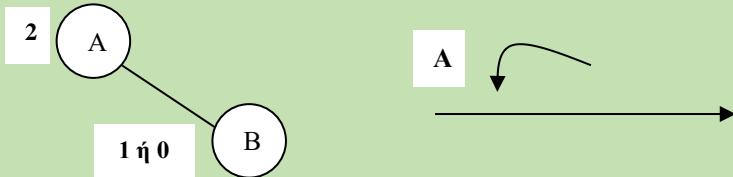
- **Δυαδικό** ισοζυγισμένο δέντρο. Σε κάθε κόμβο τα ύψη των υποδέντρων του διαφέρουν το πολύ κατά ένα
- **Υψοζύγιση ενός κόμβου $hb(u)=\text{Υψος Δεξιού Υποδέντρου (με ρίζα το } u\text{)} - \text{Υψος Αριστερού Υποδέντρου (με ρίζα το } u\text{)}$**
- **Επιτρεπτές Τιμές Υψοζύγισης $-1, 0, +1$**
- Το ύψος h ενός δέντρου με n στοιχεία είναι $O(\log n)$
- Είναι δέντρο αναζήτησης δηλ. **$\Delta\Pi \leq \text{Ρίζα} < \Delta\Pi$**
- **Οι πράξεις Access, Insert και Delete σε ένα AVL δέντρο με n φύλλα έχουν κόστος $\Theta(\log n)$ στη χειρότερη περίπτωση**

13.2 Διορθωτικές Πράξεις σε AVL για την επαναφορά της υψοζύγισης

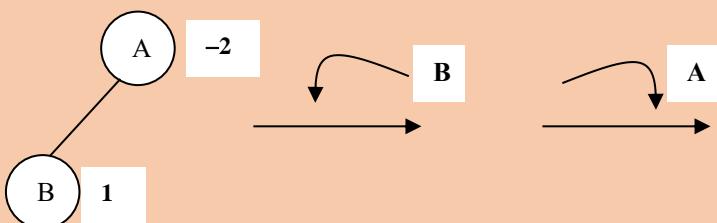
A Διόρθωση – Δεξιά Στροφή στον κόμβο A



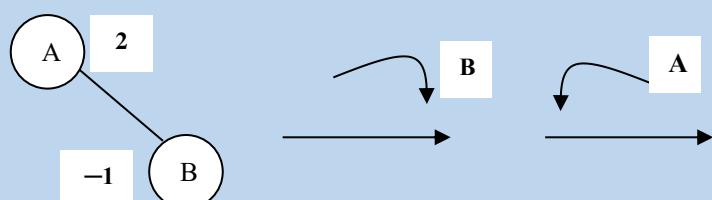
B Διόρθωση – Αριστερή Στροφή στον κόμβο A



Γ Διόρθωση – Αριστερή Στροφή στο B και μετά δεξιά στροφή στο A



Δ Διόρθωση – Δεξιά Στροφή στο B και μετά αριστερή στροφή στο A



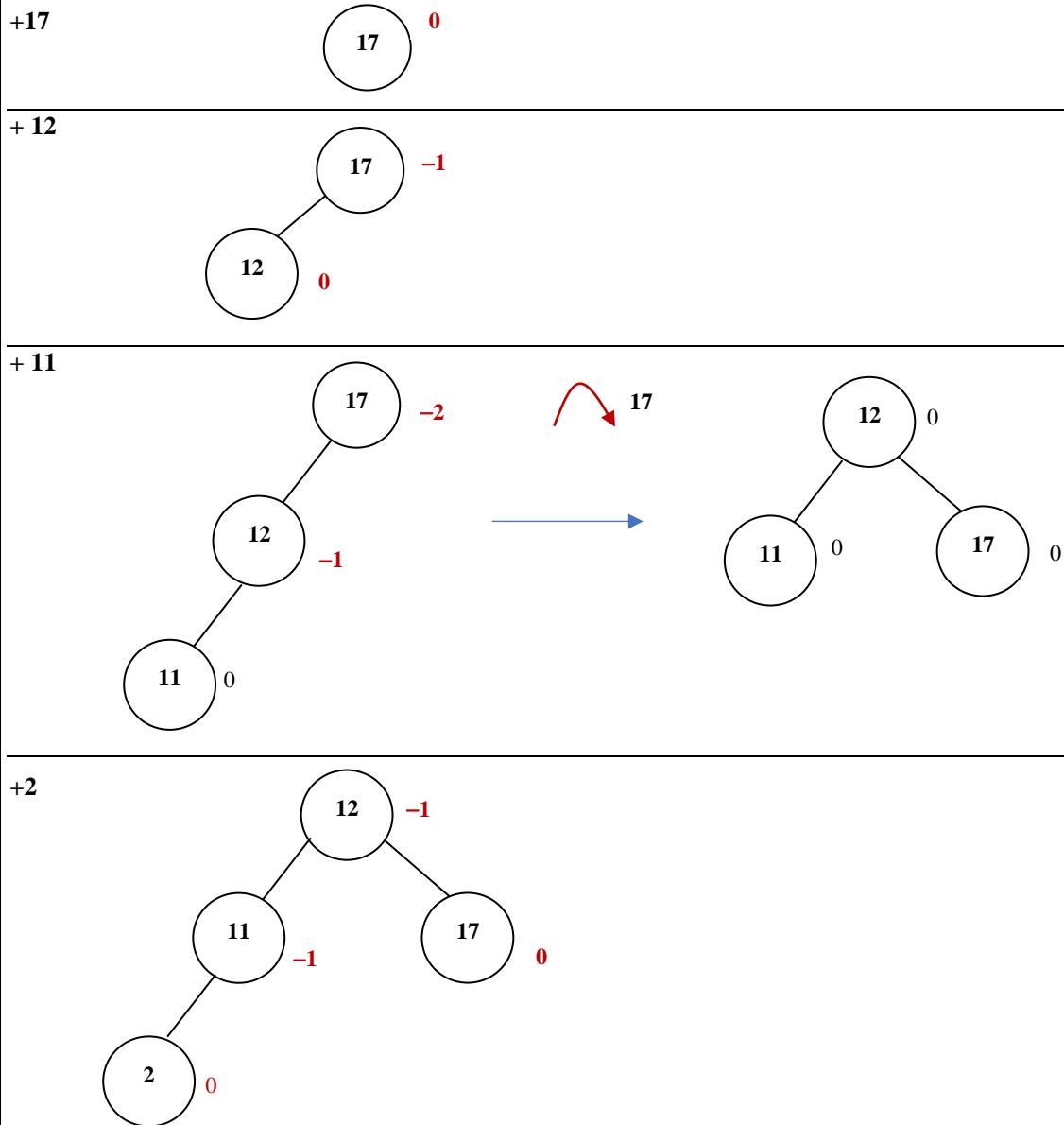
13.3 Θέμα 7 Ιούνιος 2023

Να σχεδιαστεί βήμα-βήμα το AVL δέντρο που προκύπτει από την εισαγωγή σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων:

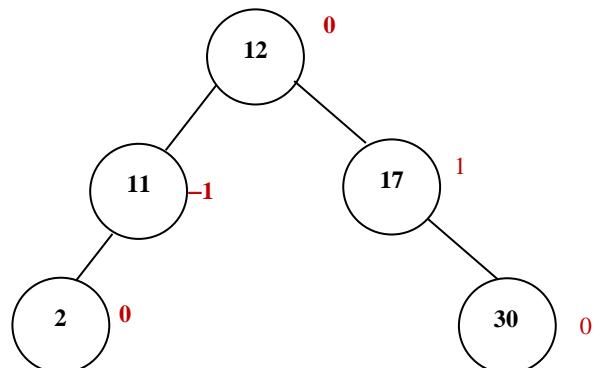
{17, 12, 11, 2, 30, 41, 20, 32, 19, 6}

Στη συνέχεια διαγράψτε βήμα-βήμα τα στοιχεία {11, 2}

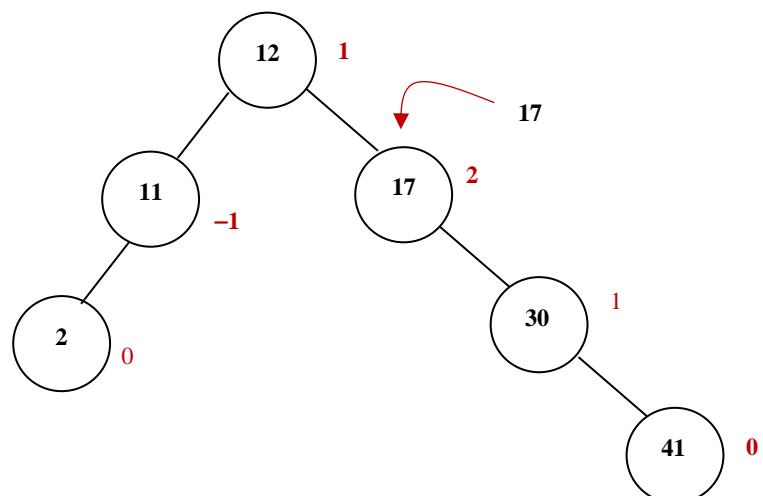
Απάντηση



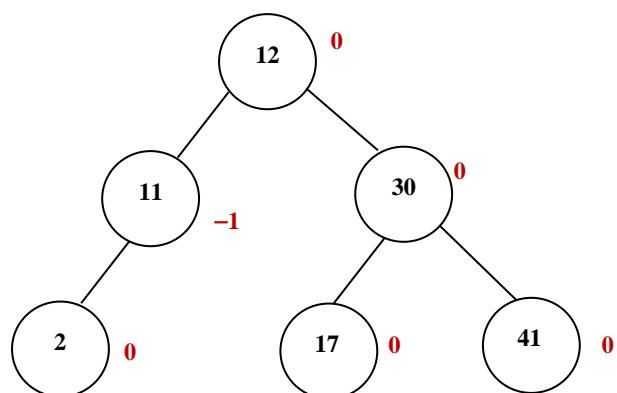
+30



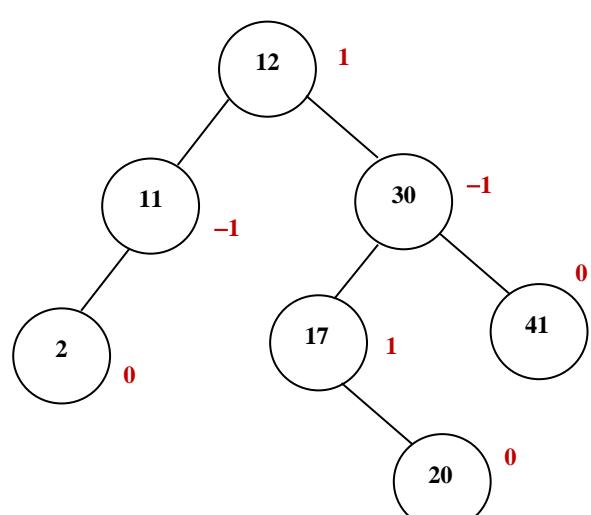
+41



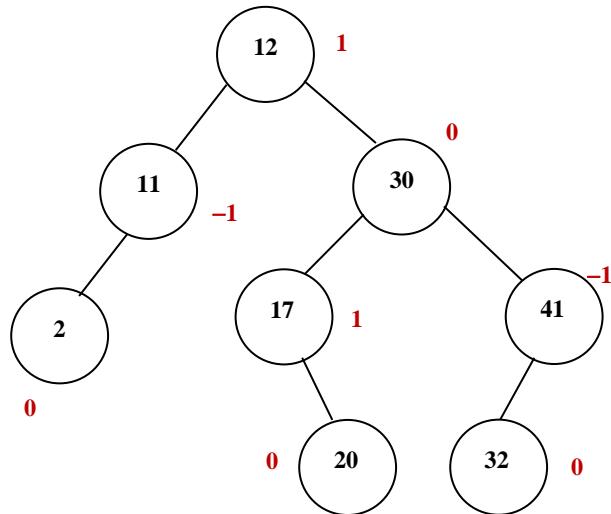
+41



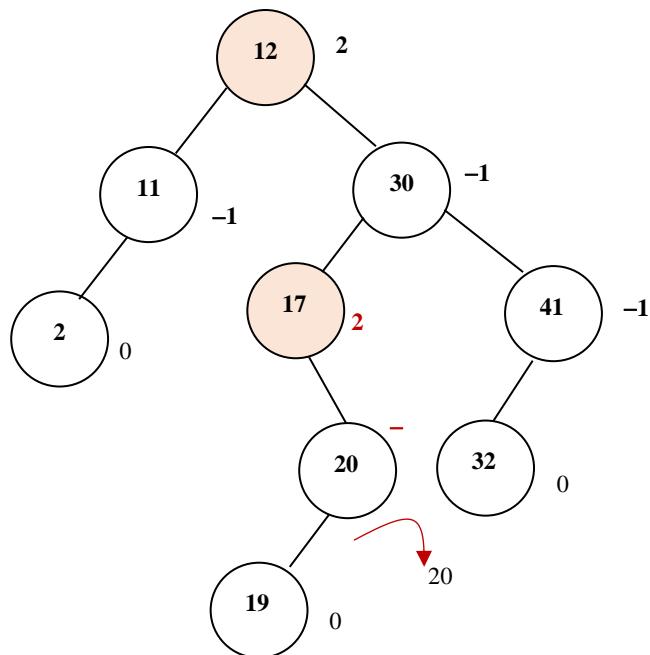
+20



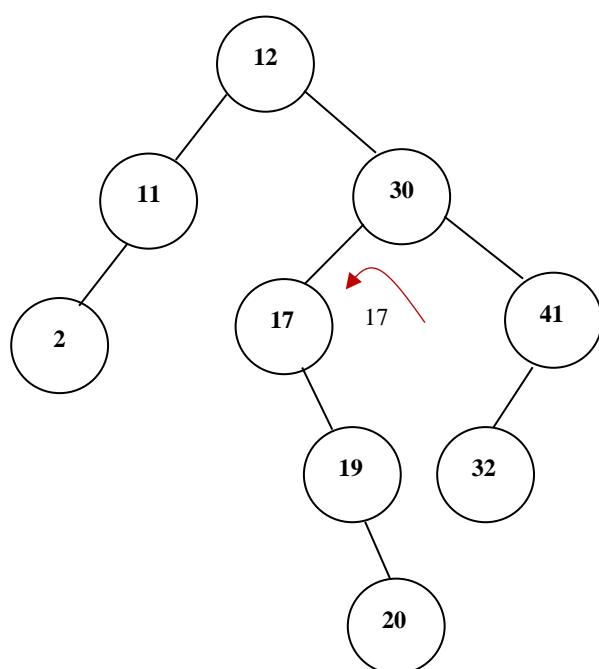
+32

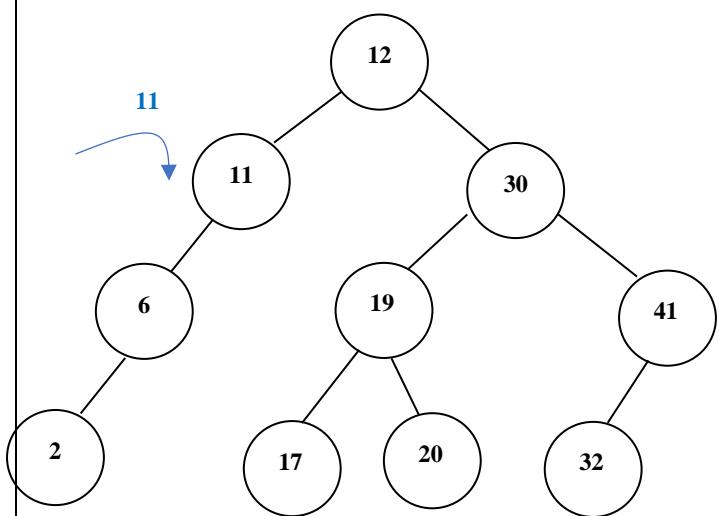
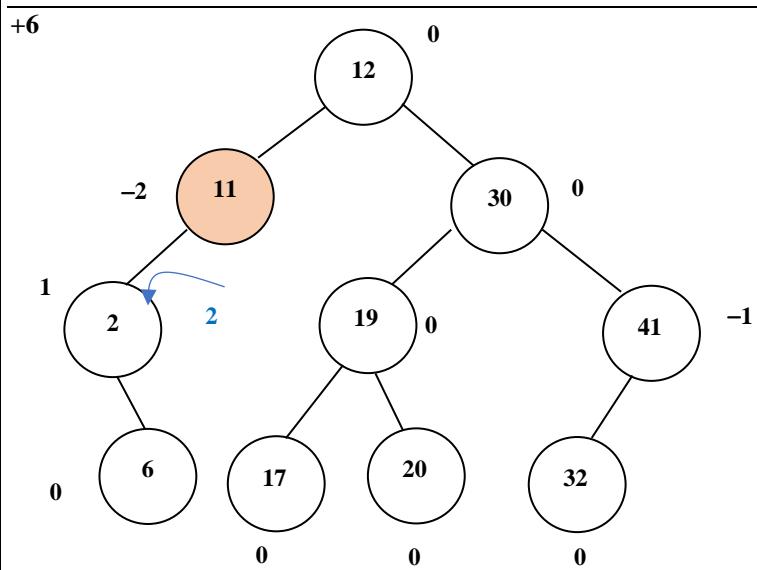
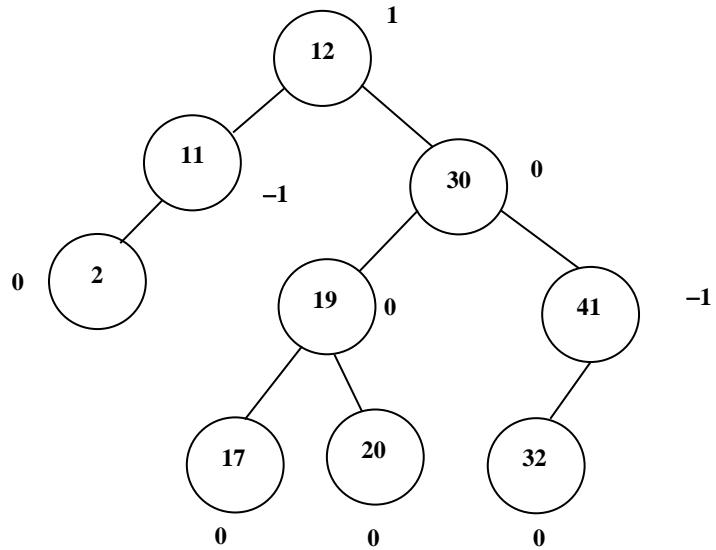


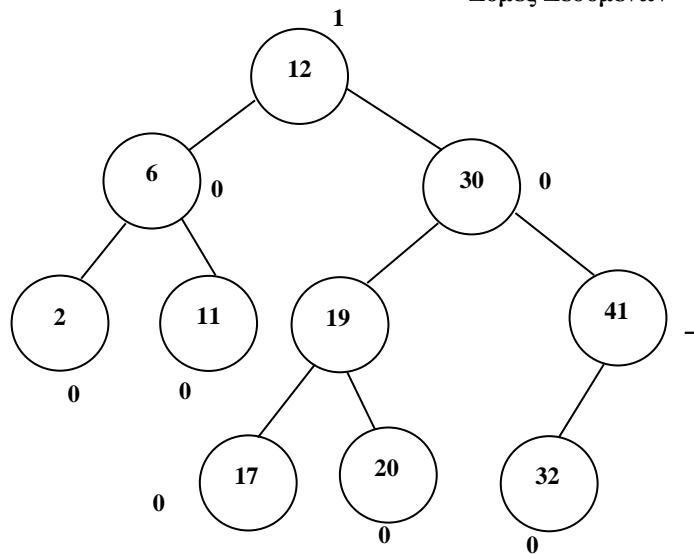
+19



Όταν παραβιάζεται η υψοζύγιση σε περισσότερους κόμβους πηγαίνουμε από κάτω προς τα πάνω και διορθώνουμε την υψοζύγιση στον 1^ο κόμβο που την παραβιάζει. Εδώ ο 1^{ος} κόμβος πάντα από κάτω προς τα πάνω που παραβιάζει την υψοζύγιση είναι το 17.



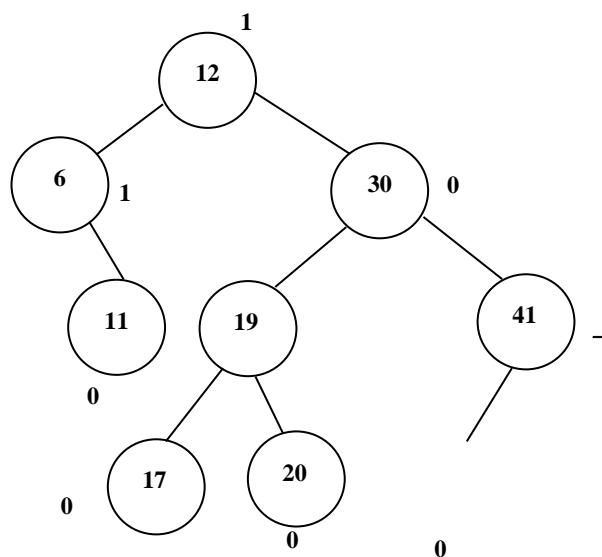




-2

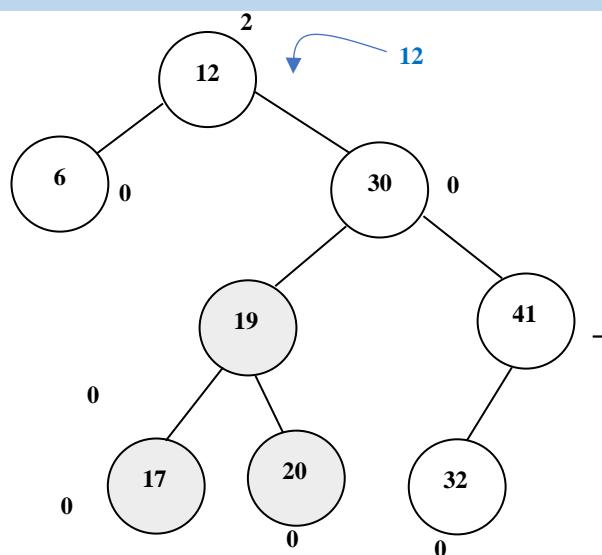
Όταν διαγράφουμε ΦΥΛΛΟ, απλά το διαγράφουμε και ελέγχουμε την υψοζύγιση. Εδώ απλά διαγράφουμε το φύλλο 2.

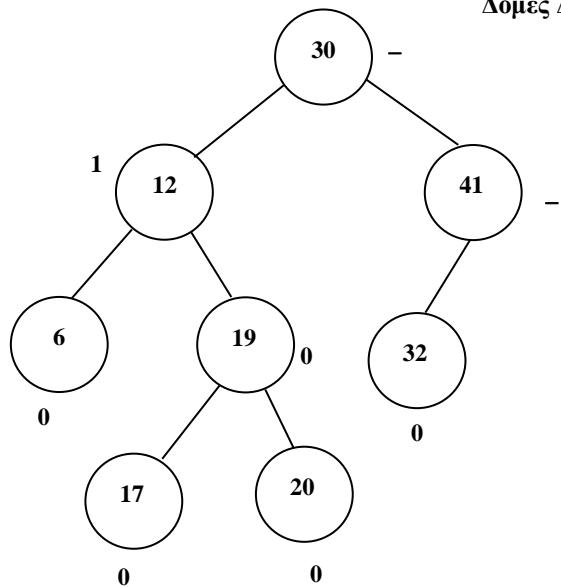
Όταν διαγράφουμε KOMBO, **ΒΑΖΟΥΜΕ ΣΤΗ ΘΕΣΗ ΤΟΥ ΤΗ ΜΕΓΑΛΥΤΕΡΗ ΤΙΜΗ ΤΟΥ ΑΡΙΣΤΕΡΟΥ ΥΠΟΔΕΝΤΡΟΥ ΤΟΥ**



-11

Όταν διαγράφουμε ΦΥΛΛΟ, απλά το διαγράφουμε και ελέγχουμε την υψοζύγιση. Εδώ απλά διαγράφουμε το φύλλο 11 και παραβιάζεται η υψοζύγιση του AVL δέντρου





Αυτό είναι το τελικό AVL Δέντρο

13.4 Θέμα 2 Φεβρουάριος και Ιούνιος 2017

Να σχεδιαστεί βήμα-βήμα το AVL δέντρο που προκύπτει από τις ακόλουθες πράξεις σε ένα αρχικά άδειο δέντρο με τη σειρά όπως δίνονται:

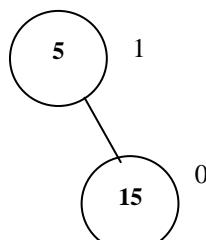
$\{+5, +15, +13, +11, +25, +14, +2, -15, +26, +28, +22\}$ όπου + εισαγωγή και - διαγραφή

Απάντηση

+5

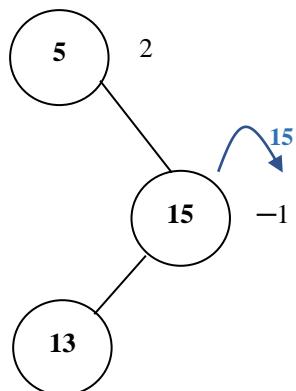


+15

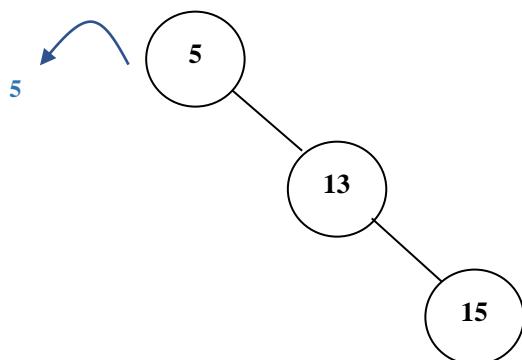


+13

$$\begin{aligned} 0-1 &= -1 \\ 2-0 &= 2 \end{aligned}$$

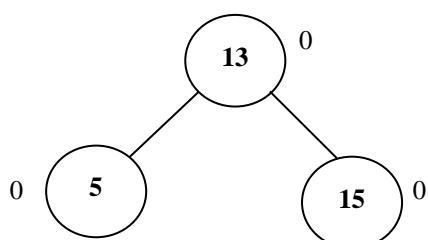


Δεξιά στροφή στο 15

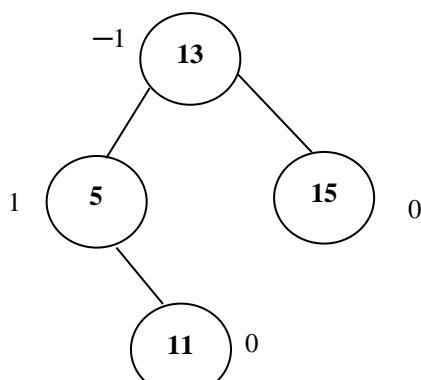


Αριστερή στροφή στο 5

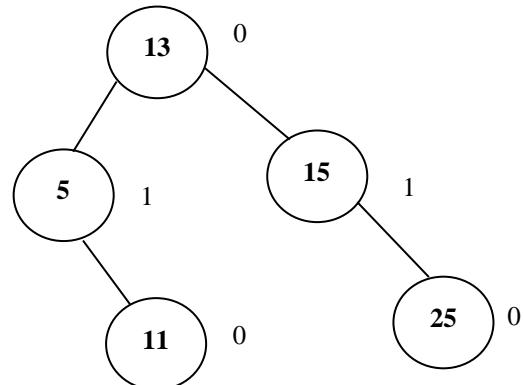
$$1-1=0$$



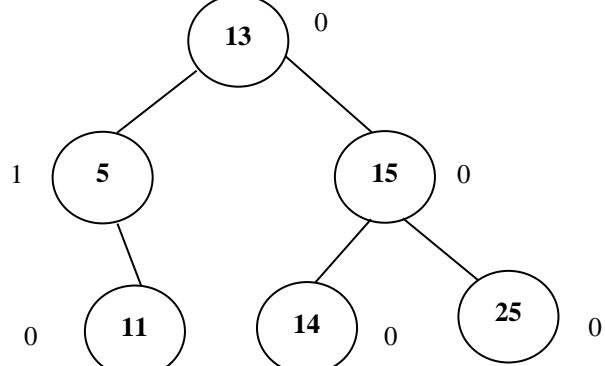
+11



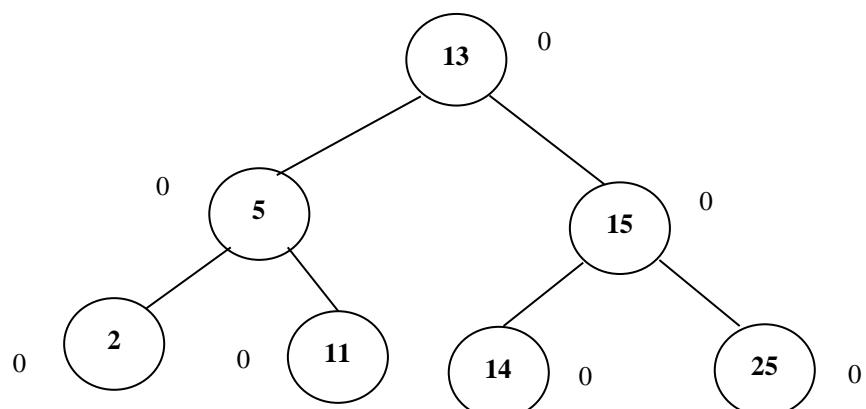
+25



+14



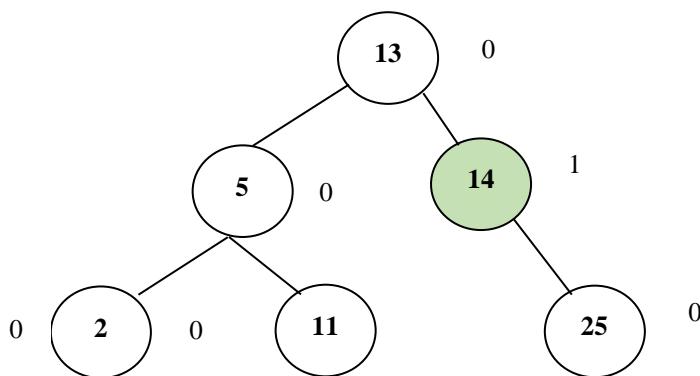
+2



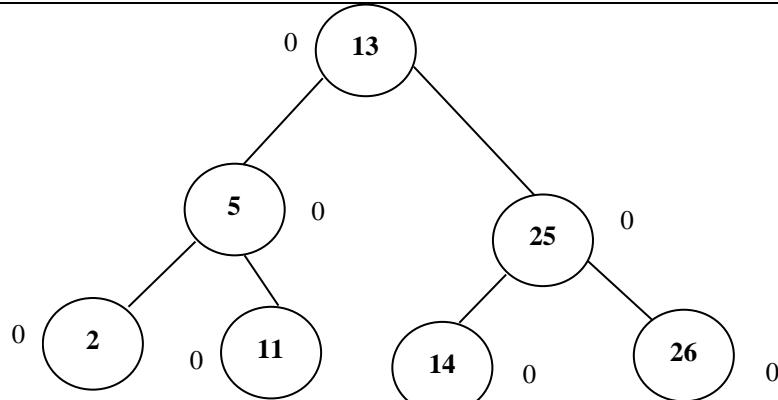
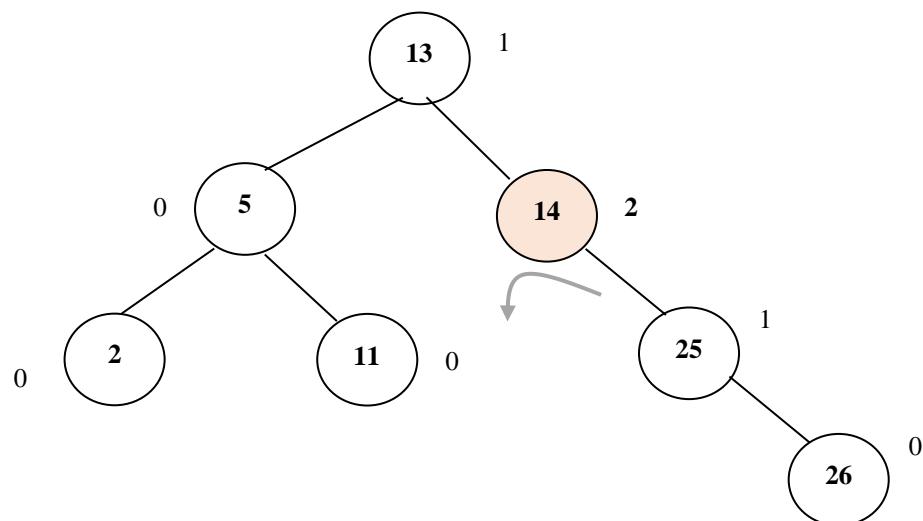
–15

Όταν διαγράφουμε ΦΥΛΛΟ, απλά το διαγράφουμε και ελέγχουμε την υψοζύγιση.

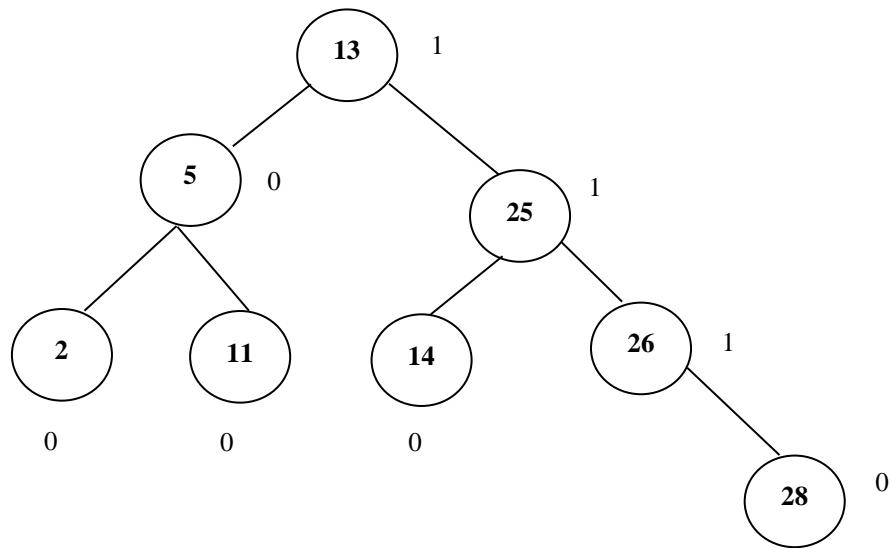
Όταν διαγράφουμε KOMBO, ΒΑΖΟΥΜΕ ΣΤΗ ΘΕΣΗ ΤΟΥ ΤΗ **max ΤΙΜΗ ΤΟΥ ΑΡΙΣΤΕΡΟΥ ΥΠΟΔΕΝΤΡΟΥ ΤΟΥ**. Εδώ η μεγαλύτερη τιμή του αριστερού υποδέντρου του κόμβου **15** που διαγράφεται είναι το 14 και βάζουμε το 14 στη θέση του.



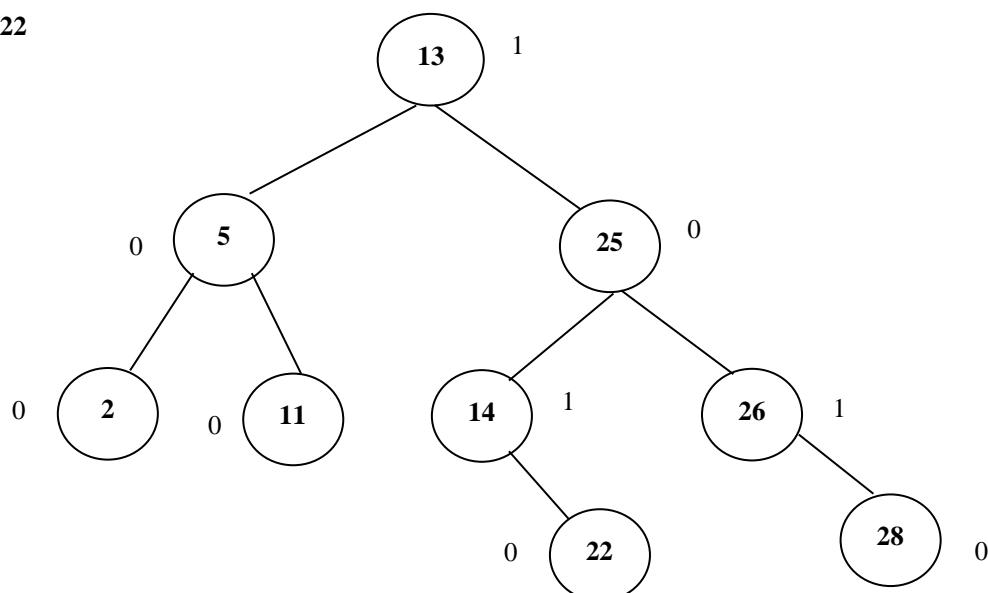
+26



+28



+ 22



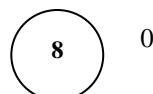
Αυτό είναι το τελικό AVL δέντρο

13.5 Θέμα 5 Σεπτέμβριος 2016

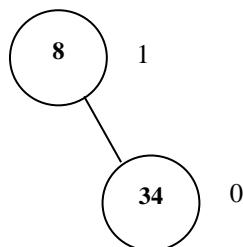
Να σχεδιαστεί βήμα προς βήμα το AVL δέντρο που προκύπτει από τις πράξεις εισαγωγής και διαγραφής σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων $\{+8, +34, +12, +1, +3, +14, +23, +37, +41, +32, -1, -34\}$ όπου $+$ εισαγωγή στοιχείου και $-$ διαγραφή στοιχείου.

Απάντηση

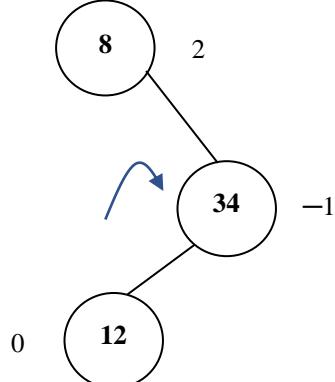
+ 8



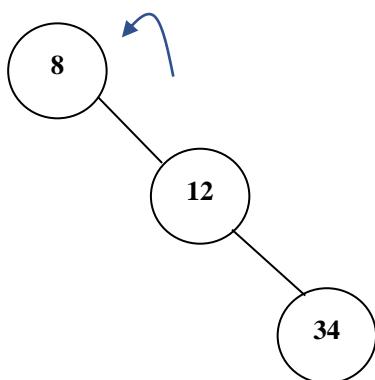
+ 34



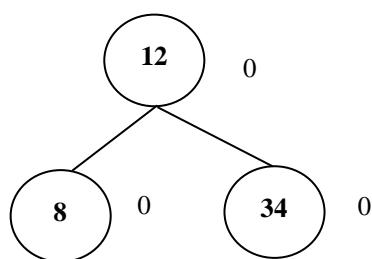
+ 12



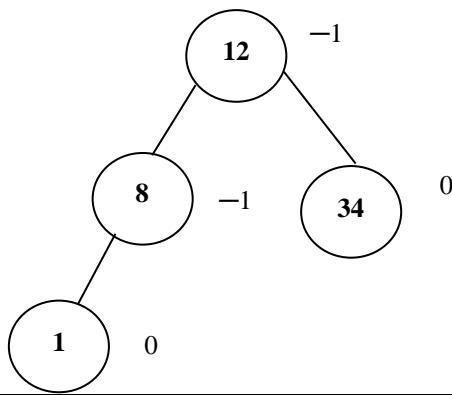
Δεξιά στροφή στο 34



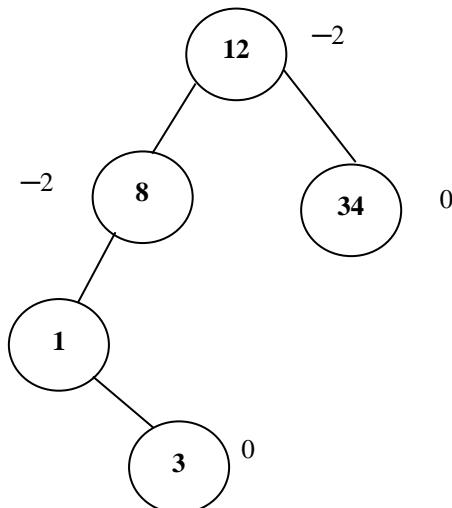
Αριστερή στροφή στο 8



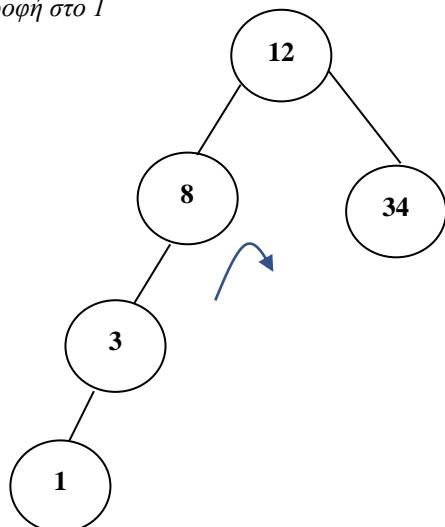
+ 1



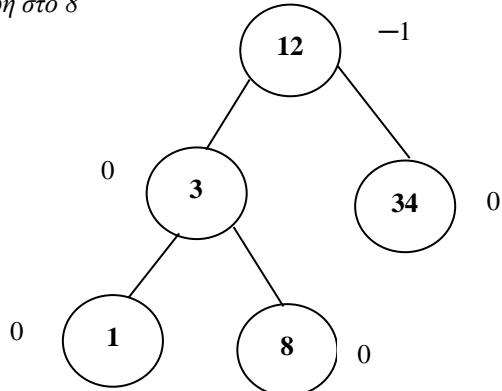
+ 3



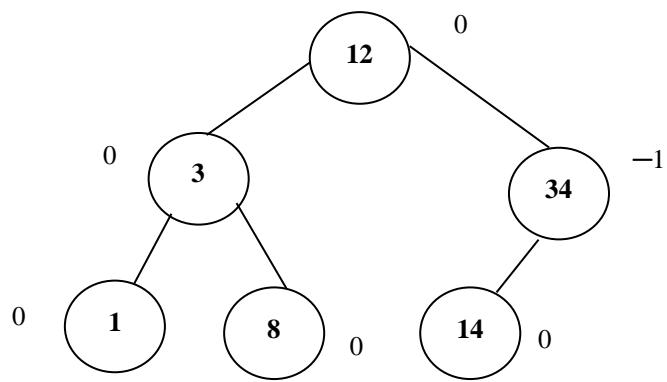
Αριστερή στροφή στο 1



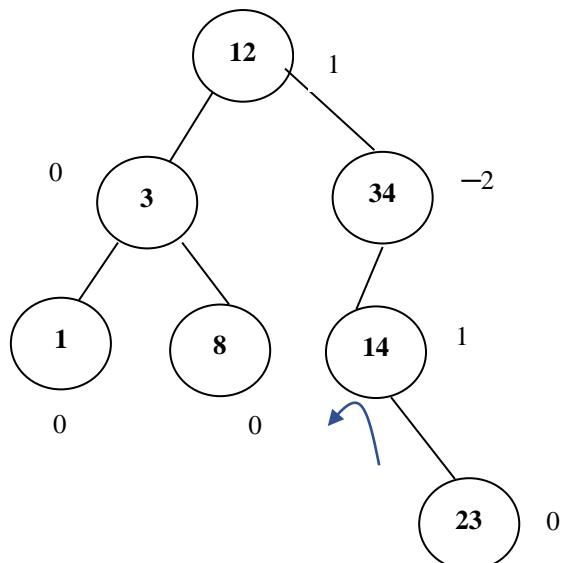
Δεξιά στροφή στο 8



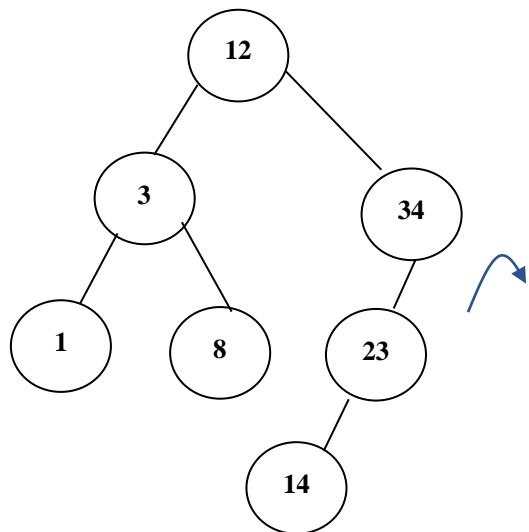
+14



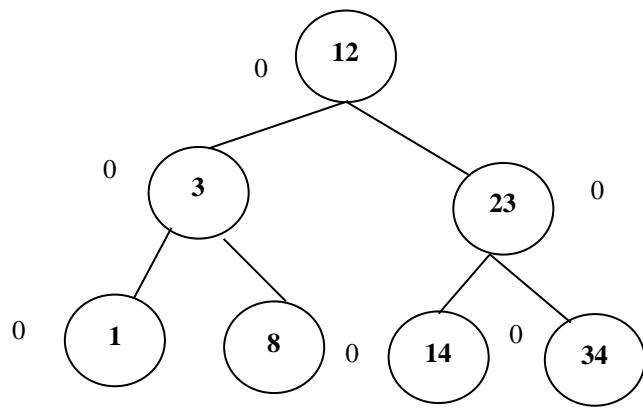
+23



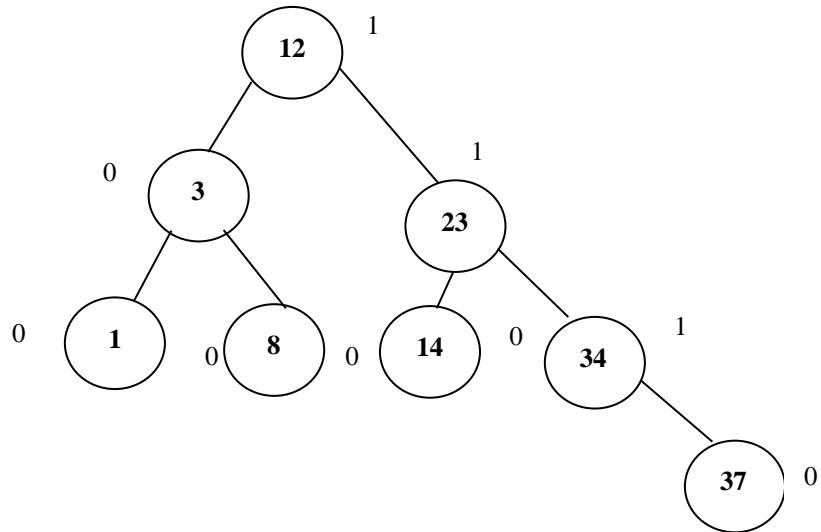
Αριστερή στροφή στο 14



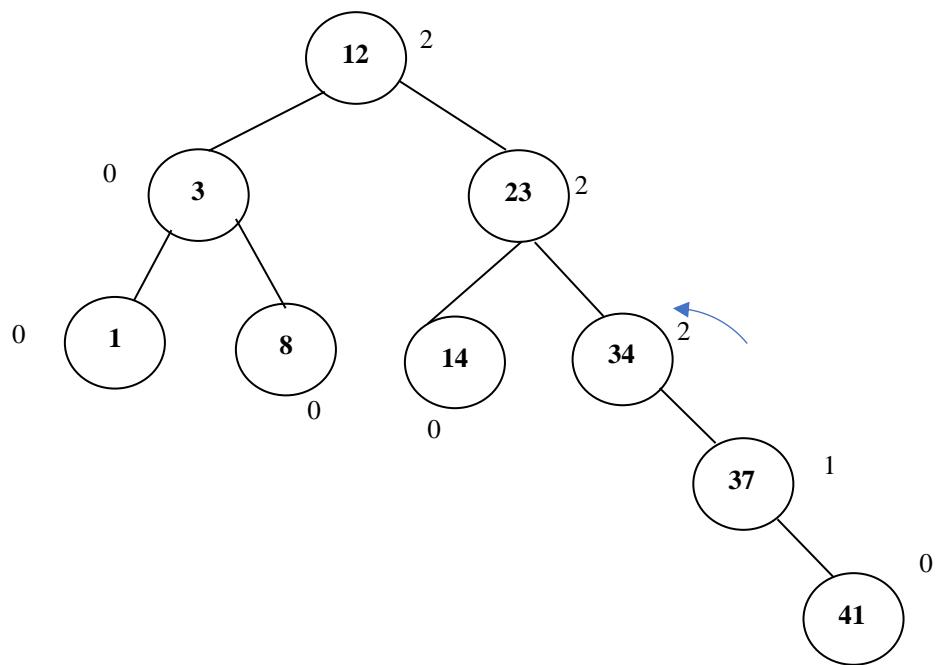
Δεξιά στροφή στο 34



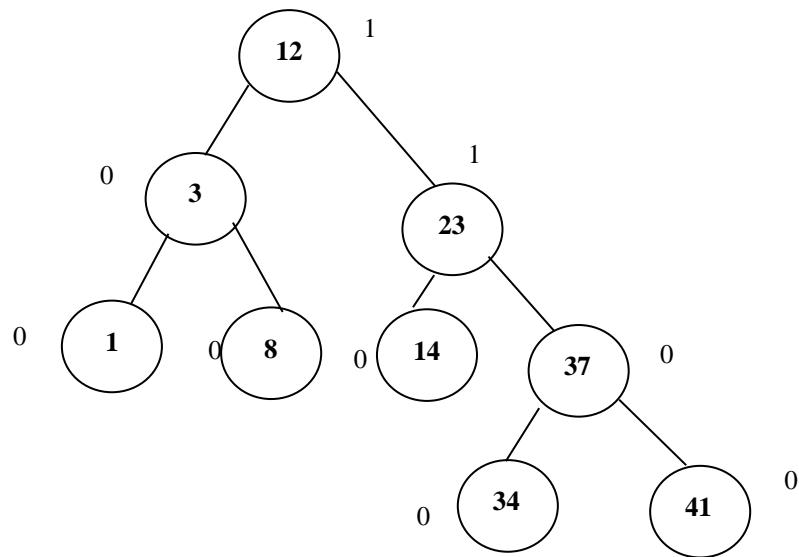
+ 37



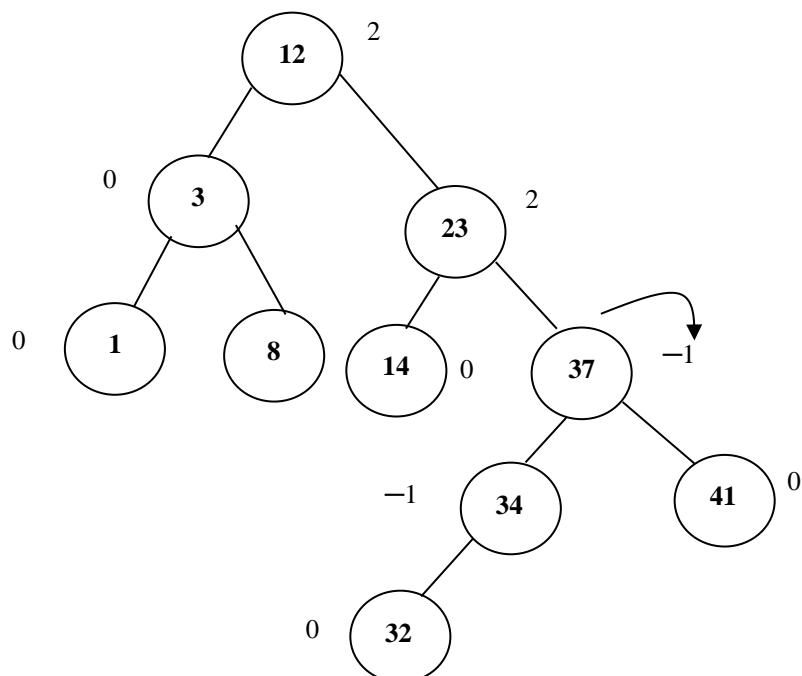
+ 41



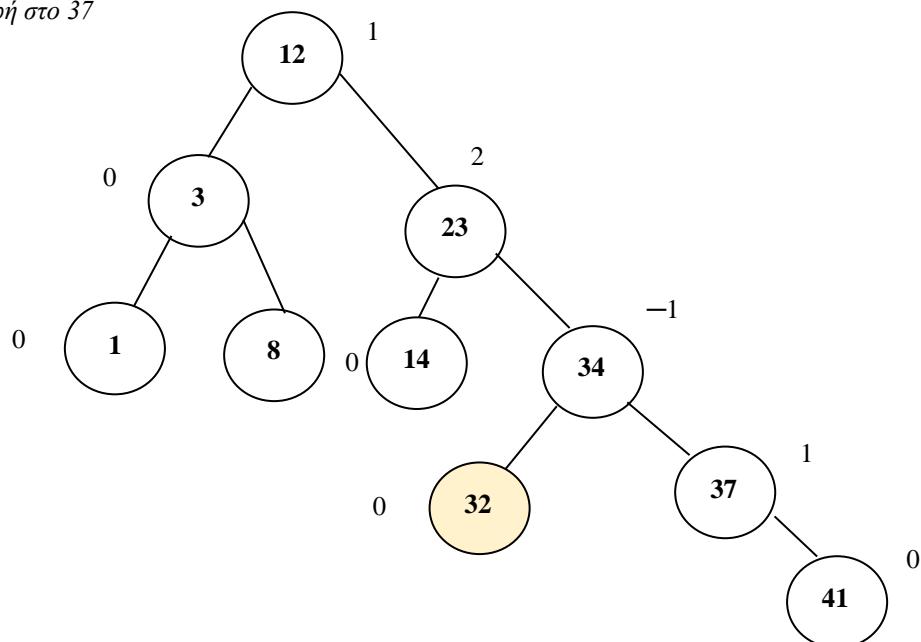
Αριστερή στροφή στο 34



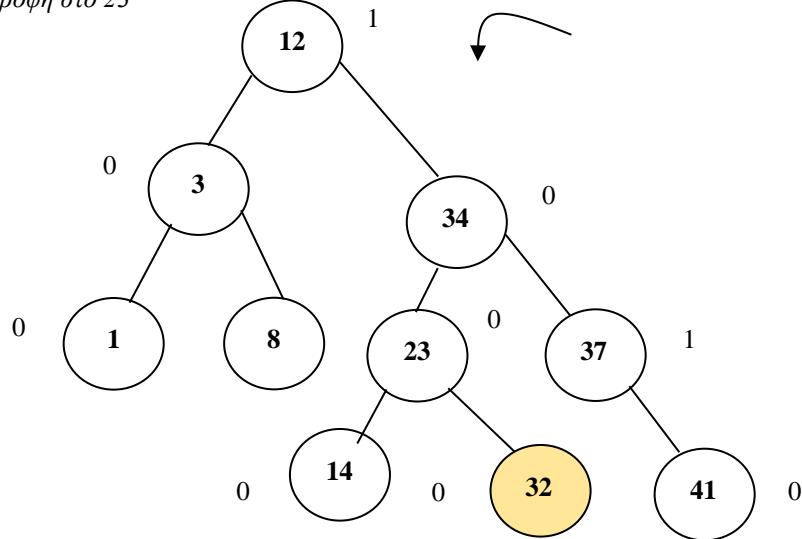
+ 32



Δεξιά στροφή στο 37

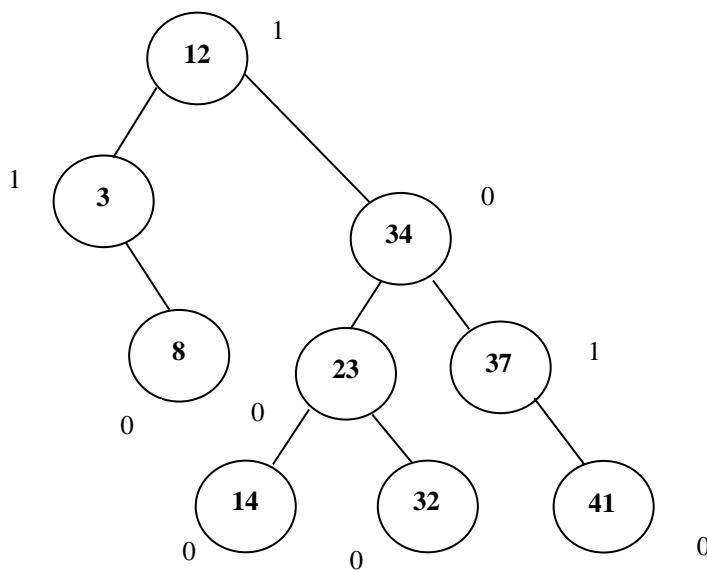


Αριστερή στροφή στο 23

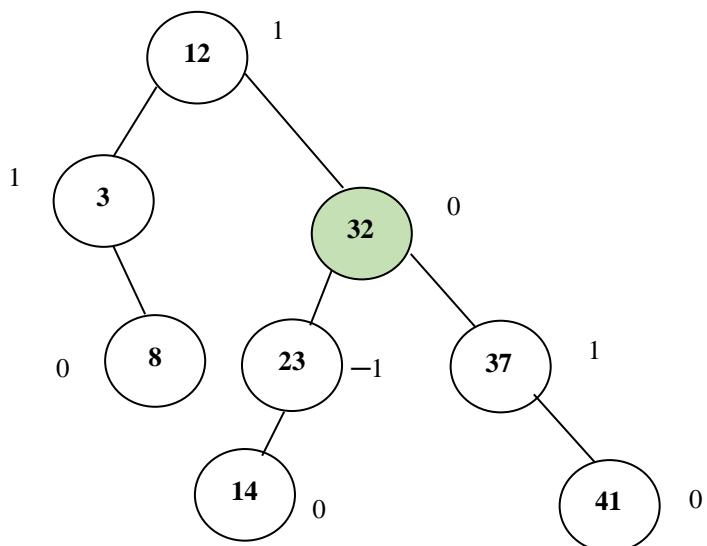


Προσοχή το αριστερό παιδί του 34 (που είναι το 32) θα γίνει δεξί παιδί του 23

-1



-34



Όταν διαγράφουμε KOMBO, ΒΑΖΟΥΜΕ ΣΤΗ ΘΕΣΗ ΤΟΥ ΤΗ ΜΕΓΑΛΥΤΕΡΗ ΤΙΜΗ ΤΟΥ ΑΡΙΣΤΕΡΟΥ ΥΠΟΔΕΝΤΡΟΥ ΤΟΥ. Εδώ η μεγαλύτερη τιμή του αριστερού υποδέντρου είναι το 32 και το βάζουμε στη θέση του 34

Αυτό είναι το τελικό AVL Δέντρο

13.6 Θέμα 10 – Ιούνιος 2009

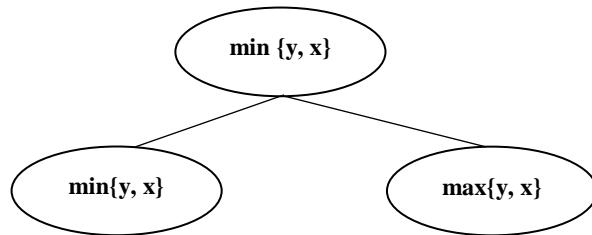
Περιγράψτε την πράξη εισαγωγής σε ένα AVL δέντρο. Στη συνέχεια κατασκευάστε το AVL δέντρο που προκύπτει μετά την ακολουθία των παρακάτω εισαγωγών: 20, 10, 30, 5, 3, 25, 22, 35, 40

Απάντηση

Η εισαγωγή σε ένα AVL δέντρο είναι μια διαδικασία 3 βημάτων:

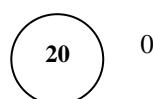
i) εύρεση της κατάλληλης θέσης στο δέντρο για την εισαγωγή του φύλλου με τιμή x. Πρόκειται για τη διαδικασία Access(x) μετά το πέρας της οποίας αν βρεθεί ότι η τιμή x υπάρχει ήδη στο δέντρο, η εισαγωγή δεν γίνεται, αλλιώς η αναζήτηση τερματίζει σε ένα φύλλο έστω y

ii) Αντικατάσταση του y με το δέντρο:

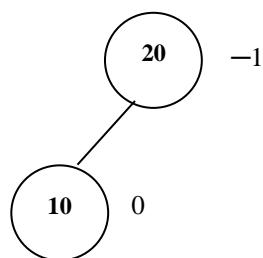


iii) Κάνουμε τις απαιτούμενες αλλαγές αν χρειάζεται για επαναφορά της ζύγισης στο δέντρο

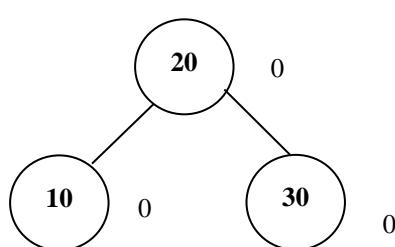
+ 20



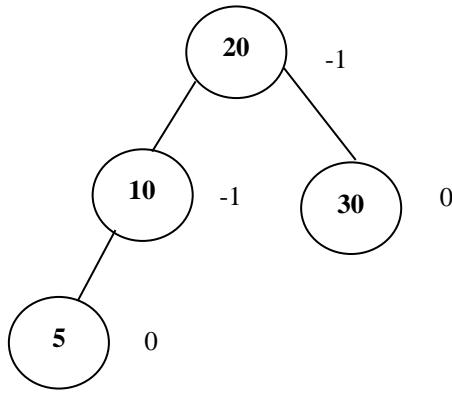
+ 10



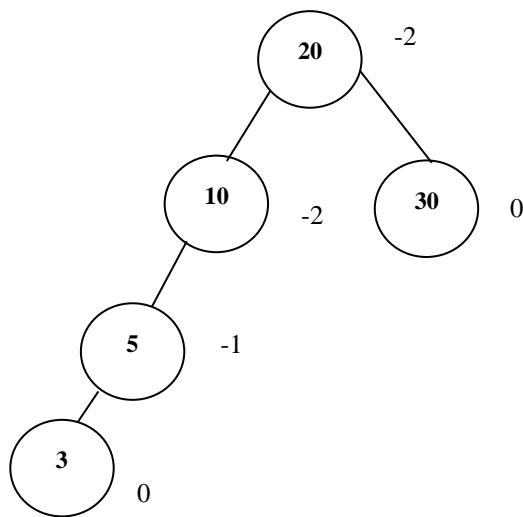
+ 30



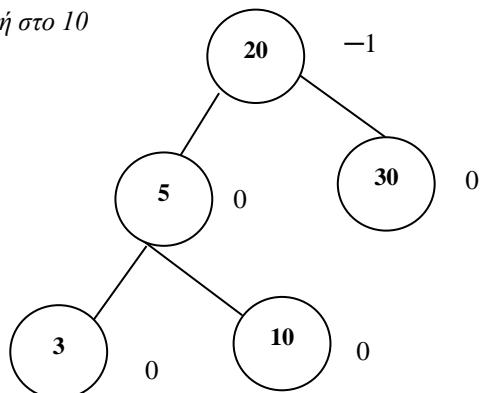
+ 5



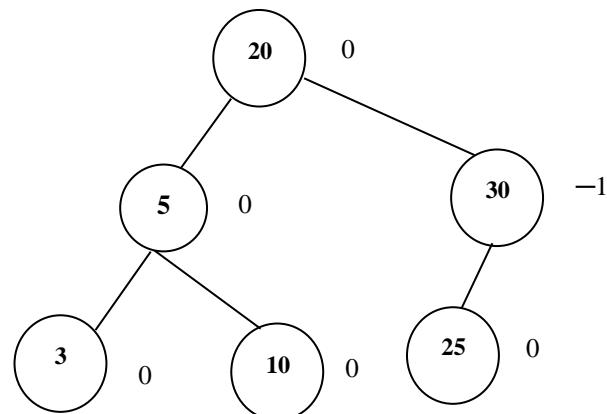
+ 3



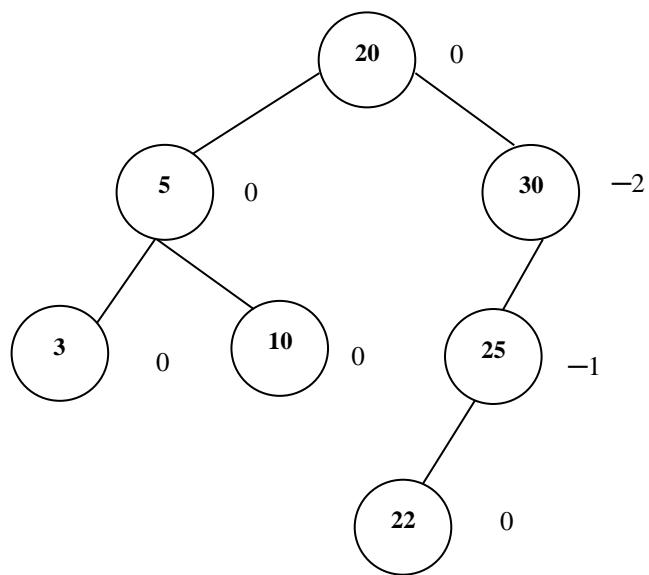
Αιώρθωση: Δεξιά στροφή στο 10



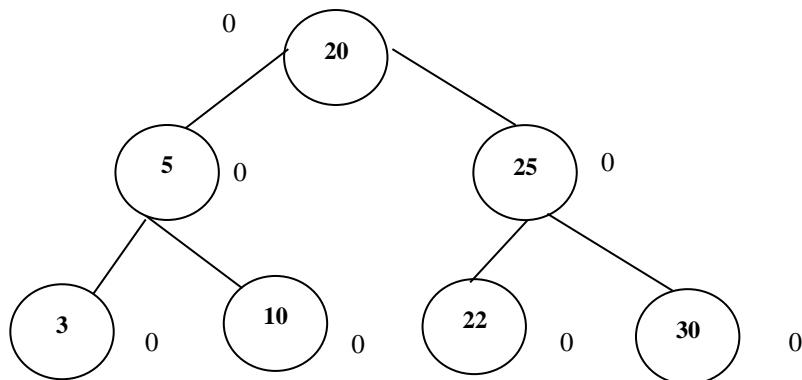
+ 25



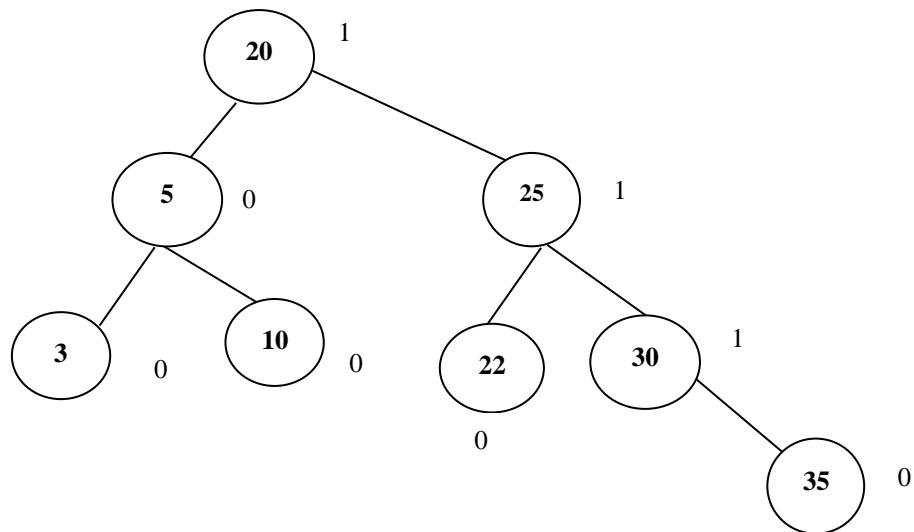
+ 22



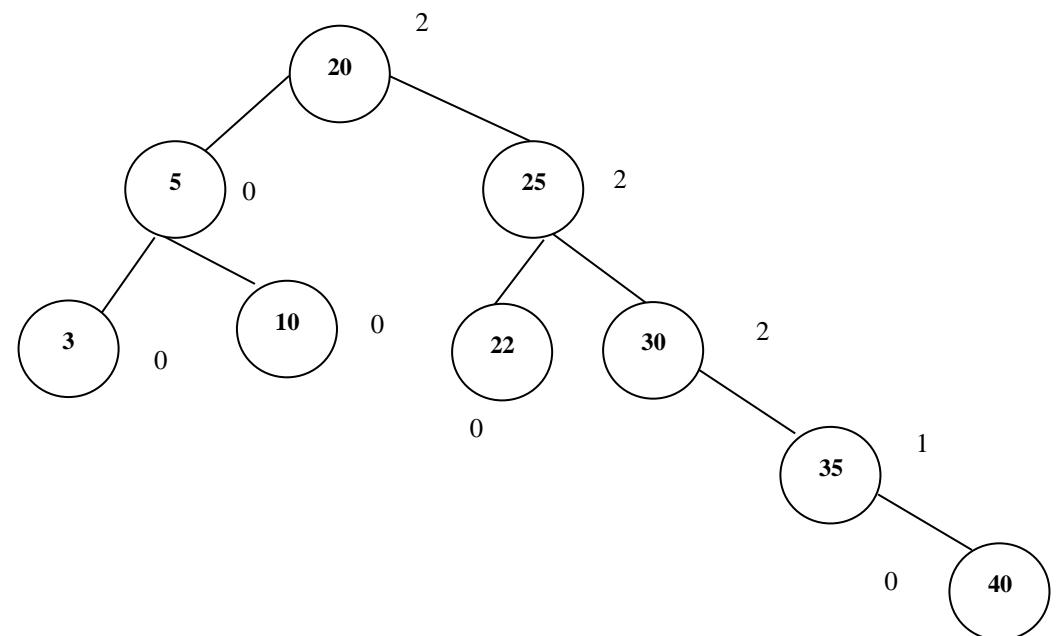
Διόρθωση: Δεξιά στροφή στο 30



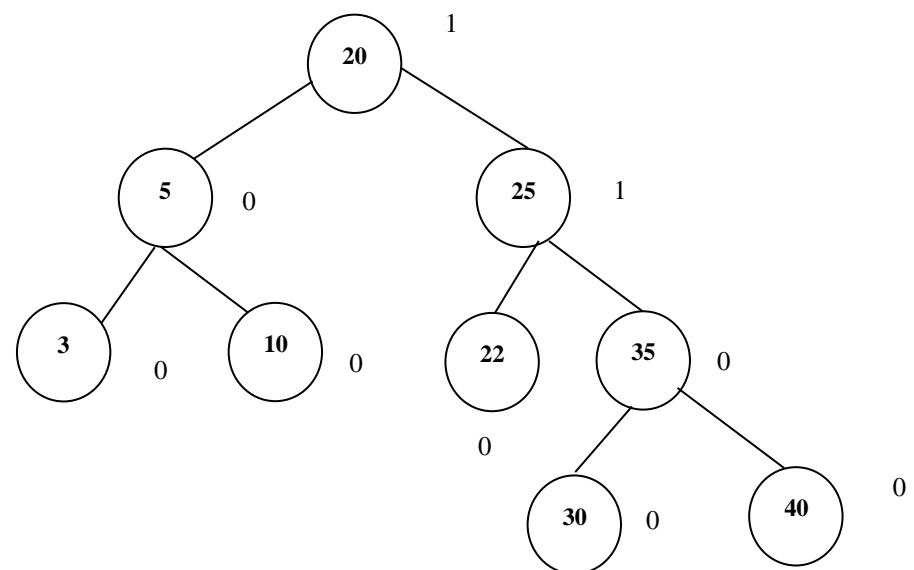
+ 35



+ 40



Διόρθωση: Αριστερή στροφή στο 30

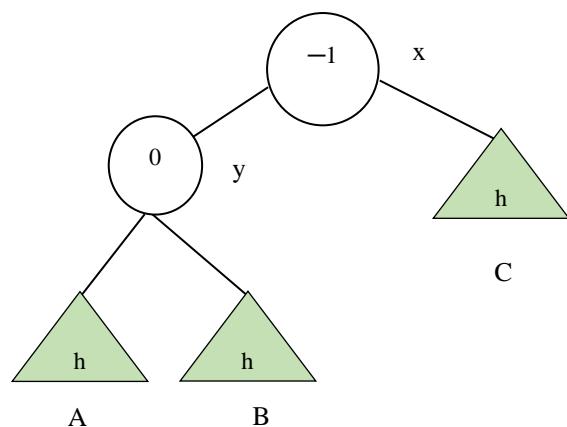


13.7 Θέμα 7 Ιούνιος 2011

Στα AVL δέντρα του παρακάτω σχήματος έστω ότι λαμβάνει χώρα μια ένθεση στο υποδέντρο **A** για το δέντρο **a**) και μια στο υποδέντρο **B** για το δέντρο **b**) με αλλαγή του ύψους αυτών των υποδέντρων. Διαταράσσεται η υψομέτρη; Αν ναι δείξτε πως μπορεί να επαναφερθεί.

Απάντηση

α) Αρχικό Δέντρο

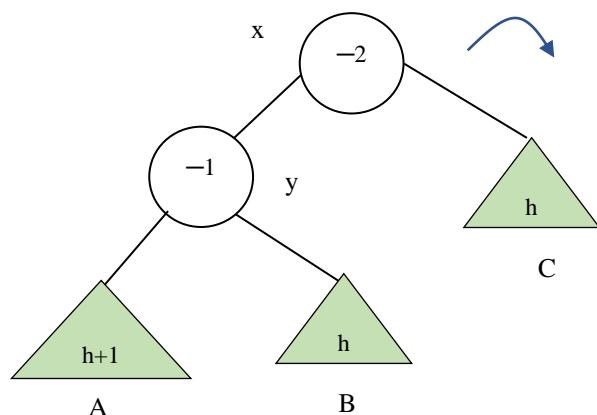


Παρατήρηση

$h+1 - (h+1) = 0$ για τον κόμβο γ

$$h+1 - (h+1+1) = -1 \text{ για τον κόμβο } x$$

Μετά την ένθεση στο υποδέντρο Α το σχήμα παίρνει τη μορφή:

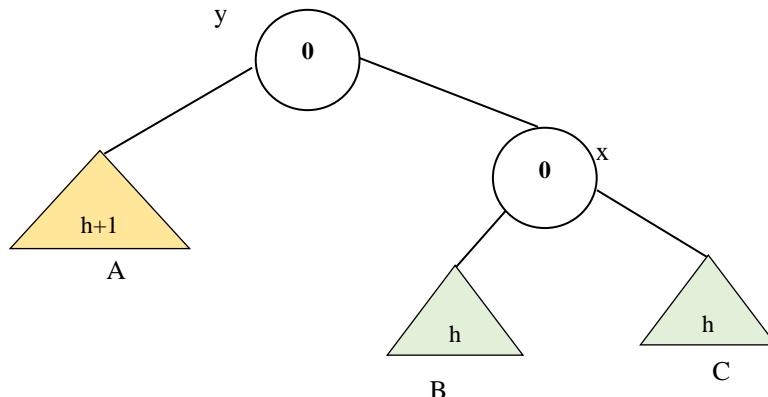


Παρατήρηση

$h+1-(h+1+1) = -1$ για τον κόμβο y

$h+1-(h+1+1+1) = -2$ για τον κόμβο x

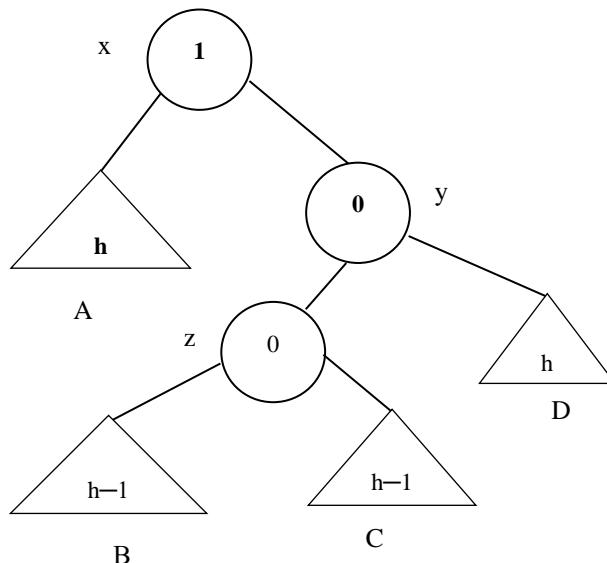
Παραβιάζεται η υψοζύγιση του κόμβου x άρα το δέντρο δεν είναι πλέον AVL. Το διορθώνουμε με δεξιά στροφή στον κόμβο x



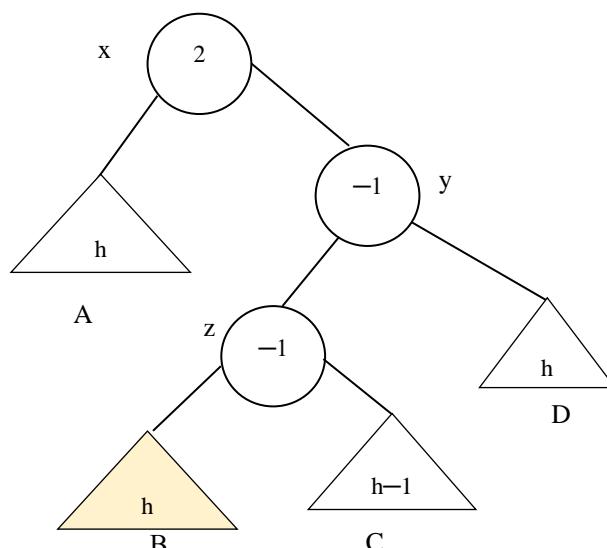
Παρατήρηση

Το Υποδέντρο B μπαίνει ως αριστερό παιδί του x. Παρατηρούμε ότι το δέντρο προκύπτει είναι τώρα AVL

β) Αρχικό Δέντρο



Μετά την ένθεση στο υποδέντρο B το σχήμα παίρνει τη μορφή:



Παρατήρηση

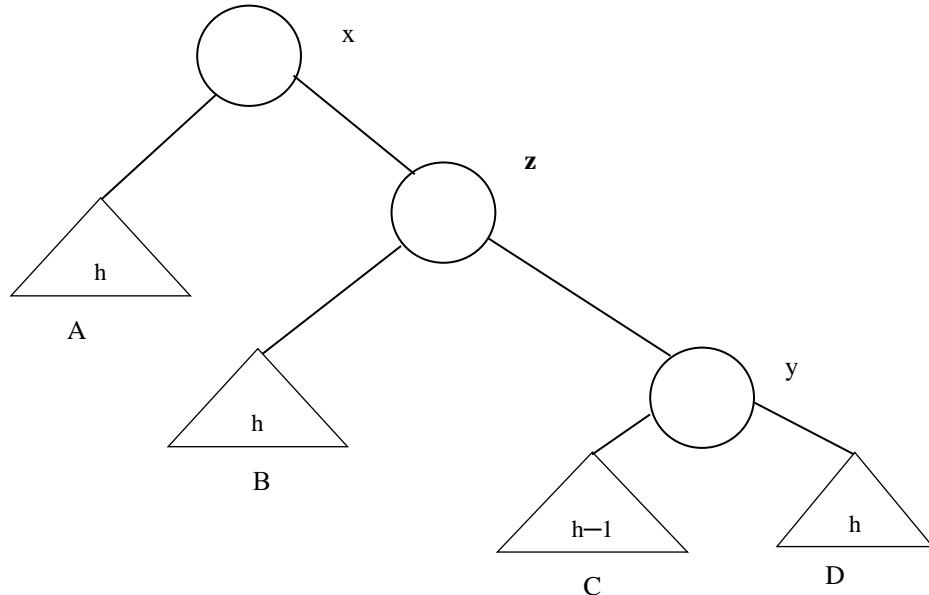
$h-1+1(h+1) = -1$ για τον κόμβο z

$h+1-(h+1+1) = -1$ για τον κόμβο y

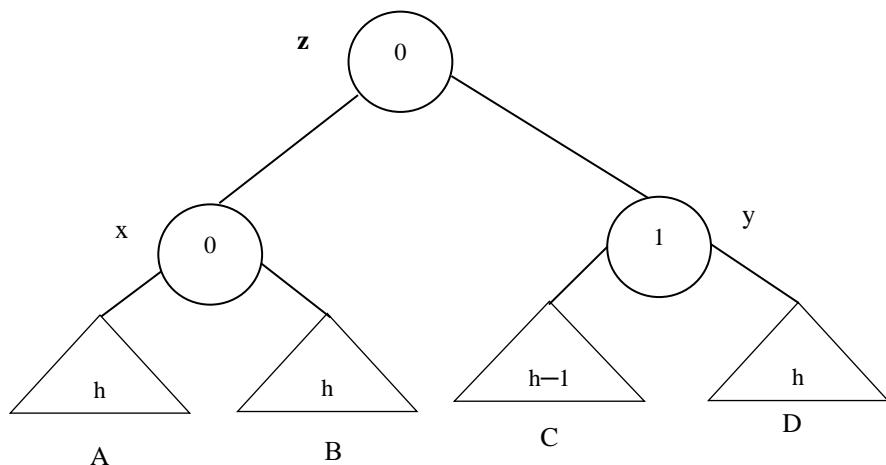
$h+3-(h+1) = 2$ για τον κόμβο x

Παραβιάζεται η υψοζύγιση του κόμβου x άρα το δέντρο δεν είναι πλέον AVL.

Το διορθώνουμε κάνοντας **αρχικά δεξιά στροφή στον κόμβο y** και προκύπτει το ακόλουθο δέντρο:



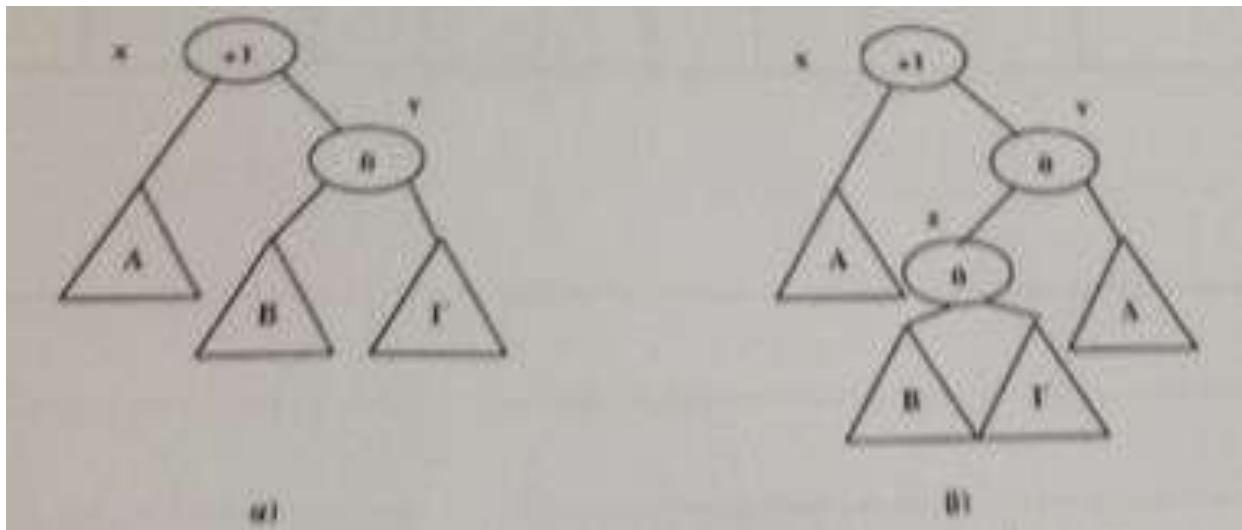
Το τελικό δέντρο προκύπτει με μια επιπλέον **αριστερή στροφή στον x** και είναι το ακόλουθο:



Παρατηρούμε ότι το δέντρο προκύπτει είναι τώρα AVL

13.8 Θέμα 5 Σεπτέμβριος 2012

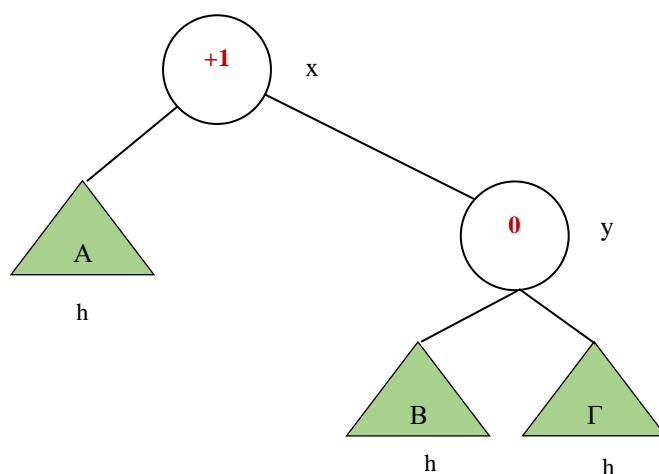
Στα παρακάτω AVL Δέντρα ισχύει ότι στοιχεία(A)<στοιχεία(B)<στοιχεία(Γ)<Στοιχεία(Δ)



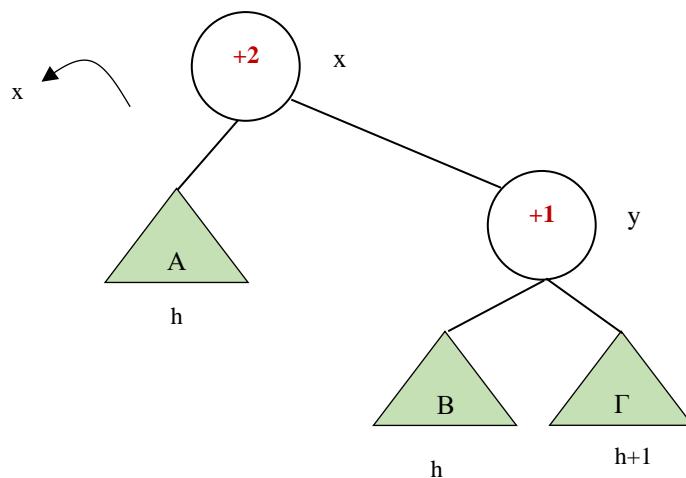
Επαναφέρετε την υψοζύγιση στον x στα επιτρεπτά της όρια μετά **την Εισαγωγή** ενός φύλλου στο υποδέντρο Γ και **στα δύο παραδείγματα**. Αναφέρετε επιγραμματικά την (τις) πράξη (πράξεις) που εκτελέσατε

Απάντηση

a) Αρχικό Δέντρο

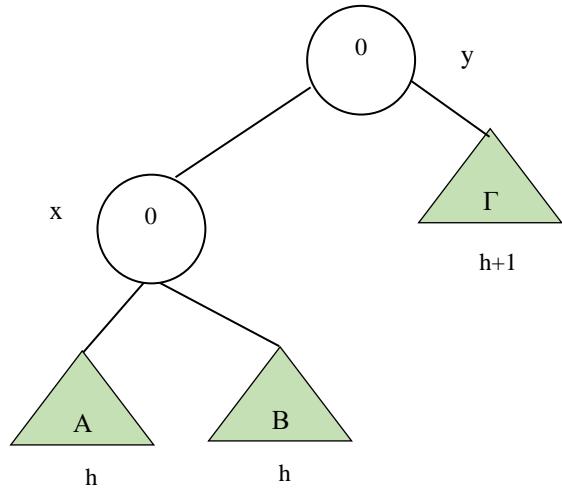


Μετά την ένθεση στο υποδέντρο Γ έχουμε:

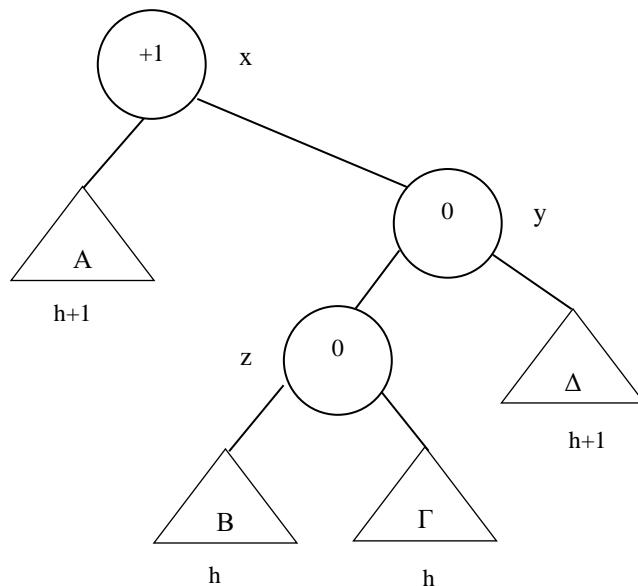


Παραβιάζεται η υψοζύγιση του κόμβου x άρα το δέντρο δεν είναι πλέον AVL. Το διορθώνουμε με αριστερή στροφή στον κόμβο x.

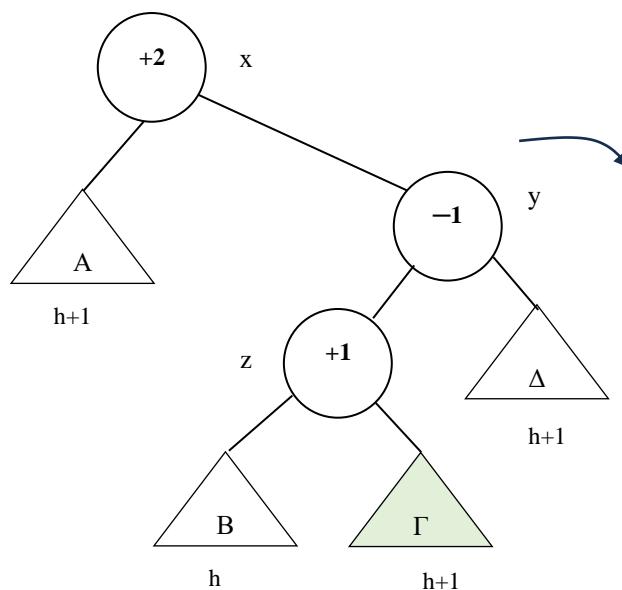
Δομές Δεδομένων –Computer Ανάλυση



β) Αρχικό Δέντρο

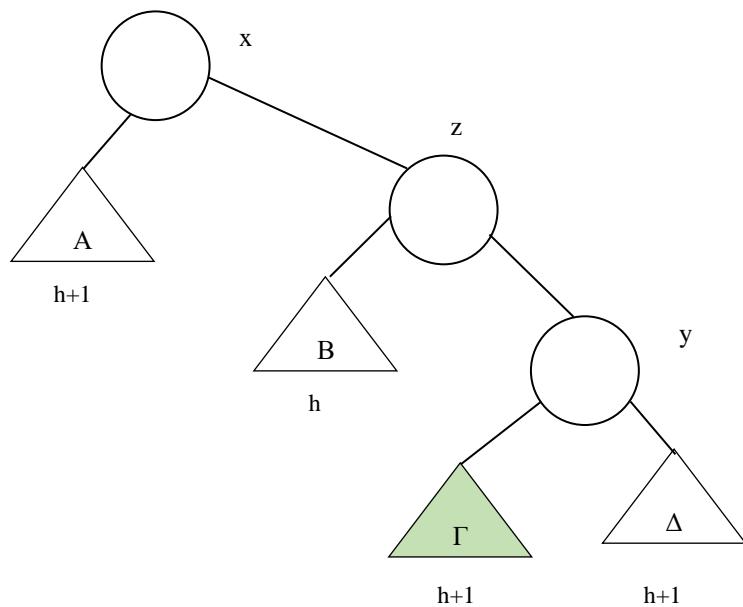


Μετά την ένθεση στο υποδέντρο Γ έχουμε:

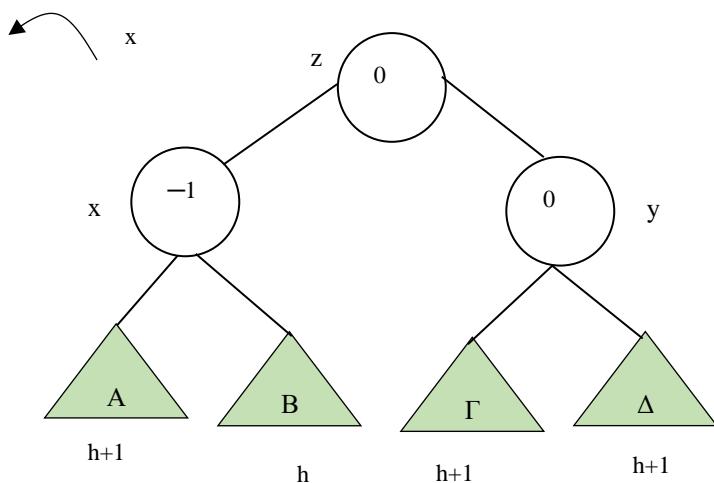


Παραβιάζεται η υψοζύγιση του κόμβου x άρα το δέντρο δεν είναι πλέον AVL.

Το διορθώνουμε αρχικά με δεξιά στροφή στον κόμβο y



και μετά με αριστερή στροφή στον κόμβο x



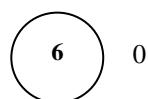
Παρατηρούμε ότι το δέντρο προκύπτει είναι τώρα AVL

13.9 Θέμα 2 Σεπτέμβριος 2020

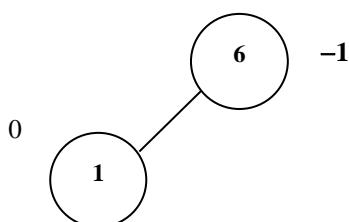
Να σχεδιαστεί βήμα-βήμα το AVL δέντρο που προκύπτει από την εισαγωγή των εξής στοιχείων $\{+6, +1, +2, +69, +7, +31, +11, +42, +20, +74\}$.

Απάντηση

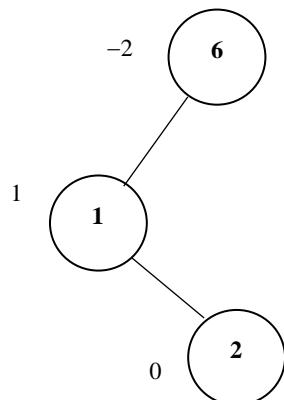
+6



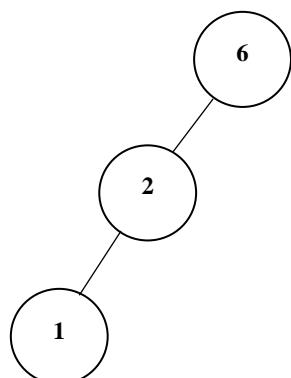
+1



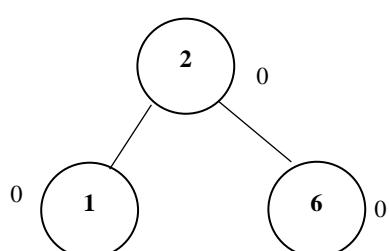
+2



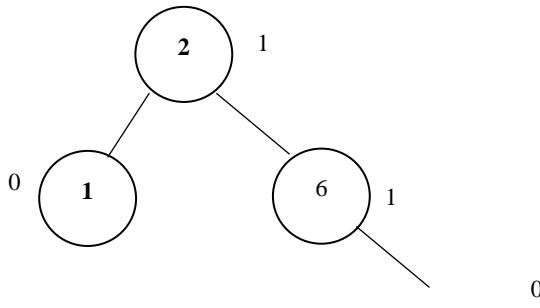
Αριστερή στροφή στο 1



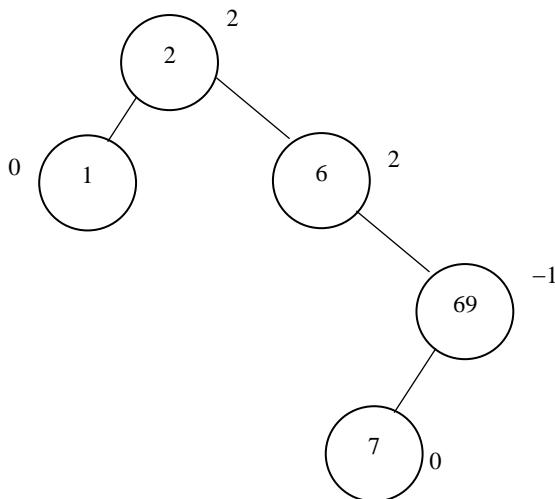
Δεξιά στροφή στο 6



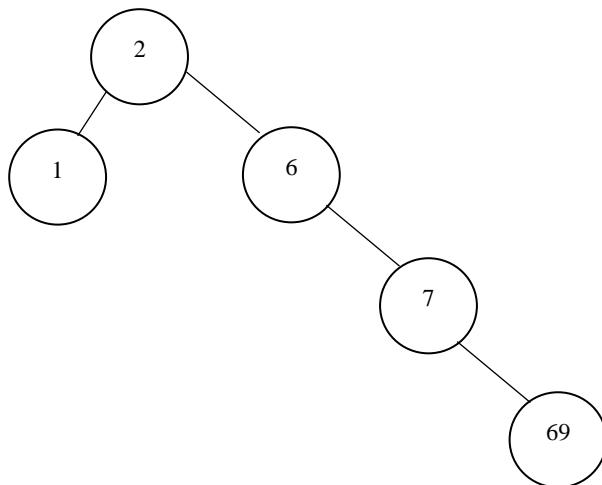
+69



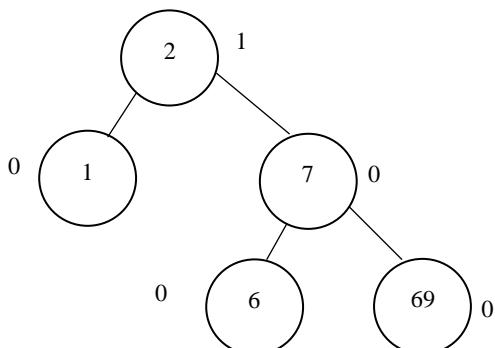
+7



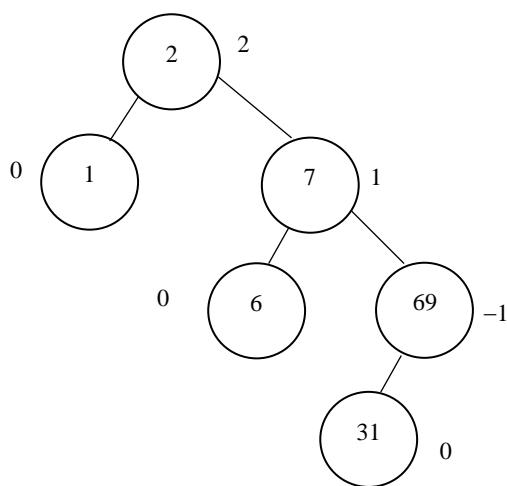
Δεξιά στροφή στο 69



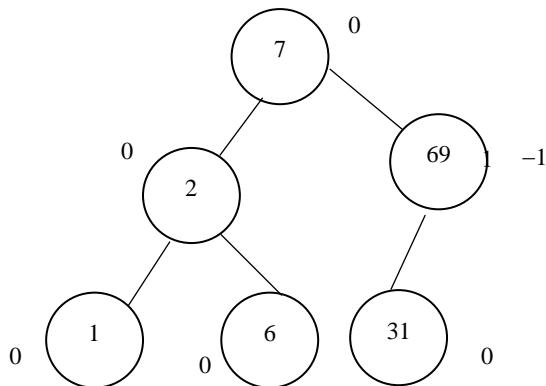
Αριστερή στροφή στο 6



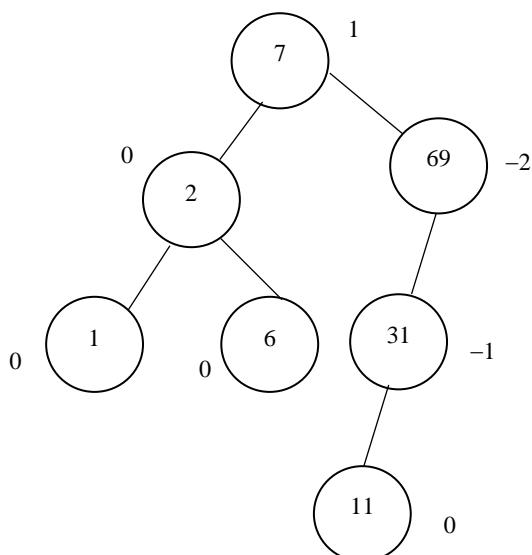
+31



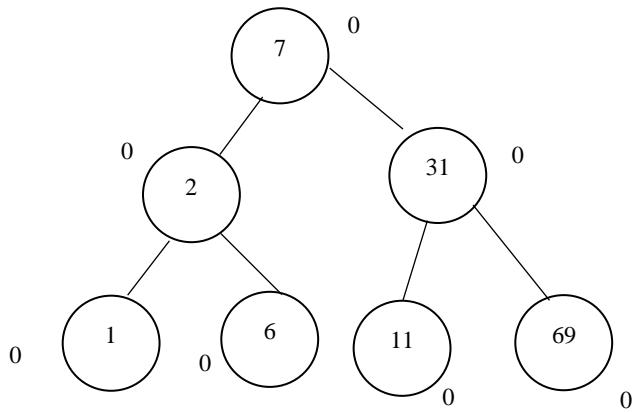
Αριστερή στροφή στο 2



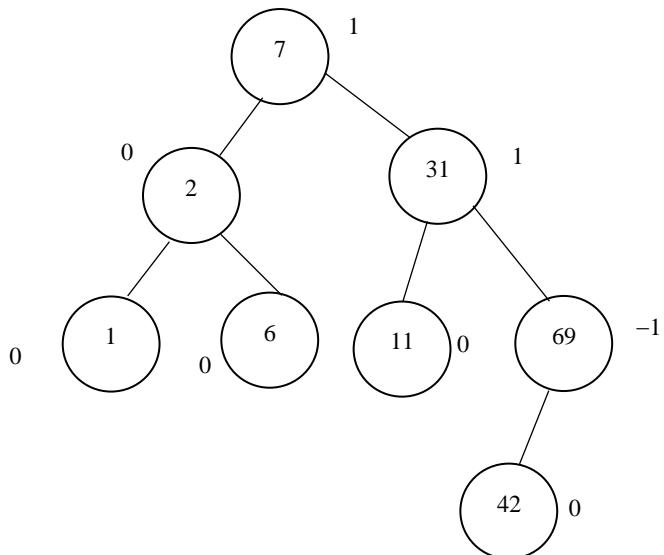
+11



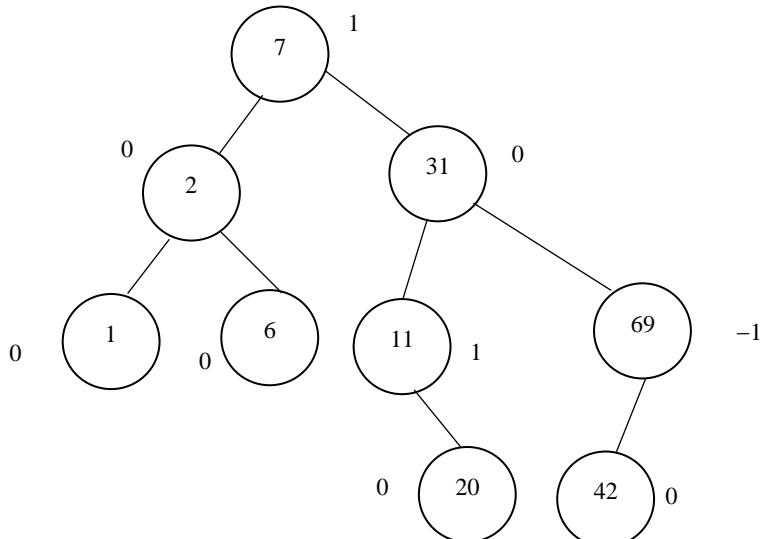
Δεξιά στροφή στο 69

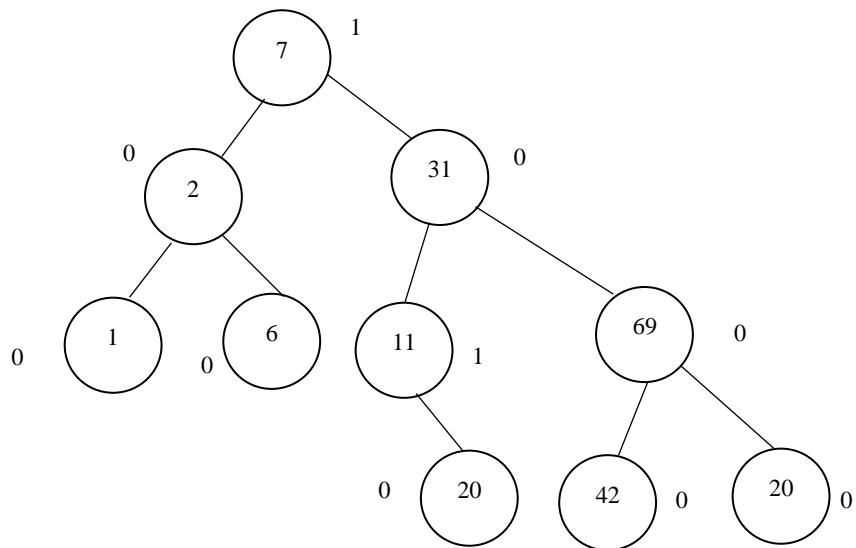


+42

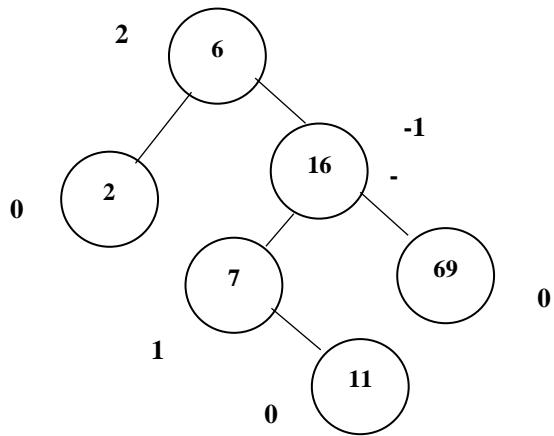


+20

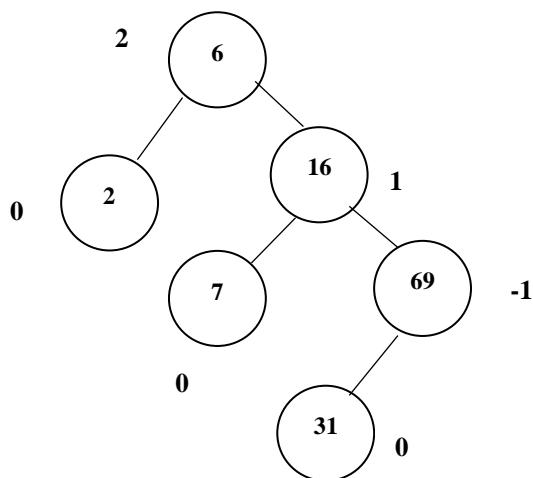




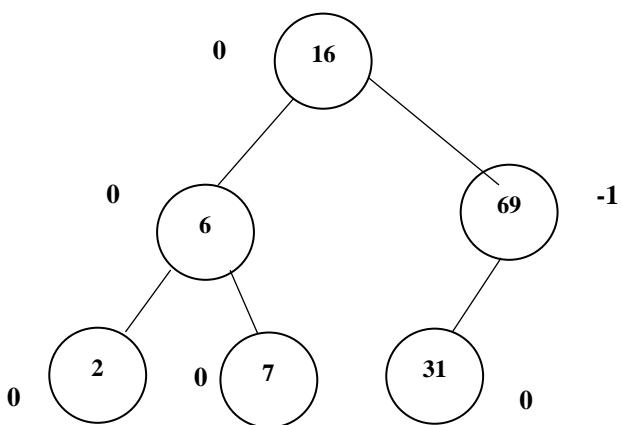
+11

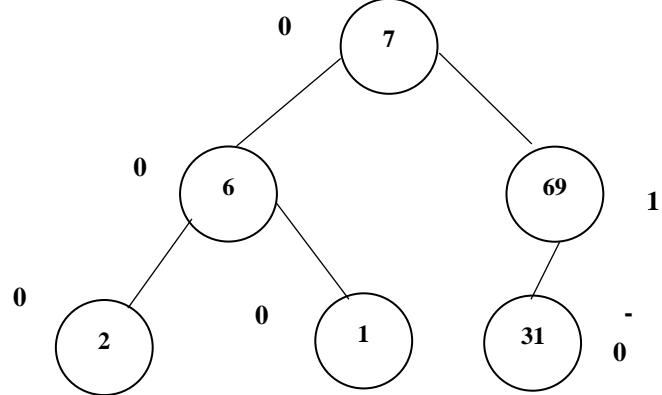


+31

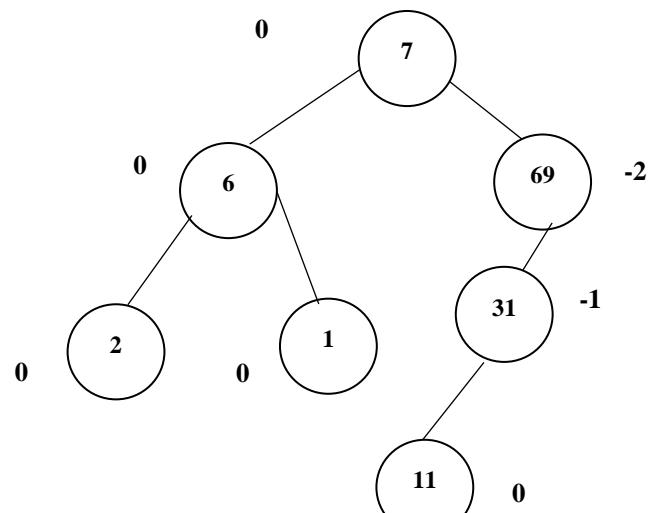


ΔΙΟΡΘΩΣΗ ΔΕΝΤΡΟΥ ΑΡΙΣΤΕΡΗ στροφή στον κόμβο 6

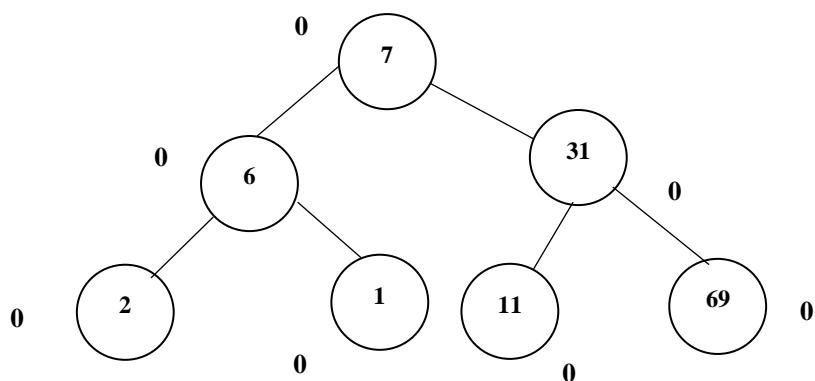




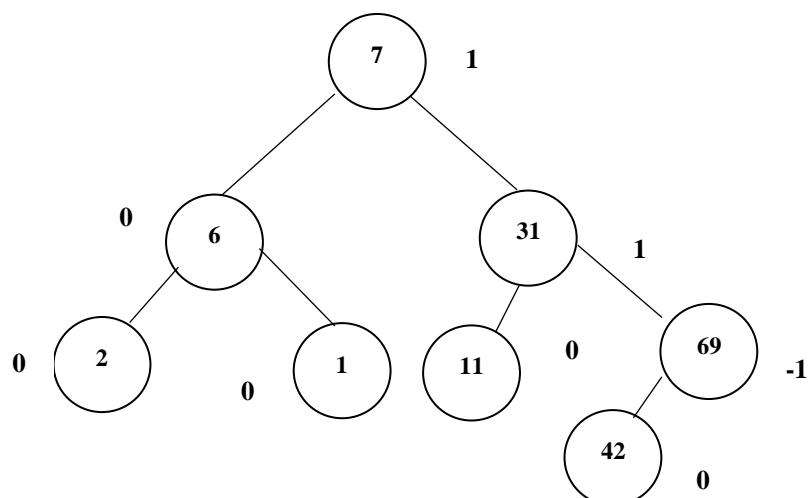
+11



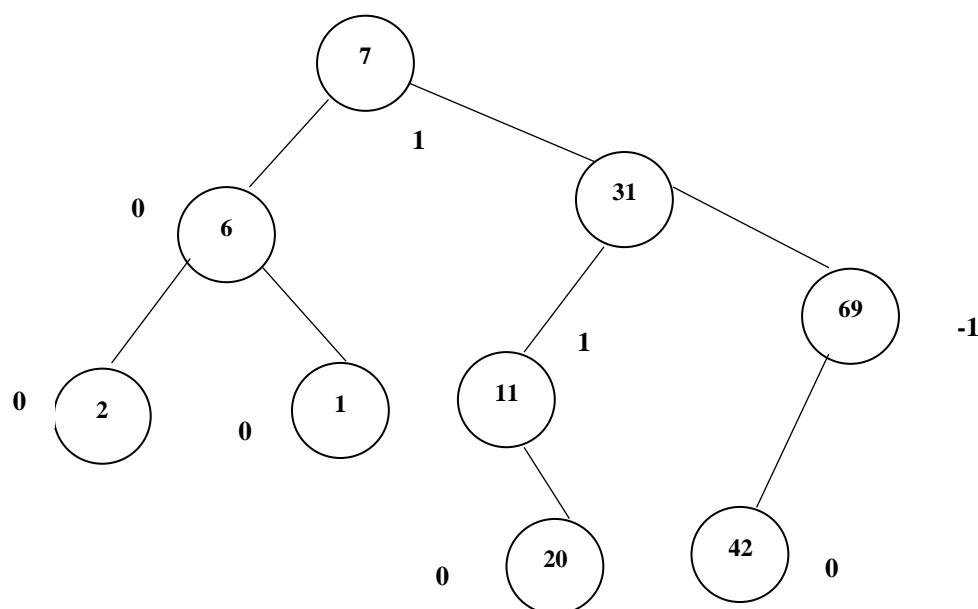
ΔΙΟΡΘΩΣΗ ΔΕΝΤΡΟΥ. ΔΕΞΙΑ στροφή στον KOMBO 69



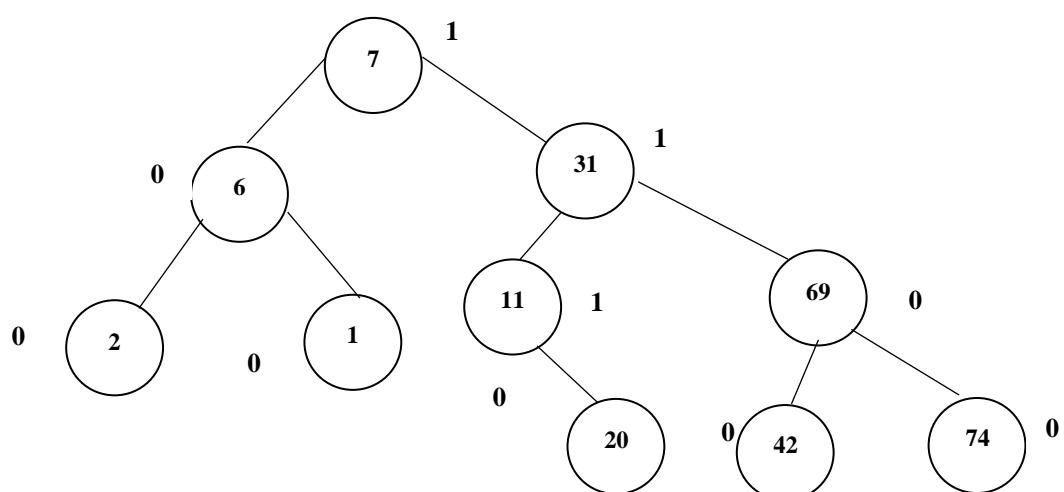
+ 42



+20

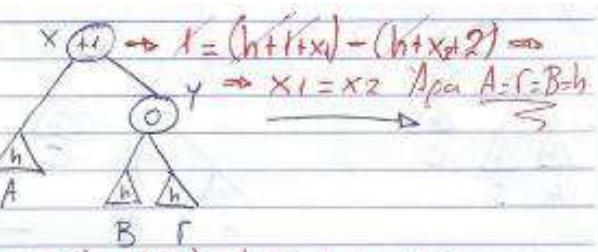


+74



13.10 Θέμα 10 με AVL Ιούνιος 2008

i) Αρχική Μορφή



$$X \rightarrow (h+1+x_1) - (h+x_2+2) \Rightarrow +1 - 2 = -1$$

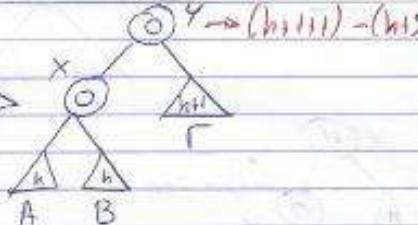
$$Y \rightarrow x_1 = x_2 \text{ Άρα } A = C = B = h$$

$$(h+1+2) - (h+1) = h+3-h-1 = +2$$

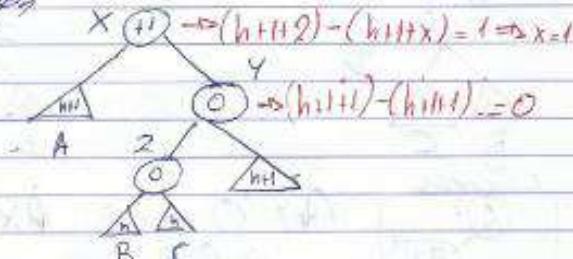
$$(h+1+1) - (h+1) = h+2-h-1 = +1$$

$$O \rightarrow (h+1+1) - (h+2) = 0$$

Kανός
A 2
B 1
C A



ii) Αρχική Μορφή



$$X \rightarrow (h+1+2) - (h+1+x) = 1 \Rightarrow x=1$$

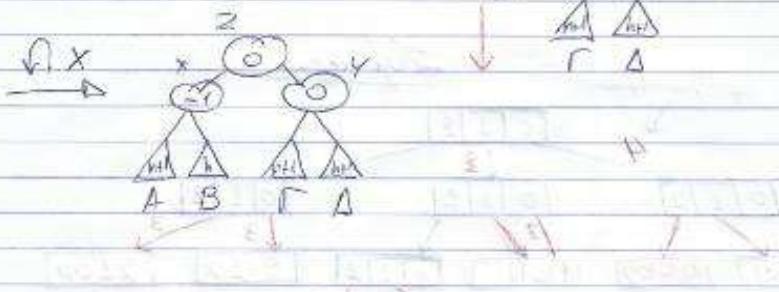
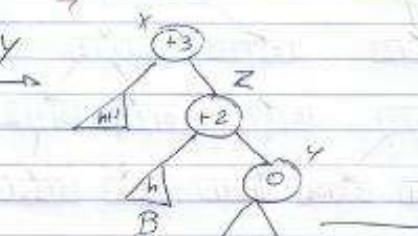
$$Y \rightarrow (h+1+1) - (h+1+1) = 0$$

$$(h+1+1) - (h+1) = +1$$

$$(h+1+1) - (h+1+1) = -1$$

$$(h+1+1) - (h+1) = +1$$

Kανός
A 2
B 1
C B
D A



13.11 Θέμα 2 Φεβρουάριος 2021 – Ομάδα Α

Να σχεδιαστεί βήμα-βήμα το AVL δέντρο που προκύπτει από την εισαγωγή σε ένα αρχικά άδειο δέντρο των εξής στοιχείων:

{**2, 23, 5, 17, 1, 11, 3, 33, 4, 35**}

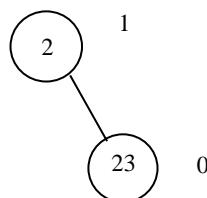
Στη συνέχεια διαγράψτε βήμα-βήμα τα στοιχεία {**11, 23**}.

Απάντηση

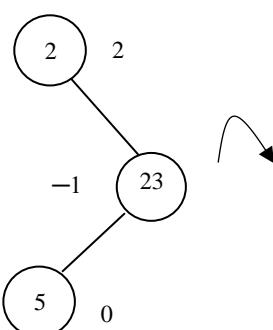
+2



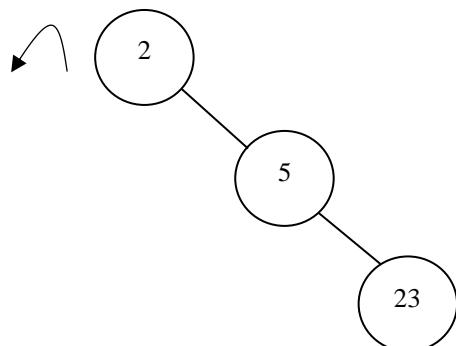
+23



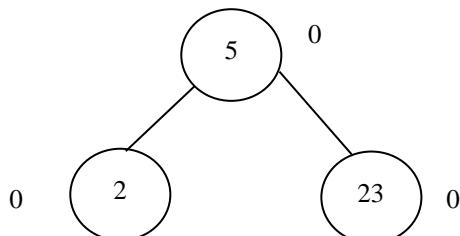
+5



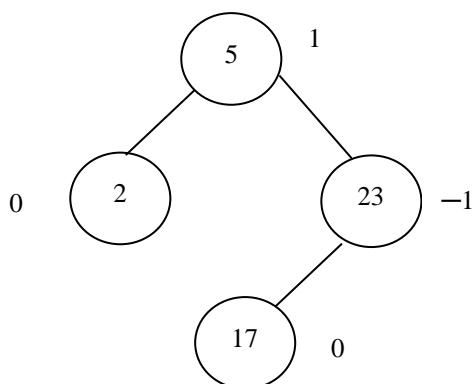
Δεξιά στροφή στο 23



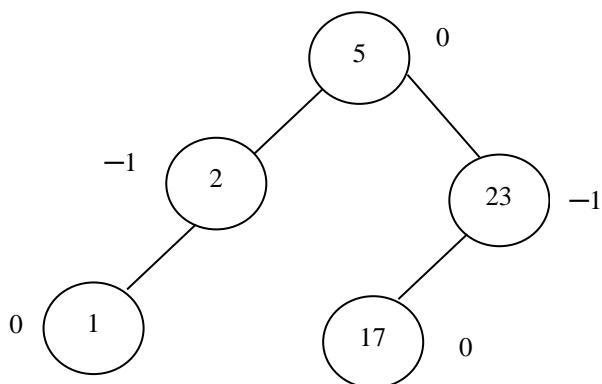
Αριστερή στροφή στο 2



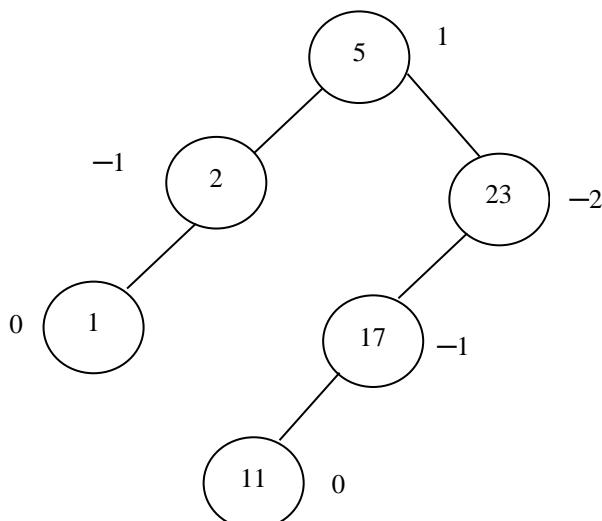
+17



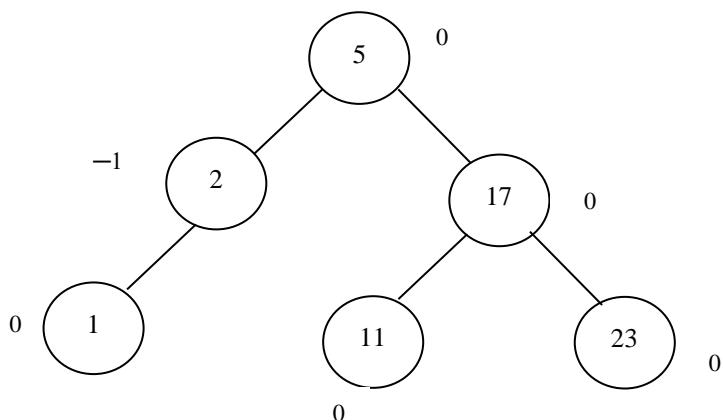
+1



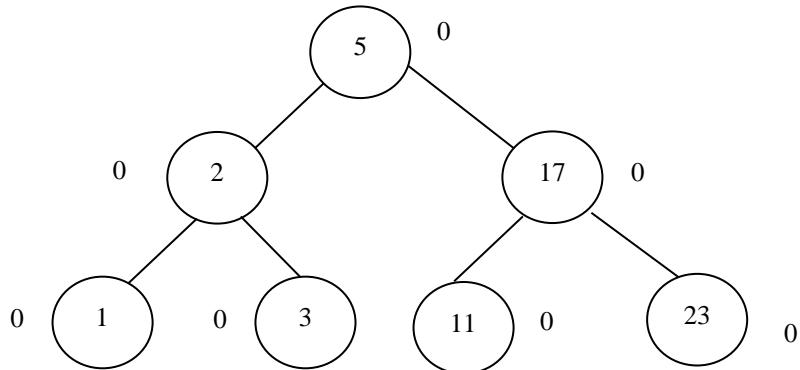
+11



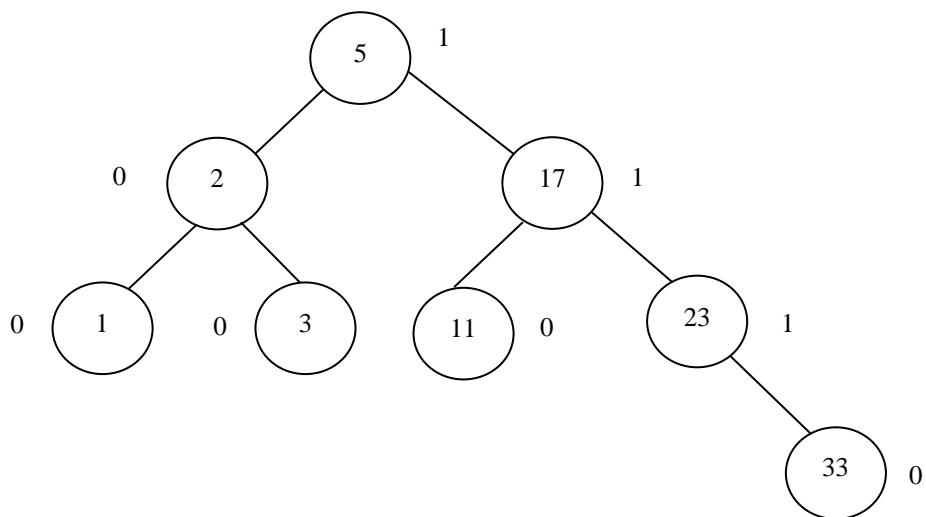
Δεξιά Στροφή στο 23



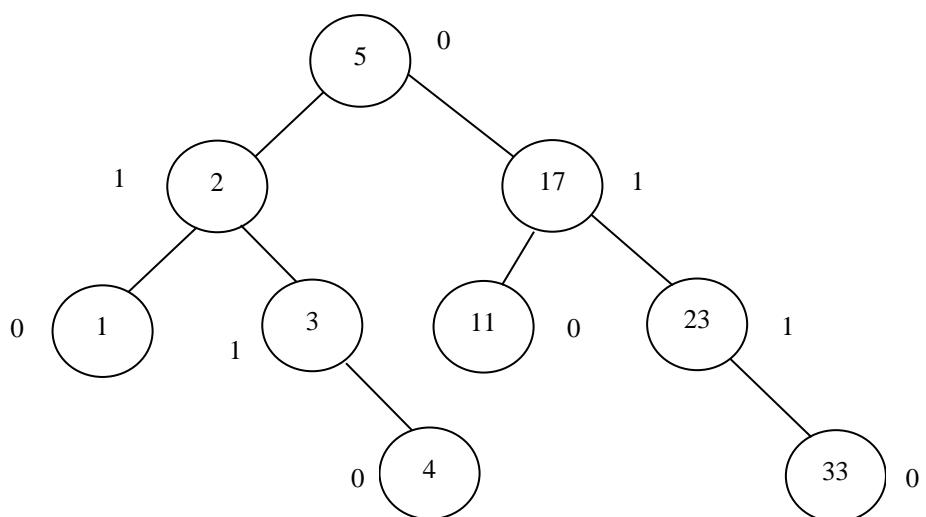
+3



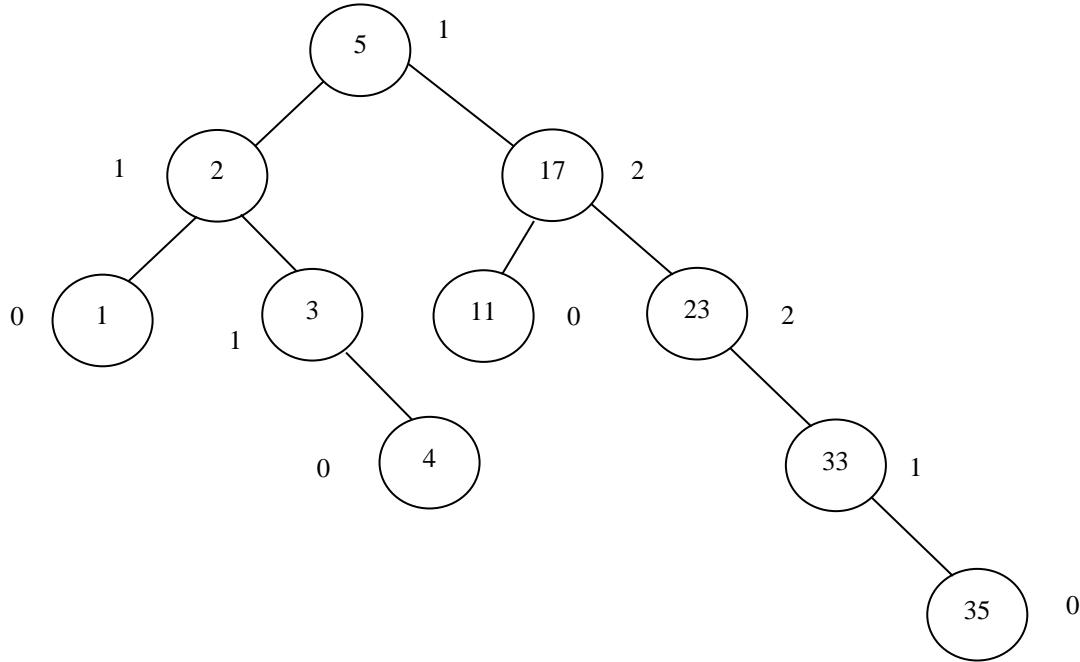
+33



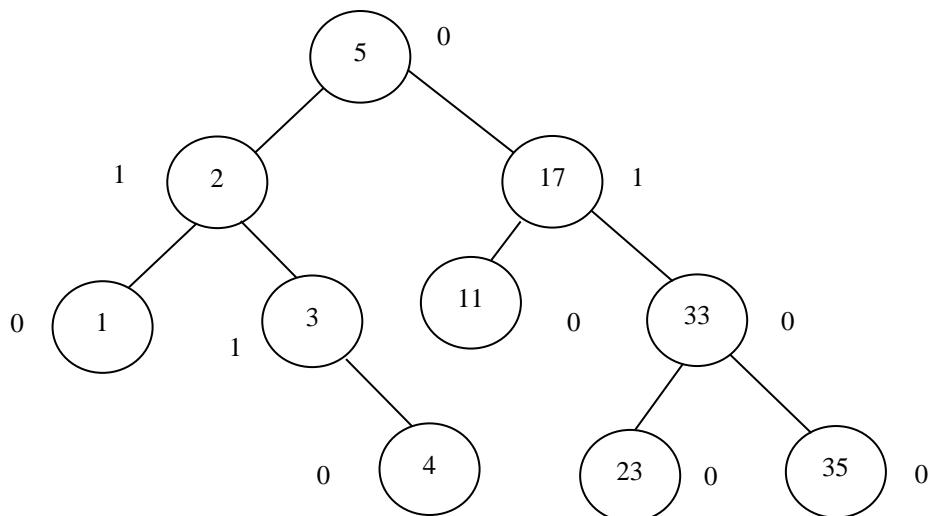
+4



+35



Αριστερή Στροφή στο 23

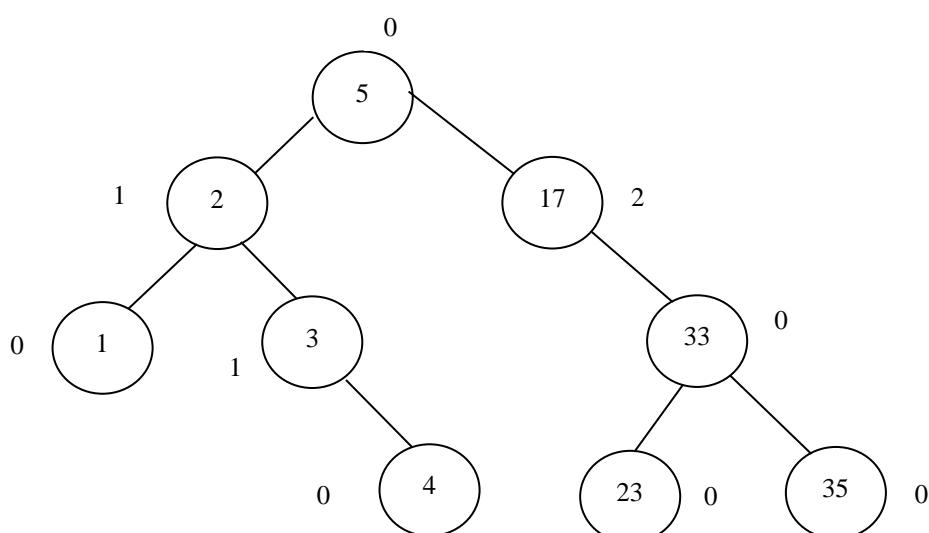


-11

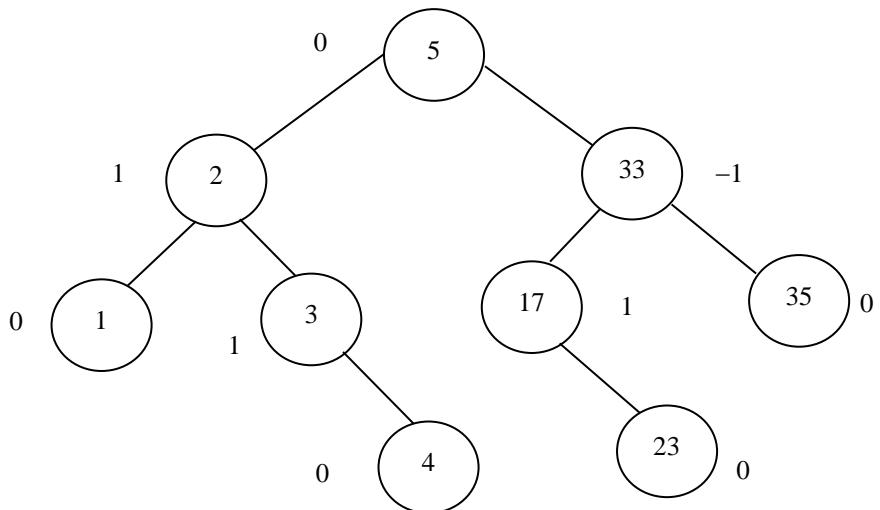
Όταν διαγράφουμε **KOMBO**, βαζούμε στη θέση του τη μεγαλύτερη τιμή του αριστερού υποδέντρου του.

Όταν διαγράφουμε **ΦΥΛΛΟ**, δεν βαζούμε κατί στη θέση του.

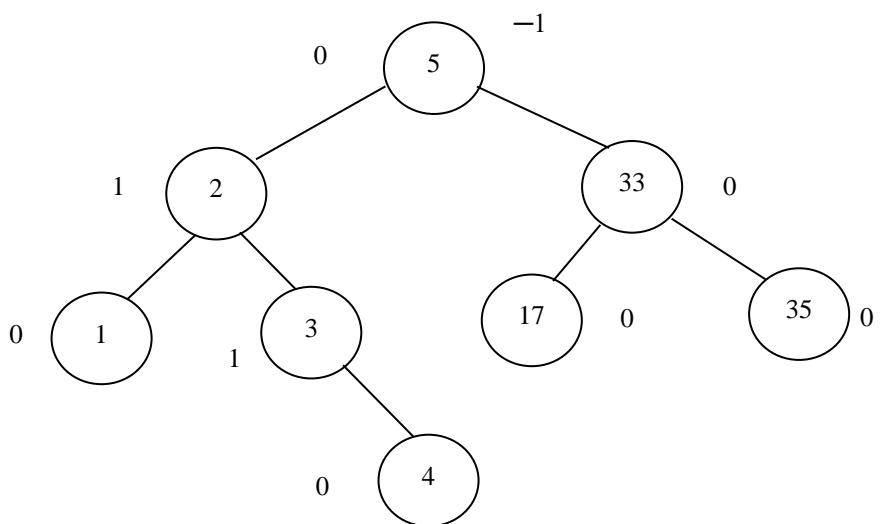
ΣΕ ΚΑΘΕ ΠΕΡΙΠΤΩΣΗ ΕΛΕΓΧΟΥΜΕ ΤΗΝ ΥΨΟΖΥΓΙΣΗ



Κάνονμε Αριστερή Στροφή στο 17



-23

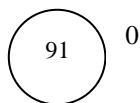


13.12 Θέμα 2 Φεβρουάριος 2021 – Ομάδα Β

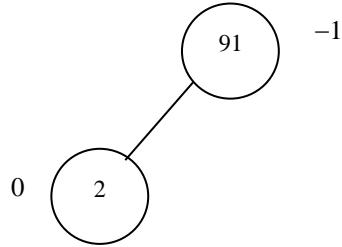
Να σχεδιαστεί βήμα-βήμα το AVL δέντρο που προκύπτει από την εισαγωγή σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων {91, 2, 8, 7, 12, 3, 1, 4, 22, 8}. Διαγράψτε 22 και 7 και δείξτε τη μορφή του δένδρου.

Απάντηση

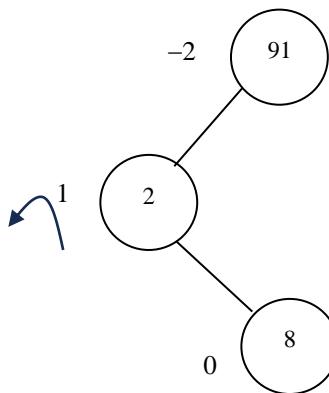
+91



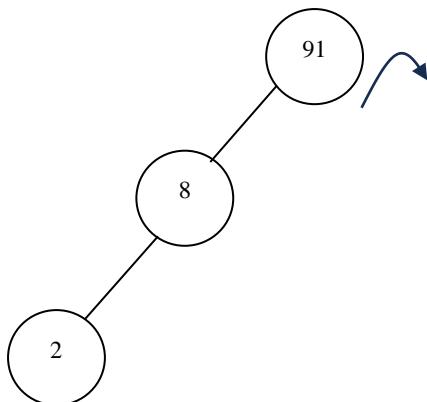
+2



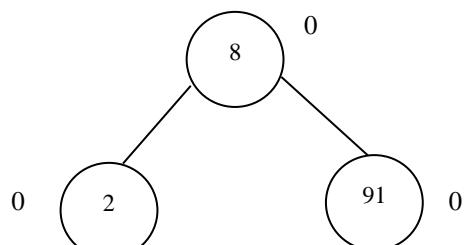
+8



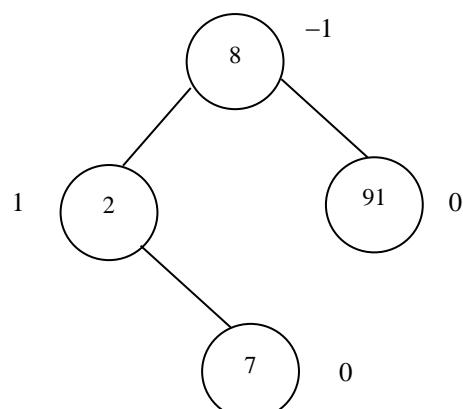
Κάνουμε πρώτα αριστερή στροφή στο 2



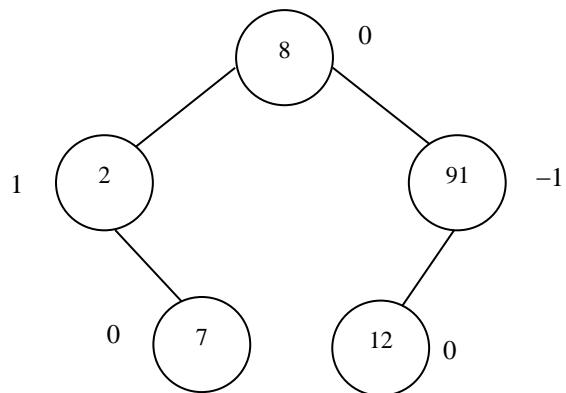
Μετά κάνουμε δεξιά στροφή στο 91



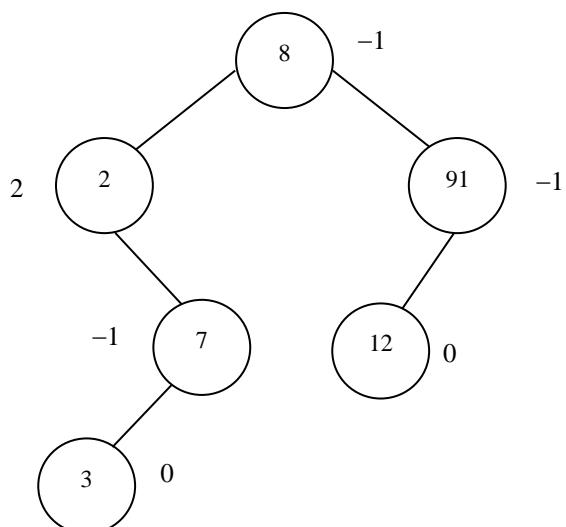
+7



+12

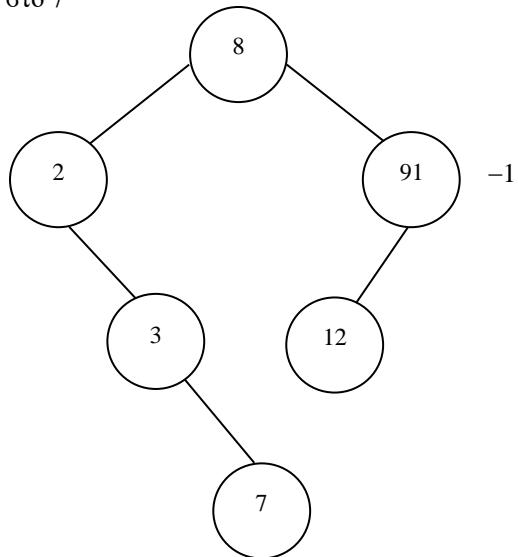


+3

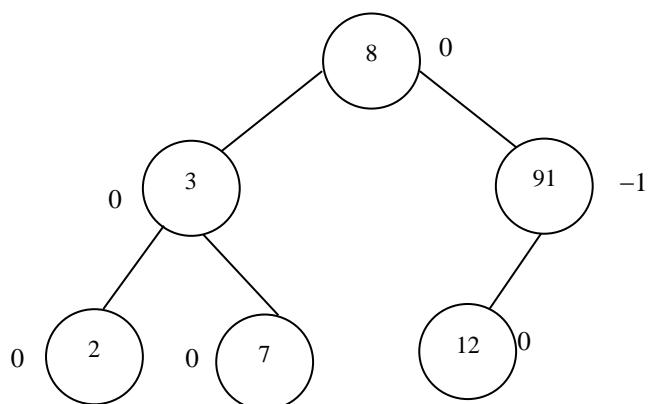


Δομές Δεδομένων –Computer Ανάλυση

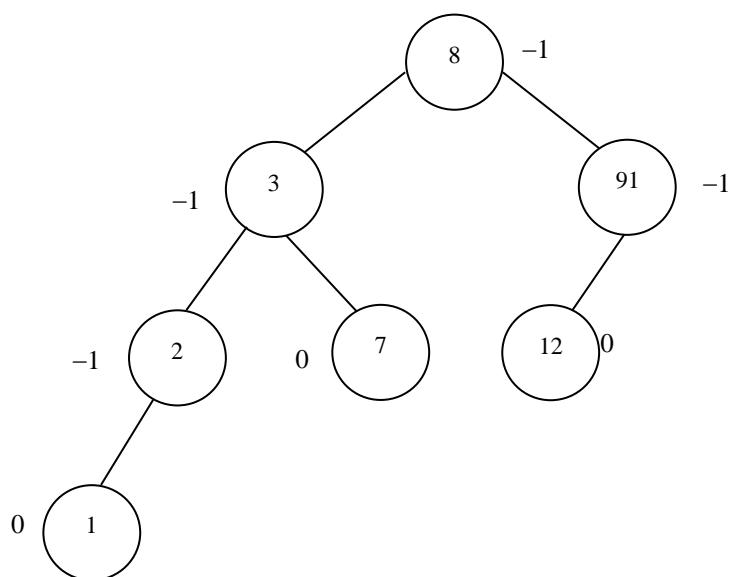
Κάνουμε πρώτα δεξιά στροφή στο 7



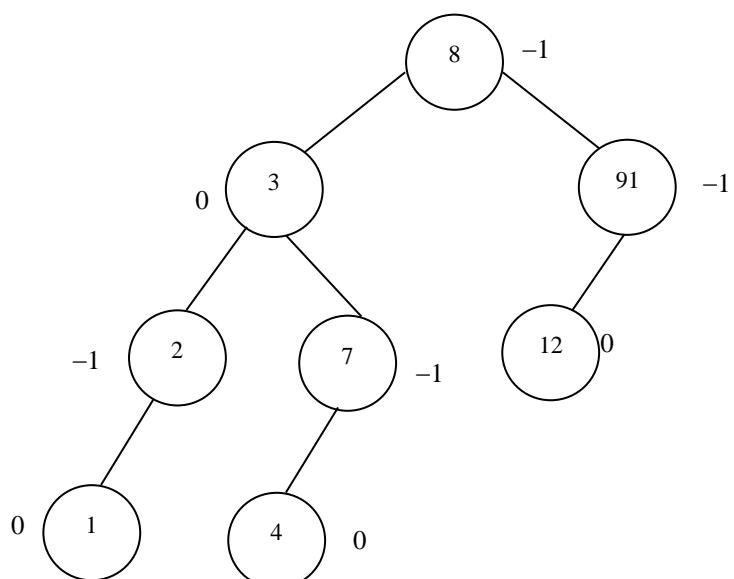
Μετά κάνουμε αριστερή στροφή στο 2



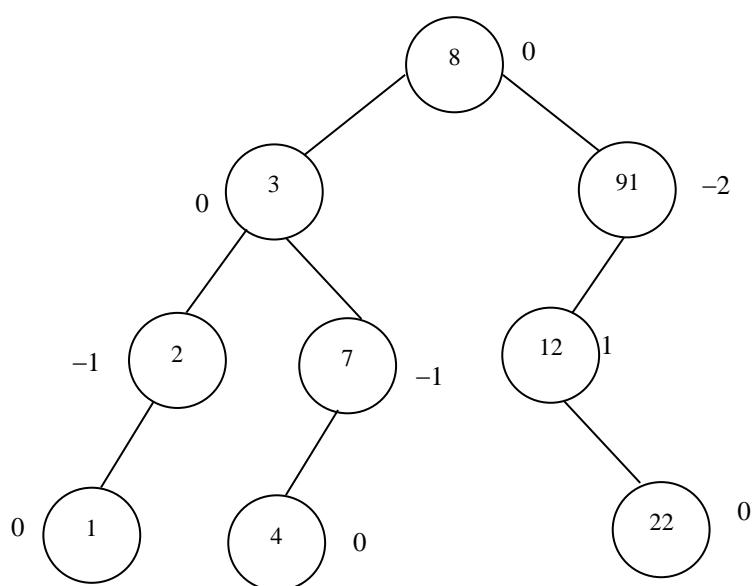
+1



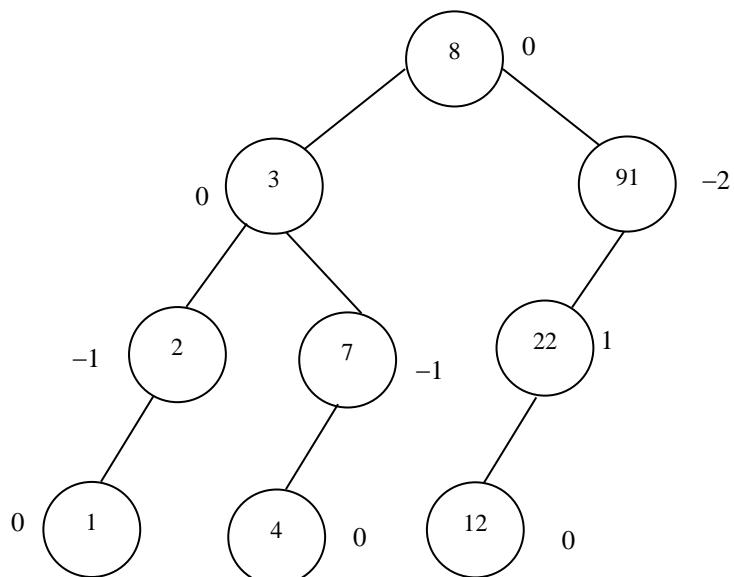
+4



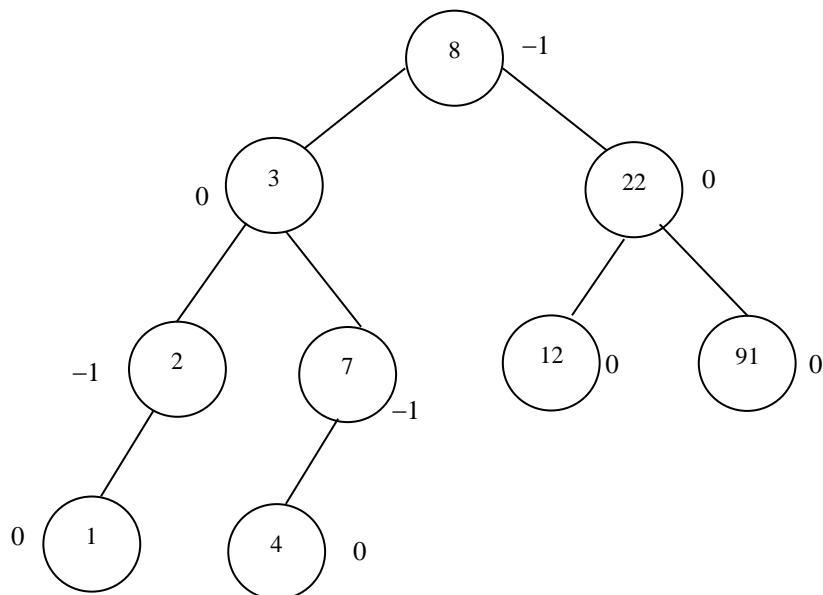
+22



Αριστερή στροφή στο 12

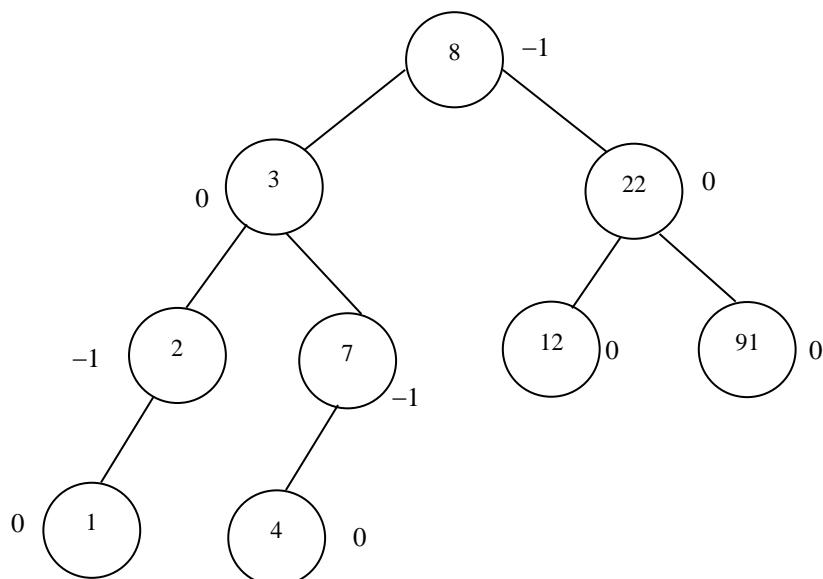


Δεξιά στροφή στο 91

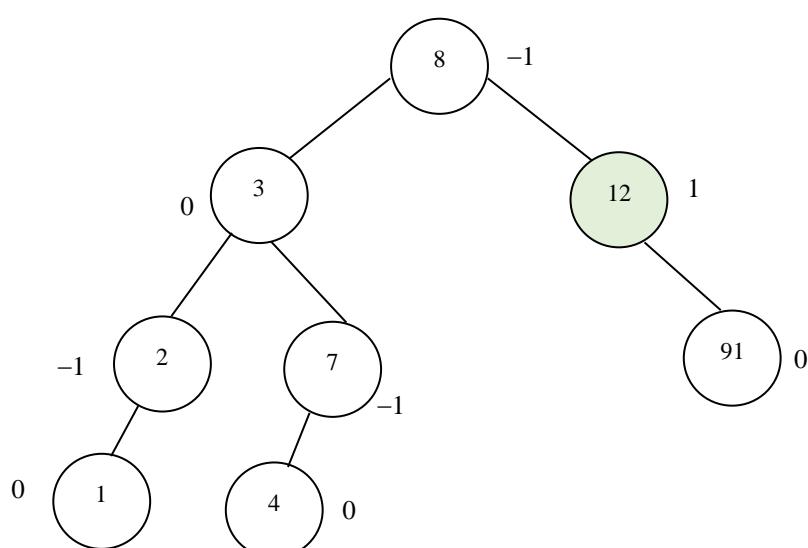


+8

Η τιμή 8 υπάρχει ήδη στο δέντρο, η εισαγωγή δεν γίνεται και το AVL tree μένει ως έχει

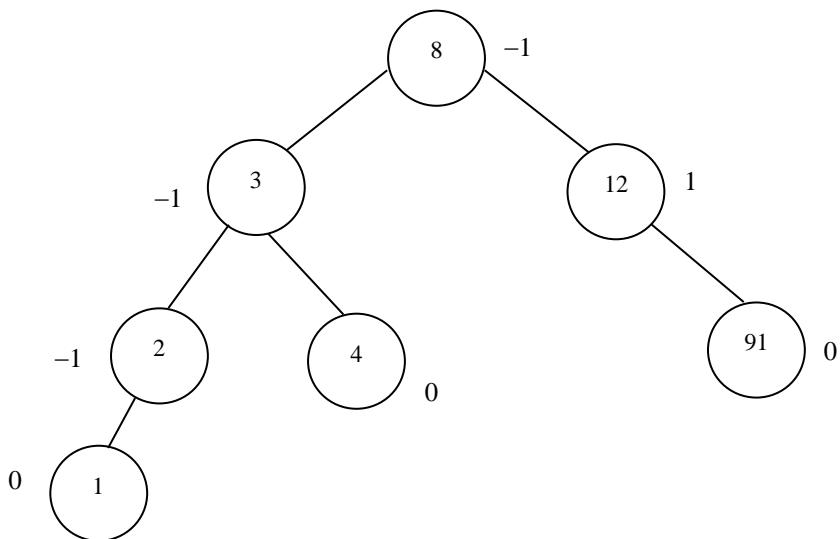


-22



Όταν διαγράφουμε KOMBO, ΒΑΖΟΥΜΕ ΣΤΗ ΘΕΣΗ ΤΟΥ ΤΗ ΜΕΓΑΛΥΤΕΡΗ ΤΙΜΗ ΤΟΥ ΑΡΙΣΤΕΡΟΥ ΥΠΟΔΕΝΤΡΟΥ ΤΟΥ. Εδώ η μεγαλύτερη τιμή του αριστερού υποδέντρου είναι το 12 και το βάζουμε στη θέση του 22

-7



Όταν διαγράφουμε KOMBO, ΒΑΖΟΥΜΕ ΣΤΗ ΘΕΣΗ ΤΟΥ ΤΗ ΜΕΓΑΛΥΤΕΡΗ ΤΙΜΗ ΤΟΥ ΑΡΙΣΤΕΡΟΥ ΥΠΟΔΕΝΤΡΟΥ ΤΟΥ. Εδώ η μεγαλύτερη τιμή του αριστερού υποδέντρου είναι το 12 και το βάζουμε στη θέση του 22

Αυτό είναι το τελικό AVL Δέντρο

13.13 Ερωτήσεις Θεωρίας στα AVL Δέντρα

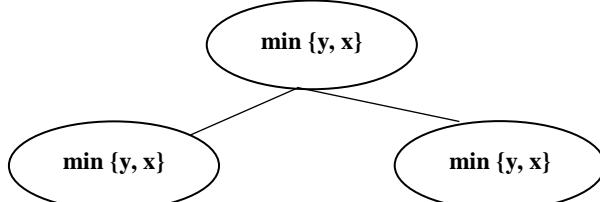
- Έστω ένα μονοπάτι v_0, v_1, \dots, v_k από τη ρίζα v_0 στο φύλλο v_k . Έστω το μέγιστο i τέτοιο ώστε πριν την ένθεση να ισχύει ότι:
$$hb(v_i) = hb(v_{i+1}) = \dots = hb(v_k) = 0 \text{ και } hb(v_{i-1}) = 1$$
- Τότε όταν $i \geq 1$ ο κόμβος v_{i-1} ονομάζεται **Κρίσιμος Κόμβος Ένθεσης (KKE)** και το μονοπάτι v_i, v_{i+1}, \dots, v_k ονομάζεται **Κρίσιμο Μονοπάτι Ένθεσης (KME)**. Πιο απλά KKE είναι ο τελευταίος μη ζυγισμένος κόμβος που συναντάμε κατεβαίνοντας το μονοπάτι από τη ρίζα προς τα φύλλα πριν από μια εισαγωγή
- Κρίσιμος Κόμβος Απόσβεσης (KKA)** είναι ο πρώτος κόμβος στο μονοπάτι από τον πατέρα του διαγραφέντος φύλλου προς τη ρίζα στον οποίο είτε η ζύγιση μετά την απόσβεση έχει μη επιτρεπτή τιμή (± 2) και λόγω αυτής προκαλούνται επανορθωτικές πράξεις είτε δεν απαιτούνται άλλες αλλαγές ζύγισης από εκεί και πάνω
- Κρίσιμο Μονοπάτι Απόσβεσης** ονομάζεται το ονομάζεται το μονοπάτι από το φύλλο που αποσβέστηκε προς τον Κρίσιμο Κόμβο Απόσβεσης

• Οι πράξεις Access, Insert και Delete σε ένα AVL δέντρο με η φύλλα έχουν κόστος $\Theta(\log n)$ στη χειρότερη περίπτωση

- Η εισαγωγή σε ένα AVL δέντρο είναι μια διαδικασία 3 βημάτων:

i) εύρεση της κατάλληλης θέσης στο δέντρο για την εισαγωγή του φύλλου με τιμή x. Πρόκειται για τη διαδικασία Access(x) μετά το πέρας της οποίας αν βρεθεί ότι η τιμή x υπάρχει ήδη στο δέντρο, η εισαγωγή δεν γίνεται, αλλιώς η αναζήτηση τερματίζει σε ένα φύλλο έστω $y \neq XX$

ii) Αντικατάσταση του y με το δέντρο



iii) Κάνουμε τις απαιτούμενες αλλαγές αν χρειάζεται για επαναφορά της ζύγισης στο δέντρο

- Η διαγραφή κόμβου σε ένα AVL δέντρο είναι μια διαδικασία των ακόλουθων 3 βημάτων:

Αρχικά, εφαρμόζουμε τον αλγόριθμο διαγραφής όπως στο απλό δυαδικό δένδρο αναζήτησης, διαγράφοντας τον κόμβο x' όπου:

- 1) $x' = x$ αν ο x είναι φύλλο
- 2) $x' =$ παιδί του x αν ο x έχει μόνο ένα παιδί
- 3) $x' =$ διάδοχος του x αν ο x έχει 2 παιδιά

Στις περιπτώσεις 2 και 3 αντιγράφουμε τα περιεχόμενα του x' στον x πριν τη διαγραφή. Κατόπιν, ελέγχουμε τις τιμές του hb: Το hb (υψοζύγιση) του κόμβου – γονέα z του κόμβου x' που διαγράφεται αλλάζει

Αν το b f(z) του κόμβου -γονέα z του κόμβου x' που διαγράφεται αλλάζει

- από 0 σε +1 ή σε -1, τότε ο αλγόριθμος τερματίζει.
- από +1 ή -1 σε 0, το ύψος του z μειώνεται και άρα και το bf άλλων προγόνων του x' ενδεχομένως αλλάζει.
- από +1 ή -1 σε +2 ή -2, τότε γίνεται μία ή περισσότερες περιστροφές.

14 (a,b) ΔΕΝΤΡΑ

14.1 Χαρακτηριστικά (a, b) δέντρων

- Τα (a, b) δέντρα είναι **φυλλοπροσανατολισμένα δέντρα (τα κλειδιά βρίσκονται μόνο στα φύλλα)**
- Κάθε εσωτερικός κόμβος έχει το **λιγότερο α και το περισσότερο b παιδιά** δηλ. κάθε εσωτερικός κόμβος έχει τόσα κλειδιά (τιμές) όσα είναι τα παιδιά του μείον 1 δηλ. **κλειδιά =παιδιά–1**. Τα κλειδιά είναι οι εσωτερικές τιμές ενός κόμβου
- **Τα κλειδιά σε κάθε κόμβο είναι ταξινομημένα**
- **Κάθε νέα τιμή εισάγεται σε νέο φύλλο**
- **To (a,b) είναι δέντρο αναζήτησης** δηλ για κάθε κόμβο X
 - Όλοι οι κόμβοι στο αριστερό υποδέντρο του X έχουν τιμή $\leq X$
 - Όλοι οι κόμβοι στο δεξιό υποδέντρο του X περιέχουν τιμή $> X$
- Όλα τα **φύλλα είναι στο ίδιο βάθος**
- **To ύψος του δέντρου είναι O(logn)**
- Όταν γίνεται διαγραφή φύλλου, ο κόμβος που το περιείχε παραμένει εφόσον ο γείτονας του (είτε ο αριστερός είτε ο δεξιός) είναι πλούσιος
- Όταν $b=2a-1$ τότε το (a, b) tree ονομάζεται **B-tree**
- **Οι πράξεις Access, Insert και Delete σε ένα (a, b) δέντρο εκτελούνται σε χρόνο $O(h)=O(\log |S|)$ όπου S είναι το σύνολο τιμών που θα τοποθετήσουμε στο δέντρο**
- Όταν γίνεται διάσπαση ενός κόμβου λόγω υπερχείλισης ο αριστερός πατέρας θα πάρει το κάτω ακέραιο μέρος του κλάσματος παιδιά/2 δηλ. το **[παιδιά/2]** ενώ ο δεξιός πατέρας θα πάρει το πάνω ακέραιο μέρος του κλάσματος παιδιά/2 δηλ. το **[παιδιά/2]**

14.2 Φροντιστήριο 2018

Να σχεδιαστεί βήμα-βήμα το (2, 4)-δέντρο που προκύπτει από την εισαγωγή με τη σειρά των εξής στοιχείων: [15 12 9 45 21 67 10].

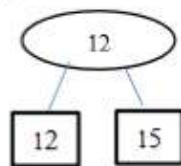
Μετά διαγράψτε τα κλειδιά **10, 12, 15**

Απάντηση

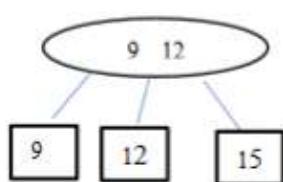
Βήμα 1



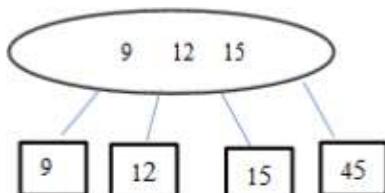
Βήμα 2



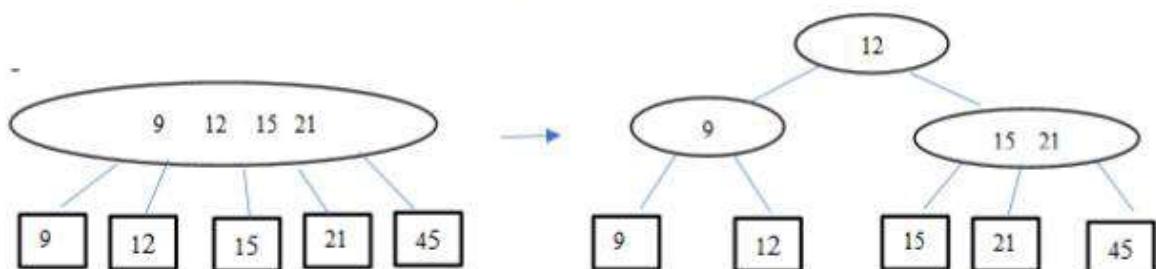
Βήμα 3



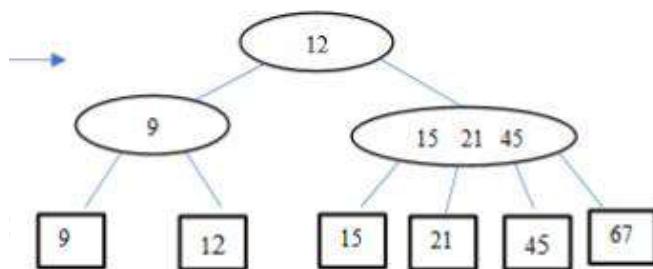
Βήμα 4



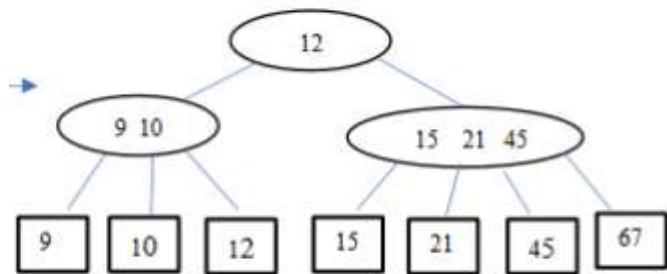
Βήμα 5



Βήμα 6

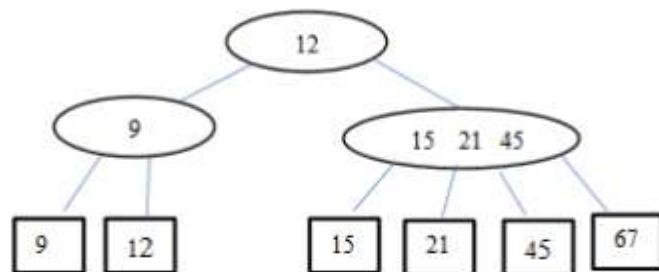


Βήμα 7

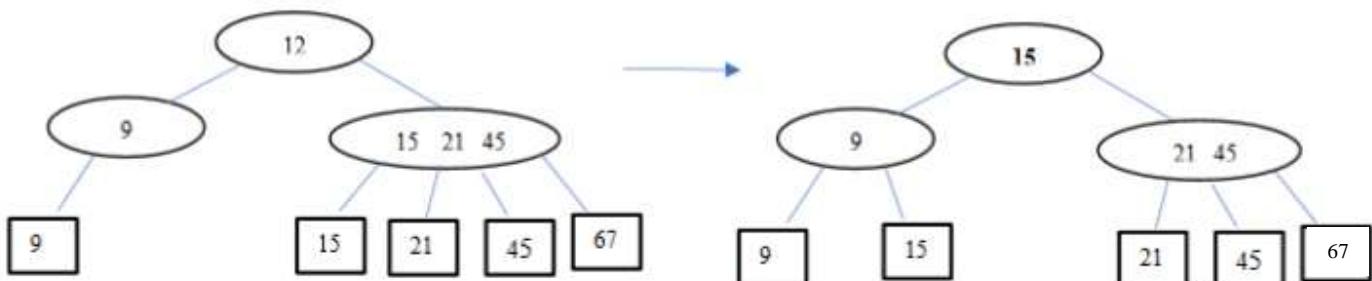


ΖΗΤΟΥΜΕΝΟ: διαγράψτε το στοιχείο 10, 12, 15

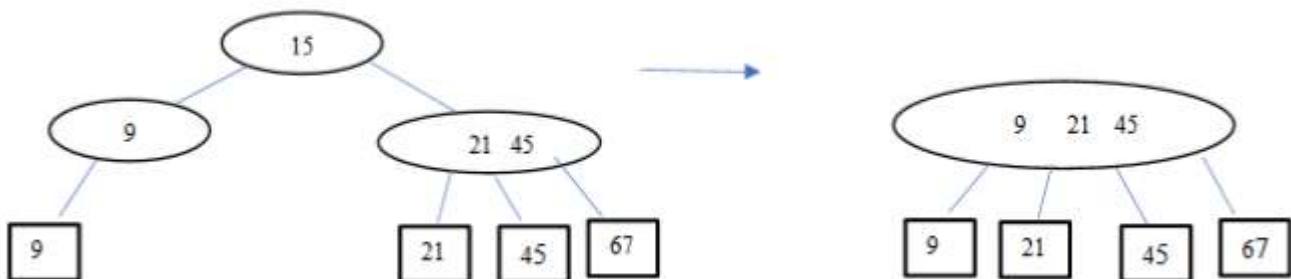
ΔΙΑΓΡΑΦΗ 10



ΔΙΑΓΡΑΦΗ 12



ΔΙΑΓΡΑΦΗ 15



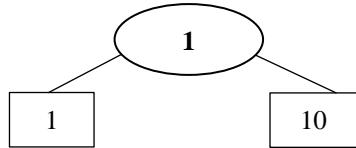
14.3 Θέμα 7 Ιούνιος 2015

Να σχεδιαστεί βήμα-βήμα το (a, b) δέντρο που προκύπτει από την εισαγωγή με τη σειρά σε ένα αρχικά άδειο δέντρο των εξής στοιχείων $[+10, +1, +5, +6, +20, +19, +16, +13, +12, +25]$ αν το $a=2$ και το $b=3$. Στη συνέχεια διαγράψτε τα στοιχεία **5, 25**.

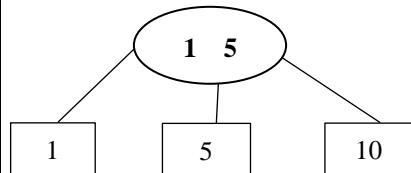
Απάντηση

+10 

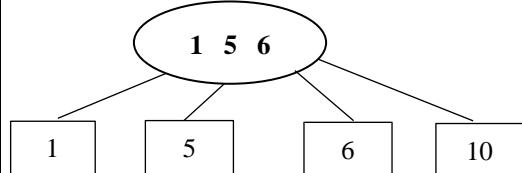
+1



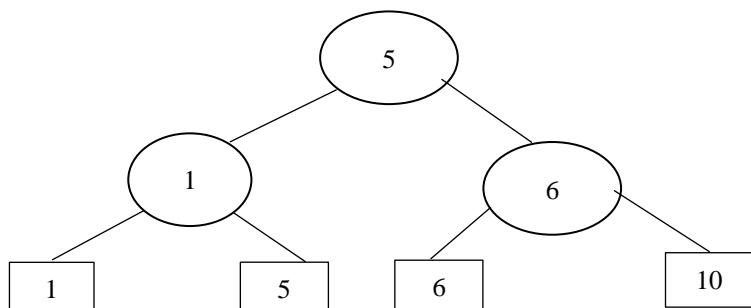
+5



+6

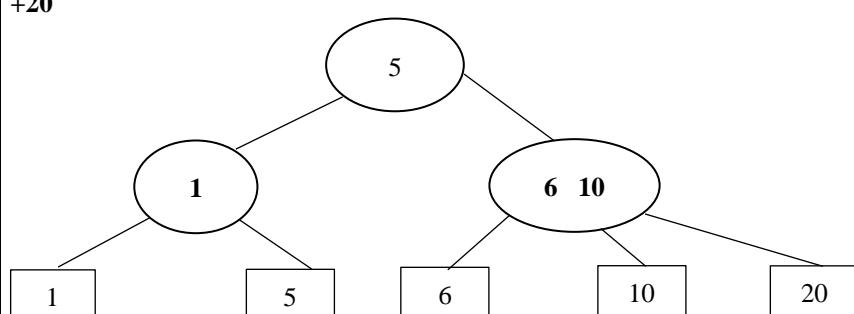


Επειδή το δέντρο είναι $(2, 3)$ θα γίνει διάσπαση του κόμβου της ρίζας και μετά τη διάσπαση του κόμβου θα έχουμε:

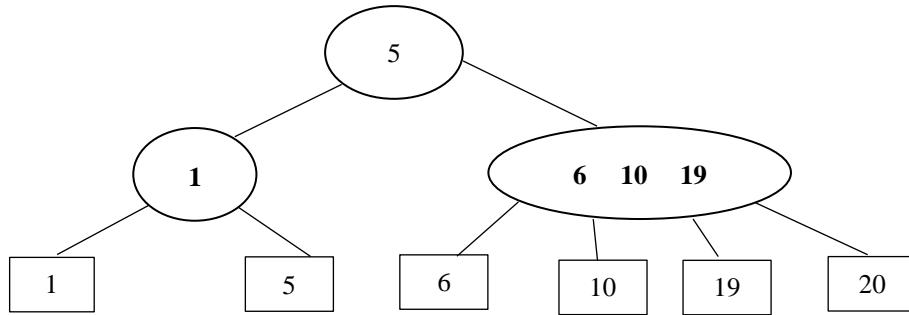


Ως ρίζα θέτουμε τη μεγαλύτερη τιμή φύλλου του αριστερού υποδέντρου

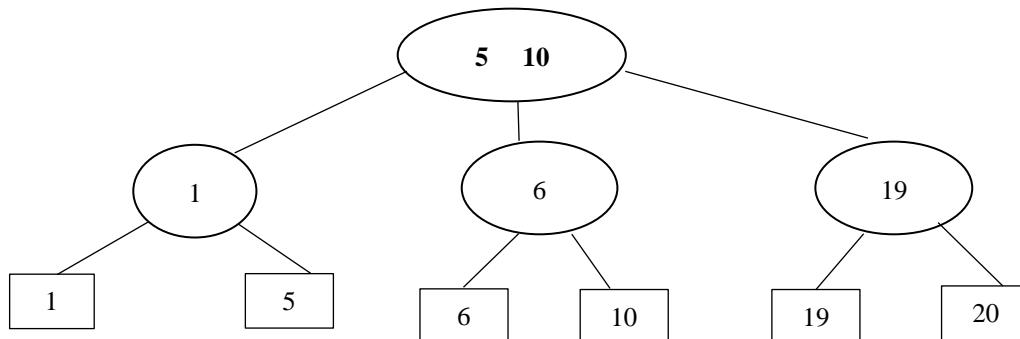
+20



+19

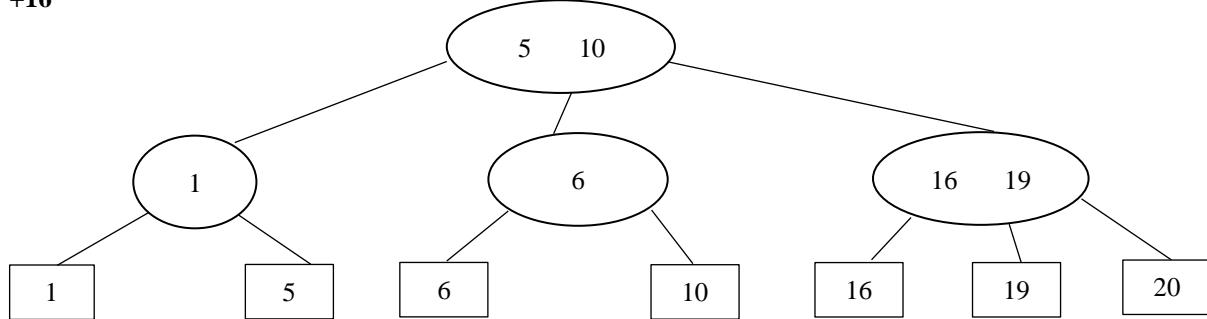


Μετά τη διάσπαση του κόμβου (6, 10, 19) θα έχουμε:

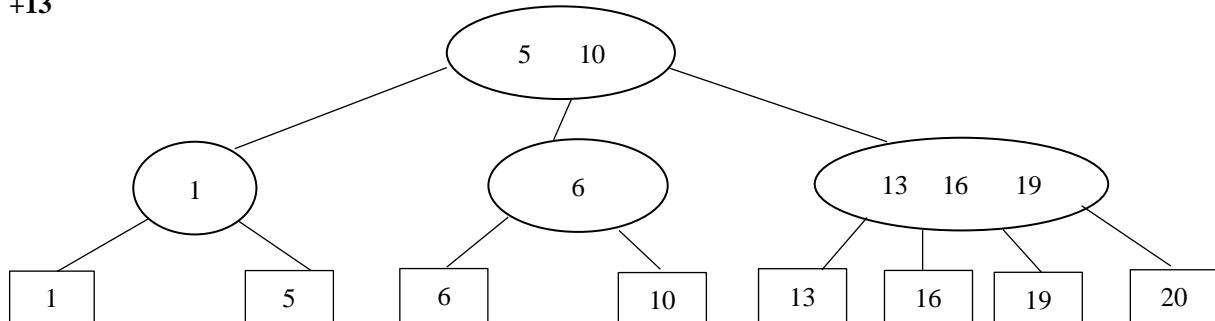


Ως ρίζα θέτουμε τη μεγαλύτερη τιμή φύλλου του αριστερού υποδέντρου και τη μεγαλύτερη τιμή φύλλου του μεσαίου υποδέντρου

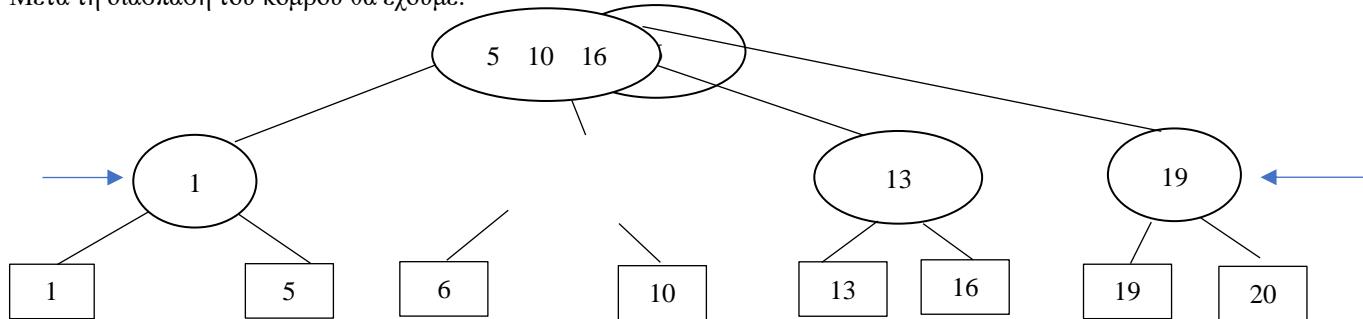
+16



+13

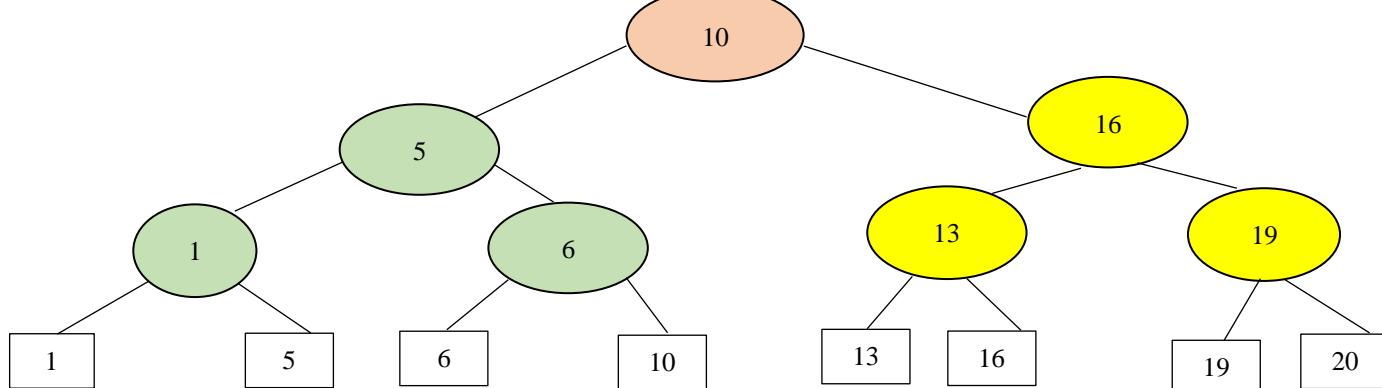


Μετά τη διάσπαση του κόμβου θα έχουμε:



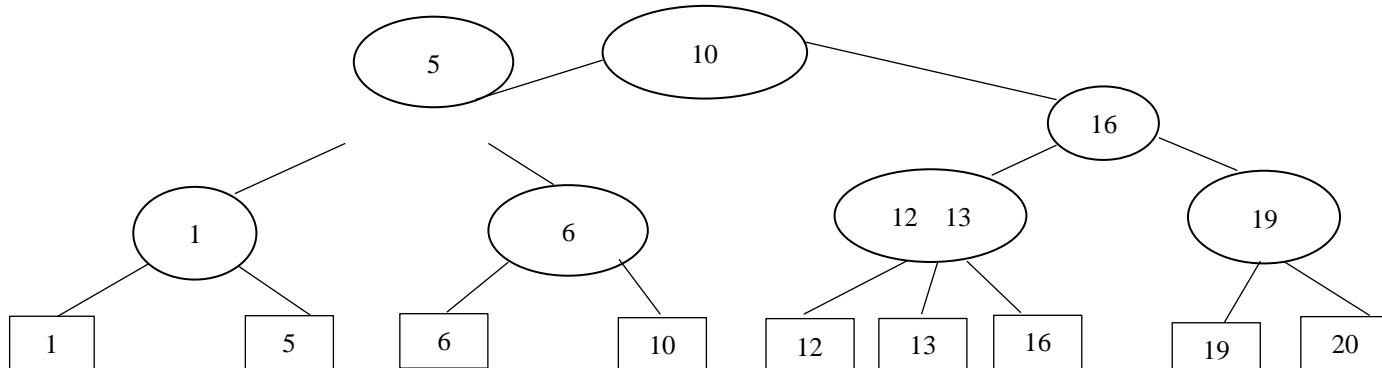
ΑΥΤΟ ΔΕΝ ΕΠΙΤΡΕΠΕΤΑΙ ΔΙΟΤΙ ΕΙΝΑΙ ΔΕΝΤΡΟ 2, 3

Δομές Δεδομένων –Computer Ανάλυση

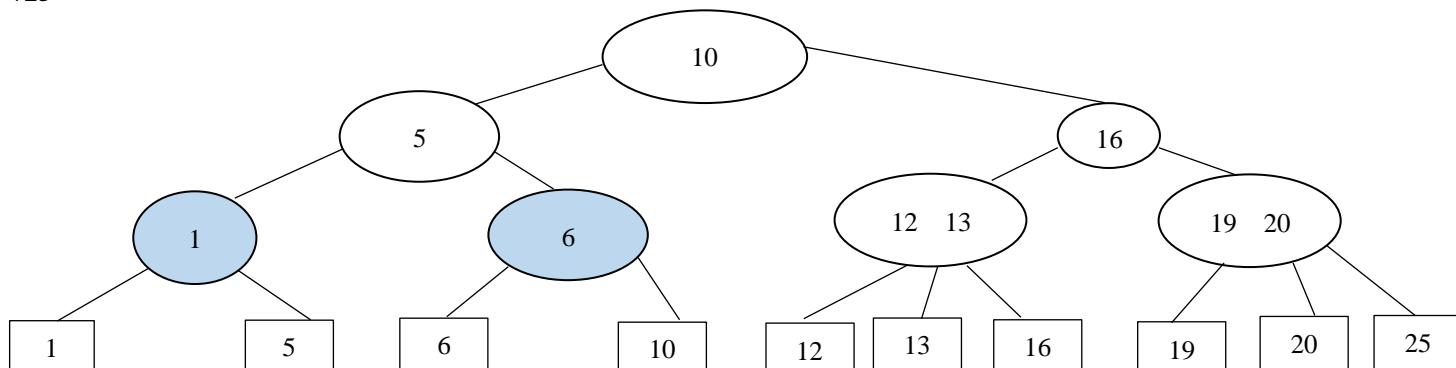


Το δέντρο λόγω υπερχείλισης ανεβαίνει και πάλι επίπεδο. Το κλειδί της ρίζας είναι το μεγαλύτερο φύλλο του αριστερού υποδέντρου

+12



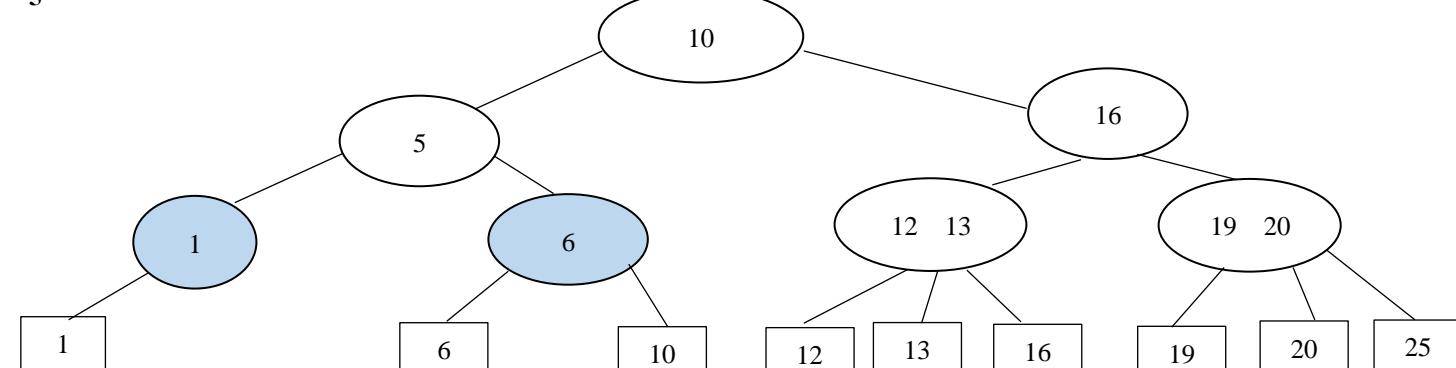
+25

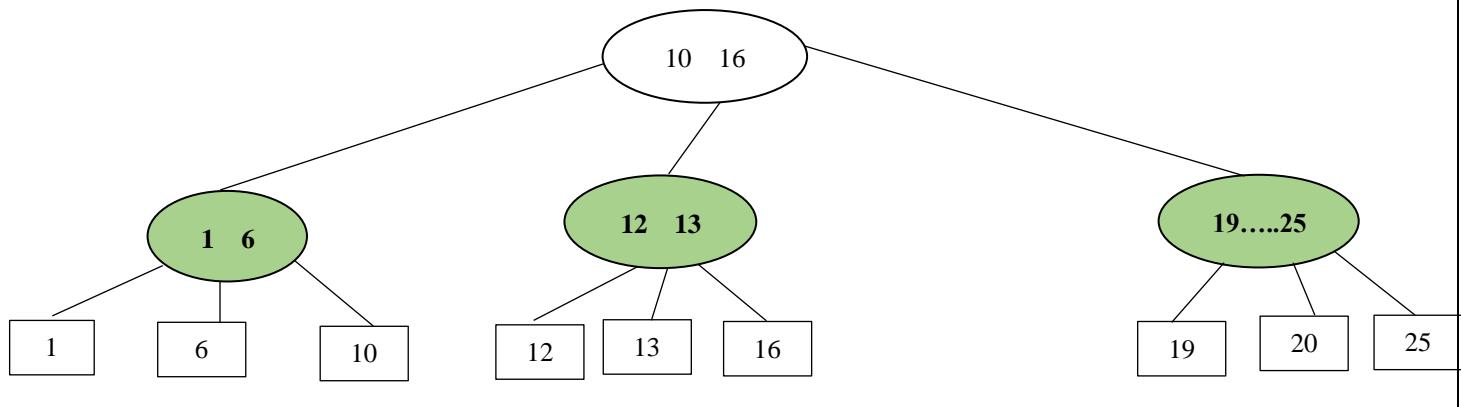


Παρατήρηση για Διαγραφές

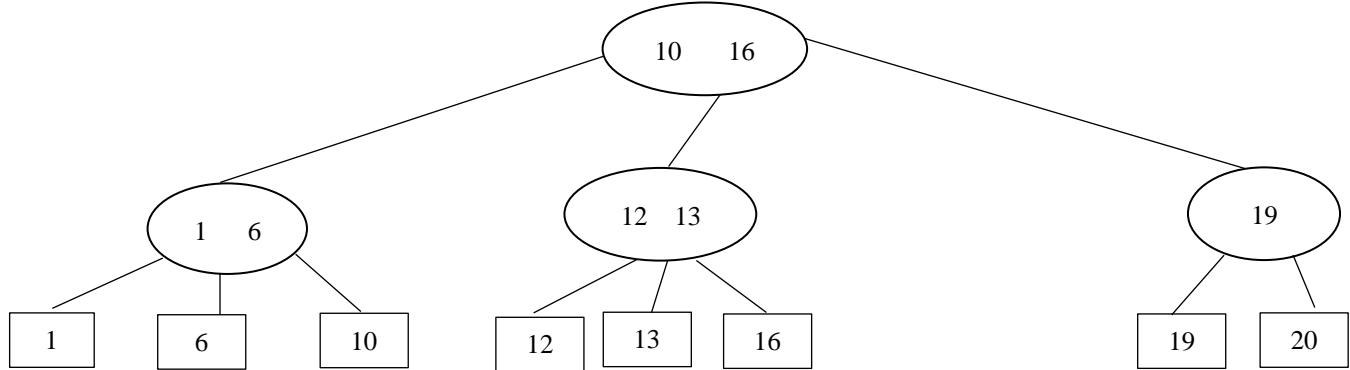
Όταν κάνουμε διαγραφή φύλλου από το (α, β) δέντρο, εξετάζουμε τον αριστερό ή το δεξιό γείτονα. Αν είναι πλούσιος δηλ. αν έχει τουλάχιστον $\alpha+1$ παιδιά τότε δανείζει κλειδιά στον κόμβο που είναι ελλιπής προκειμένου το δέντρο να διατηρήσει τη δομή του. Αν είναι φτωχός γείτονας τότε τα κλειδιά του ελλιπή κόμβου προστίθενται στο φτωχό γείτονα. Με τη διαγραφή ενός φύλλου ενδέχεται να μειωθούν τα επίπεδα του δέντρου.

-5





-25



14.4 Θέμα 4 Ιούνιος 2023

Να σχεδιαστεί βήμα-βήμα το (2, 3) δέντρο που προκύπτει από την εισαγωγή σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων:

[17, 12, 11, 2, 30, 41, 20, 32, 19, 6].

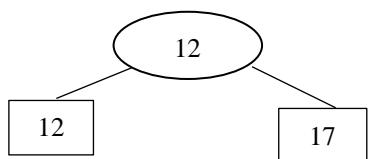
Στη συνέχεια να διαγράψετε βήμα-βήμα τα στοιχεία {11, 2}.

Απάντηση

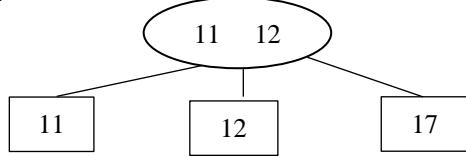
+17



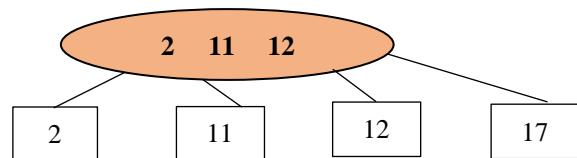
+12



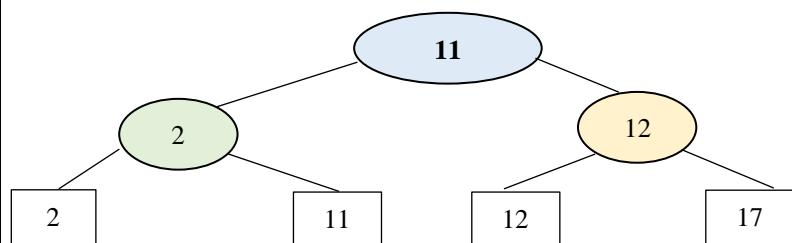
+11



+2

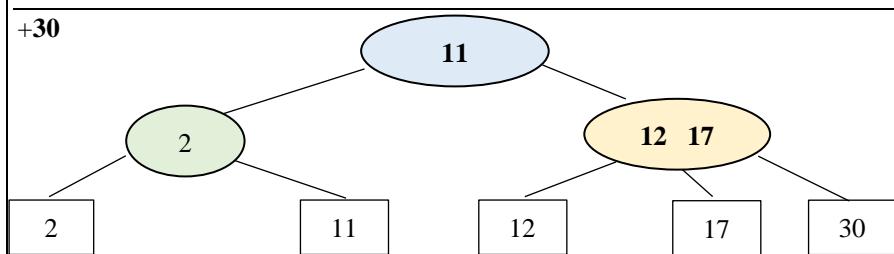


Με την προσθήκη του 2 συμβαίνει υπερχείλιση διότι το δέντρο είναι (2, 3). Το δέντρο ανεβαίνει επίπεδο δηλ. τα επίπεδα του δέντρου αυξάνονται κατά 1. Μετά τη διάσπαση της ρίζας θα έχουμε:



Στη ρίζα βάζουμε πάντα ως κλειδί τη μεγαλύτερη τιμή φύλλου του αριστερού υποδέντρου

+30



+41

11

12 17

2

11

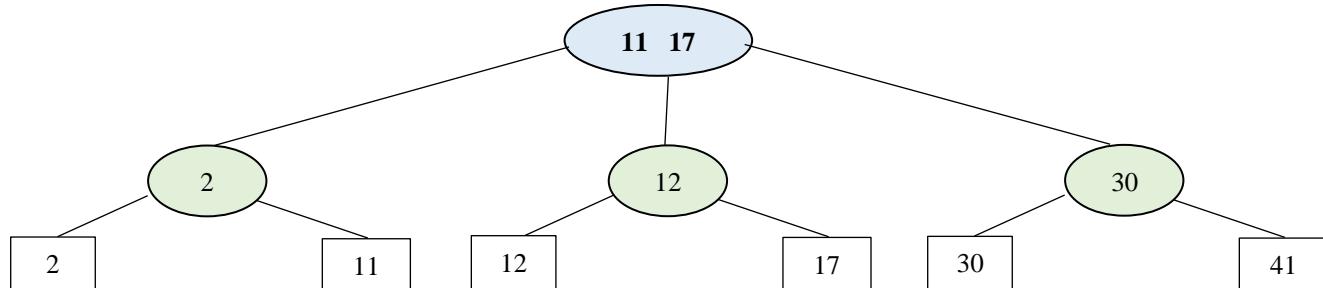
12

17

30

41

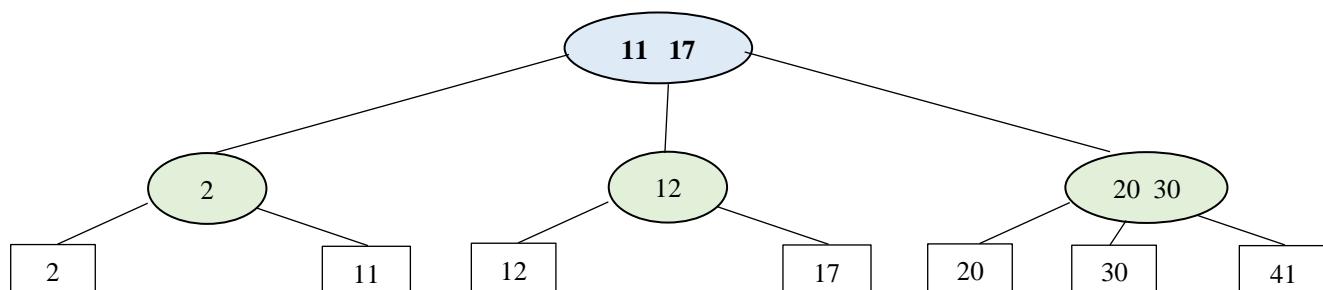
Με την προσθήκη του 41 συμβαίνει υπερχείλιση διότι το δέντρο είναι (2, 3) και ο κόμβος που μπαίνει το 41 διασπάται σε δύο



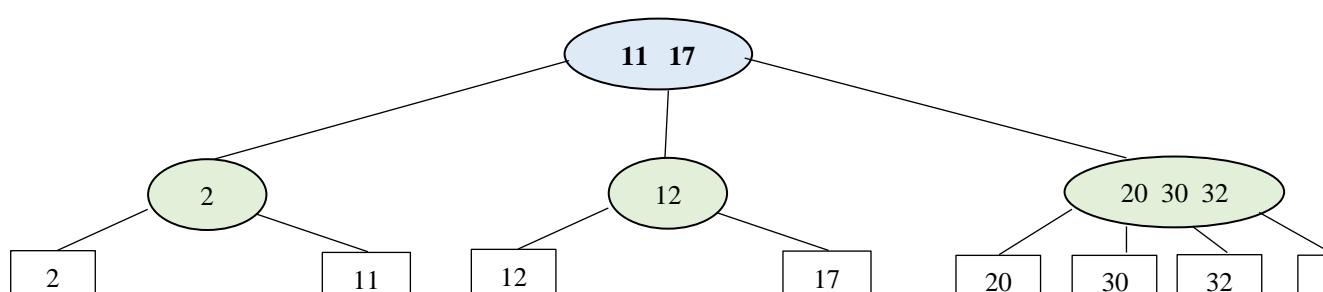
Ως ρίζα θέτουμε τη μεγαλύτερη τιμή φύλλου του αριστερού υποδέντρου και τη μεγαλύτερη τιμή φύλλου του μεσαίου υποδέντρου

+20

Το 20 συγκρίνεται με τη ρίζα. Επειδή το (2, 3) δέντρο είναι δέντρο αναζήτησης (δηλ. $\Delta\Pi \leq \text{Ρίζα} < \Delta\Pi$) αυτό σημαίνει ότι επειδή το $20 > 17$ τοποθετείται στο υποδέντρο δεξιά από το 17.

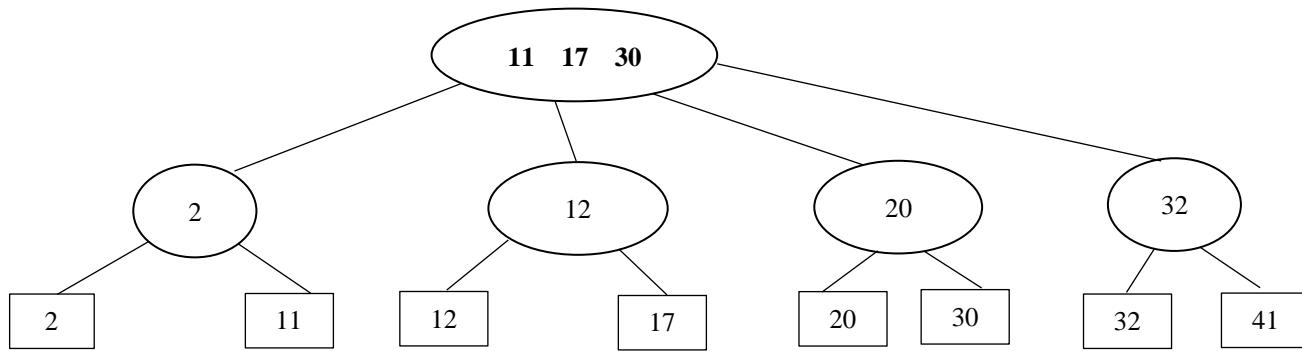


+32

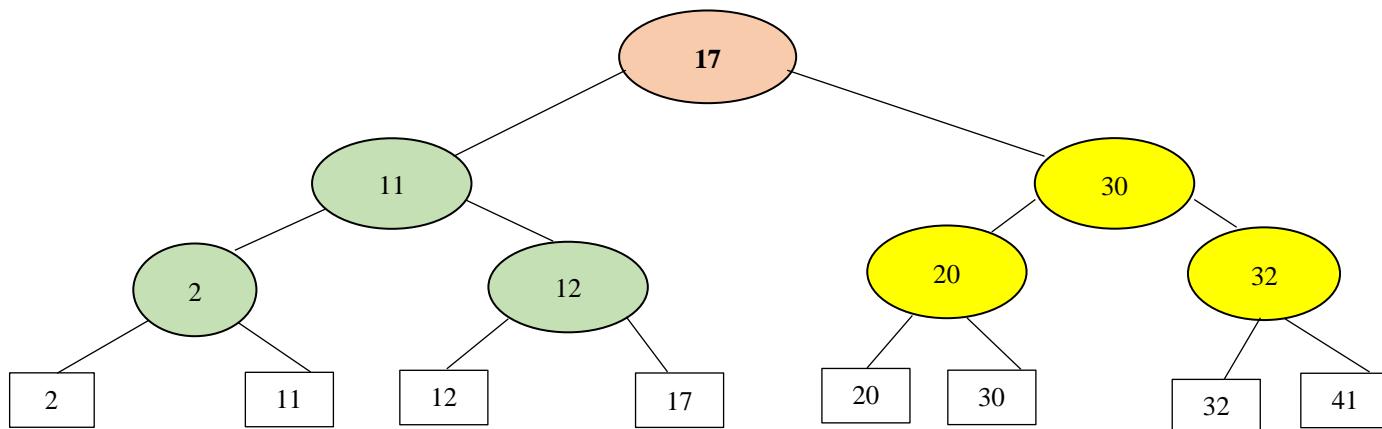


Με την προσθήκη του 32 συμβαίνει υπερχείλιση διότι το δέντρο είναι (2, 3) και ο κόμβος που μπαίνει το 32 διασπάται σε δύο

Μετά τη διάσπαση του κόμβου θα έχουμε:



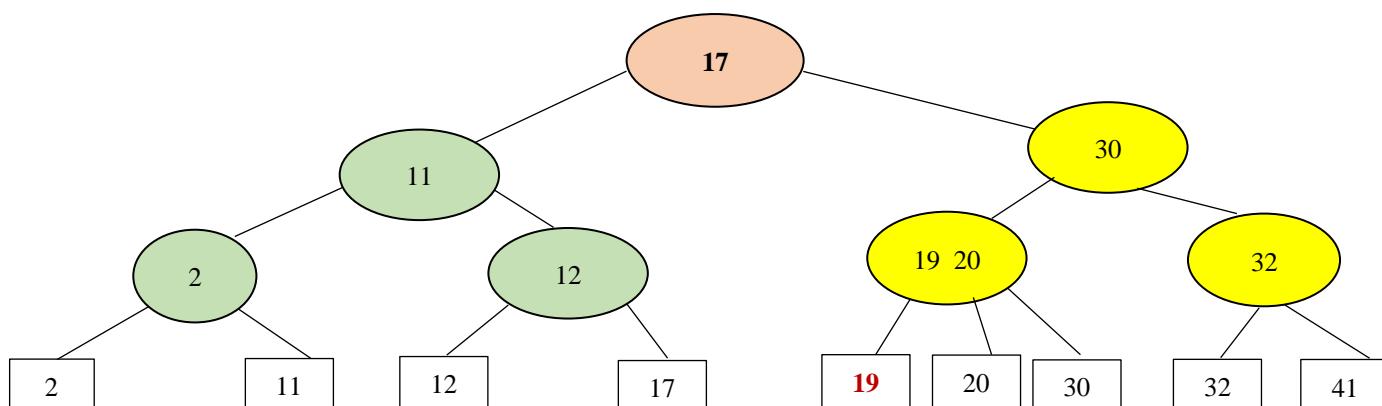
ΑΥΤΟ ΔΕΝ ΕΠΙΤΡΕΠΕΤΑΙ ΔΙΟΤΙ ΕΙΝΑΙ ΔΕΝΤΡΟ 2,3



Το δέντρο λόγω υπερχείλισης ανεβαίνει ξανά επίπεδο. Το κλειδί στη ρίζα είναι το μεγαλύτερο φύλλο του αριστερού υποδέντρου

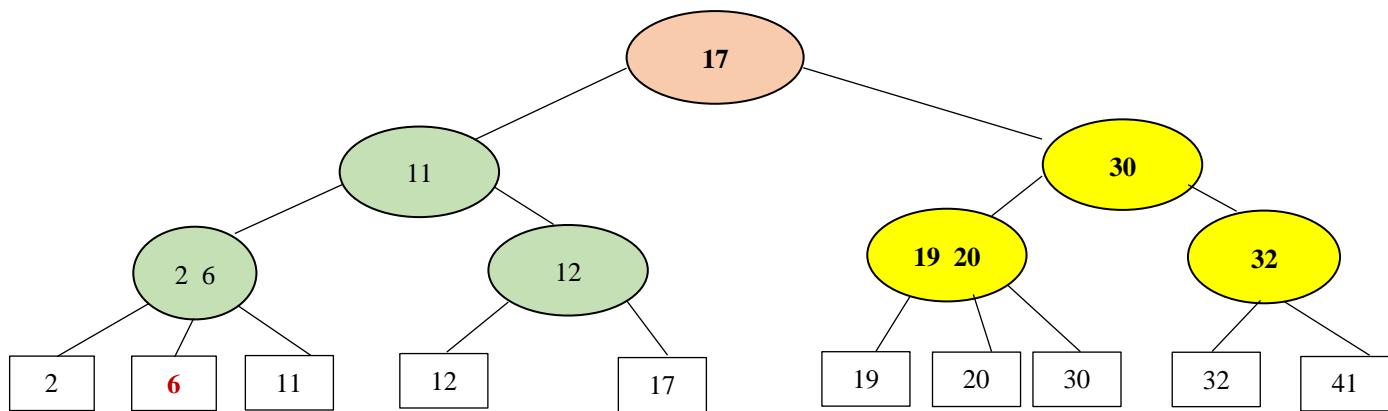
+19

Το 19 συγκρίνεται με τη ρίζα. Επειδή το (2, 3) δέντρο είναι δέντρο αναζήτησης (δηλ. $\Delta\Pi \leq \text{Ρίζα} < \Delta\Pi$) αυτό σημαίνει ότι επειδή το $19 > 17$ εξετάζεται το υποδέντρο δεξιά από το 17. Εκεί συγκρίνεται με το 30 και επειδή $19 < 30$ τοποθετείται ως φύλλο στο υποδέντρο αριστερά από το 30.



+6

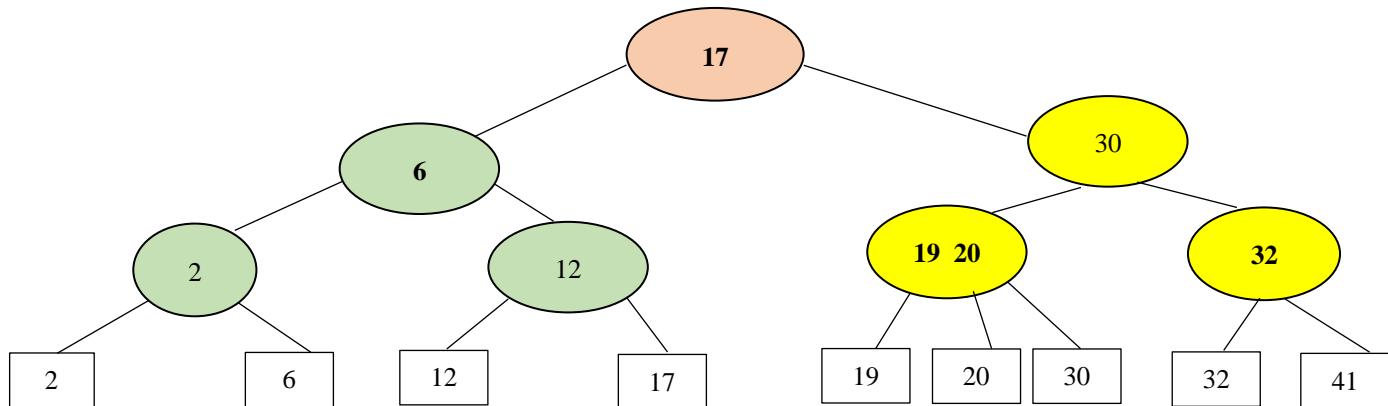
Το 6 συγκρίνεται με τη ρίζα. Επειδή το (2, 3) δέντρο είναι δέντρο αναζήτησης (δηλ. $\text{ΑΠ} \leq \text{Ρίζα} < \Delta\text{Π}$) αυτό σημαίνει ότι επειδή το $6 < 17$ εξετάζεται το υποδέντρο αριστερά από το 17. Εκεί συγκρίνεται με το 11 και επειδή $6 < 11$ τοποθετείται ως φύλλο στο υποδέντρο αριστερά από το 11.



Διαγραφές

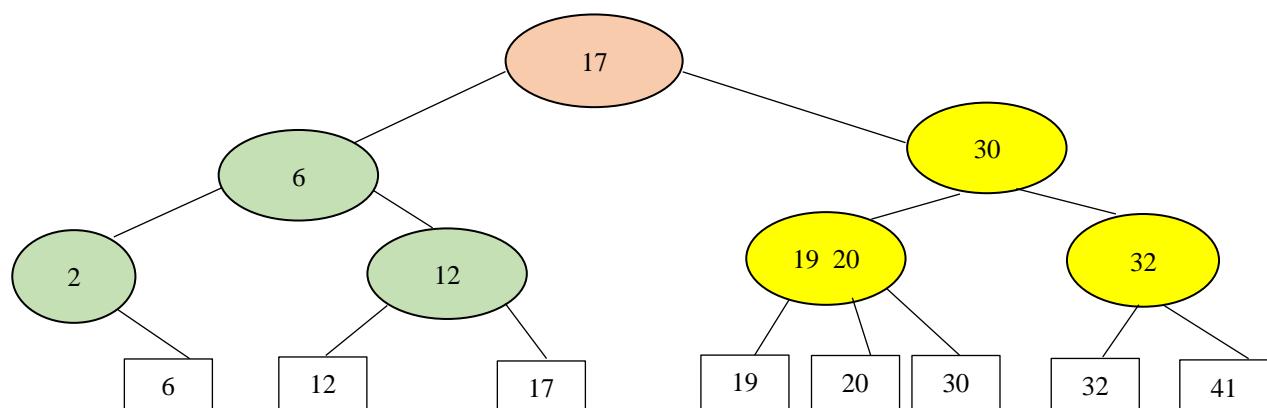
-11

Το 11 διαγράφεται ως φύλλο και ο κόμβος που το περιείχε θα μείνει με 2 φύλλα. Αυτό όμως δεν παραβιάζει τη δομή του (2, 3) δέντρου καθώς κάθε κόμβος έχει (2, 3) παιδιά



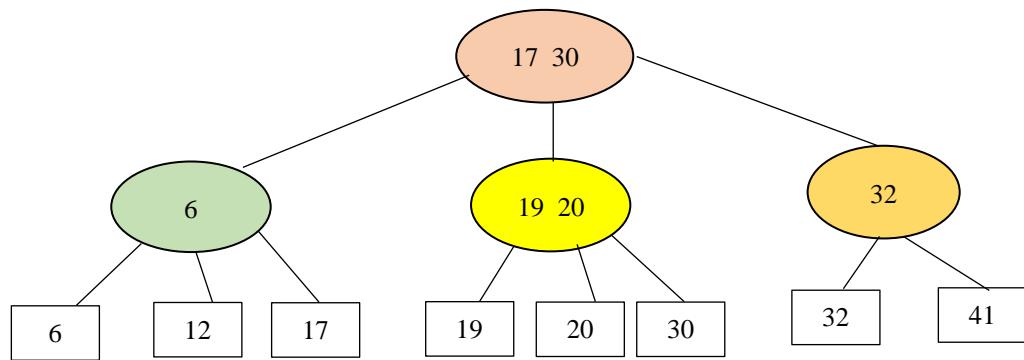
-2

Το 2 διαγράφεται ως φύλλο και ο κόμβος που το περιείχε θα μείνει με 1 φύλλο. Αυτό όμως παραβιάζει τη δομή του (2, 3) δέντρου καθώς κάθε κόμβος έχει τουλάχιστον 2 παιδιά



Δομές Δεδομένων –Computer Ανάλυση

Ο γείτονας δεξιά από το 6 είναι φτωχός και δεν μπορεί να δανείσει κλειδιά. Γιαυτό το 6 ενώνεται σε ένα κόμβο μαζί με 12 και το 17



Το δέντρο λόγω υποχείλισης κατεβαίνει επίπεδο. Αυτό είναι το τελικό 2-3 Δέντρο

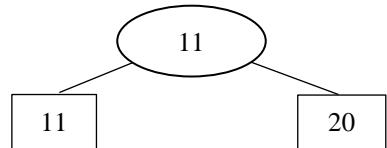
14.5 Θέμα 7 Ιούνιος 2016

Να σχεδιαστεί βήμα-βήμα το (a, b) δέντρο που προκύπτει από την εισαγωγή σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων [11, 20, 16, 7, 25, 2, 12, 13, 9, 21, 3] αν το **a=2 και το b=3**. Μετά να γίνει διαγραφή του 7 και του 11.

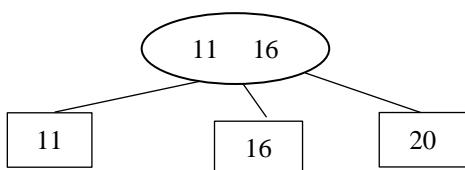
Απάντηση



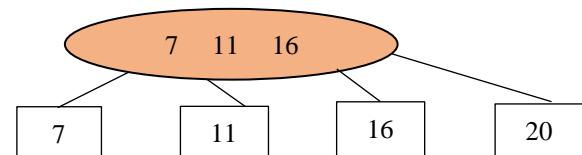
+20



+16

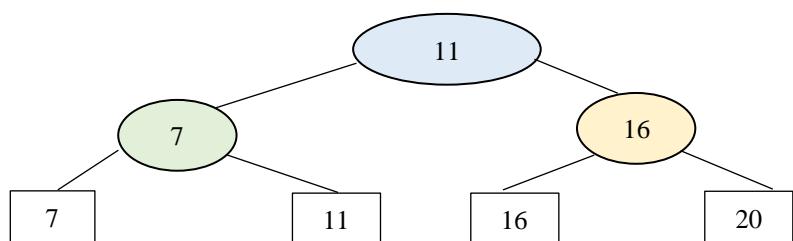


+7



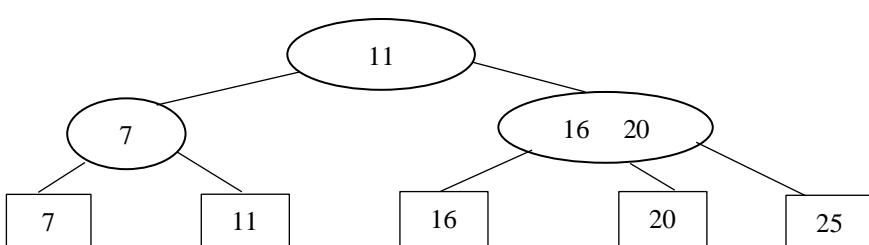
Τώρα συμβαίνει υπερχείλιση. Το δέντρο ανεβαίνει επίπεδο δηλ. τα επίπεδα του δέντρου αυξάνονται κατά 1.

Μετά τη διάσπαση της ρίζας θα έχουμε:

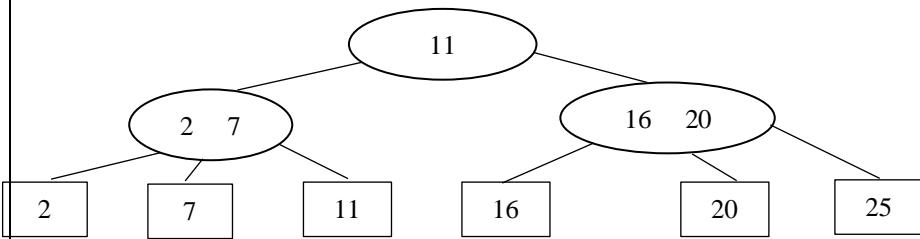


Στη ρίζα βάζουμε ως κλειδί τη μεγαλύτερη τιμή του αριστερού υποδέντρου

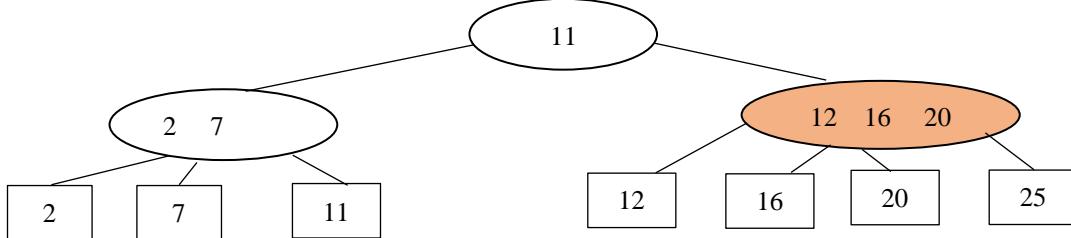
+25



+2

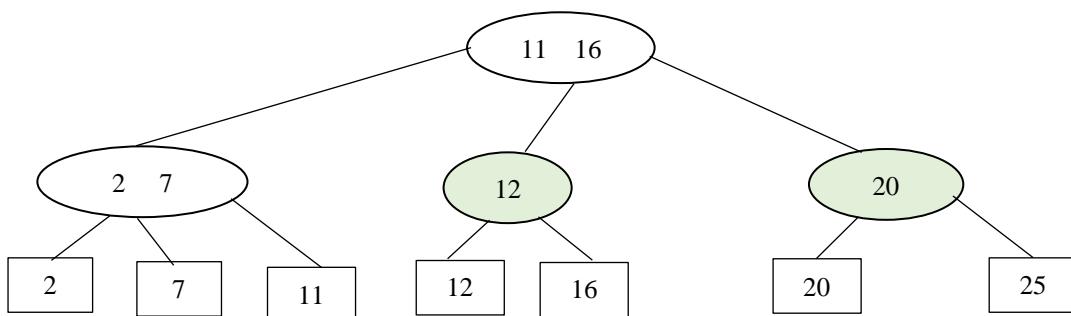


+12

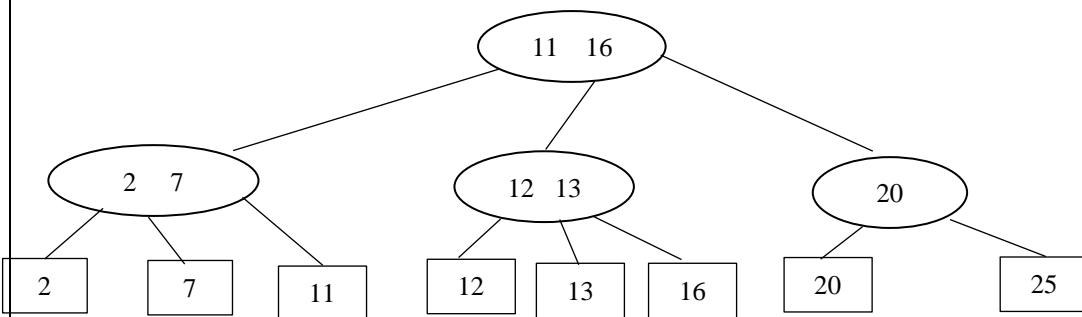


Τώρα συμβαίνει υπερχείλιση. Το δέντρο ανεβαίνει επίπεδο δηλ. τα επίπεδα του δέντρου αυξάνονται κατά 1.

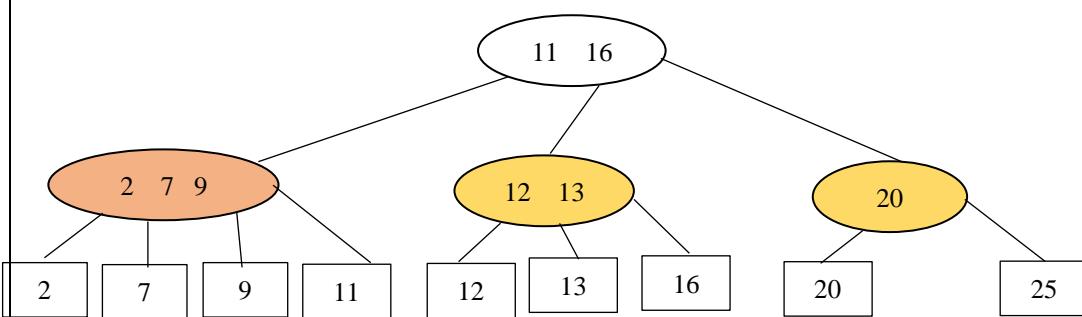
Μετά τη διάσπαση του κόμβου (12, 16, 20) θα έχουμε:



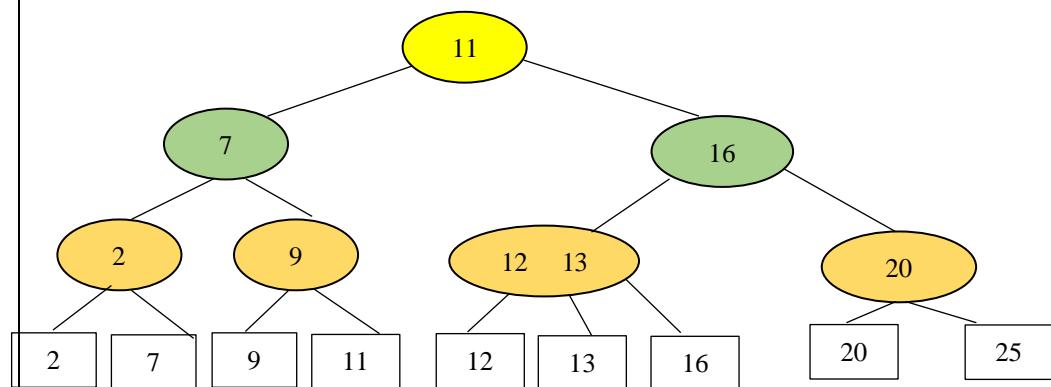
+13



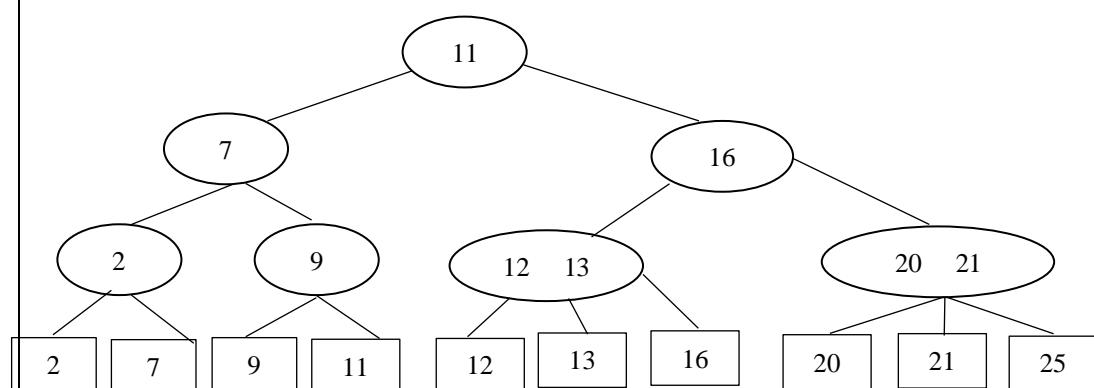
+9



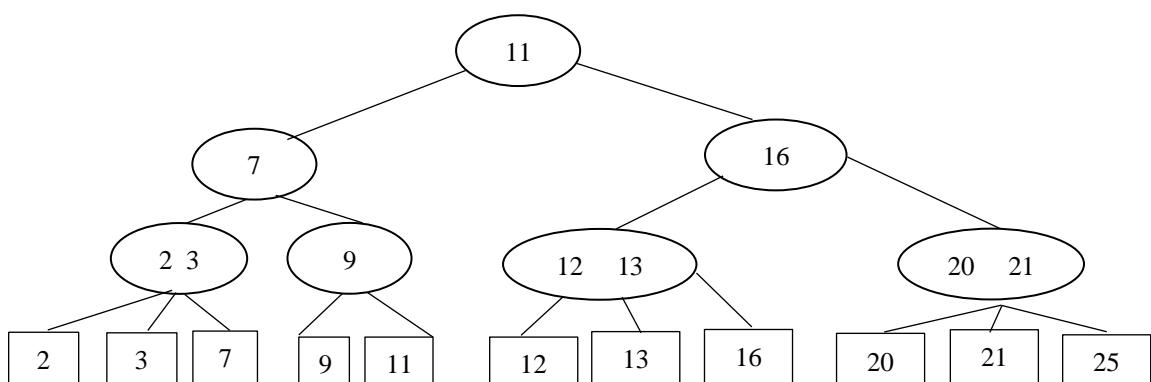
Τώρα συμβαίνει υπερχείλιση. Το δέντρο ανεβαίνει επίπεδο δηλ. τα επίπεδα του δέντρου αυξάνονται κατά 1.



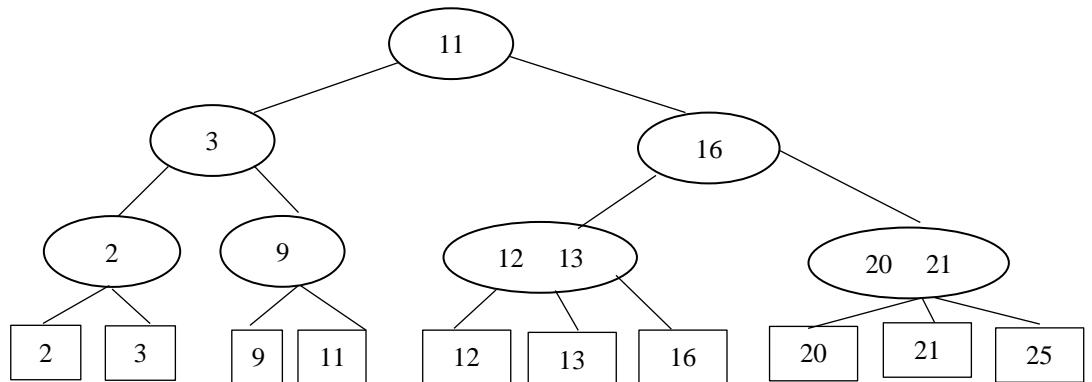
+21



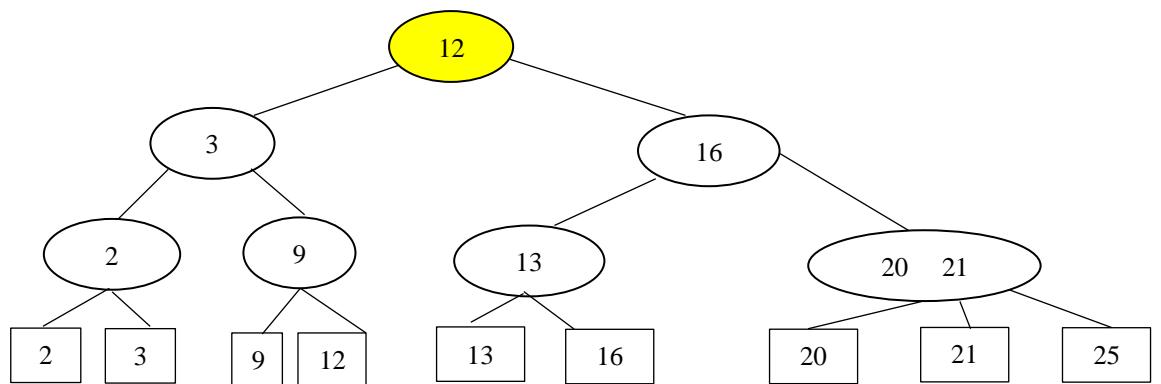
+3



-7

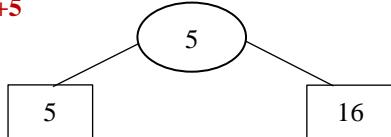
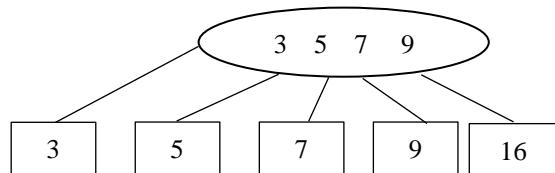


-11

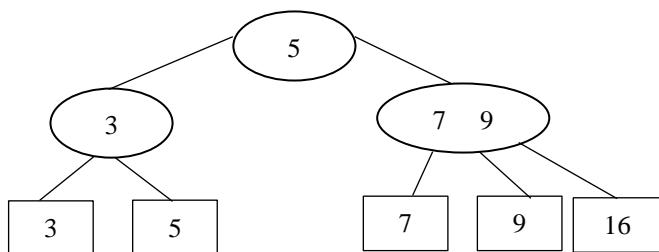
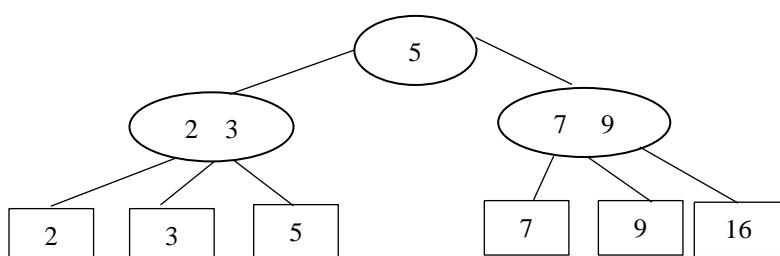
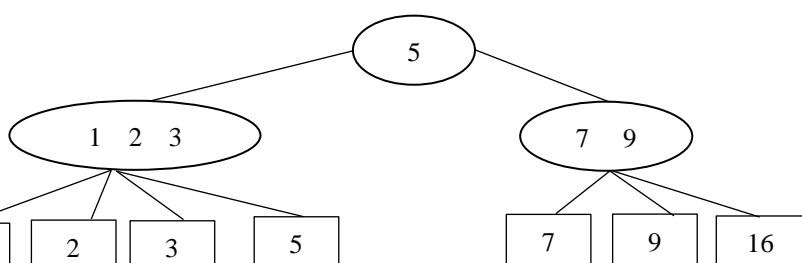


14.6 Θέμα 2 Ιούνιος 2008

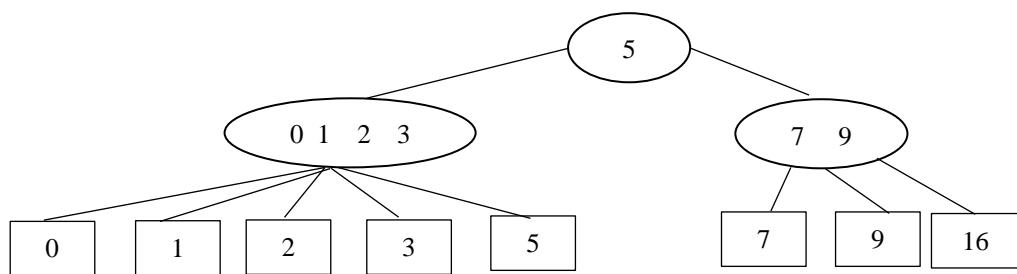
Έστω η ακολουθία Εισαγωγών (Ε)/Διαγραφών (Δ) των στοιχείων $\{+16, +5, +7, +9, +3, +2, +1, +0, +20, +25, +22, +27, -16, -9, -5, +100, +128\}$ σε ένα αρχικά άδειο (2, 4) δέντρο. Περιγράψτε την κατάσταση του δέντρου κατά τη διάρκεια των πράξεων

Απάντηση**+16****+5****+7****+3**

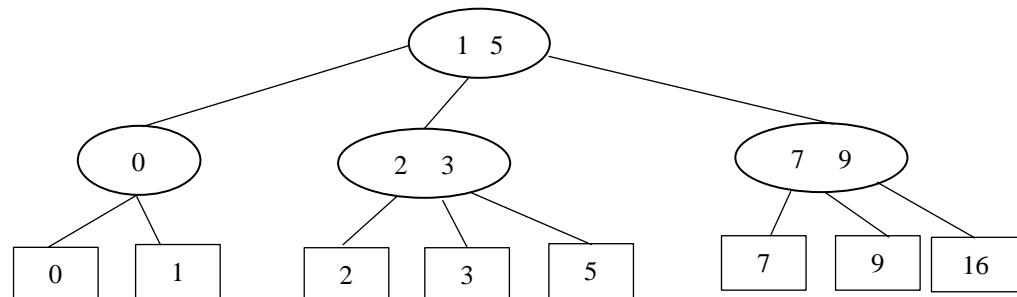
Μετά τη διάσπαση του κόμβου θα έχουμε $\lfloor 5/2 \rfloor = 2$ στο αριστερό παιδί και $\lceil 5/2 \rceil = 3$ στο δεξιό παιδί και το δέντρο ανεβαίνει επίπεδο

**+2****+1**

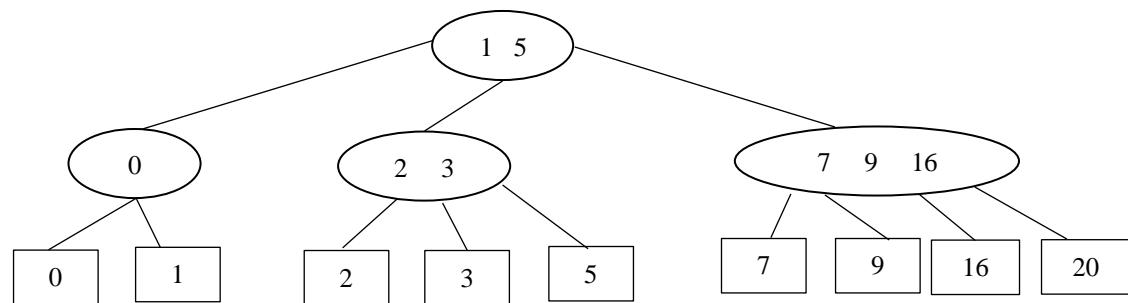
+0



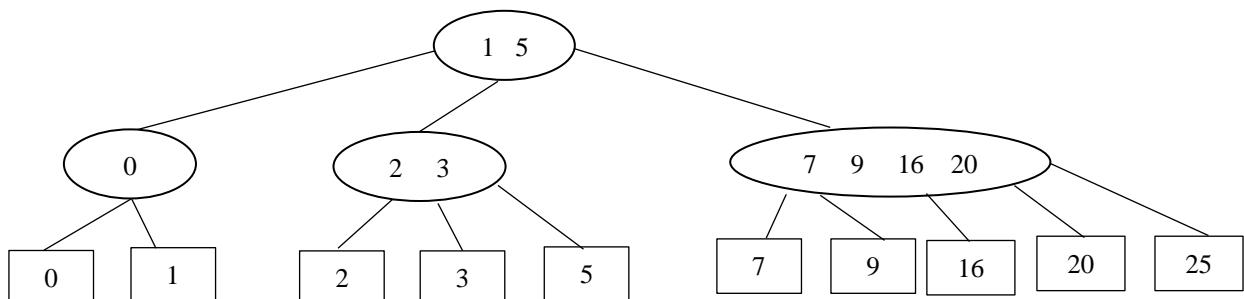
Μετά τη διάσπαση του κόμβου θα έχουμε:



+20

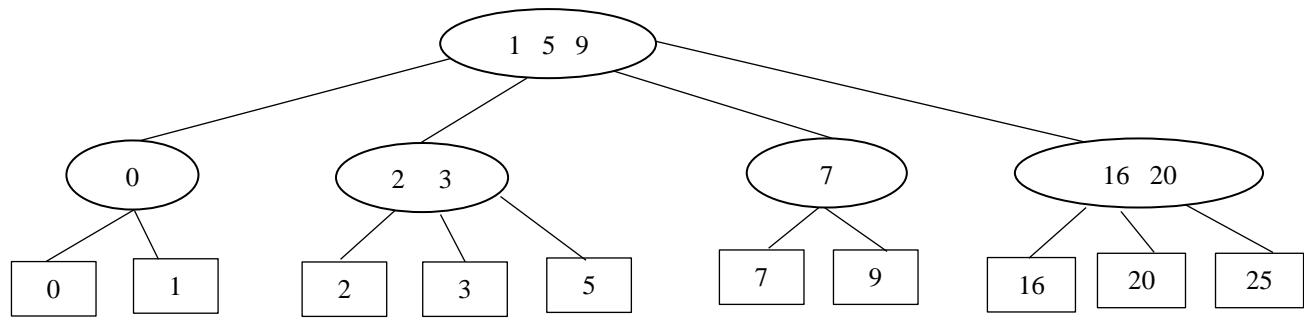


+25

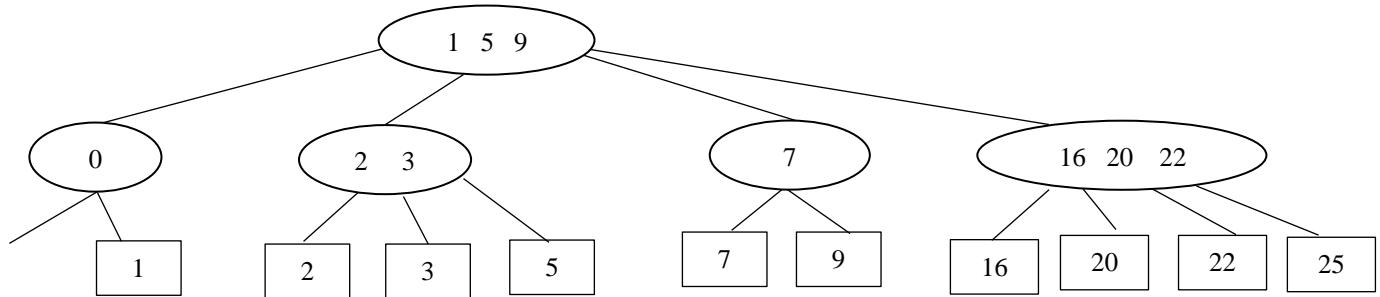


Δομές Δεδομένων –Computer Ανάλυση

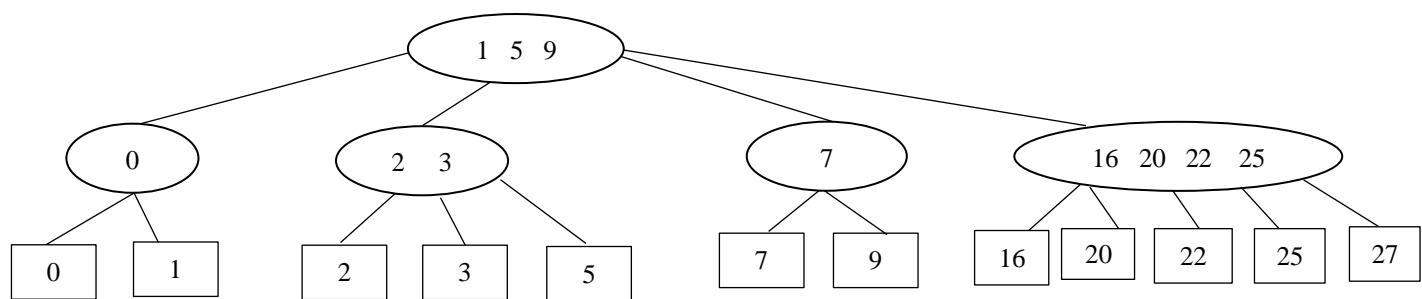
Μετά τη διάσπαση του κόμβου θα έχουμε:



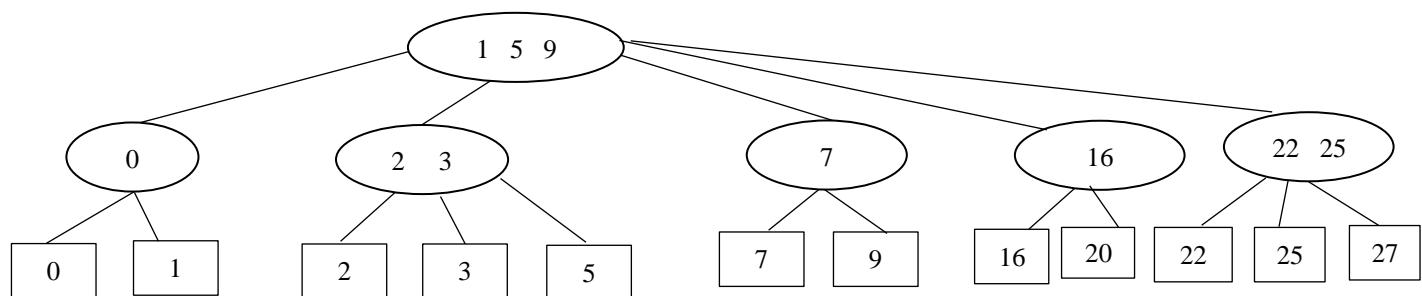
+22



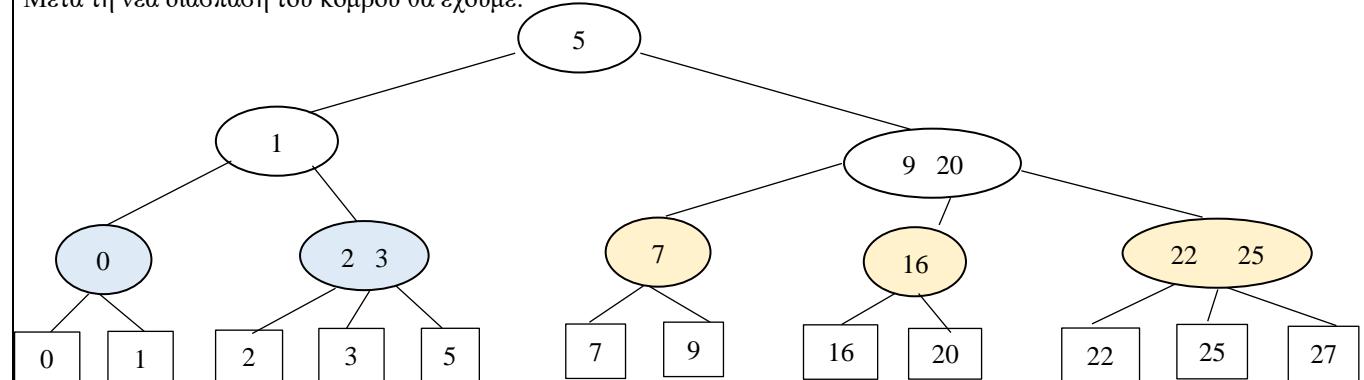
+27



Μετά τη διάσπαση του κόμβου θα έχουμε



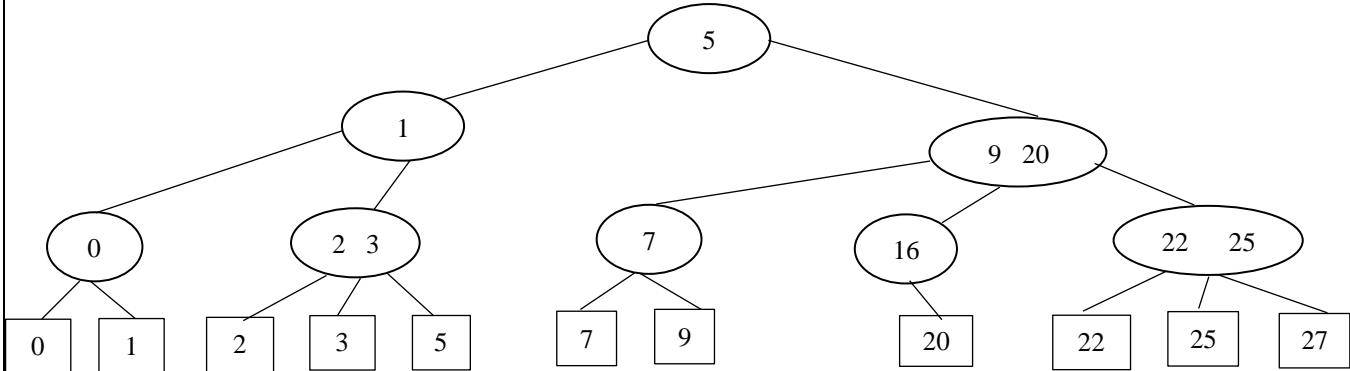
Μετά τη νέα διάσπαση του κόμβου θα έχουμε:



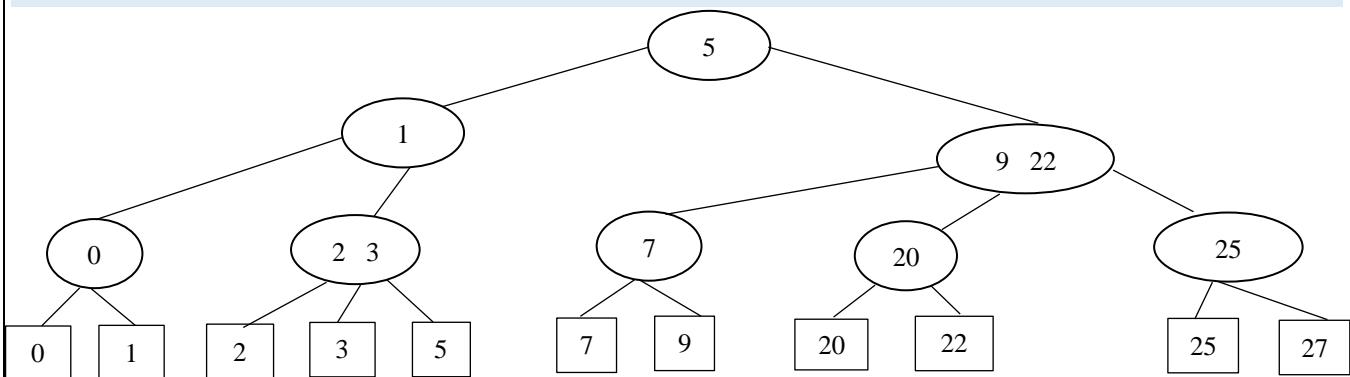
εδώ παρατηρούμε ότι το δέντρο ανεβαίνει επίπεδο

Σελίδα 22 - Computer Ανάλυση

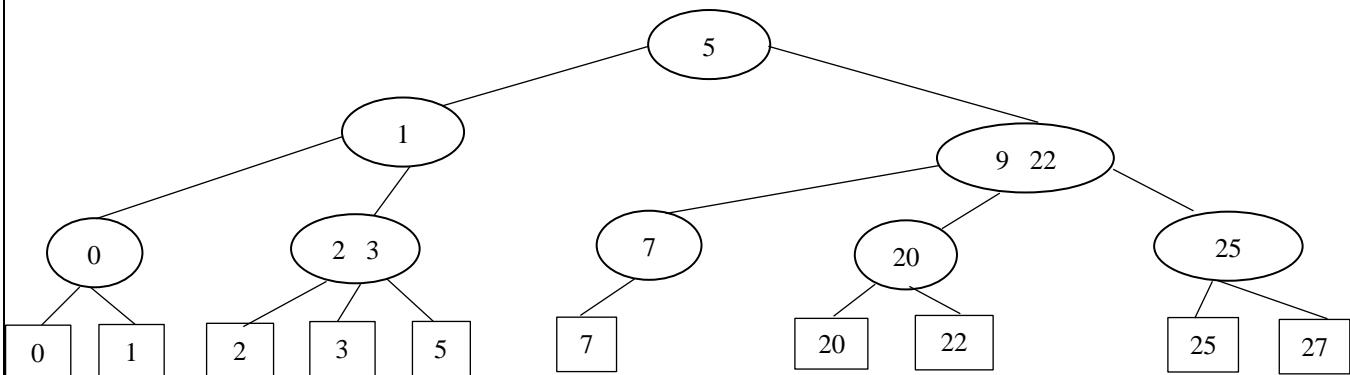
-16



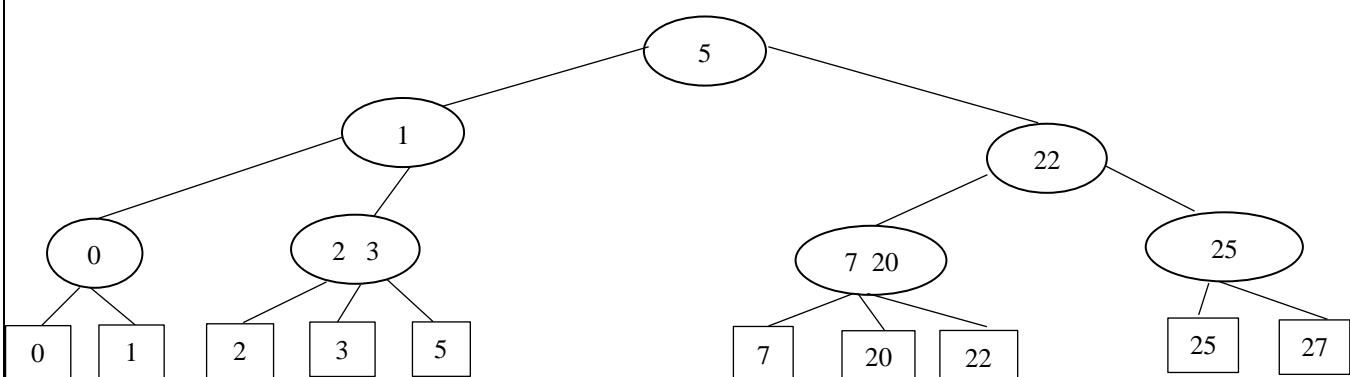
Ο γείτονας δεξιά από το 20 είναι πλούσιος και δανείζει ένα κλειδί προκειμένου η δομή του δέντρου να παραμείνει ως έχει



-9



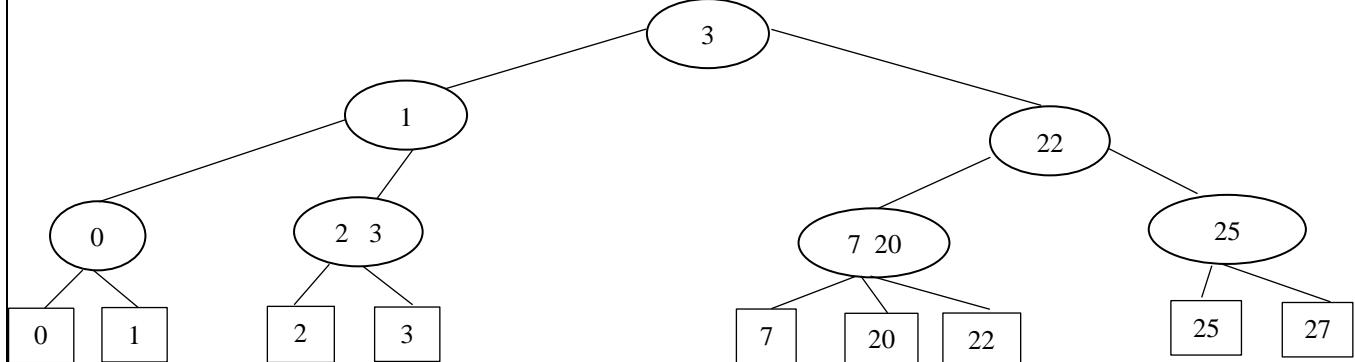
Ο γείτονας δεξιά από το 20 είναι φτωχός και δεν μπορεί να δανείσει κλειδί προκειμένου η δομή του δέντρου να παραμείνει ως έχει. Γιαυτό το 7 ενσωματώνεται στο δεξιό γείτονα.



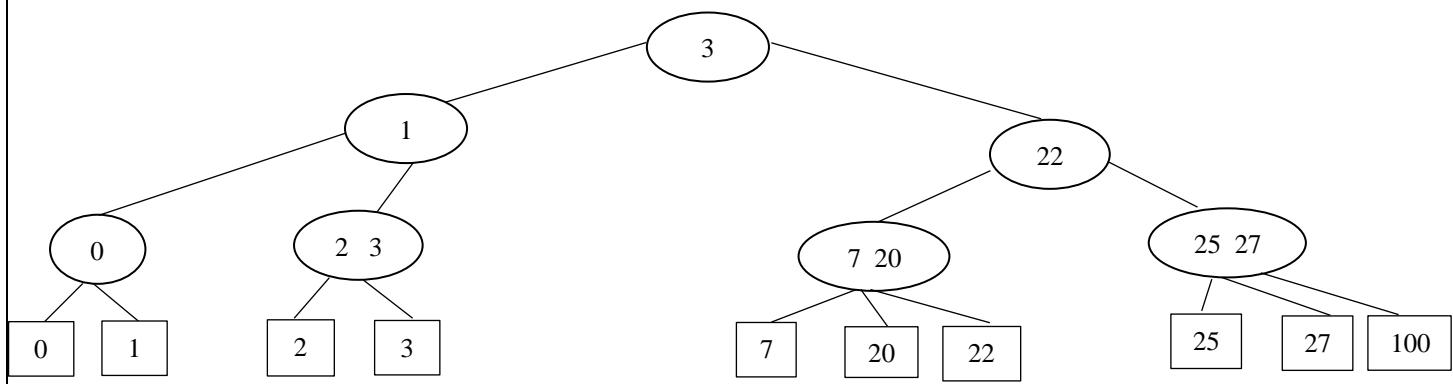
Δομές Δεδομένων –Computer Ανάλυση

-5

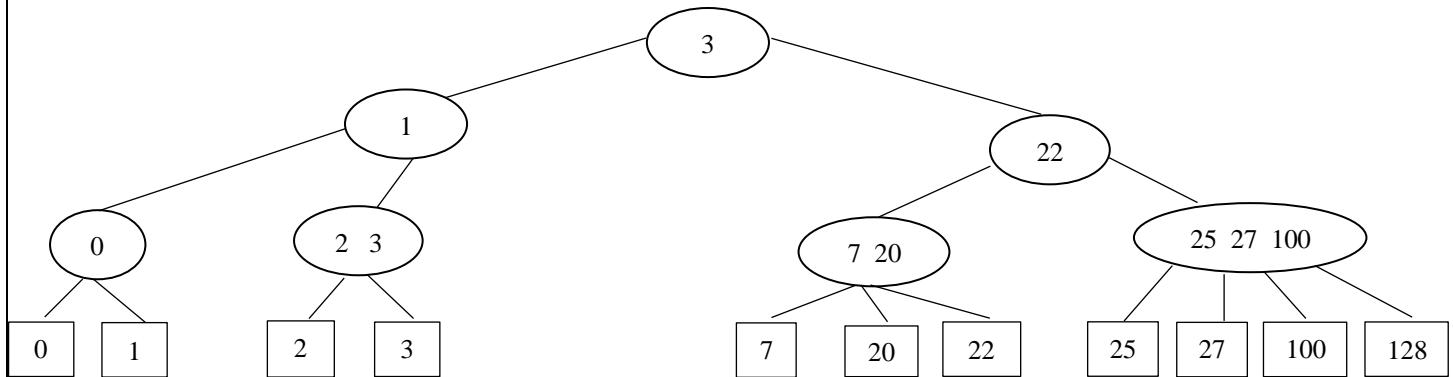
Διαγράφοντας το 5 ο κόμβος έχει 2 κλειδιά και παραμένει ως έχει γιατί είναι σε επιτρεπτή κατάσταση



+100



+128



14.7 Θέμα 2 Ιούνιος 2017

Να σχεδιαστεί βήμα-βήμα το (a,b)-δέντρο που προκύπτει από την εισαγωγή σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων:

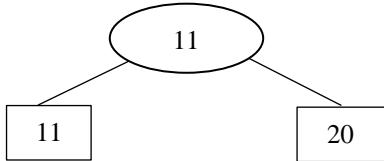
[11, 20, 16, 7, 25, 2, 12, 13, 9, 21, 3] με $a=2$ και $b=4$.

Απάντηση

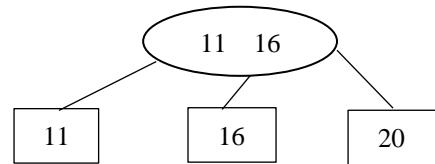
+11



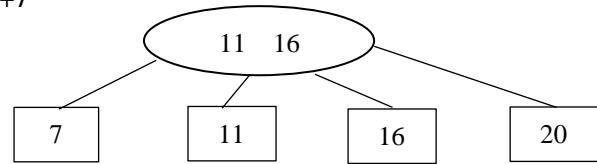
+20



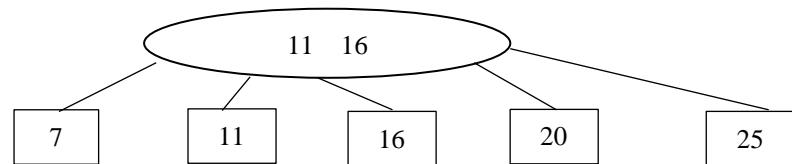
+16



+7



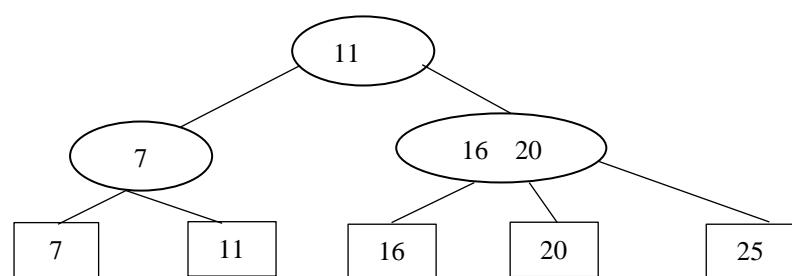
+25



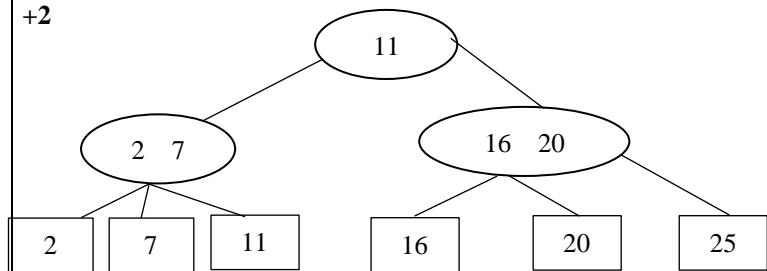
Συμβαίνει υπερχείλιση. Το δέντρο ανεβαίνει επίπεδο δηλ. τα επίπεδα του δέντρου αυξάνονται κατά 1.

Μετά τη διάσπαση της ρίζας θα έχουμε:

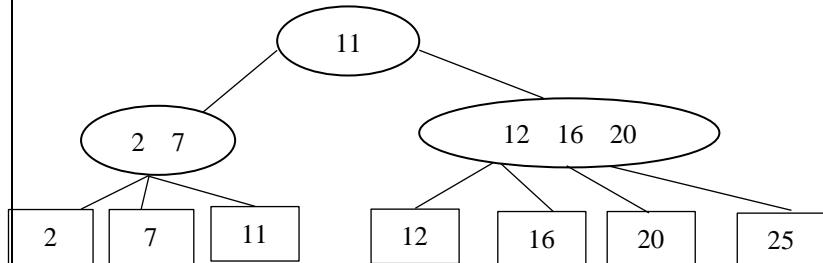
Μετά τη διάσπαση του κόμβου θα έχουμε:



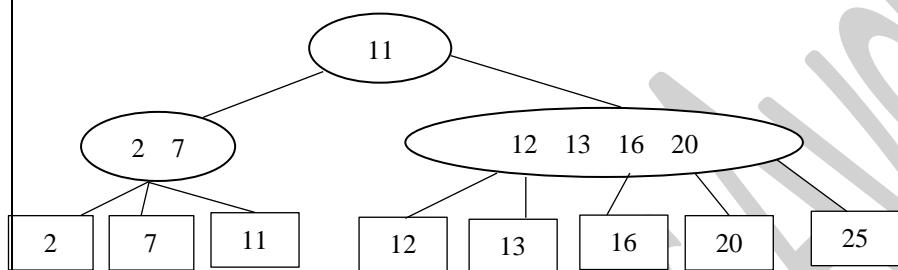
+2



+12

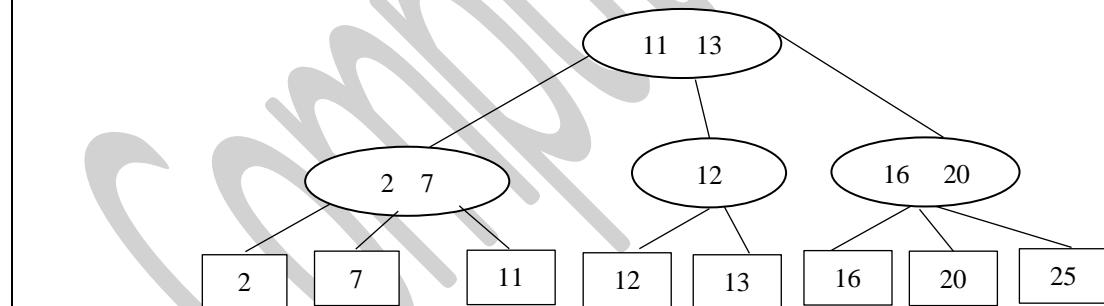


+13

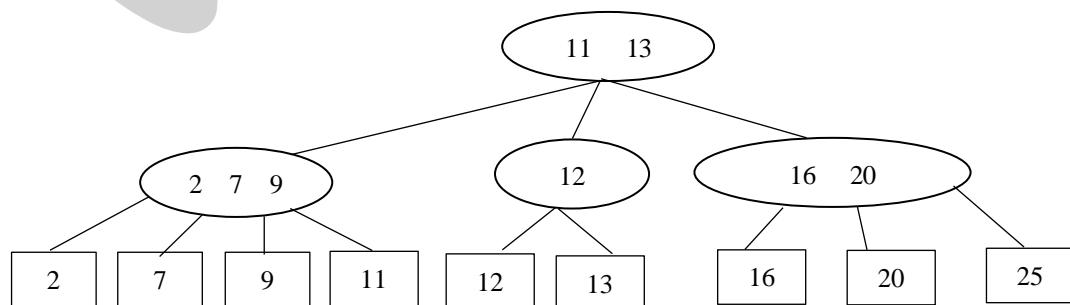


Συμβαίνει υπερχεύλιση. Το δέντρο ΔΕΝ ανεβαίνει επίπεδο δηλ. τα επίπεδα του δέντρου αυξάνονται κατά 1.

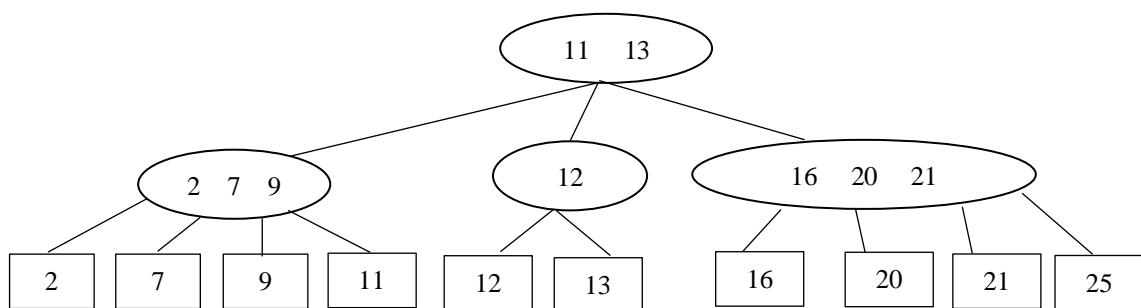
Μετά τη διάσπαση του κόμβου (12, 13, 16, 20) θα έχουμε θα έχουμε:



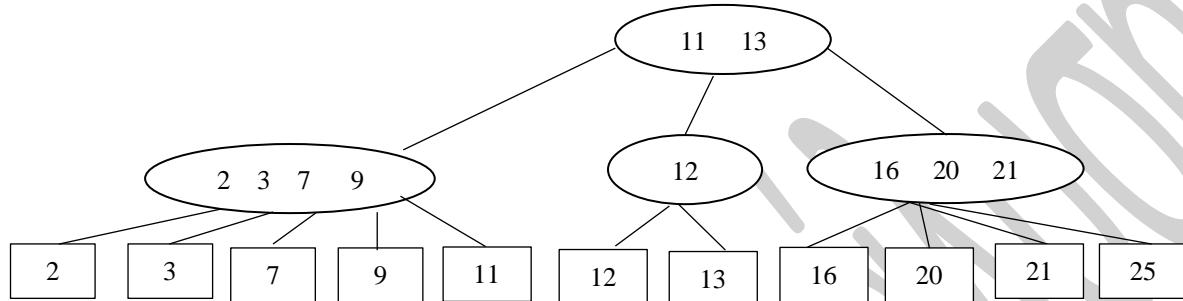
+9



+21

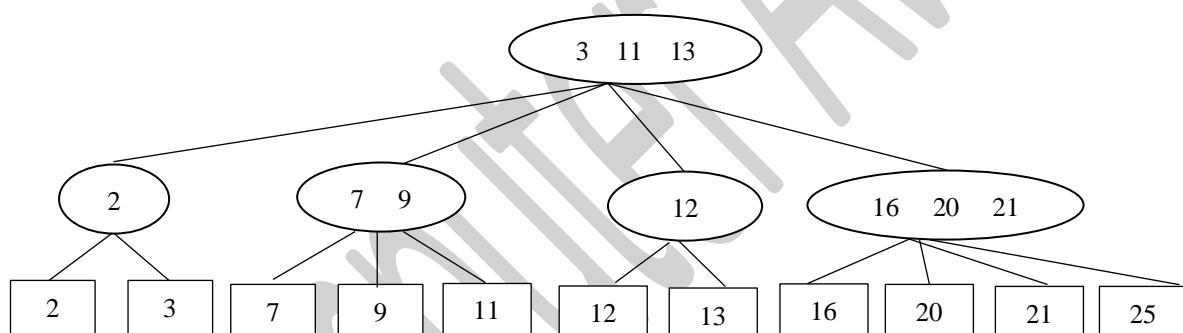


+3



Συμβαίνει υπερχεύλιση. Το δέντρο ΔΕΝ ανεβαίνει επίπεδο δηλ. τα επίπεδα του δέντρου αυξάνονται κατά 1.

Μετά τη διάσπαση του κόμβου (2, 3, 7, 9) θα έχουμε:



14.8 Θέμα 4 Φεβρουάριος 2019

Να σχεδιαστεί βήμα-βήμα το (a,b)-δέντρο που προκύπτει από την εισαγωγή σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων:

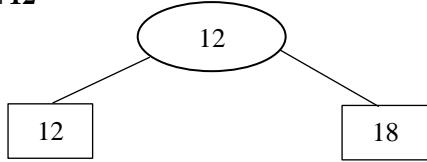
[18, 12, 19, 27, 31, 1, 29, 30, 4, 3, 34] με $a=2$ και $b=4$. Στη συνέχεια διαγράψτε βήμα-βήμα τα στοιχεία [12, 29, 18]

Απάντηση

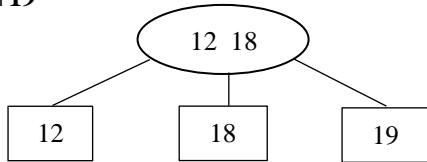
+18



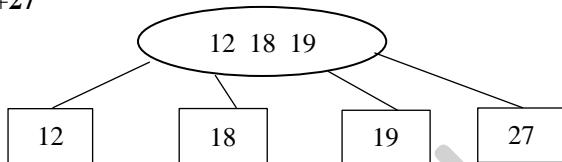
+12



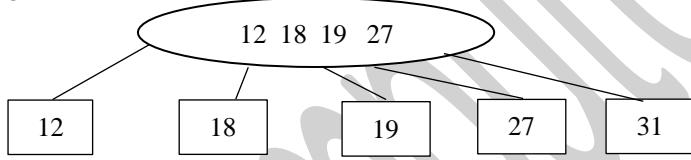
+19



+27



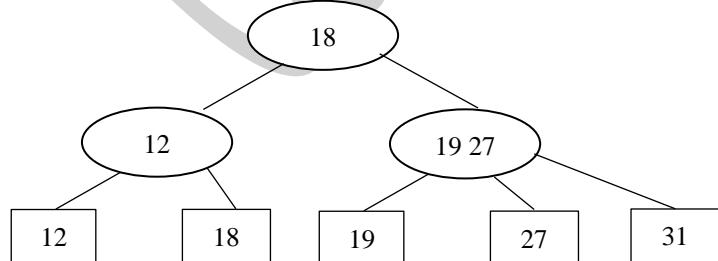
+31



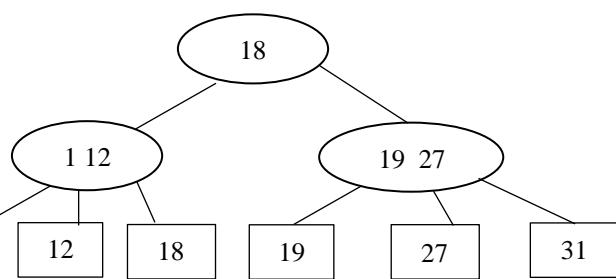
Γίνεται υπερχείλιση του κόμβου (12 18 19 27). Το δέντρο ανεβαίνει επίπεδο δηλ. τα επίπεδα του δέντρου αυξάνονται κατά 1.

Ο αριστερός κόμβος που προκύπτει από τη διάσπαση θα έχει $\lfloor 5/2 \rfloor = \lfloor 2,5 \rfloor = 2$ παιδιά ενώ ο δεξιός κόμβος θα έχει $\lceil 5/2 \rceil = \lceil 2,5 \rceil = 3$ παιδιά

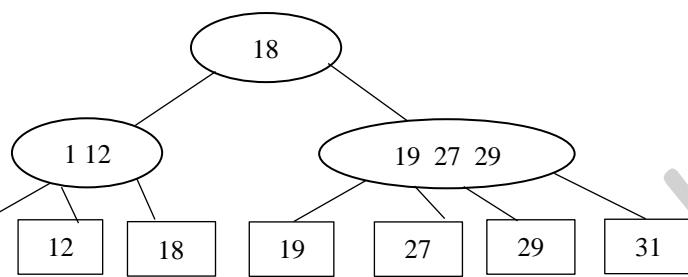
Μετά τη διάσπαση της ρίζας θα έχουμε: (Στη ρίζα θέτουμε τη μεγαλύτερη τιμή του αριστερού υποδέντρου)



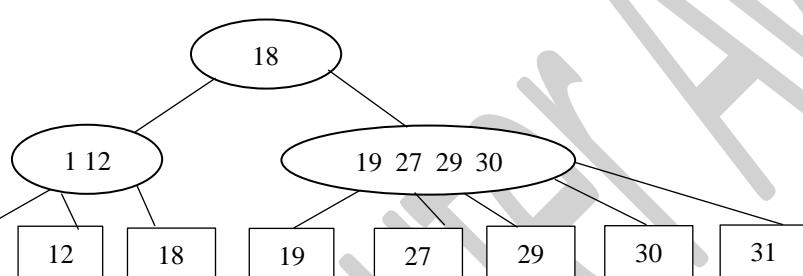
+1



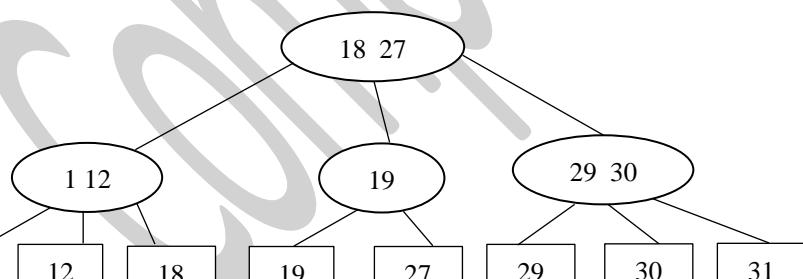
+29



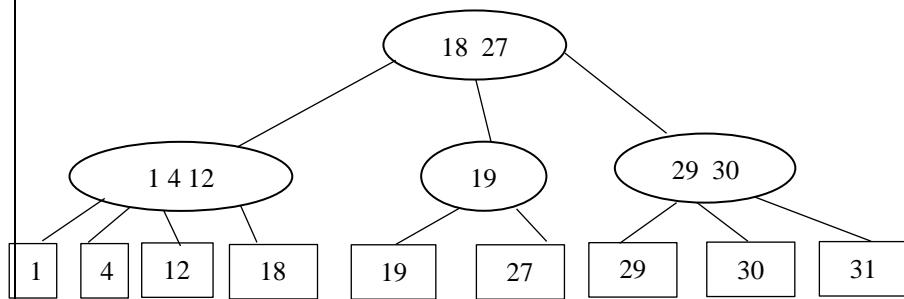
+30



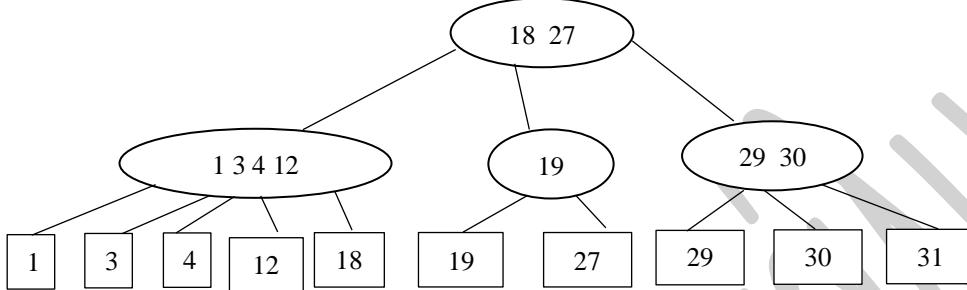
Γίνεται Διάσπαση του κόμβου (19, 27, 29, 31) αλλά το δέντρο ΔΕΝ ανεβαίνει επίπεδο. Στη ρίζα θέτουμε τη μεγαλύτερη τιμή των αριστερού υποδέντρου και τη μεγαλύτερη τιμή των μεσαίου υποδέντρου



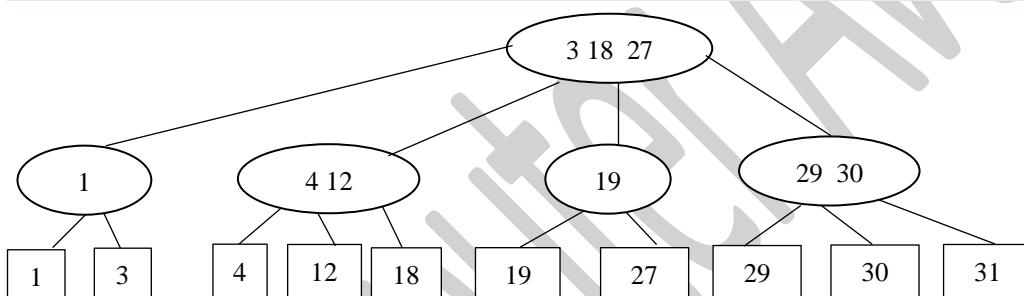
+4



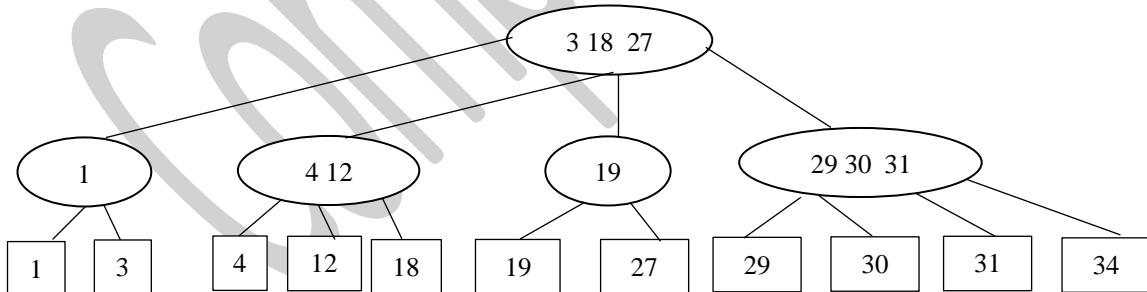
+3



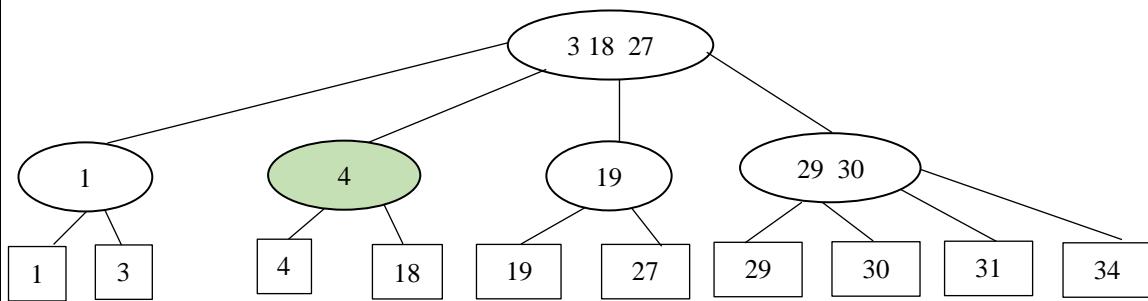
Γίνεται Διάσπαση του κόμβου (1, 3, 4, 12) αλλά το δέντρο ΔΕΝ ανεβαίνει επίπεδο. Στη ρίζα θέτουμε τη μεγαλύτερη τιμή των αριστερού υποδέντρου και τη μεγαλύτερη τιμή των μεσαίου υποδέντρου



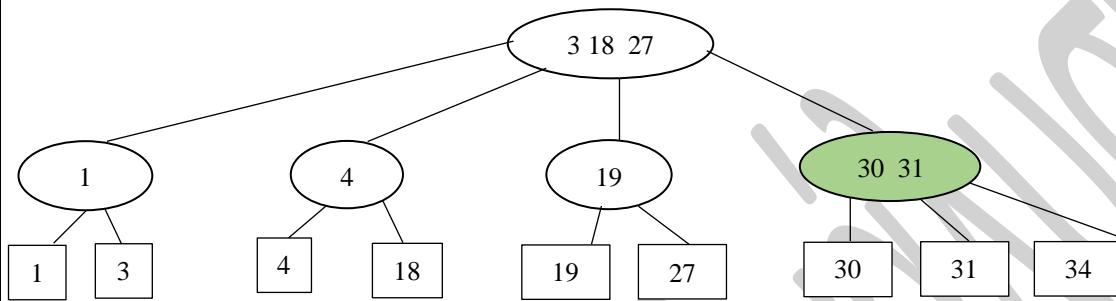
+34



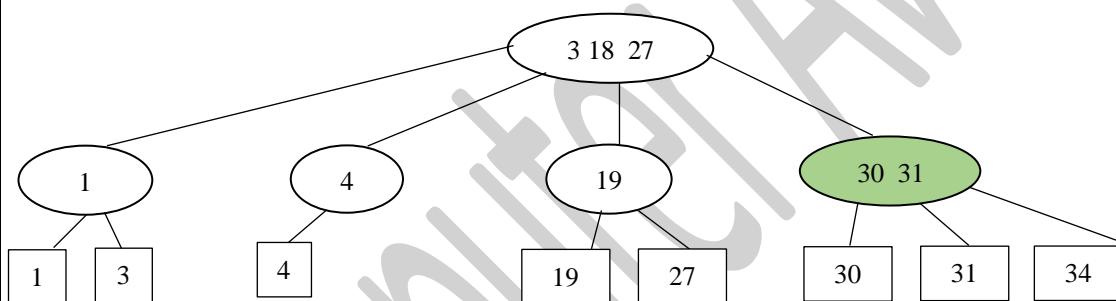
Διαγραφή 12



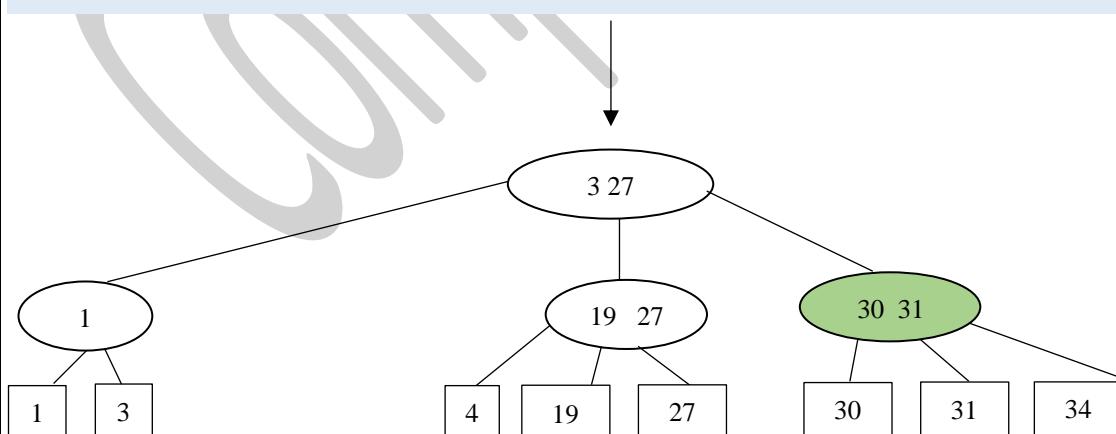
Διαγραφή 29



Διαγραφή 18



Ο αριστερός γείτονας του 4 είναι φτωχός και δεν μπορεί να δανείσει κλειδί. Ομοίως και ο δεξιός γείτονας του 4 είναι φτωχός και επίσης δεν μπορεί να δανείσει κλειδί. Άρα το 4 ενώνεται είτε με τον αριστερό είτε με το δεξιό γείτονα σε ένα ενιαίο κόμβο



14.9 Θέμα 2 Φεβρουάριος 2020

Να σχεδιαστεί βήμα-βήμα το (a,b)-δέντρο που προκύπτει από την εισαγωγή σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων

{1, 80, 13, 71, 25, 36, 43, 12, 21, 24, 5} με $a=2$ και $b=4$.

Διαγράψτε 25 και 80 και 71

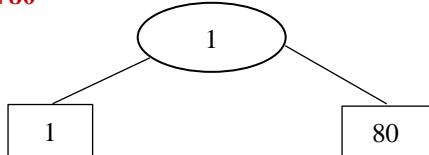
και δείξτε τη μορφή του δένδρου.

Απάντηση

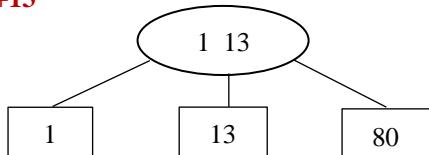
+1



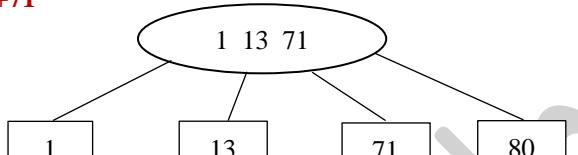
+80



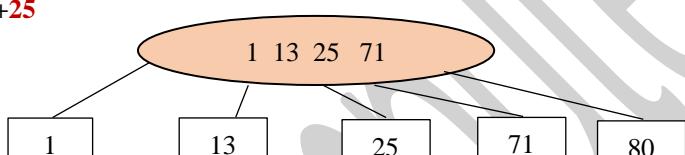
+13



+71



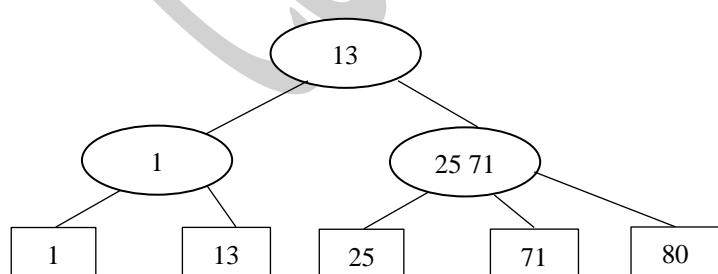
+25



Γίνεται υπερχείλιση του κόμβου (1, 13, 25, 71). Το δέντρο ανεβαίνει επίπεδο δηλ. τα επίπεδα του δέντρου αυξάνονται κατά 1.

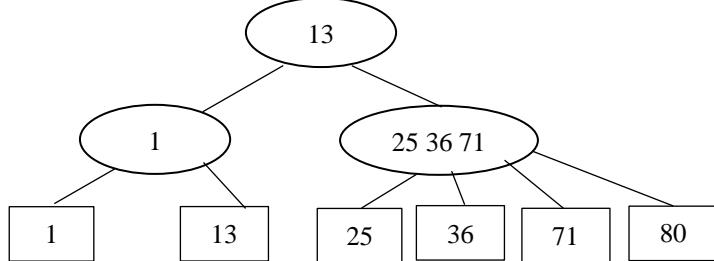
Ο αριστερός κόμβος που προκύπτει από τη διάσπαση θα έχει $\lfloor 5/2 \rfloor = \lfloor 2.5 \rfloor = 2$ παιδιά ενώ ο δεξιός κόμβος θα έχει $\lceil 5/2 \rceil = \lceil 2.5 \rceil = 3$ παιδιά

Μετά τη διάσπαση της ρίζας θα έχουμε: (Στη ρίζα θέτουμε τη μεγαλύτερη τιμή του αριστερού υποδέντρου)



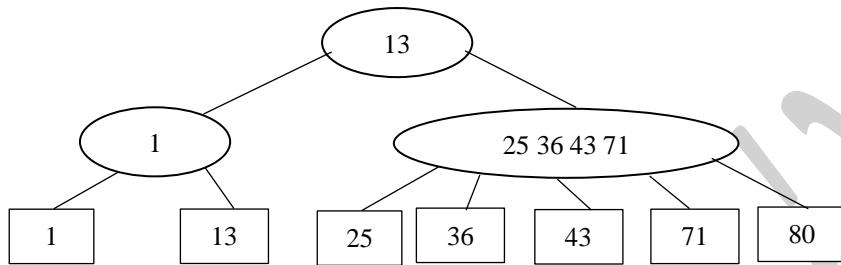
Στη ρίζα μπαίνει πάντα η μεγαλύτερη τιμή του αριστερού υποδέντρου

+36



+43

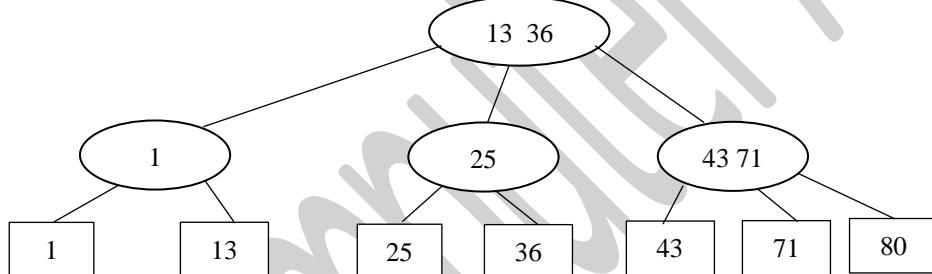
Το 43 συγκρίνεται με τη ρίζα και επειδή το $(2, 4)$ δέντρο είναι δέντρο αναζήτησης και αφού $43 > 13$ τοποθετείται ως φύλλο στο υποδέντρο δεξιά του 13



Γίνεται υπερχείλιση του κόμβου $(25, 36, 43, 71)$. Το δέντρο δεν ανεβαίνει επίπεδο

Ο αριστερός κόμβος που προκύπτει από τη διάσπαση θα έχει $\lfloor 5/2 \rfloor = 2$ παιδιά ενώ ο δεξιός κόμβος θα έχει $\lceil 5/2 \rceil = 3$ παιδιά

Μετά τη διάσπαση της ρίζας θα έχουμε:

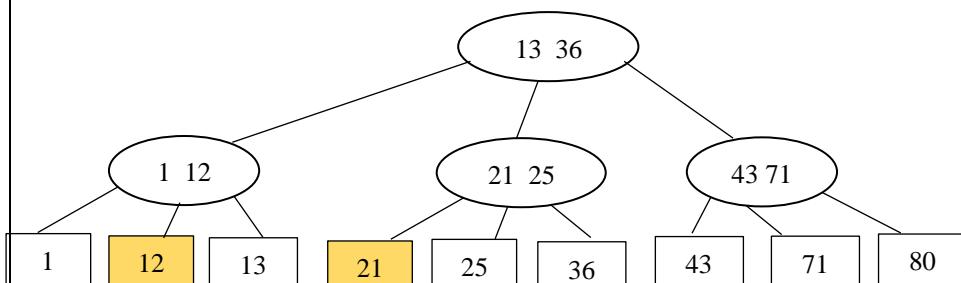


Στη ρίζα θέτουμε τη μεγαλύτερη τιμή του αριστερού και τη μεγαλύτερη τιμή του μεσαίου υποδέντρου

+12, +21

Το 12 συγκρίνεται με τη ρίζα και επειδή το $(2, 4)$ δέντρο είναι δέντρο αναζήτησης και αφού $12 < 13$ τοποθετείται ως φύλλο στο υποδέντρο αριστερά του 13

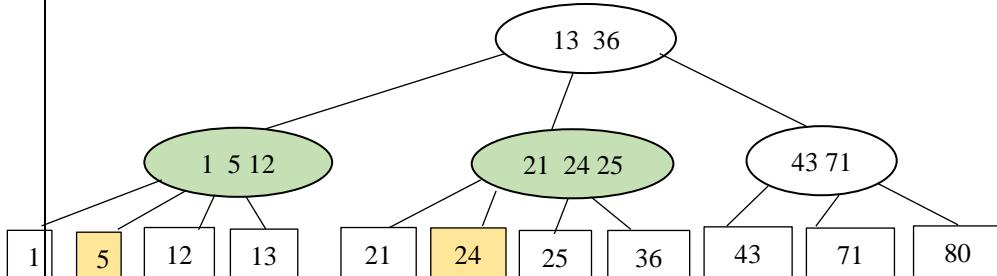
Το 21 συγκρίνεται με τη ρίζα και επειδή το $(2, 4)$ δέντρο είναι δέντρο αναζήτησης και αφού $13 < 21 < 36$ τοποθετείται ως φύλλο στο ενδιάμεσο υποδέντρο



+24,+5

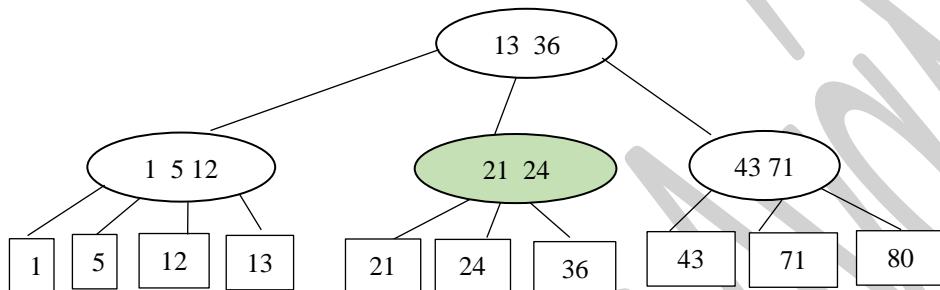
Το 24 συγκρίνεται με τη ρίζα και επειδή το (2, 4) δέντρο είναι δέντρο αναζήτησης και αφού $13 < 24 < 36$ τοποθετείται ως φύλλο στο ενδιάμεσο υποδέντρο.

Το 5 συγκρίνεται με τη ρίζα και επειδή το (2, 4) δέντρο είναι δέντρο αναζήτησης και αφού $5 < 13$ τοποθετείται ως φύλλο στο υποδέντρο αριστερά του 13.

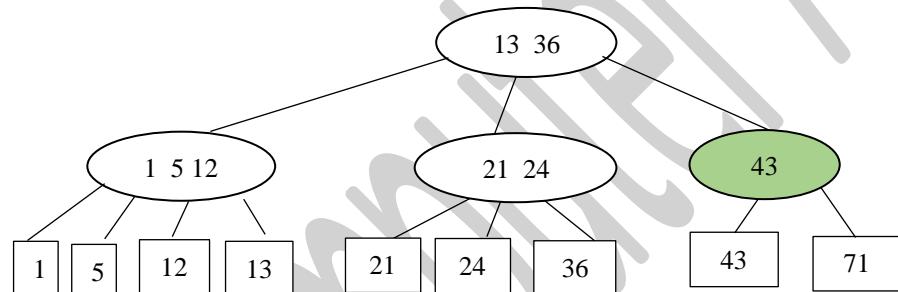


Διαγραφές

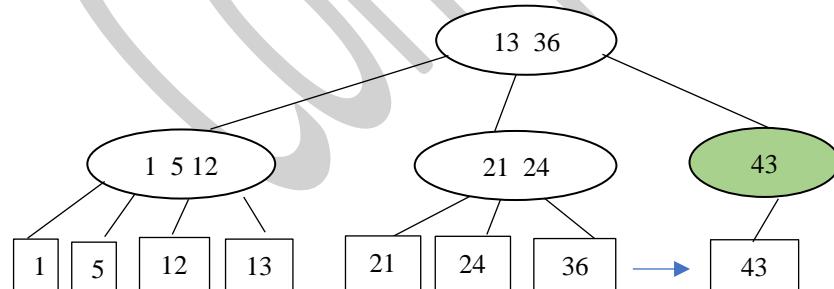
-25



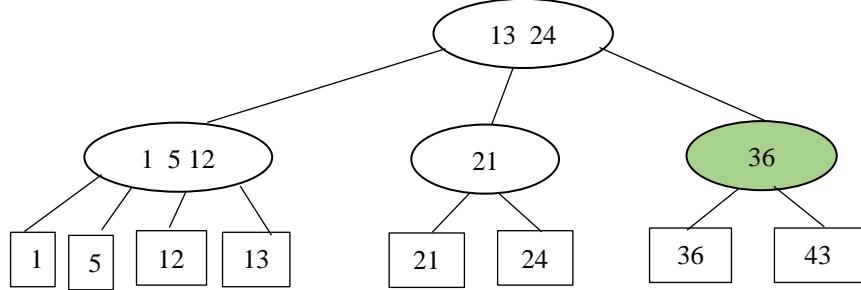
-80



-71



Μετά τη διαγραφή του 71 ο κόμβος με το 43 ξεμένει με 1 φύλλο. Αυτό δεν αποτελεί επιτρεπτή κατάσταση διότι ένα δέντρο (2, 4) έχει τουλάχιστον 2 παιδιά. Εξετάζουμε τον αριστερό γείτονα και αυτός είναι πλούσιος άρα μπορεί να δανείσει ένα κλειδί και η δομή του δέντρου να παραμείνει ως έχει. Συνεπώς δανείζει το κλειδί 36 και το δέντρο μετατρέπεται σε (2, 4) και είναι το ακόλουθο:



15 TRIE ΔΕΝΤΡΑ

15.1 Χαρακτηριστικά TRIE Δέντρου

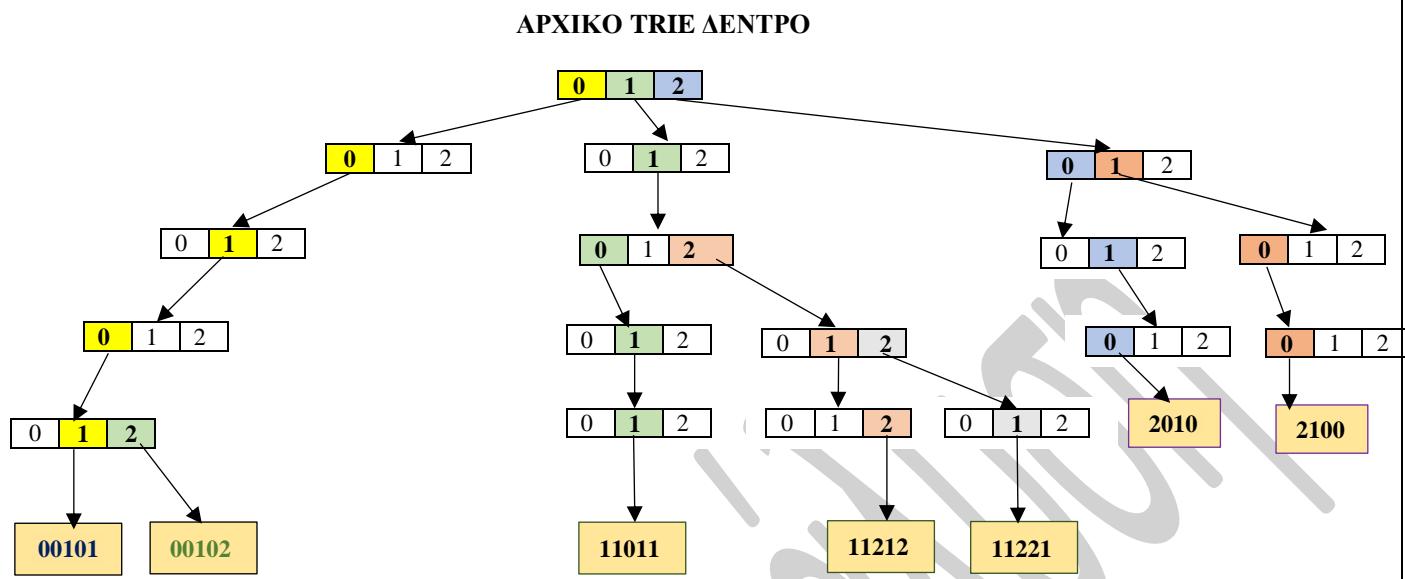
- **To TRIE δέντρο είναι φυλλοπροσανατολισμένο**
- **Κάθε εσωτερικός κόμβος περιέχει όλο το αλφάριθμο της γλώσσας**
- **Το ψηφιακό (TRIE) δέντρο αναπαριστάνει συμβολοσειρές. Κάνει εύκολη και γρήγορη αποθήκευση και ανάκτηση πληροφορίας για λέξεις, συμβολοσειρές, επιθέματα κ.λ.π.**
- **Οι συμβολοσειρές τοποθετούνται στα φύλλα του TRIE δέντρου και είναι ταξινομημένες από αριστερά προς τα δεξιά**
- **Για την κατασκευή του συμπαγούς TRIE όλοι οι κόμβοι με βαθμό εξόδου 1 (δηλ. με πλήθος εξερχόμενων ακμών=1) συμπιέζονται (διαγράφονται)**
- Έστω σύμπαν U του οποίου τα στοιχεία είναι συμβολοσειρές μήκους λ πάνω σε ένα αλφάριθμο K με $|K|=k$. Ένα υποσύνολο $S \subseteq U$ αναπαρίσταται ως ένα k -αδικό δέντρο που περιέχει όλα τα προθέματα των στοιχείων του S
- Κάθε εσωτερικός κόμβος του δέντρου είναι ένας πίνακας μήκους k από δείκτες
- Κάθε θέση του πίνακα αντιστοιχίζεται σε ένα γράμμα του αλφαριθμού
- Κάθε θέση του πίνακα σε ένα κόμβο σε βάθος i θα λάβει τιμή αν κάποιο από τα στοιχεία του S στην i -οστή θέση έχει τον αντίστοιχο χαρακτήρα
- Το δέντρο έχει ύψος λ και καταλαμβάνει χώρο $O(k)$ όπου λ το μήκος των συμβολοσειρών και k το πλήθος χαρακτήρων του αλφαριθμού
- Οι βασικές πράξεις απαιτούν χρόνο $O(\lambda)=O(\log_k N)$ δηλ. εξαρτώνται από το μέγεθος του αλφαριθμού k
- Ο χώρος στη χειρότερη περίπτωση είναι $O(n\lambda k)$ όπου $n=|S|$ είναι το πλήθος των συμβολοσειρών του συνόλου S . Αυτό συμβαίνει όταν τα στοιχεία δεν έχουν κοινά προθέματα οπότε έχουμε:
 - n πλήρη μονοπάτια
 - $n \times \lambda$ κόμβοι συνολικά
 - $n \times \lambda \times k$ συνολικός χώρος
- Στο συμπαγές Trie ο χώρος μειώνεται από $O(n\lambda k)$ σε $O(nk)$
- Ο χώρος που καταλαμβάνει στη χειρότερη περίπτωση ένα συμπαγές (συμπιεσμένο) TRIE που κρατά όλα τα επιθέματα μιας συμβολοσειράς n χαρακτήρων είναι $O(n)$.

15.2 SOS Πολυπλοκότητες TRIE Δέντρων

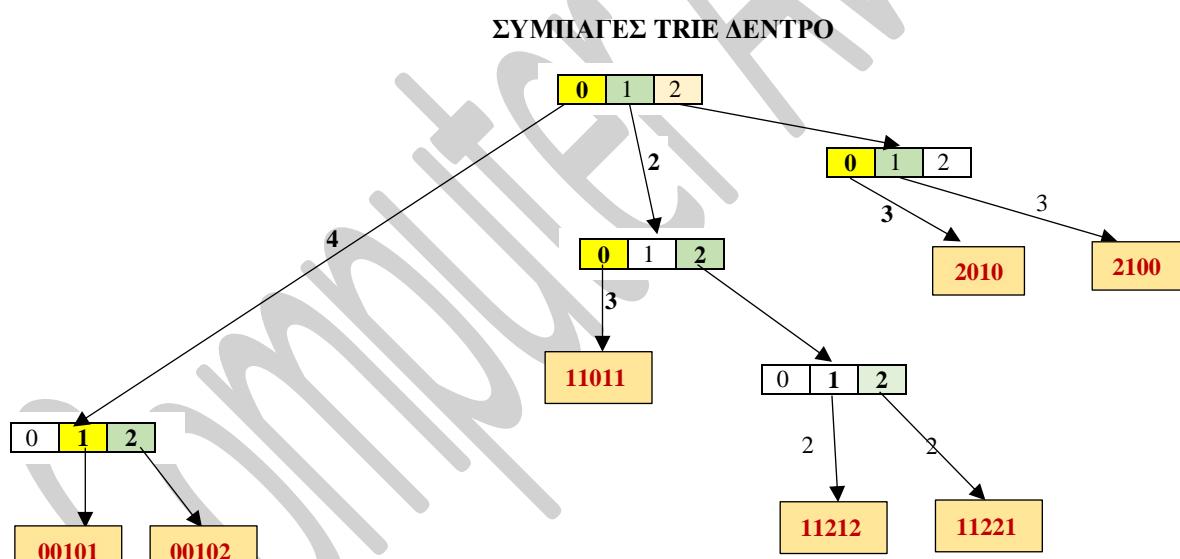
- Ο χώρος που καταλαμβάνει στη χειρότερη περίπτωση ένα TRIE δέντρο είναι **$O(n\lambda k)$** όπου $|S|=n$ είναι το πλήθος συμβολοσειρών, $|K|=k$ πλήθος χαρακτήρων αλφαριθμού και κάθε στοιχείο του S είναι συμβολοσειρά μήκους λ . Αυτό το χειρότερο ενδεχόμενο συμβαίνει όταν **και τα π στοιχεία δεν έχουν κανένα κοινό πρόθεμα**
- Ο χώρος του συμπαγούς TRIE μειώνεται από $O(n\lambda k)$ σε **$O(nk)$**
- Ο χώρος που καταλαμβάνει στη χειρότερη περίπτωση ένα συμπαγές (συμπιεσμένο) TRIE συμβαίνει όταν κρατά όλα τα επιθέματα μιας συμβολοσειράς n χαρακτήρων και είναι **$O(n)$**

15.3 Θέμα 5 Ιούνιος 2008

Αποθηκεύστε σε μια δομή TRIE χτισμένη στο τριαδικό αλφάριθμο $\Sigma = \{0, 1, 2\}$ τα στοιχεία **{00101, 00102, 11011, 11212, 11221, 2100, 2010}**. Προτείνετε την αποθήκευση που καταλαμβάνει το λιγότερο χώρο.

Απάντηση

Κανόνας Συμπίεσης: Για την κατασκευή του συμπαγούς TRIE όλοι οι κόμβοι με βαθμό εξόδου=1 δηλ. με πλήθος εξερχόμενων ακμών=1 συμπιέζονται (διαγράφονται).

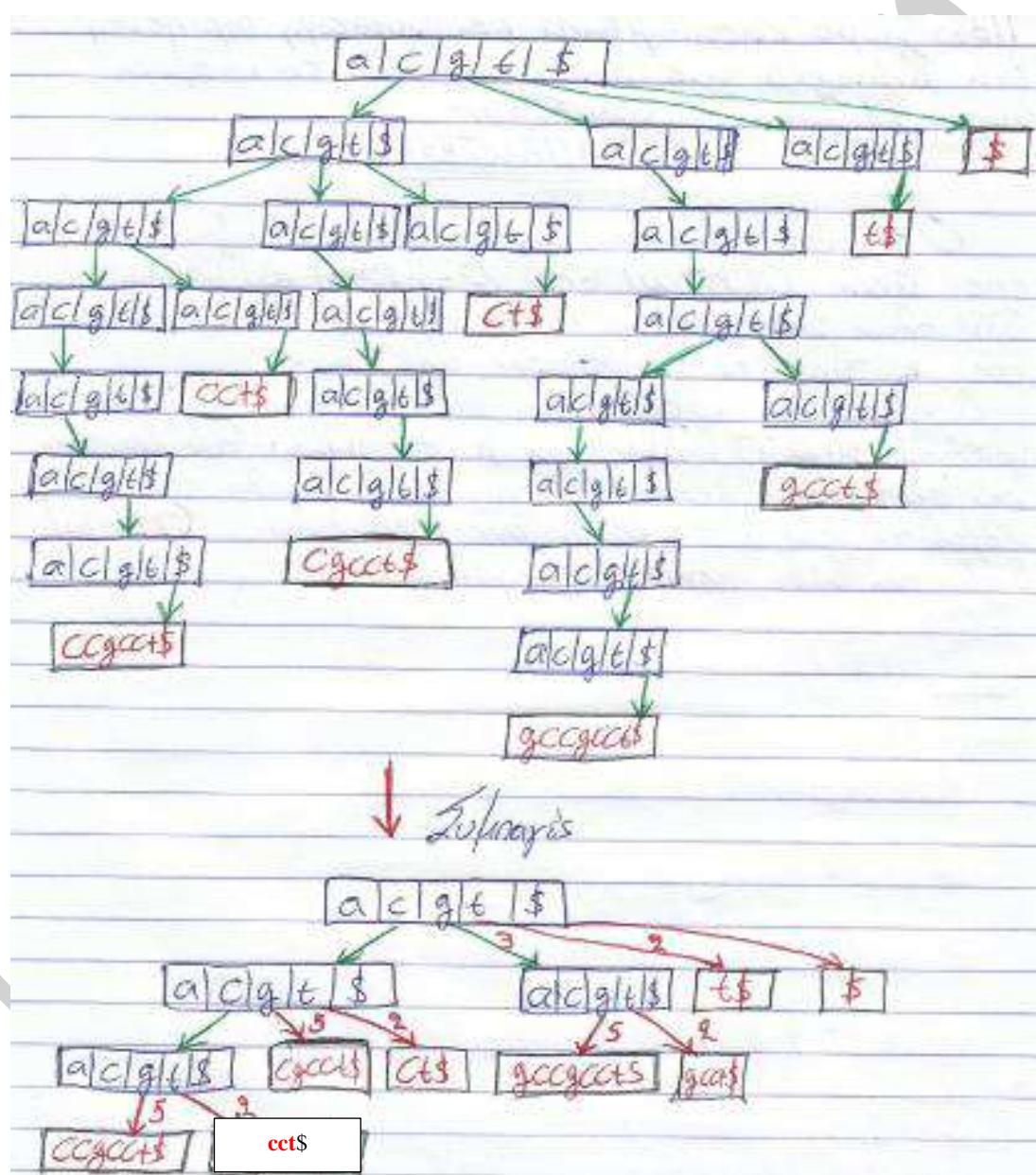


15.4 Θέμα 4 Ιούνιος 2011

Όλα τα δυνατά επιθέματα της συμβολοσειράς $gccgcct\$$ είναι τα: $gccgcct\$, ccgcct\$, cgccct\$, gcct\$, cct\$, ct\$, t\$, \$$. Αποθηκεύστε τα παραπάνω επιθέματα στα φύλλα ενός TRIE στο αλφάριθμο $\Sigma = \{a, c, g, t, \$\}$. Φροντίστε για την οικονομικότερη αποθήκευση. Πόσο χώρο καταλαμβάνει στη χειρότερη περίπτωση ένα συμπαγές TRIE που κρατά όλα τα επιθέματα μιας συμβολοσειράς **η χαρακτήρων**:

Απάντηση

Πρώτα ταξινομούμε αλφαριθμητικά όλες τις συμβολοσειρές: $\{ccgcct\$, cct\$, cgccct\$, ct\$, gccgcct\$, gcct\$, t\$, \$\}$

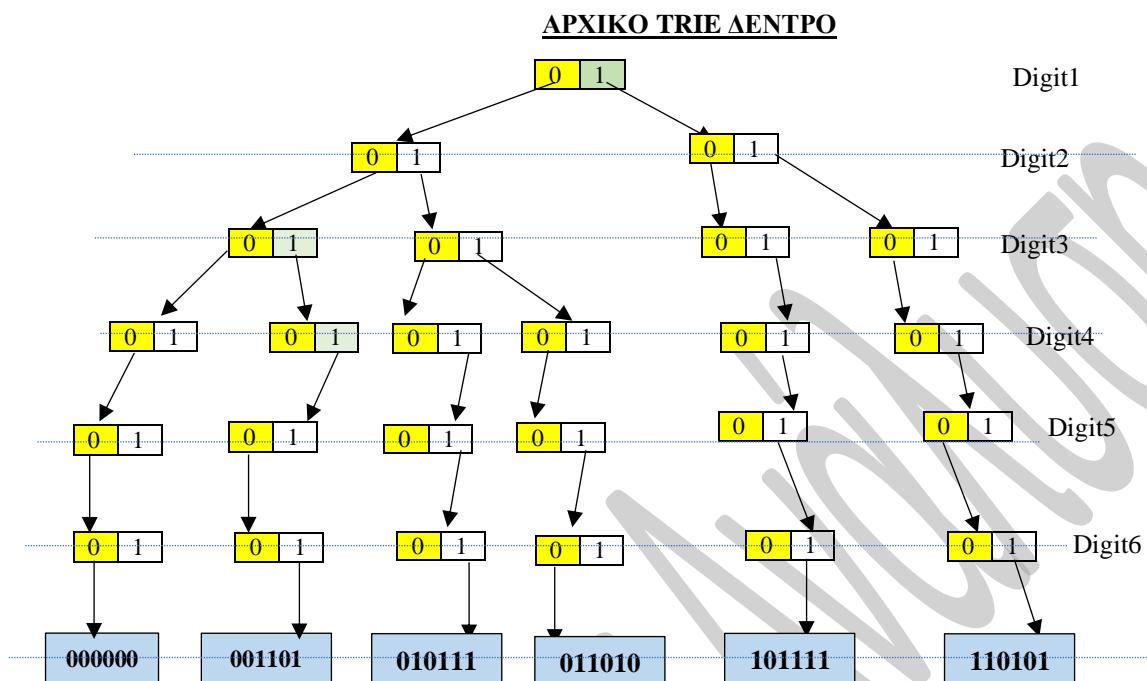


Ο χώρος που καταλαμβάνει στη χειρότερη περίπτωση ένα συμπαγές (συμπιεσμένο) TRIE που κρατά όλα τα επιθέματα μιας συμβολοσειράς η χαρακτήρων είναι **$O(n)$** .

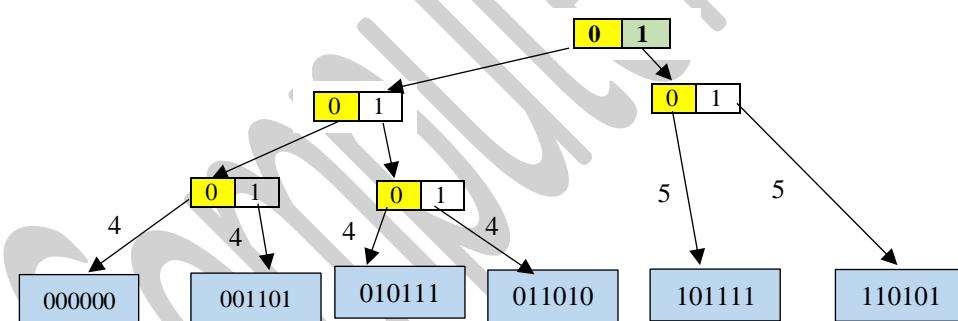
15.5 Θέμα 3 Ιούνιος 2012

Να αποθηκευτούν σε ένα Trie με αλφάριθμο {0, 1} το σύνολο {000000, 011010, 110101, 010111, 001101, 101111}. Φροντίστε για την οικονομικότερη σε χώρο αποθήκευση. Πόσο χώρο καταλαμβάνει στη χειρότερη περίπτωση ένα συμπαγές trie που κρατά όλα τα επιθέματα μιας συμβολοσειράς ή χαρακτήρων;

Απάντηση



ΣΥΜΠΑΓΕΣ TRIE ΔΕΝΤΡΟ



Ο χώρος που καταλαμβάνει στη χειρότερη περίπτωση ένα συμπαγές TRIE που κρατά όλα τα επιθέματα μιας συμβολοσειράς ή χαρακτήρων είναι $O(n)$.

15.6 Θέμα 9 Ιούνιος 2015

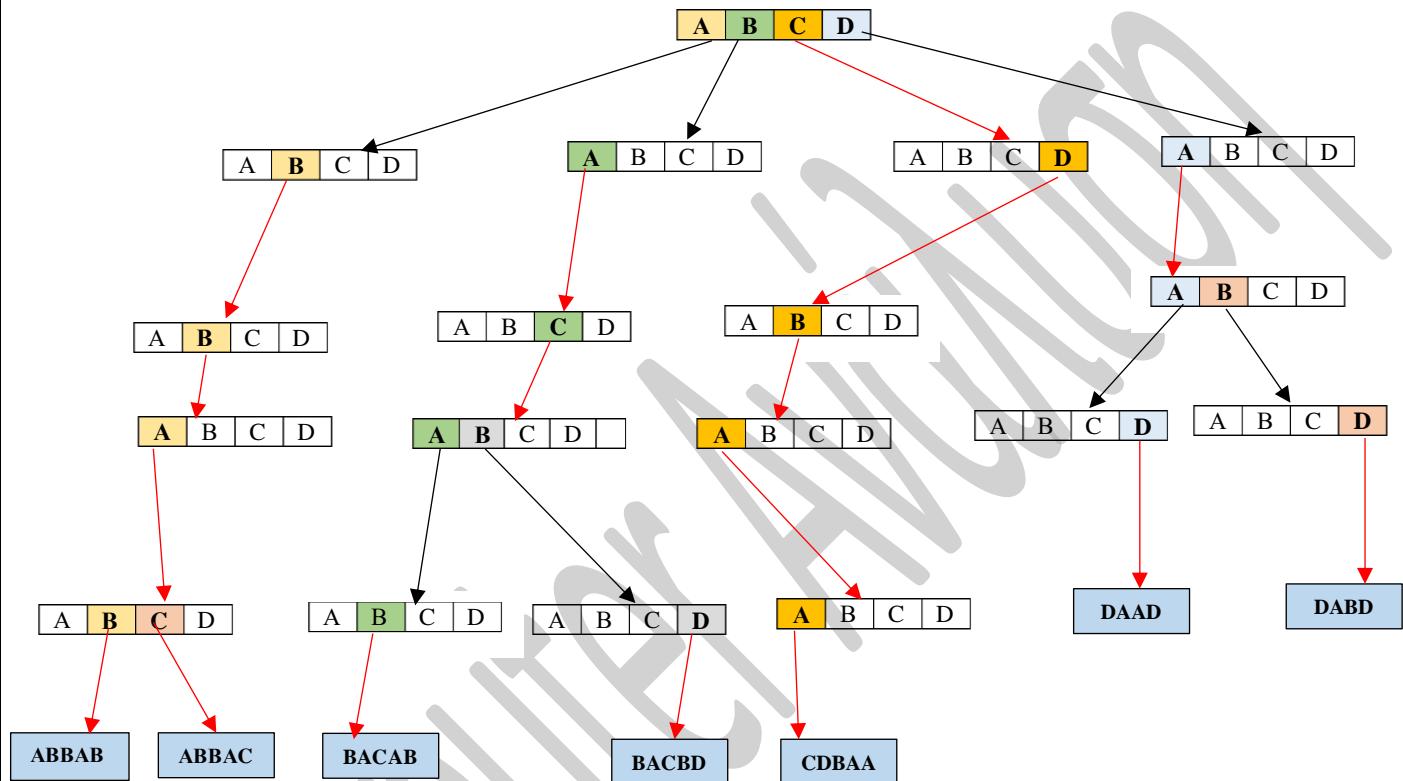
Αποθηκεύστε σε δομή Trie χτισμένη στο αλφάριθμο $\Sigma = \{A, B, C, D\}$ τα στοιχεία {**ABBAB, ABBAC, BACAB, BACBD, CDBAA, DAAD, DABD**}.

Προτείνετε την αποθήκευση που καταλαμβάνει το μικρότερο χώρο.

Απάντηση

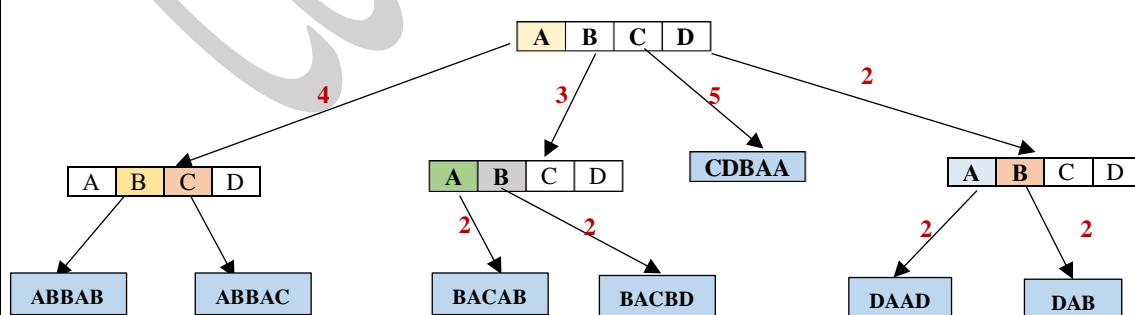
Απαραίτητη προϋπόθεση για να κατασκευαστεί σωστά ένα TRIE Δέντρο είναι οι συμβολοσειρές που τοποθετούνται σε αυτό να είναι ταξινομημένες. Αν δεν είναι πρώτα τις ταξινομούμε και μετά τις τοποθετούμε στο TRIE Δέντρο.

ΑΡΧΙΚΟ ΔΕΝΤΡΟ TRIE



Κανόνας Συμπίεσης: Για την κατασκευή του συμπαγούς TRIE όλοι οι κόμβοι με βαθμό εξόδου=1 δηλ. με πλήθος εξερχόμενων ακμών=1 συμπιέζονται (διαγράφονται)

ΣΥΜΠΑΓΕΣ ΔΕΝΤΡΟ TRIE



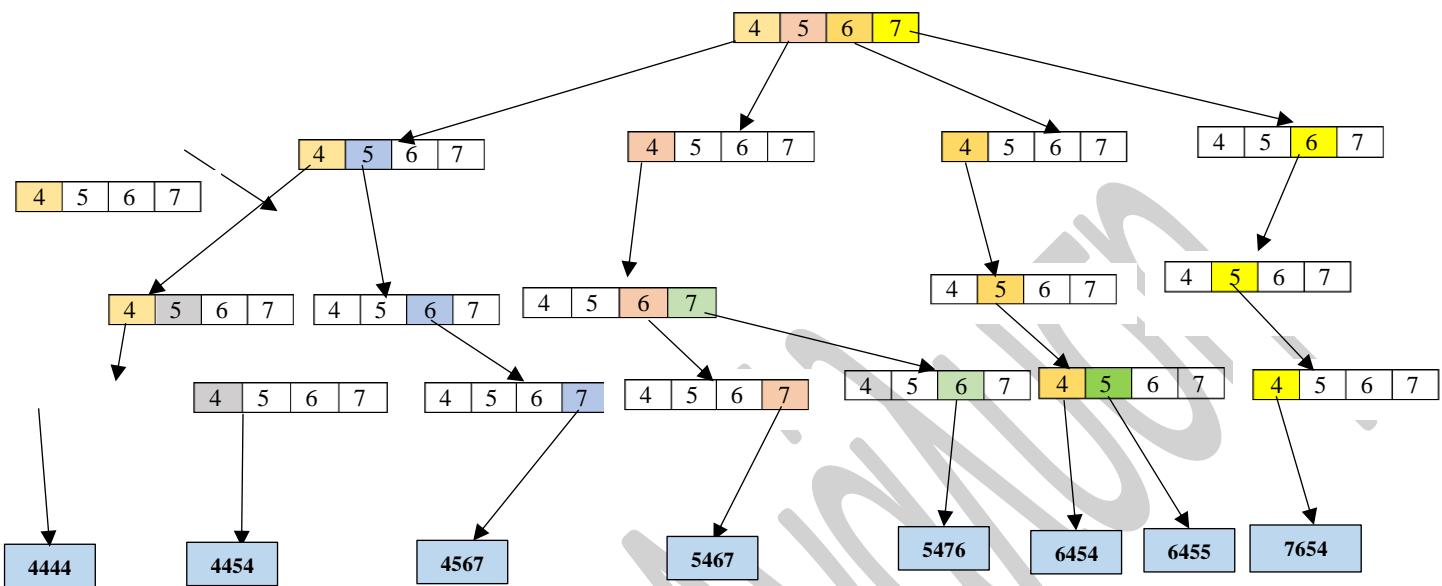
Ο χώρος που καταλαμβάνει το αρχικό Trie είναι $O(n \cdot \lambda \cdot k)$. Ο χώρος που καταλαμβάνει το συμπαγές Trie είναι $O(n \cdot k)$ όπου $n = |S|$ είναι το πλήθος των συμβολοσειρών του συνόλου S , λ το μήκος των συμβολοσειρών και k το πλήθος χαρακτήρων του αλφαρίθμου

15.7 Θέμα 7 Σεπτέμβριος 2016 και Ιούνιος 2017

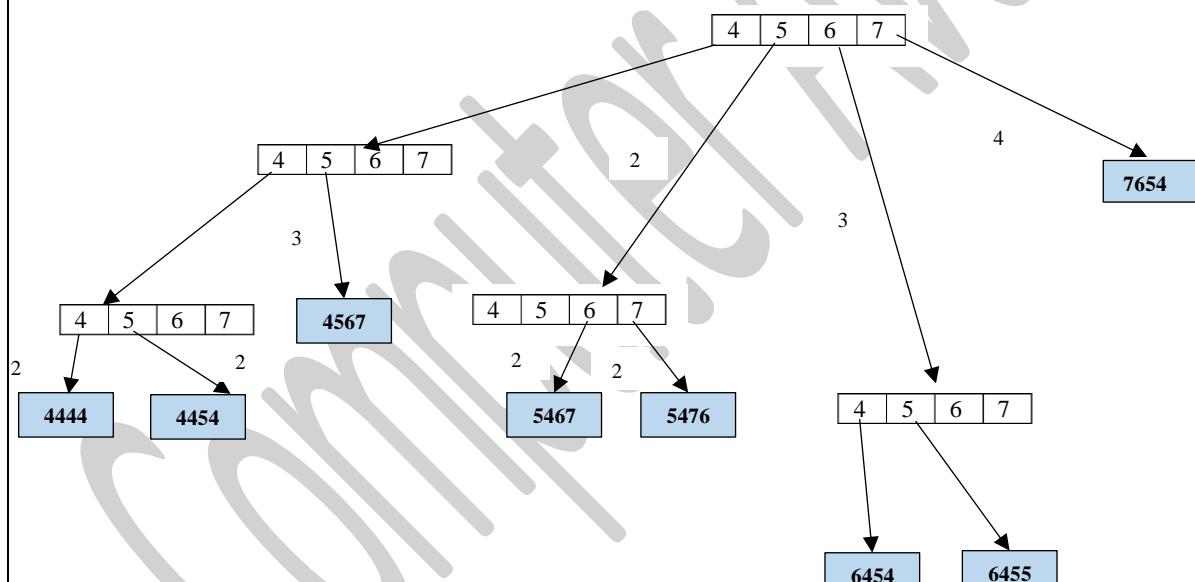
Αποθηκεύστε σε δομή Trie χτισμένη στο αλφάριθμο $\Sigma = \{4, 5, 6, 7\}$ τα στοιχεία $\{4444, 4454, 4567, 5467, 5476, 6454, 6455, 7654\}$. Σχεδιάστε την πλήρη αποθήκευση και την αποθήκευση που καταλαμβάνει το μικρότερο χώρο.

Απάντηση

ΑΡΧΙΚΟ ΔΕΝΤΡΟ TRIE



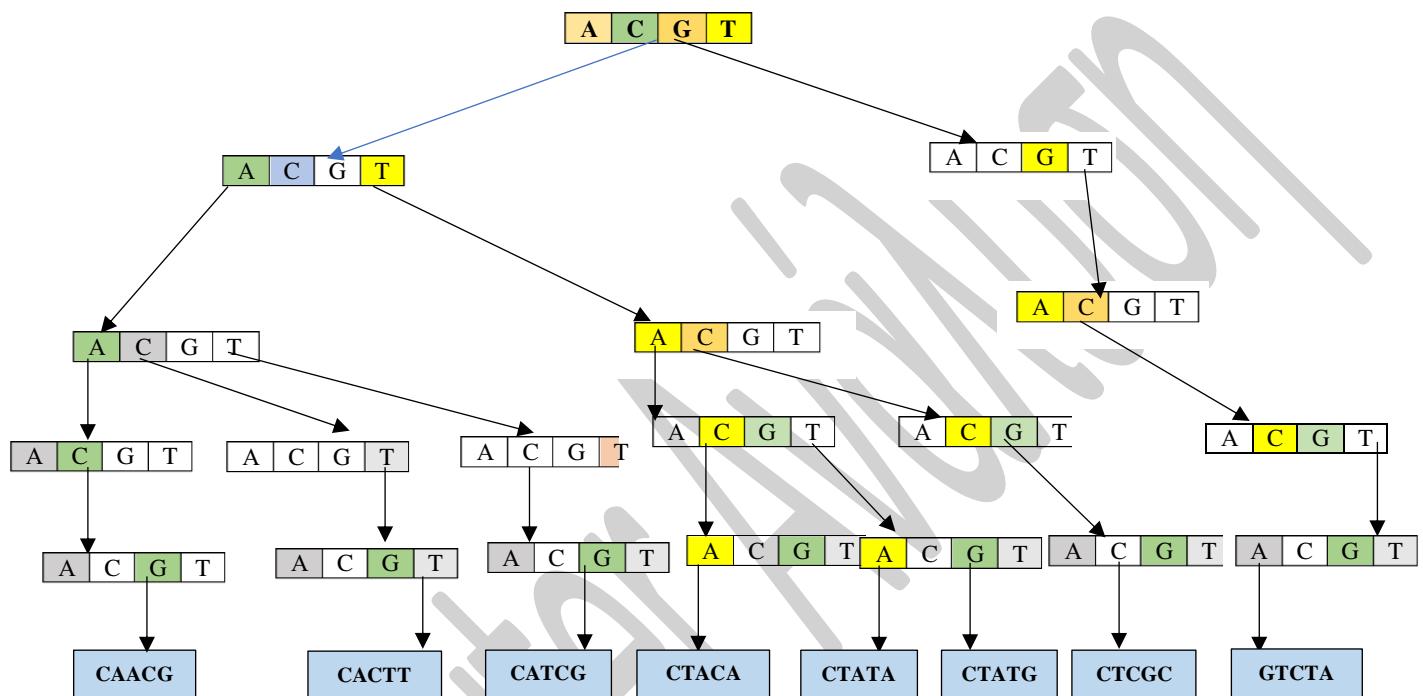
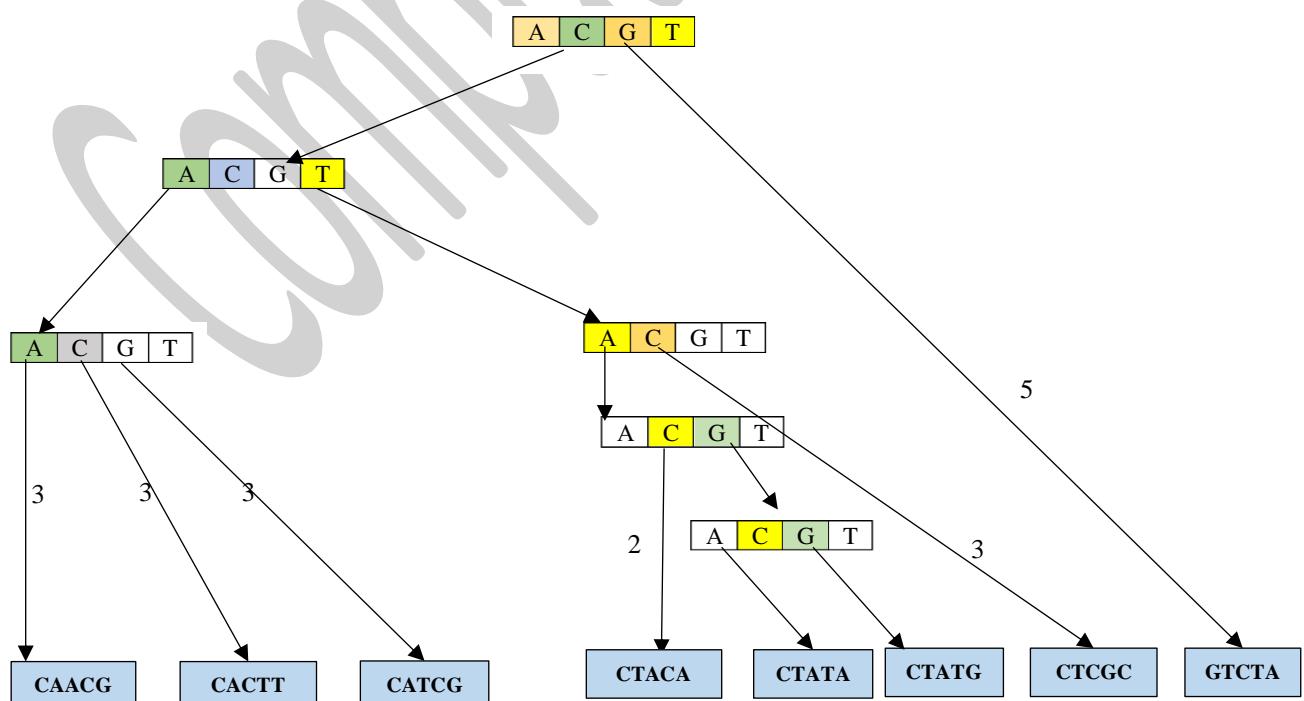
ΣΥΜΠΑΓΕΣ ΔΕΝΤΡΟ TRIE



15.8 Θέμα 3 Φεβρουάριος 2021

Σας δίνετε το αλφάριθμο $\Sigma = \{A, T, C, G\}$ που αποτελεί τις βάσεις του ανθρώπινου γονιδιώματος (DNA). Σύμφωνα με το αλφάριθμο αυτό δημιουργούνται τα στοιχεία {**CTATA**, **CTACA**, **CATCG**, **CAACG**, **CTCGC**, **CACTT**, **CTATG**, **GTCTA**} τα οποία καλείστε να αποθηκεύστε σε δομή Trie χτισμένη στο παραπάνω αλφάριθμο. Σχεδιάστε:

- Την πλήρη αποθήκευση
- Την αποθήκευση που καταλαμβάνει το μικρότερο χώρο

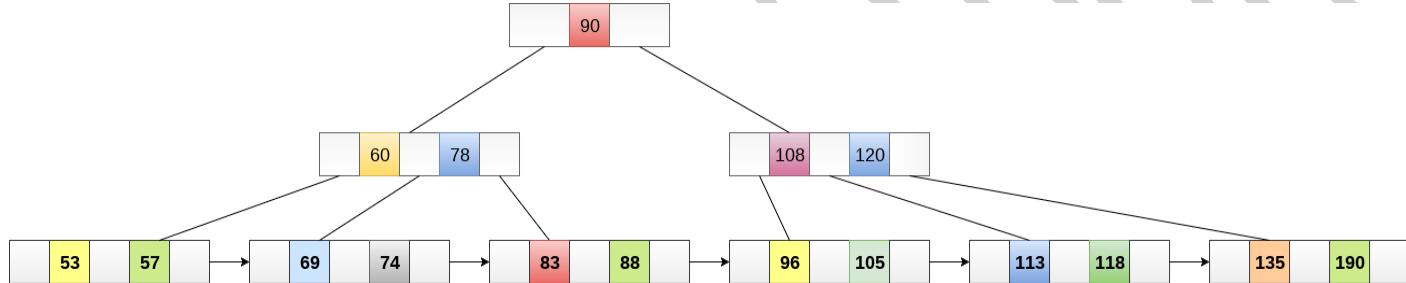
ΑπάντησηΑΡΧΙΚΟ ΔΕΝΤΡΟ TRIE ΜΕ ΠΛΗΡΗ ΑΠΟΘΗΚΕΥΣΗΤΕΛΙΚΟ ΔΕΝΤΡΟ TRIE ΜΕ ΤΟ ΛΙΓΟΤΕΡΟ ΧΩΡΟ

16 ΘΕΜΑΤΑ ΜΕ B+ TREES-OXI

Το B⁺ Tree είναι μια επέκταση του B Tree που επιτρέπει την αποτελεσματική εισαγωγή, διαγραφή και αναζήτηση. Στο B Tree, τα κλειδιά και τα δεδομένα μπορούν να αποθηκευτούν τόσο στους εσωτερικούς κόμβους όσο και στα φύλλα, ενώ στο δέντρο B⁺ **τα δεδομένα μπορούν να αποθηκευτούν μόνο στα φύλλα**, ενώ οι εσωτερικοί κόμβοι αποθηκεύουν μόνο τις τιμές των κλειδιών.
Άρα το B tree είναι ένα Κομβοπροσανατολισμένο δέντρο, ενώ το B⁺ tree είναι ένα φυλλοπροσανατολισμένο δέντρο.

Τα φύλλα ενός δέντρου B⁺ συνδέονται μεταξύ τους με δείκτες για να κάνουν **πιο αποτελεσματική την αναζήτηση**. Τα B⁺ δέντρα χρησιμοποιούνται για την αποθήκευση μεγάλης ποσότητας δεδομένων που δεν μπορούν να αποθηκευτούν στην κύρια μνήμη, λόγω του γεγονότος ότι το μέγεθος της κύριας μνήμης είναι πάντα περιορισμένο. Μόνο οι εσωτερικοί κόμβοι του δέντρου B⁺ αποθηκεύονται στην κύρια μνήμη ενώ τα φύλλα αποθηκεύονται στη δευτερεύουσα μνήμη.

Οι εσωτερικοί κόμβοι του B⁺ δέντρου ονομάζονται συχνά κόμβοι ευρετηρίου. Ένα δέντρο B⁺ τάξης 3 παρουσιάζεται στο παρακάτω σχήμα. Η τάξη του B⁺ δέντρου εκφράζει το πλήθος δεικτών κάθε κόμβου.



Χαρακτηριστικά B+ Tree

- Οι εγγραφές μπορούν να ληφθούν με ίσο αριθμό προσπελάσεων στο δίσκο.
- Το ύψος του δέντρου παραμένει ισορροπημένο και μικρότερο σε σύγκριση με το δέντρο B.
- Μπορούμε να έχουμε πρόσβαση στα δεδομένα που είναι αποθηκευμένα σε ένα δέντρο B⁺ σειριακά καθώς και απευθείας
- Γρήγορες Αναζητήσεις, καθώς τα δεδομένα αποθηκεύονται μόνο στα φύλλα**

Πλεονεκτήματα B+ Tree

- Μπορεί να βρίσκονται πλεονάζοντα κλειδιά.
- Τα δεδομένα αποθηκεύονται στα φύλλα**
- Γρηγορότερη αναζήτηση καθώς τα δεδομένα είναι μόνο στα φύλλα**
- Η διαγραφή είναι γρήγορη γιατί τα στοιχεία διαγράφονται πάντα από τα φύλλα**
- Τα φύλλα διασυνδέονται μεταξύ τους με δείκτες για να κάνουν την αναζήτηση πιο γρήγορη

Κατασκευή B+ Tree

- Το B⁺ Tree κατασκευάζεται με την παράμετρο n που ονομάζεται τάξη δέντρου και εκφράζει το πλήθος δεικτών σε ένα κόμβο
- Κάθε κόμβος εκτός της ρίζας περιέχει από $\left\lceil \frac{n}{2} \right\rceil$ έως n δείκτες και από $\left\lceil \frac{n}{2} \right\rceil - 1$ έως n-1 τιμές
- Κάθε κόμβος εκτός της ρίζας περιέχει κλειδιά (τιμές)=παιδιά-1
- Οι τιμές-κλειδιά είναι ταξινομημένες σε αύξουσα σειρά

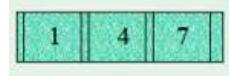
16.1 Παράδειγμα 1 Κατασκευής B+ Δέντρου

Να κατασκευαστεί B⁺ Δέντρο με τις ακόλουθες τιμές: 1, 4, 7, 10, 17, 21, 31, 25, 19, 20, 28, 42 με n=4

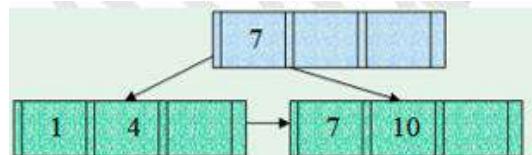
Απάντηση

Το n=4 εκφράζει το πλήθος δεικτών σε κάθε κόμβο. Άρα n-1=3 θα είναι οι εσωτερικές τιμές (κλειδιά) κάθε κόμβου.

+1, +4, +7

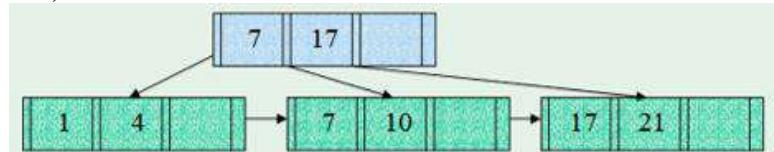


+10



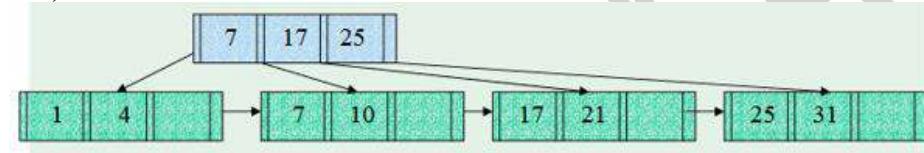
Στη ρίζα τοποθετείται η μικρότερη τιμή του δεξιού φύλλου

+17, +21



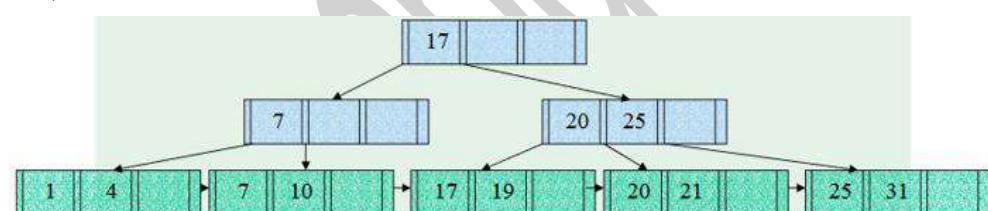
Στη ρίζα τοποθετούνται η μικρότερη τιμή του μεσαίου φύλλου και η μικρότερη τιμή του δεξιού φύλλου

+31, +25



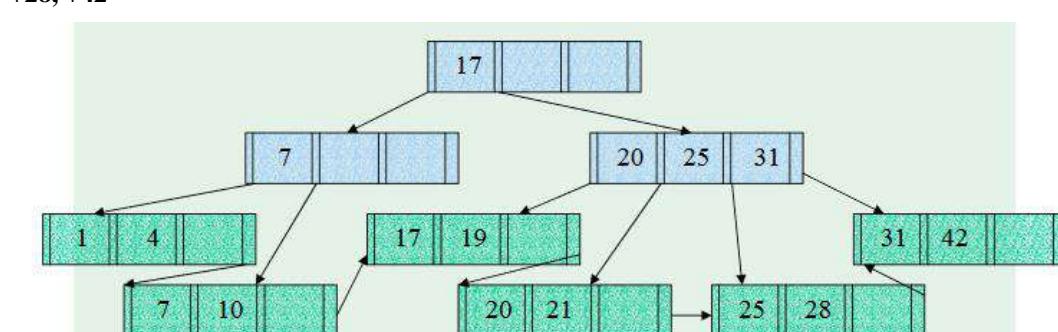
Στη ρίζα τοποθετούνται η μικρότερη τιμή του δεύτερου, τρίτου και τέταρτου φύλλου.

+19, +20



Εδώ υπάρχουν 5 φύλλα και ο αριστερός πατέρας θα πάρει $\left\lfloor \frac{\text{φύλλα}}{2} \right\rfloor = \left\lfloor \frac{5}{2} \right\rfloor = 2$ ενώ ο δεξιός πατέρας τα υπόλοιπα 5-3=2 φύλλα και το δέντρο ανεβαίνει επίπεδο.

+28, +42



16.2 Παράδειγμα 2 Κατασκευής B+ Δέντρου

Να κατασκευαστεί ένα B⁺ Δέντρο για τις ακόλουθες τιμές: 1, 3, 5, 7, 9, 2, 4, 6, 8, 10, 11, 12 με n=4.

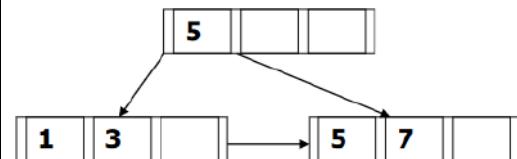
Απάντηση

Το n αντιπροσωπεύει το πλήθος των δεικτών σε κάθε κόμβο, ενώ το πλήθος των κλειδιών είναι n-1. Τα κλειδιά τοποθετούνται ταξινομημένα σε ένα κόμβο. Εδώ σε κάθε κόμβο θα έχουμε 4 δείκτες και 3 κλειδιά το πολύ.

+1, +3, +5

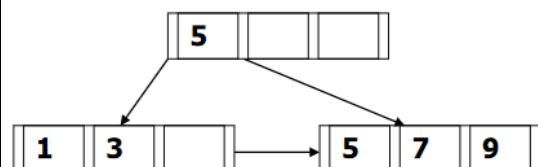


+7

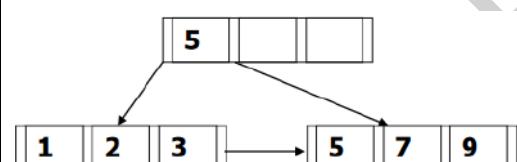


Στον πατέρα τοποθετείται πάντα η μικρότερη τιμή του δεξιού υποδέντρου.

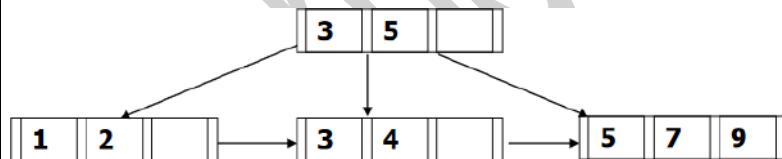
+9



+2

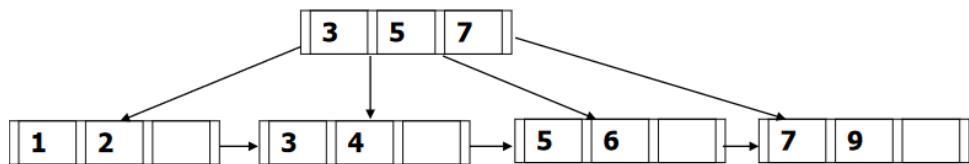


+4

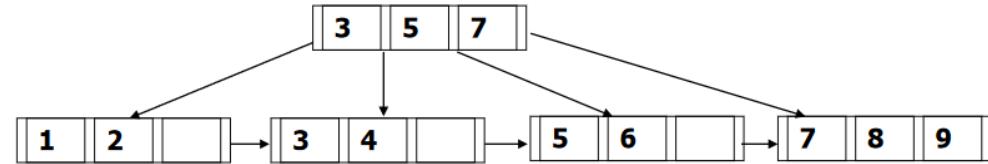


Το 4 τοποθετείται στο φύλλο με τις τιμές 1, 2, 3. Ο κόμβος διασπάται σε δύο. Ο αριστερός κόμβος που προκύπτει από τη διάσπαση παίρνει τις τιμές 1 και 2 ενώ ο δεξιός κόμβος που προκύπτει από τη διάσπαση παίρνει τις τιμές 3 και 4. Η μικρότερη τιμή του μεσαίου φύλλου που είναι το 3 και η μικρότερη τιμή του δεξιού φύλλου που είναι το 5 τοποθετούνται στη ρίζα του δέντρου.

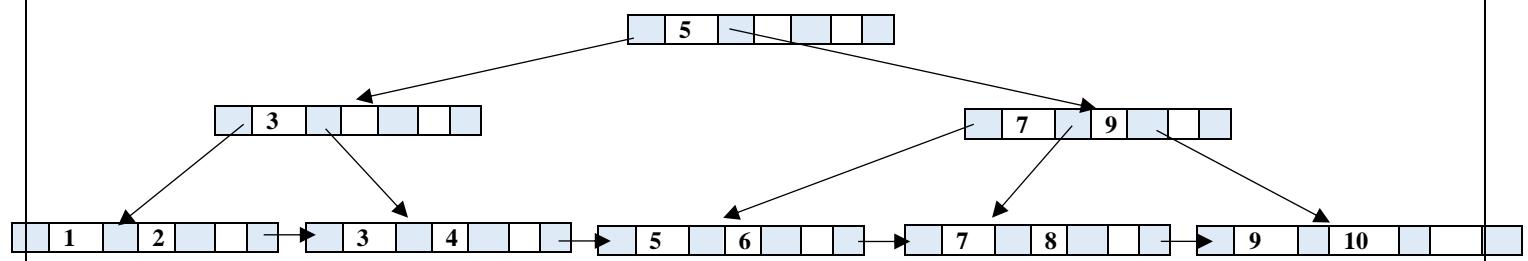
+6



+8

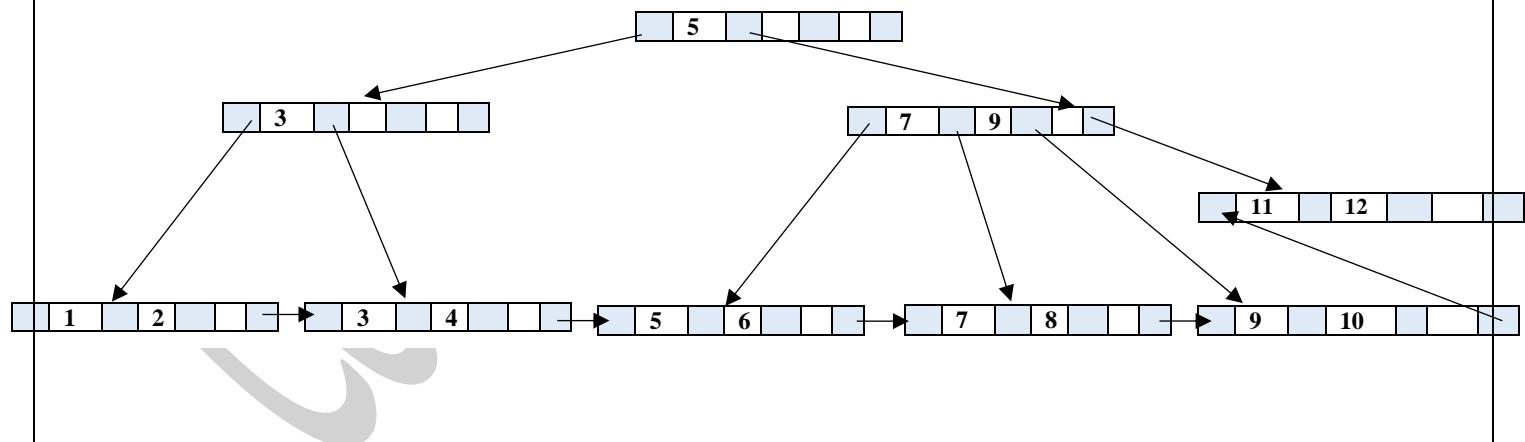


+10



Εδώ υπάρχουν 5 φύλλα και ο αριστερός πατέρας θα πάρει $\left\lfloor \frac{\text{φύλλα}}{2} \right\rfloor = \left\lfloor \frac{5}{2} \right\rfloor = 2$ ενώ ο δεξιός πατέρας τα υπόλοιπα $5-3=2$ φύλλα και το δέντρο ανεβαίνει επίπεδο.

+11, +12

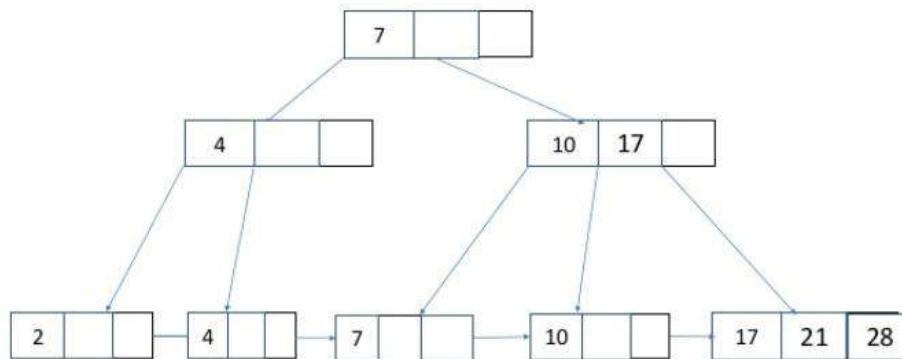


16.3 Παράδειγμα 1 με Διαγραφές από B+ Δέντρο

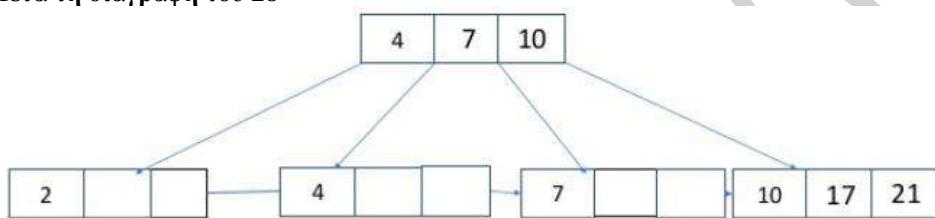
Στο επόμενο B⁺ δέντρο με n=4 εκτελούμε διαδοχικές διαγραφές των τιμών 28, 4, 7 και 2.

Απάντηση

Αρχικό Δέντρο Πριν τη Διαγραφή 28

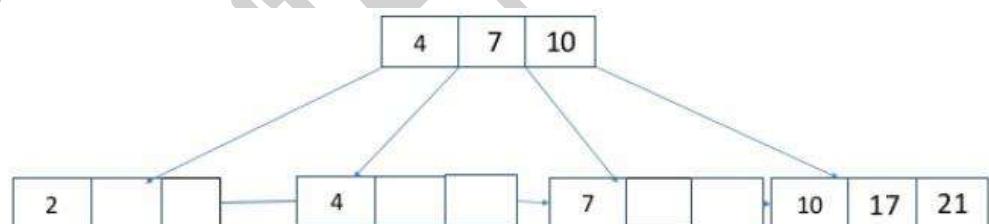


Μετά τη διαγραφή του 28

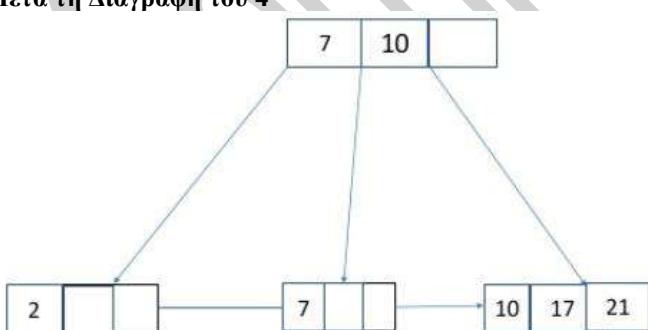


Όταν διαγραφεί το 28 αδειάζει μια θέση στο τελευταίο φύλλο. Στη θέση αυτή μπαίνει το 10. Άρα τα 2 δεξιά φύλλα συνενώνονται σε ένα φύλλο, οπότε το τελευταίο επίπεδο περιλαμβάνει 4 φύλλα και συνεπώς αρκεί μια ρίζα και για 4 φύλλα. Το δέντρο αποκτά λιγότερα επίπεδα

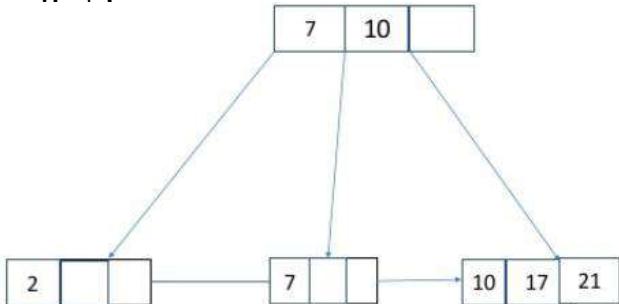
Πριν τη Διαγραφή 4



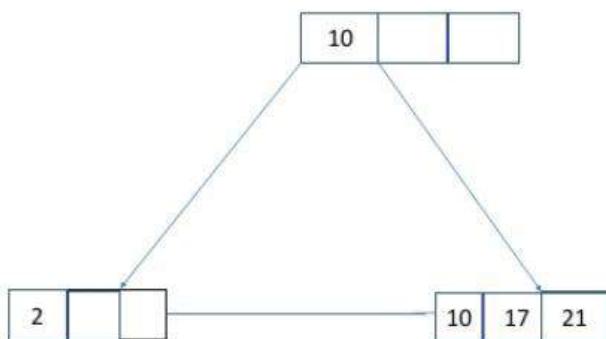
Μετά τη Διαγραφή του 4



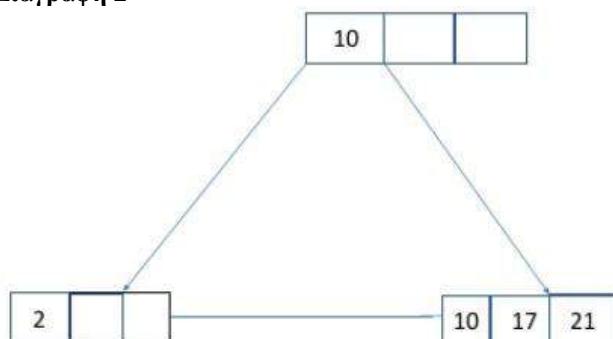
Διαγραφή 7



Μετά τη Διαγραφή του 7



Διαγραφή 2



Μετά τη Διαγραφή του 2 (απομένει μόνο η ρίζα)



17 EXTERNAL SORTING-OXI

17.1 Παράδειγμα με External Sorting

Ο αλγόριθμος External Sorting χρησιμοποιεί 2p ταινίες (p ταινίες εισόδου και p ταινίες εξόδου) και κύρια μνήμη μεγέθους M για να ταξινομήσει στοιχεία. Ο αλγόριθμος αυτός ανήκει στην κατηγορία των αλγορίθμων ταξινόμησης στη δευτερεύουσα μνήμη. Έστω M=3 μέγεθος κύριας μνήμης και p=3 (άρα έχουμε 3 ταινίες εισόδου και 3 ταινίες εξόδου) και θέλουμε να ταξινομήσουμε τη φράση (αγνοούμε τα κενά):

“A SORTING AND MERGING EXAMPLE”

Φάση Α

Εφόσον M = 3, η ακολουθία διαβάζεται κατά μπλοκ μήκους 3 ως εξής:

ASORTINGANDMERGINGEXAMPLE

Κάθε i – οστό ταξινομημένο μπλοκ εγγράφεται στην ταινία ($i \bmod p$)

Π.χ. Μπλοκ 4 = DMN, το γράφουμε ταξινομημένα στην ($4 \bmod 3 = 1$) ταινία, άρα στην 1η ταινία δίπλα από το 1ο μπλοκ.

TAINIA 1	AOS	DMN	AEX
TAINIA 2	IRT	EGR	LMP
TAINIA 3	AGN	GIN	E

Φάση Β

Συγχώνευση (επαναληπτικά) των μπλοκ της Φάσης Α και εγγραφή στην ταινία ($i \bmod p$) + p .

Συγχωνεύουμε τα p = 3 πρώτα μπλοκ από τις ταινίες εισόδου στην κύρια μνήμη και τα γράφουμε στην ταινία p+1. Στη συνέχεια, συγχωνεύουμε τα επόμενα p = 3 μπλοκ από τις ταινίες εισόδου στην κύρια μνήμη και τα γράφουμε στην ταινία p+2 κ.ο.κ.

TAINIA 4	AAGINORST
TAINIA 5	DEGGIMNNR
TAINIA 6	AEELMPX

Φάση Β

Οι p ταινίες (1-3) είναι οι ταινίες εισόδου και οι υπόλοιπες p (4-6) οι ταινίες εξόδου.

Συγχώνευση: Οι ταινίες εισόδου(1-3) “ξανατυλίγονται” σε μία ταινία εξόδου, την Ταινία 1. Οι ταινίες εξόδου(4-6) γίνονται ταινίες εισόδου.

Σαρώνουμε τα πρώτα στοιχεία της κάθε ταινίας και διώχνουμε το μικρότερο στην ταινία εξόδου. Στη θέση του στοιχείου που έφυγε ολισθαίνει το αμέσως επόμενο στοιχείο της ταινίας που αυτό ανήκει.

ΦΑΣΗ Γ

- ▶ Συνεχίζουμε την προηγούμενη διαδικασία μέχρι να εμφανιστεί ένα και μόνο μπλοκ στην έξοδο, δηλαδή στην Ταινία 1 να εμφανιστούν τα γράμματα της ακολουθίας εισόδου ταξινομημένα.

Ταινία 1: **AAADEEEGGGIILMMNNNNOPRRSTX**

18 EXTERNAL SORTING KAI REPLACEMENT SELECTION

Περιγραφή Replacement Selection

Ο Replacement Selection είναι αλγόριθμος που υλοποιεί τη Φάση A του External Sorting δηλ. κάνει την κατασκευή των μπλοκ χαρακτήρων και τα τοποθετεί στις ταινίες. Ο Replacement Selection έχει 2 φάσεις που τις εκτελεί εναλλακτικά και επαναληπτικά: Στην 1^η φάση του όταν ο επόμενος χαρακτήρας είναι μικρότερος (αλφαριθμητικά) από τη Ρίζα του Heap, τότε ο επόμενος χαρακτήρας μπαίνει στο Heap στη θέση της ρίζας με * και δεν συμμετέχει περαιτέρω. Στη 2^η φάση του Replacement Selection όταν ο επόμενος χαρακτήρας είναι μικρότερος αλφαριθμητικά από τη Ρίζα Heap, τότε ο επόμενος χαρακτήρας μπαίνει στο Heap στη θέση της ρίζας χωρίς * και δεν συμμετέχει περαιτέρω.

18.1 Θέμα 6 Ιούνιος 2011

Ταξινομήστε την ακολουθία **SAD WINGS OF DESTINY** με τη μέθοδο ταξινόμησης **External Sorting με Replacement Selection** δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η **κύρια μνήμη** έχει μέγεθος **M=3** και ότι **έχετε 6 ταινίες** στη διάθεση σας (**p=3**).

Απάντηση

Ο αλγόριθμος Replacement Selection είναι η 1^η φάση του External Sorting. Δημιουργεί τα μπλοκ χαρακτήρων και τα τοποθετεί σε ταινίες:

Heap	Ρίζα Heap	Επόμενο	Έξοδος	Σχόλιο
SAD	A	W	A	Επειδή W>A μπαίνει το W στη θέση του A χωρίς *
SWD	D	I	AD	Επειδή I>D μπαίνει το I στη θέση του D χωρίς *
SWI	I	N	ADI	Επειδή N>I μπαίνει το N στη θέση του I χωρίς *
SWN	N	G	ADIN	Επειδή G<N μπαίνει το G στη θέση του N με *
SWG*	S	S	ADINS	Επειδή S=S μπαίνει στη θέση του S χωρίς *
SWG*	S	O	ADINSS	Επειδή O<S μπαίνει O στη θέση του S με *
O*WG*	W	F	ADINSSW-1 ^ο μπλοκ –Ταινία 1	Επειδή F<W μπαίνει F στη θέση του W με *
O*F*G*	F*	D	F*	Επειδή D<F* μπαίνει στη θέση του F* χωρίς *
O*DG*	G*	E	F*G*	Επειδή E<G* μπαίνει στη θέση του G* χωρίς *
O*DE	O*	S	F*G*O*	Επειδή S>O* μπαίνει στη θέση του O* με *
S*DE	S*	T	F*G*O*S*	Επειδή T>S* μπαίνει στη θέση του S* με *
T*DE	T*	I	F*G*O*S*T*-2 ^ο μπλοκ–Ταινία 2	
IDE	D	N	D	Επειδή N>D μπαίνει στη θέση του D χωρίς *
INE	E	Y	DE - 3 ^ο μπλοκ –Ταινία 3	
INY		-	INY - 4 ^ο μπλοκ –Ταινία 1	Ότι είναι στο heap στο τέλος της συμβολοσειράς; εισόδου, ταξινομείται αλφαριθμητικά και μπαίνει ταξινομημένο ως ένα ξεχωριστό μπλοκ στην Ταινία 1

Φάση A – Ταινίες Εισόδου

Ταινία 1	ADINSSW	INY
Ταινία 2	FGOST	
Ταινία 3	DE	

Ταινία 4	ADDEFGINOSSSTW
Ταινία 5	INY

Φάση B-Συγχώνευση στην Ταινία 1

Ταινία 1	ADDEFGHIINNOSSSTWY
----------	--------------------

Computer Ανάλυση

Θέμα 5 Ιούνιος 2016 με External Sorting και Replacement Selection

Ταξινομήστε την ακολουθία **SUMMER BETTER IN FLIP FLOP<5** με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος $M=3$ και ότι έχετε 6 ταινίες στη διάθεση σας ($p=3$). **Υπόδειξη:** Προτεραιότητα Ταξινόμησης $<A\dots Z0\dots 9$

Απάντηση

Το Replacement Selection είναι η Α Φάση του External Sorting και κάνει την κατασκευή των μπλοκ. Δίνεται ότι $M=3$.

Heap	Pίζα Heap	Επόμενο	Έξοδος
SUM	M	M	M
SUM	M	E	MM
SUE*	S	R	MMS
R*UE*	U	B	MMSU- 1 ^o μπλοκ Ταινία 1
R*B*E*	B*	E	B*
R*E*E*	E*	T	B*E*
R*T*E*	E*	T	B*E*E*
R* T*T*	R*	E	B*E*E*R*
ET*T*	T*	R	B*E*E*R*T*
ERT*	T*	I	B*E*E*R*T*T*- 2 ^o μπλοκ Ταινία 2
ERI	E	N	E
NRI	I	F	EI
NRF*	N	L	EIN
L*RF*	R	I	EINR - 3 ^o μπλοκ Ταινία 3
L*I*F*	F*	P	F*
L*I*P*	I*	F	F*I*
L*FP*	L*	L	F*I*L*
L*FP*	L*	O	F*I*L*L*
O*FP*	O*	P	F*I*L*L*O*
P*FP*	P*	<	F*I*L*L*O*P*
<FP*	P*	5	F*I*L*L*O*P*P*- 4 ^o μπλοκ Ταινία 1
<F5*	–	–	<F5*- 5 ^o μπλοκ Ταινία 2

Τα μπλοκ που τοποθετούνται στις ταινίες είναι τα ακόλουθα:

Ταινία 1	MMSU	FILLOPP
Ταινία 2	BEERTT	<F5
Ταινία 3	EINR	

Φάση B-Συγχώνευση ταξινομημένων υπο-ακολουθιών. Συγχωνεύονται οι ταινίες που παρήχθησαν κατά τη Φάση A. Συγκεκριμένα παίρνουμε (συγχωνεύουμε) τα $p=3$ πρώτα μπλοκ από τις ταινίες εισόδου στην κύρια μνήμη και τα γράφουμε στην ταινία $p+1$. Μετά παίρνουμε (συγχωνεύουμε) τα επόμενα $p=3$ μπλοκ από τις ταινίες εισόδου στην κύρια μνήμη και τα γράφουμε στην ταινία $p+2$ κ.λ.π. Παράγονται τα ακόλουθα:

Ταινία 4	BEEEIMMNRRSTTU
Ταινία 5	<FFILLOPP5

Φάση B-Συγχώνευση στην Ταινία 1

Στο τέλος παράγεται η ακολουθία ταξινομημένη ακολουθία στην Ταινία 1: <BEEEFFIILLMMNOPPRRSTTU5

18.2 Θέμα 1 Ιούνιος 2023

Ταξινομήστε την ακολουθία **CHATGPTQUANTUMCOMPUTERS** με τη μέθοδο ταξινόμησης **External Sorting με Replacement Selection**

Selection δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος **M=4** και ότι έχετε **συνολικά 8 ταινίες στη διάθεση σας** ($p=4$). **Υπόδειξη:** Προτεραιότητα Ταξινόμησης ABCDEFGHIJKLMNOPQRSTUVWXYZ.

Απάντηση

Πρώτα εφαρμόζουμε τον αλγόριθμο Replacement Selection που είναι η Φάση A του External Sorting και κάνει την κατασκευή των μπλοκ. Θέλουμε να ταξινομήσουμε τη φράση **CHATGPTQUANTUMCOMPUTERS**. Δίνονται $M=4$ και $p=4$ ταινίες εισόδου και $p=4$ ταινίες εξόδου

Heap	Pίζα Heap	Επόμενο	Έξοδος
CHAT	A	G	A
CHGT	C	P	AC
PHGT	G	T	ACG
PHTT	H	Q	ACGH
PQTT	P	U	ACGHP
UQTT	Q	A	ACGHPQ
UA*TT	T	N	ACGHPQT
UA*N*T	T	T	ACGHPQTT
UA*N*T	T	U	ACGHPQT
UA*N*U	U	M	ACGHPQT
M*A*N*U	U	C	ACGHPQT -1 ^o Μπλοκ-Ταινία 1
M*A*N*C*	A*	O	A*
M*O*N*C*	C*	M	A*C*
M*O*N*M*	M*	P	A*C*M*
P*O*N*M*	M*	U	A*C*M*M*
P*O*N*U*	N*	T	A*C*M*M*N*
P*O*T*U*	O*	E	A*C*M*M*N*O*
P*ET*U*	P*	R	A*C*M*M*N*O*P*
R*ET*U*	R*	S	A*C*M*M*N*O*P*R* -2 ^o Μπλοκ-Ταινία 2
S*ET*U*		-	ES*T*U* - 3 ^o Μπλοκ-Ταινία 3

Τα μπλοκ που τοποθετούνται στις ταινίες εισόδου είναι τα ακόλουθα:

Ταινία 1	ACGHPQT	
Ταινία 2	ACMMNOPR	
Ταινία 3	ESTU	
Ταινία 4		

Δομές Δεδομένων –Computer Ανάλυση

Φάση Β-Συγχώνευση ταξινομημένων υπο-ακολουθιών. Συγχωνεύονται οι ταινίες που παρήχθησαν κατά τη Φάση Α. Τα μπλοκ που παρήχθησαν κατά τη Φάση Α συγχωνεύονται και το αποτέλεσμα γράφεται στις p ταινίες που έχουν απομείνει και οι οποίες ονομάζονται ταινίες εξόδου. Η συγχώνευση επαναλαμβάνεται με την παραγωγή στην έξοδο ενός και μόνο μπλοκ τι οποίο θα αποτελεί την ταξινομημένη έξοδο.

Ταινία 5	AACCEGHMMNOPPQRSTTTUUU
Ταινία 6	
Ταινία 7	
Ταινία 8	

Φάση Β-Συγχώνευση στην Ταινία 1

Ταινία 1: AACCEGHMMNOPPQRSTTTUUU

18.3 Θέμα 4 Σεπτέμβριος 2016

Ταξινομήστε την ακολουθία #DATASTRUCTURES#EXAMS#2016 με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection δεύχοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος $M=3$ και ότι έχετε 6 ταινίες στη διάθεση σας ($p=3$). Υπόδειξη: Προτεραιότητα Ταξινόμησης #A...Z0...9

Απάντηση

Πρώτα εφαρμόζουμε τον αλγόριθμο Replacement Selection που είναι η Φάση A του External Sorting και κάνει την κατασκευή των μπλοκ. Θέλουμε να ταξινομήσουμε τη φράση #DATASTRUCTURES#EXAMS#2016. Δίνεται $M=3$.

Heap	Pίζα Heap	Επόμενο	Έξοδος
#DA	#	T	#
TDA	A	A	#A
TDA	A	S	#AA
TDS	D	T	#AAD
TTS	S	R	#AADS
TTR*	T	U	#AADST
UTR*	T	C	#AADSTT
UC*R*	U	T	#AADSTTU → 1 ^ο Μπλοκ-Ταινία 1
T*C*R*	C*	U	C*
T*U*R*	R*	R	C*R*
T*U*R*	R*	E	C*R*R*
T*U*E	T*	S	C*R*R*T*
SU*E	U*	#	C*R*R*T*U* → 2 ^ο Μπλοκ-Ταινία 2
S#E	#	E	#
SEE	E	X	#E
SXE	E	A	#EE
SXA*	S	M	#EES
M*XA*	X	S	#EESX → 3 ^ο Μπλοκ-Ταινία 3
M*S*A*	A*	#	A*
M*S*#	M*	2	A*M*
2*S*#	S*	0	A*M*S*
2*0*#	0*	1	A*M*S*0*
2*1*#	1*	6	A*M*S*0*1* → 4 ^ο Μπλοκ-Ταινία 1
2*6*#	–	–	#2*6* → 5 ^ο Μπλοκ-Ταινία 2

Τα μπλοκ που τοποθετούνται στις ταινίες εισόδου είναι τα ακόλουθα:

Ταινία 1	#AADSTTU	AMS01
Ταινία 2	CRRTU	#26
Ταινία 3	#EESX	

Δομές Δεδομένων –Computer Ανάλυση

Φάση Β-Συγχώνευση ταξινομημένων υπο-ακολουθιών. Συγχωνεύονται οι ταινίες που παρήχθησαν κατά τη Φάση Α. Τα μπλοκ που παρήχθησαν κατά τη Φάση Α συγχωνεύονται και το αποτέλεσμα γράφεται στις ρ ταινίες που έχουν απομείνει και οι οποίες ονομάζονται ταινίες εξόδου. Η συγχώνευση επαναλαμβάνεται με την παραγωγή στην έξοδο ενός και μόνο μπλοκ τι οποίο θα αποτελεί την ταξινομημένη έξοδο.

Ταινία 4	##AACDEERRSSTTUUX
Ταινία 5	#AMS0126

Φάση Β-Συγχώνευση στην Ταινία 1

Ταινία 1: ###AAACDEEMRRSSSTTUUX0126

18.4 Θέμα 3 Φεβρουάριος 2017

Ταξινομήστε την ακολουθία #PATRASCARNIVAL2017 με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος $M=3$ και ότι έχετε **6 ταινίες στη διάθεση σας** ($p=3$). **Υπόδειξη:** Προτεραιότητα Ταξινόμησης #A...Z0...9

Απάντηση

To Replacement Selection είναι η Φάση A του External Sorting και κάνει την κατασκευή των μπλοκ και την τοποθέτηση στους σε ταινίες. Θέλουμε να ταξινομήσουμε τη φράση #PATRASCARNIVAL2017. Δίνεται ότι $M=3$. Όταν δίνεται $p=3$ σημαίνει ότι έχουμε 3 ταινίες εισόδου και 3 ταινίες εξόδου.

Heap	Πίζα Heap	Επόμενο	Έξοδος
#PA	#	T	#
TPA	A	R	#A
TPR	P	A	#AP
TA*R	R	S	#APR
TA*S	S	C	#APRS
TA*C*	T	A	#APRST-1 ^o μπλοκ Ταινία 1
A*A*C*	A*	R	A*
R*A*C*	A*	N	A*A*
R*N*C*	C*	I	A*A*C*
R*N*I*	I*	V	A*A*C*I*
R*N*V*	N*	A	A*A*C*I*N*
R*AV*	R*	L	A*A*C*I*N*R*
LAV*	V*	2	A*A*C*I*N*R*V*
LA2*	2*	0	A*A*C*I*N*R*V*2*-2 ^o μπλοκ Ταινία 2
LA0	A	1	A
L10	L	7	AL - 3 ^o μπλοκ Ταινία 3
710	—	—	017- 4 ^o μπλοκ Ταινία 1

Τα μπλοκ που τοποθετούνται στις ταινίες είναι τα ακόλουθα:

Ταινία 1	#APRST	017
Ταινία 2	AACINRV2	
Ταινία 3	AL	

Φάση Β-Συγχώνευση ταξινομημένων υπο-ακολουθιών. Συγχωνεύονται οι ταινίες που παρήχθησαν κατά τη Φάση A. Τα μπλοκ που παρήχθησαν κατά τη Φάση A συγχωνεύονται και το αποτέλεσμα γράφεται στις p ταινίες που έχουν απομείνει και οι οποίες ονομάζονται ταινίες εξόδου. Η συγχώνευση επαναλαμβάνεται με την παραγωγή στην έξοδο ενός και μόνο μπλοκ τι οποίο θα αποτελεί την ταξινομημένη έξοδο.

Τανία 4	#AAAACI LNPRRSTV2
Τανία 5	017

Φάση Β-Συγχώνευση στην Τανία 1

Τανία 1: #AAAACILNPRRSTV0127

Computer Analytics

18.5 Θέμα 6 Σεπτέμβριος 2018

Ταξινομήστε την ακόλουθα #PATRASCARNIVAL2018 με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος M=3 και ότι έχετε 6 ταινίες στη διάθεση σας (p=3). **Υπόδειξη: Προτεραιότητα Ταξινόμησης #A...Z0...9**

Απάντηση

To Replacement Selection είναι η Φάση A του External Sorting και κάνει την κατασκευή των μπλοκ και την τοποθέτηση στους σε ταινίες. Θέλουμε να ταξινομήσουμε τη φράση #PATRASCARNIVAL2018. Δίνεται ότι M=3. Όταν δίνεται p=3 σημαίνει ότι έχουμε 3 ταινίες εισόδου και 3 ταινίες εξόδου.

Heap	Pίζα Heap	Επόμενο	Έξοδος
#PA	#	T	#
TPA	A	R	#A
TPR	P	A	#AP
TA*R	R	S	#APR
TA*S	S	C	#APRS
TA*C*	T	A	#APRST-1 ^o μπλοκ Ταινία 1
A*A*C*	A*	R	A*
R*A*C*	A*	N	A*A*
R*N*C*	C*	I	A*A*C*
R*N*I*	I*	V	A*A*C*I*
R*N*V*	N*	A	A*A*C*I*N*
R*AV*	R*	L	A*A*C*I*N*R*
LAV*	V*	2	A*A*C*I*N*R*V*
LA2*	2*	0	A*A*C*I*N*R*V*2*-2 ^o μπλοκ Ταινία 2
LA0	A	1	A
L10	L	8	AL - 3 ^o μπλοκ Ταινία 3
810	—	—	018- 4 ^o μπλοκ Ταινία 1

Τα μπλοκ που τοποθετούνται στις ταινίες είναι τα ακόλουθα:

Ταινία 1	#APRST	018
Ταινία 2	AACINRV2	
Ταινία 3	AL	

Φάση Β-Συγχώνευση ταξινομημένων υπο-ακολουθιών. Συγχωνεύονται οι ταινίες που παρήχθησαν κατά τη Φάση A. Τα μπλοκ που παρήχθησαν κατά τη Φάση A συγχωνεύονται και το αποτέλεσμα γράφεται στις p ταινίες που έχουν απομείνει και οι οποίες ονομάζονται ταινίες εξόδου. Η συγχώνευση επαναλαμβάνεται με την παραγωγή στην έξοδο ενός και μόνο μπλοκ τι οποίο θα αποτελεί την ταξινομημένη έξοδο.

Τανία 4	#AAAACI LNPRRSTV2
Τανία 5	018

Φάση Β-Συγχώνευση στην Τανία 1

Τανία 1: #AAAACILNPRRSTV0128

Computer Analytics

18.6 Θέμα 1 Σεπτέμβριος 2020

Ταξινομήστε την ακολουθία: **SEARCHINALGORITHMSORTAND** με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection, δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η κύρια μνήμη έχει **μέγεθος M=3** και ότι έχετε 6 ταινίες στις διάθεσή σας ($p=3$). (Υπόδειξη: Προτεραιότητα ταξινόμησης ABCDEFGHIJKLMNOPQRSTUVWXYZ)

Απάντηση

To Replacement Selection είναι η A Φάση του External Sorting και κάνει την κατασκευή των μπλοκ. Δίνεται ότι $M=3$.

Heap	Pίζα Heap	Επόμενο	Έξοδος
SEA	A	R	A
SER	E	C	AE
SC*R	R	H	AER
SC*H*	S	I	AERS - 1 ^o μπλοκ – Ταινία 1
I*C*H*	C*	N	C*
I*N*H*	H*	A	C*H*
I*N*A	I*	L	C*H*I*
L*N*A	L*	G	C*H*I*L*
GN*A	N*	O	C*H*I*L*N*
GO*A	O*	R	C*H*I*L*N*O*
GR*A	R*	I	C*H*I*L*N*O*R* - 2 ^o μπλοκ – Ταινία 2
GIA	A	T	A
GIT	G	H	AG
HIT	H	M	AGH
MIT	I	S	AGHI
MST	M	O	AGHIM
OST	O	R	AGHIMO
RST	R	T	AGHIMOR
TST	S	A	AGHIMORS
TA*T	T	N	AGHIMORST
N*A*T	T	D	AGHIMORSTT – 3 ^o μπλοκ – Ταινία 3
N*A*D*			A*D*N* - 4 ^o μπλοκ – Ταινία 1

Τα μπλοκ που τοποθετούνται στις ταινίες είναι τα ακόλουθα:

Ταινία 1	AERS	ADN
Ταινία 2	CHILNOR	
Ταινία 3	AGHIMORSTT	

Φάση Β-Συγχώνευση ταξινομημένων υπο-ακολουθιών. Συγχωνεύονται οι ταινίες που παρήγθησαν κατά τη Φάση Α. Συγκεκριμένα παίρνουμε (συγχωνεύουμε) τα $p=3$ πρώτα μπλοκ από τις ταινίες εισόδου στην κύρια μνήμη και τα γράφουμε στην ταινία $p+1$. Μετά παίρνουμε (συγχωνεύουμε) τα επόμενα $p=3$ μπλοκ από τις ταινίες εισόδου στην κύρια μνήμη και τα γράφουμε στην ταινία $p+2$ κ.λ.π. Παράγονται τα ακόλουθα:

Ταινία 4	AACEGHIIILMNOORRRSSTT
Ταινία 5	ADN

Φάση Β-Συγχώνευση στην Ταινία 1

Στο τέλος παράγεται η ακολουθία ταξινομημένη ακολουθία στην Ταινία 1: AAACDEGHIIILMNNOORRRSSTT

18.7 Θέμα 1 Σεπτέμβριος 2020 – Ομάδα B

Ταξινομήστε την ακόλουθη: THESECONDSEARCHINGALGORITHMUSED με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection, δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος M=3, και ότι έχετε 6 ταινίες στις διάθεσή σας (p=3). (Υπόδειξη: Προτεραιότητα ταξινόμησης ABCDEFGHIJKLMNOPQRSTUVWXYZ)

Απάντηση

To Replacement Selection είναι η A Φάση του External Sorting και κάνει την κατασκευή των μπλοκ. Δίνεται ότι M=3.

Heap	Ρίζα Heap	Επόμενο	Έξοδος
THE	E	S	E
THS	H	E	EH
TE*S	S	C	EHS
TE*C*	T	O	EHST- 1^o μπλοκ-Ταινία 1
O*E*C*	C*	N	C*
O*E*N*	E*	D	C*E*
O*DN*	N*	S	C*E*N*
O*DS*	O*	E	C*E*N*O*
EDS*	S*	A	C*E*N*O*S*-2^o μπλοκ-Ταινία 2
EDA	A	R	A
EDR	D	C	AD
EC*R		H	
		I	
		N	
		G	
		A	
		L	
		G	
		O	
		R	
		I	
		T	
		H	
		M	
		U	
		S	
		E	
		D	

Τα μπλοκ που τοποθετούνται στις ταινίες είναι τα ακόλουθα:

Ταινία 1		
Ταινία 2		
Ταινία 3		

Φάση B-Συγχώνευση ταξινομημένων υπο-ακολουθιών. Συγχωνεύονται οι ταινίες που παρήχθησαν κατά τη Φάση A. Συγκεκριμένα παίρνουμε (συγχωνεύουμε) τα p=3 πρώτα μπλοκ από τις ταινίες εισόδου στην κύρια μνήμη και τα γράφουμε στην ταινία p+1. Μετά

Δομές Δεδομένων –Computer Ανάλυση

παίρνουμε (συγχωνεύουμε) τα επόμενα $p=3$ μπλοκ από τις ταινίες εισόδου στην κύρια μνήμη και τα γράφουμε στην ταινία $p+2$ κ.λ.π.

Παράγονται τα ακόλουθα:

Ταινία 4	
Ταινία 5	

Φάση Β-Συγχώνευση στην Ταινία 1

Στο τέλος παράγεται η ακολουθία ταξινομημένη ακολουθία στην Ταινία 1:

18.8 Θέμα 2 Φεβρουάριος 2021

Ταξινομήστε την ακολουθία: **SIGCOMPWORD2VEC12LEARNING** με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection, δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος $M=3$, και ότι έχετε 6 ταινίες στις διάθεσή σας ($p=3$). (Υπόδειξη: Προτεραιότητα ταξινόμησης ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789)

Απάντηση

Το Replacement Selection είναι η Α Φάση του External Sorting και κάνει την κατασκευή των μπλοκ. Δίνεται ότι $M=3$.

Heap	Pίζα Heap	Επόμενο	Έξοδος
SIG	G	C	G
SIC*	I	O	GI
SOC*	O	M	GIO
SM*C*	S	P	GIOS → 1 ^o μπλοκ Ταινία 1
P*M*C*	C*	W	C*
P*M*W*	M*	O	C*M*
P*O*W*	O*	R	C*M*O*
P*R*W*	P*	D	C*M*O*P*
DR*W*	R*	2	C*M*O*P*R*
D2*W*	W*	V	C*M*O*P*R*W*
D2*V	2*	E	C*M*O*P*R*W*2* → 2 ^o μπλοκ Ταινία 2
DEV	D	C	D
C*EV	E	1	DE
C*1V	V	2	DEV
C*12	1	L	DEV1
C*L*2	2	E	DEV12 → 3 ^o μπλοκ Ταινία 3
C*L*E*	C*	A	C*
AL*E*	E*	R	C*E*
AL*R*	L*	N	C*E*L*
AN*R*	N*	I	C*E*L*N*
AIR*	R*	N	C*E*L*N*R* → 4 ^o μπλοκ Ταινία 1
AIN	A	G	A → 5 ^o μπλοκ Ταινία 2
GIN	–	–	GIN → 6 ^o μπλοκ Ταινία 3

Τα μπλοκ που τοποθετούνται στις ταινίες είναι τα ακόλουθα:

Ταινία 1	GIOS	CELNR
Ταινία 2	CMOPRW2	A
Ταινία 3	DEV12	GIN

Φάση B-Συγχώνευση ταξινομημένων υπο-ακολουθιών. Συγχωνεύονται οι ταινίες που παρήχθησαν κατά τη Φάση A. Συγκεκριμένα παίρνουμε (συγχωνεύουμε) τα $p=3$ πρώτα μπλοκ από τις ταινίες εισόδου στην κύρια μνήμη και τα γράφουμε στην ταινία $p+1$. Μετά παίρνουμε (συγχωνεύουμε) τα επόμενα $p=3$ μπλοκ από τις ταινίες εισόδου στην κύρια μνήμη και τα γράφουμε στην ταινία $p+2$ κ.λ.π. Παράγονται τα ακόλουθα:

Ταινία 4	CDEGIMOOPRSVW122
Ταινία 5	ACEGILNNR

Φάση B-Συγχώνευση στην Ταινία 1

Στο τέλος παράγεται η ακολουθία ταξινομημένη ακολουθία στην Ταινία 1:

Ταινία 1	ACCDEEGGIILMNNOOPRRSVW122
----------	---------------------------

18.9 Θέμα 1 Ομάδα 1 – Σεπτέμβριος 2021

Ταξινομήστε την ακόλουθη: **THE TREEDATASTRUCTURISUSEDCOND** με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection, δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. **Υποθέστε ότι η κύρια μνήμη έχει μέγεθος M=4 και ότι έχετε 8 ταινίες στις διάθεσή σας (p=4).** (Υπόδειξη: Προτεραιότητα ταξινόμησης ABCDEFGHIJKLMNOPQRSTUVWXYZ)

Απάντηση

To Replacement Selection είναι η A Φάση του External Sorting και κάνει την κατασκευή των μπλοκ. Δίνεται ότι M=4.

Heap	Pίζα Heap	Επόμενο	Έξοδος
THET	E	R	E
THRT	H	E	EH
TE*RT	R	E	EHR
TE*E*T	T	D	EHRT
D*E*E*T	T	A	EHRTT – 1^o μπλοκ Ταινία 1
D*E*E*A*	A*	T	A*
D*E*E*T*	D*	A	A*D*
AE*E*T*	E*	S	A*D*E*
AS*E*T*	E*	T	A*D*E*E*
AS*T*T*	S*	R	A*D*E*E*S*
ART*T*	T*	U	A*D*E*E*S*T*
ARU*T*	T*	C	A*D*E*E*S*T*T*
ARU*C	U*	T	A*D*E*E*S*T*T*U* – 2^o μπλοκ Ταινία 2
ARTC	A	U	A
URTC	C	R	AC
URTR	R	I	ACR
UI*TR	R	S	ACRR
UI*TS	S	U	ACRRS
UI*TU	T	S	ACRRST
UI*S*U	U	E	ACRRSTU
E*I*S*U	U	D	ACRRSTUU – 3^o μπλοκ Ταινία 3
E*I*S*D*	D*	C	D*
E*I*S*C	E*	O	D*E*
O*I*S*C	I*	N	D*E*I*
O*N*S*C	N*	D	D*E*I*N* – 4^o μπλοκ Ταινία 4
O*DS*C		–	CDO*S* – 5^o μπλοκ Ταινία 1

Τα μπλοκ που τοποθετούνται στις ταινίες είναι τα ακόλουθα:

Ταινία 1	EHRTT	CDOS
Ταινία 2	ADEESTTU	
Ταινία 3	ACRRSTUU	
Ταινία 4	DEIN	

Φάση Β-Συγχώνευση ταξινομημένων υπο-ακολουθιών. Συγχωνεύονται οι ταινίες που παρήχθησαν κατά τη Φάση Α. Τα μπλοκ που παρήχθησαν κατά τη Φάση Α συγχωνεύονται και το αποτέλεσμα γράφεται στις p ταινίες που έχουν απομείνει και οι οποίες

Δομές Δεδομένων –Computer Ανάλυση

ονομάζονται ταινίες εξόδου. Η συγχώνευση επαναλαμβάνεται με την παραγωγή στην έξοδο ενός και μόνο μπλοκ τι οποίο θα αποτελεί την ταξινομημένη έξοδο.

Ταινία 5	AACDDEEEHINRRRSSTTTUUU
Ταινία 6	CDOS
Ταινία 7	
Ταινία 8	

Φάση Β-Συγχώνευση στην Ταινία 1

Ταινία 1: AACCDDEEEHINORRRSSSTTTUUU

18.10 Θέμα 1 Ομάδα 2 – Σεπτέμβριος 2021

Ταξινομήστε την ακόλουθία: **SEARCHINGALGORITHMUSEDTHESECOND** με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection, δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος M=4, και ότι έχετε 6 ταινίες στις διάθεσή σας (p=4). (Υπόδειξη: Προτεραιότητα ταξινόμησης ABCDEFGHIJKLMNOPQRSTUVWXYZ

Απάντηση

Heap	Pίζα Heap	Επόμενο	Έξοδος
SEAR	A	C	A
SECR	C	H	AC
SEHR	E	I	ACE
SIHR	H	N	ACEH
SINR	I	G	ACEHI
SG*NR	N	A	ACEHIN
SG*A*R	R	L	ACEHINR
SG*A*L*	S	G	ACEHINRS -1 ^ο μπλοκ Ταινία 1
G*G*A*L*	A*	O	A*
G*G*O*L*	G*	R	A*G*
R*G*O*L*	G*	I	A*G*G*
R*I*O*L*	I*	T	A*G*G*I*
R*T*O*L*	L*	H	A*G*G*I*L*
R*T*O*H	O*	M	A*G*G*I*L*O*
R*T*MH	R*	U	A*G*G*I*L*O*R*
U*T*MH	T*	S	A*G*G*I*L*O*R*T*
U*SMH	U*	E	A*G*G*I*L*O*R*T*U* - 2 ^ο μπλοκ Ταινία 2
ESMH	E	D	E
D*SMH	H	T	EH
D*SMT	M	H	EHM
D*SH*T	S	E	EHMS
D*E*H*T	T	S	EHMST - 3 ^ο μπλοκ Ταινία 3
D*E*H*S*	D*	E	D*
E*E*H*S*	E*	C	D*E*
CE*H*S*	E*	O	D*E*E*
CO*H*S*	H*	N	D*E*E*H*
CO*N*S*	N*	D	D*E*E*H*N* - 4 ^ο μπλοκ Ταινία 4
CO*DS*	—	—	CDO*S* - 5 ^ο μπλοκ Ταινία 1

Τα μπλοκ που τοποθετούνται στις ταινίες είναι τα ακόλουθα:

Ταινία 1	ACEHINRS	CDOS
Ταινία 2	AGGIORTU	
Ταινία 3	EHMST	
Ταινία 4	DEEHN	

Δομές Δεδομένων –Computer Ανάλυση

Φάση Β-Συγχώνευση ταξινομημένων υπο-ακολουθιών. Συγχωνεύονται οι ταινίες που παρήχθησαν κατά τη Φάση Α. Τα μπλοκ που παρήχθησαν κατά τη Φάση Α συγχωνεύονται και το αποτέλεσμα γράφεται στις p ταινίες που έχουν απομείνει και οι οποίες ονομάζονται ταινίες εξόδου. Η συγχώνευση επαναλαμβάνεται με την παραγωγή στην έξοδο ενός και μόνο μπλοκ τι οποίο θα αποτελεί την ταξινομημένη έξοδο.

Ταινία 5	AACDEEEEGGHHHIILMNNORRSSTTU
Ταινία 6	CDOS

Φάση Β-Συγχώνευση στην Ταινία 1

Ταινία 1: AACCDDEEEEGGHHHIILMNNOORRSSSTTU

Ταξινομήστε την ακολουθία: **HASHINGINCHATINGWITHLISTS** με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection, δείχνοντας όλα τα ενδιάμεσα βήματα του αλγορίθμου. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος M=4 και ότι έχετε 8 ταινίες στις διάθεσή σας (p=4). (**Υπόδειξη:** Προτεραιότητα ταξινόμησης ABCDEFGHIJKLMNOPQRSTUVWXYZ

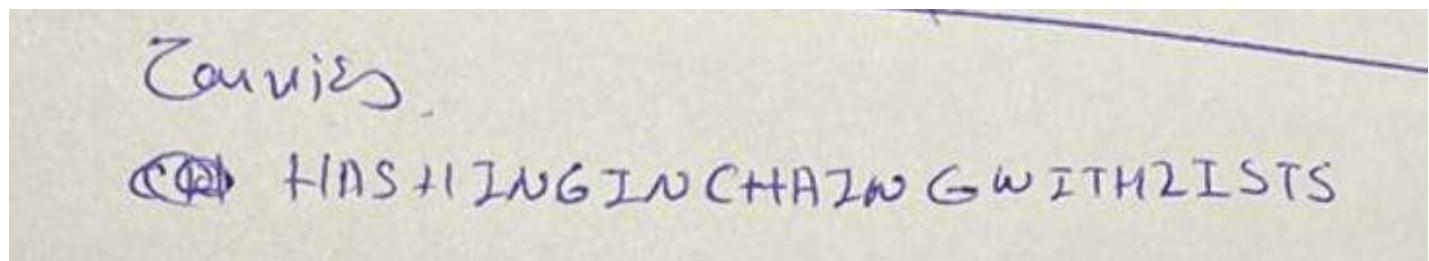
Απάντηση

Heap	Pίζα Heap	Επόμενο	Έξοδος
HASH	A	I	A
HISH	H	N	AH
NISH	H	G	AHH
NISG*	I	I	AHHI
NISG*	I	N	AHHII
NNSG*	N	C	AHHIIN
C*NSG*	N	H	AHHIINN
C*H*SG*	S	A	AHHIINNS –1^ο μπλοκ – Ταινία 1
C*H*A*G*	A*	T	A*
C*H*T*G*	C*	I	A*C*
I*H*T*G*	G*	N	A*C*G*
I*H*T*N*	H*	G	A*C*G*H*
I*GT*N*	I*	W	A*C*G*H*I*
W*GT*N*	N*	I	A*C*G*H*I*N*
W*GT*I	T*	T	A*C*G*H*I*N*T*
W*GT*I	T*	H	A*C*G*H*I*N*T*T*
W*GHI	W*	L	A*C*G*H*I*N*T*T*W* –2^ο μπλοκ – Ταινία 2
LGHI	G	I	G
LIHI	H	S	GH
LISI	I	T	GHI
LTSI	I	S	GHII* –3^ο μπλοκ – Ταινία 3
LTSS		–	LSST –4^ο μπλοκ – Ταινία 4

Ταινία 1	AHHIINNS
Ταινία 2	ACGHINTTW
Ταινία 3	GHII
Ταινία 4	GHII

Φάση Β-Συγχώνευση ταξινομημένων υπο-ακολουθιών. Συγχωνεύονται οι ταινίες που παρήχθησαν κατά τη Φάση Α. Τα μπλοκ που παρήχθησαν κατά τη Φάση Α συγχωνεύονται και το αποτέλεσμα γράφεται στις p ταινίες που έχουν απομείνει και οι οποίες ονομάζονται ταινίες εξόδου. Η συγχώνευση επαναλαμβάνεται με την παραγωγή στην έξοδο ενός και μόνο μπλοκ τι οποίο θα αποτελεί την ταξινομημένη έξοδο.

Ταινία 5	AHHIINNS ACGHINTTW GHII GHII
Ταινία 6	
Ταινία 7	
Ταινία 8	



18.11 Θέμα 1 Ομάδα 2 – Ιούνιος 2022

<u>1ος Όρθια:</u>	Zanies. HASHING IN CHAIN WITH LISTS
<u>2ος Όρθια</u>	Λ γραδικός γραμμής Λ αυτομετρία πλεονάσματος. Φυλαγές των διαδικτικών δεντρών.
<u>3ος Όρθια</u>	hashing + double hashing Στο επιπλέον είρημα είναι μεταξύ "σύγκρουσης"
<u>4ος Όρθια</u>	MVL
<u>5ος Όρθια</u>	Θετικό (2, 4)
<u>6ος Όρθια</u>	TRIE με τα γενήγενα των
<u>7ος Όρθια</u>	Heapsort.

19 ΘΕΜΑΤΑ ME B TREES

Ένα B-Tree τάξης m έχει τα ακόλουθα χαρακτηριστικά:

1. Όλα τα φύλλα είναι στο ίδιο επίπεδο
2. Όλοι οι εσωτερικοί κόμβοι (εκτός ρίζας) έχουν το πολύ m και το λιγότερο $m/2$ παιδιά
3. Στους εσωτερικούς κόμβους του δέντρου ο αριθμός κλειδιών ισούται με τον αριθμό των παιδιών -1 , ενώ για τα φύλλα ο αριθμός των κλειδιών είναι το πολύ $m-1$ και το λιγότερο $m/2$
4. Η ρίζα έχει το λιγότερο 2 παιδιά εκτός αν το δέντρο έχει μόνο ρίζα

19.1 Παράδειγμα 1 Κατασκευής B-Tree με $m=5$

1. Όλα τα φύλλα είναι στο ίδιο επίπεδο
2. Όλοι οι εσωτερικοί κόμβοι (εκτός ρίζας) έχουν το πολύ 5 και το λιγότερο 2 παιδιά
3. Στους εσωτερικούς κόμβους ο αριθμός κλειδιών ισούται με τον αριθμό των παιδιών -1 κόμβους, ενώ για τα φύλλα ο αριθμός των κλειδιών είναι το πολύ $5-1=4$ και το λιγότερο $5/2=2$
4. Η ρίζα έχει το λιγότερο 2 παιδιά εκτός αν το δέντρο έχει μόνο ρίζα

Θέλουμε να κατασκευάσουμε ένα B=δέντρο τάξης 5 για τους ακόλουθους χαρακτήρες:

A G F B K D H M J E S I R X C L N T U P

Βήμα 1

A G F B K D H M J E S I R X C L N T U P

A	B	F	G
---	---	---	---

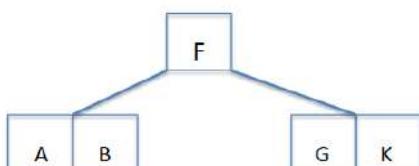
Βήμα 2

A G F B K D H M J E S I R X C L N T U P

A	B	F	G	K
---	---	---	---	---

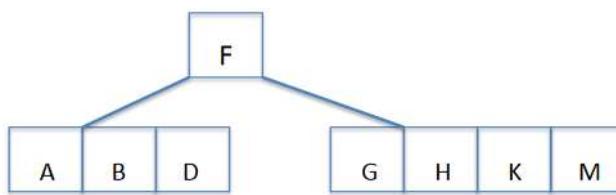
Βήμα 3

A G F B K D H M J E S I R X C L N T U P



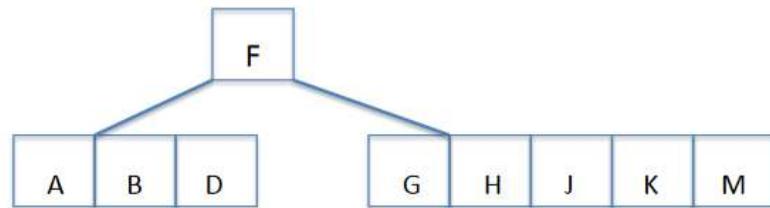
Βήμα 4

A G F B K D H M J E S I R X C L N T U P



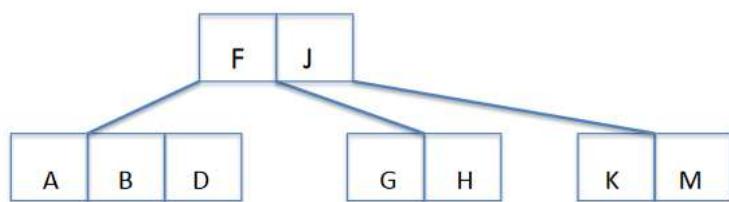
Βήμα 5

A G F B K D H M J E S I R X C L N T U P



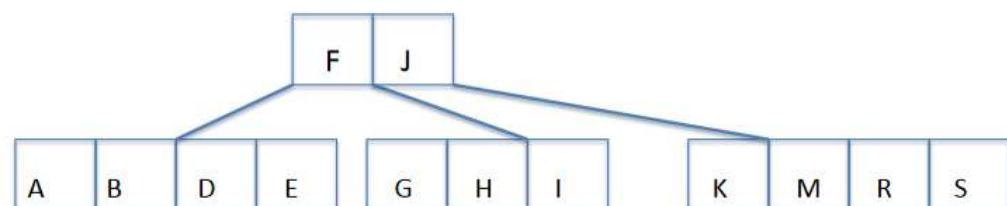
Βήμα 6

A G F B K D H M J E S I R X C L N T U P



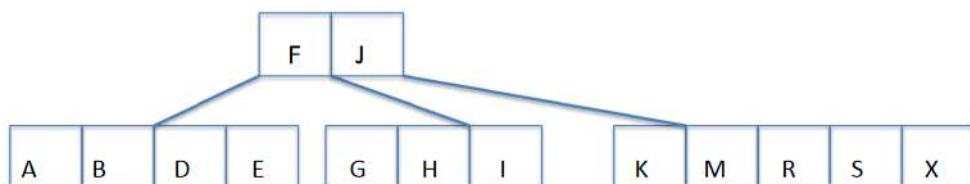
Βήμα 7

A G F B K D H M J E S I R X C L N T U P



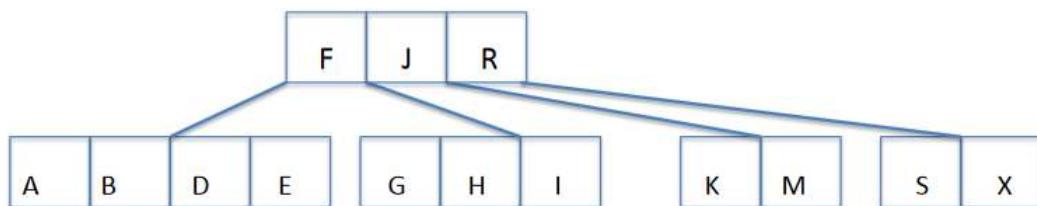
Βήμα 8

A G F B K D H M J E S I R X C L N T U P



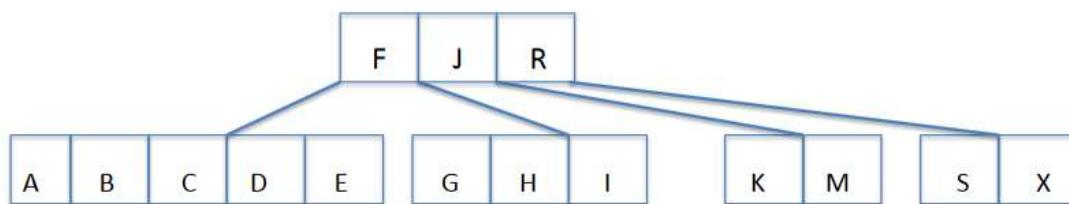
Βήμα 9

AGFBKDHMJESIRXCLNTUP



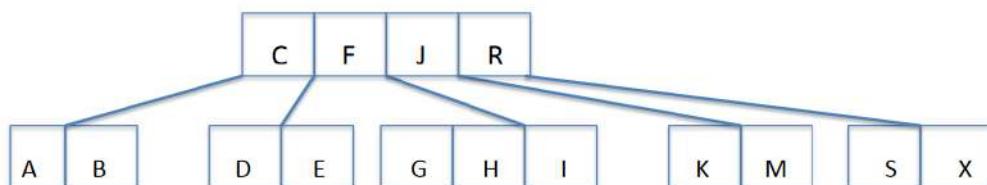
Βήμα 10

AGFBKDHMJESIRXCLNTUP



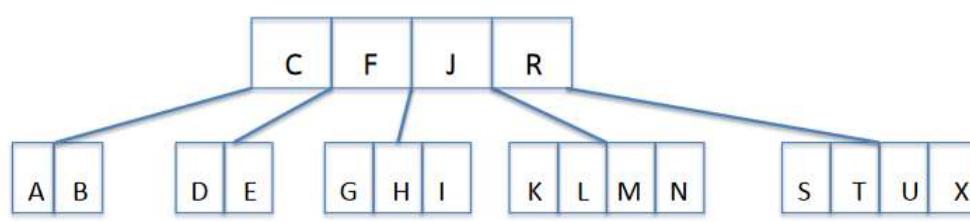
Βήμα 11

AGFBKDHMJESIRXCLNTUP



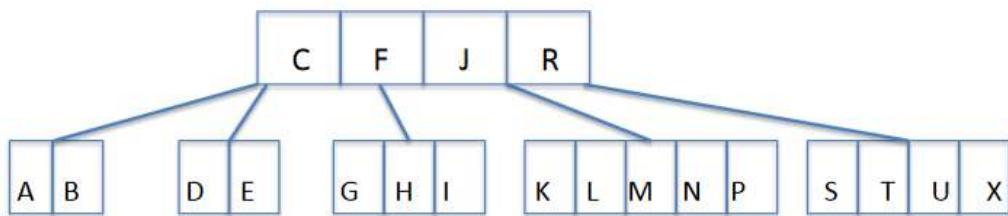
Βήμα 12

AGFBKDHMJESIRXCLNTUP

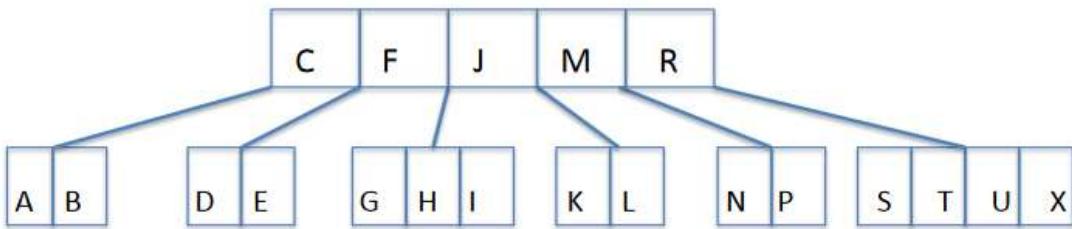


Βήμα 13

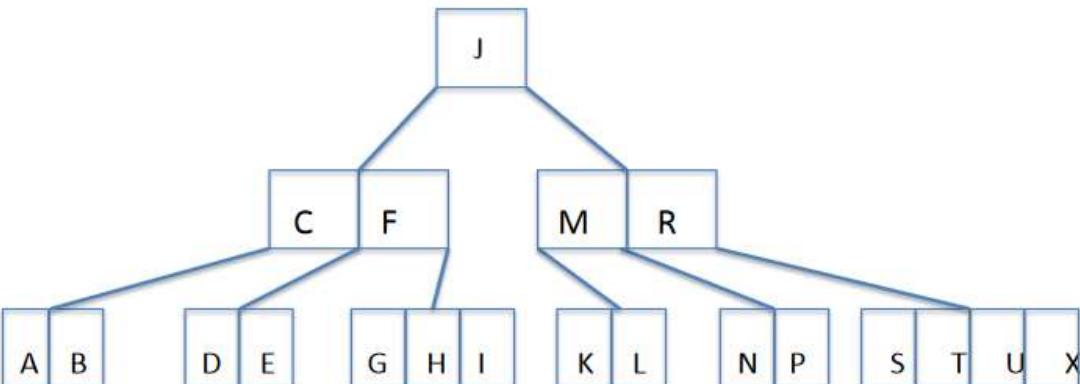
A G F B K D H M J E S I R X C L N T U P



Βήμα 14

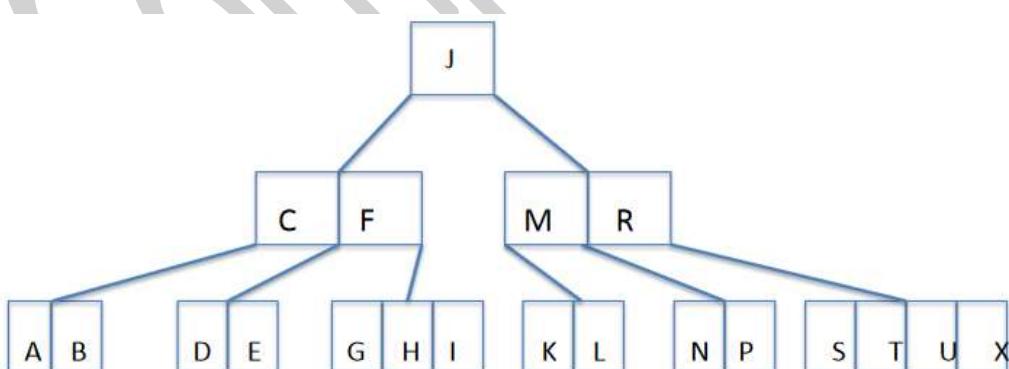


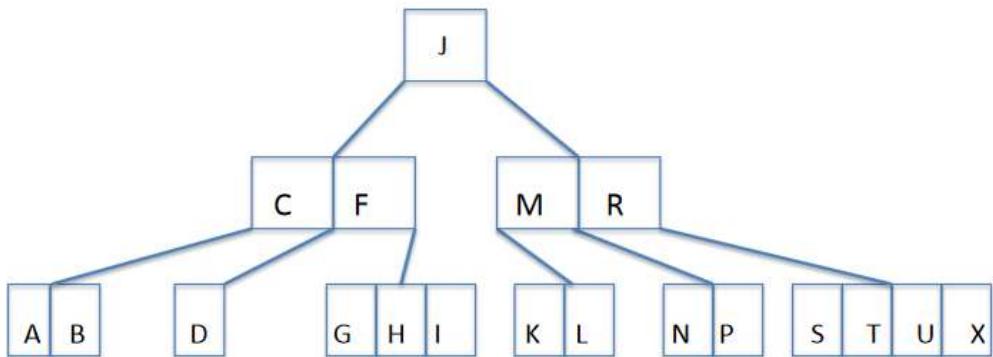
Βήμα 15



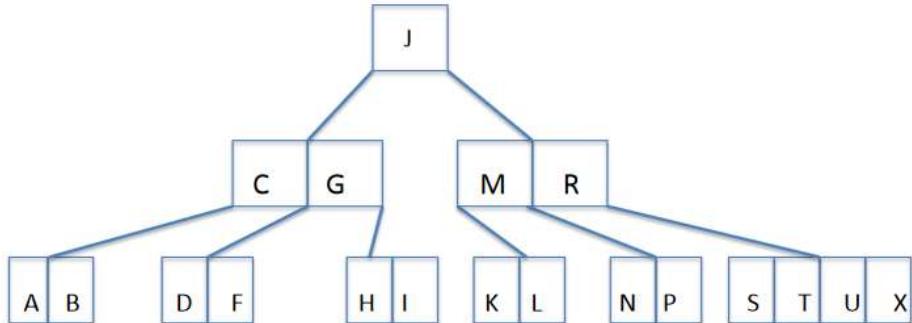
Βήμα 16

Διαγραφή E από φύλλο



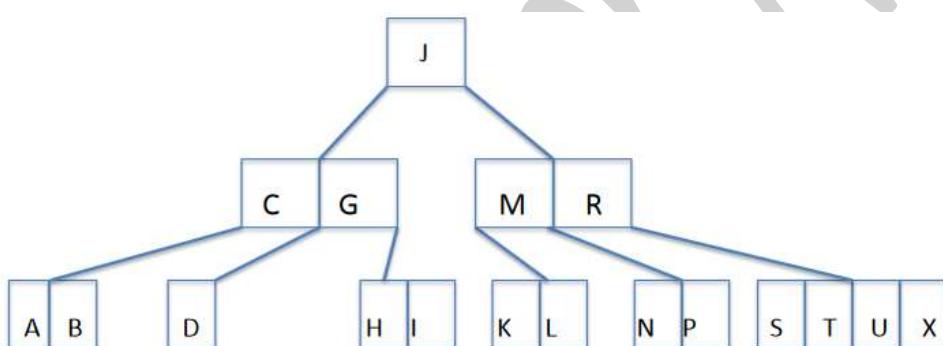


Δανεισμός από πλούσιο Γείτονα

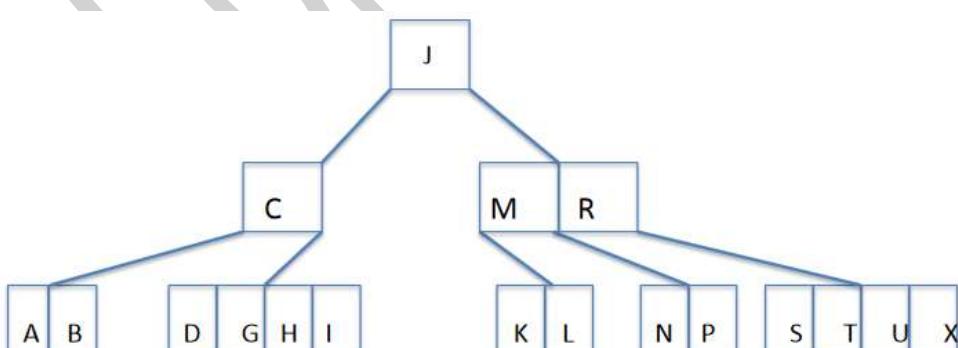


Βήμα 17

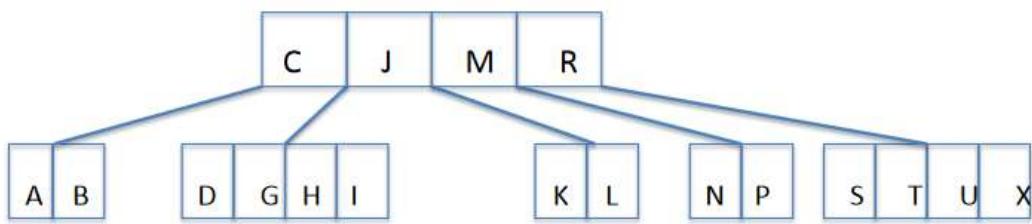
Διαγραφή F – Δεν μπορεί να γίνει δανεισμός από γείτονα



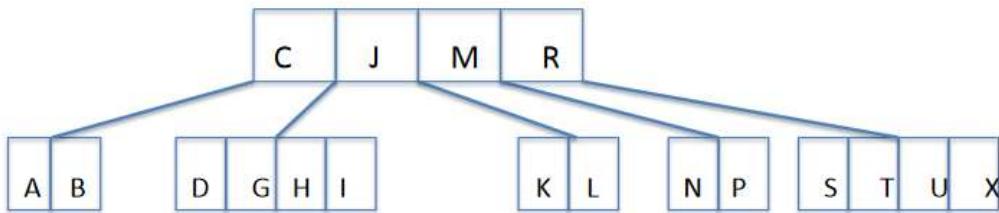
Συνδυασμός και ανέβασμα ενός επιπέδου



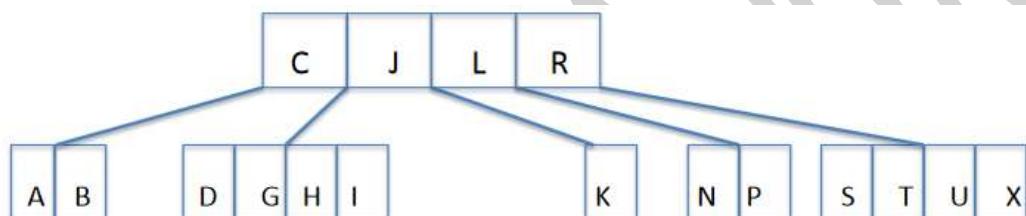
Δεν μπορεί να δανειστεί άρα συνδυασμός



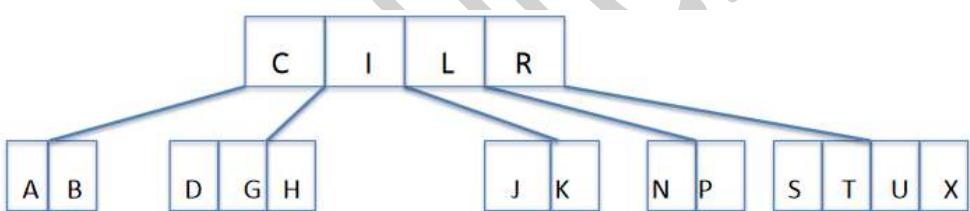
Διαγραφή M από εσωτερικό κόμβο



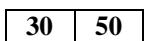
Αντικατάσταση M με τον άμεσο πρόγονο



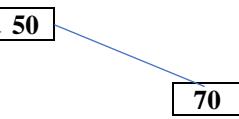
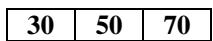
Δανεισμός από γείτονα



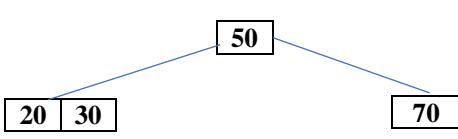
Βήμα 1



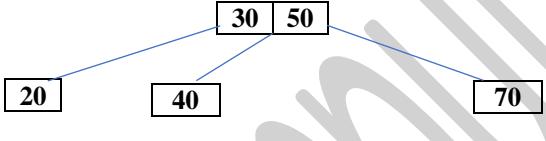
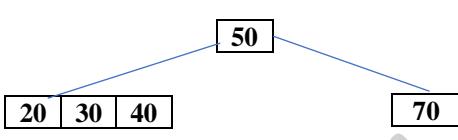
Βήμα 2



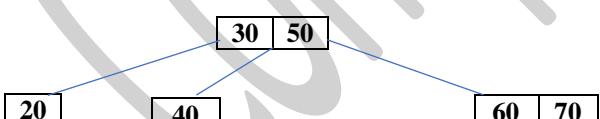
Βήμα 3



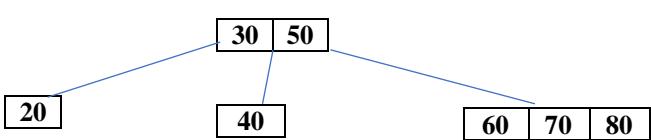
Βήμα 4

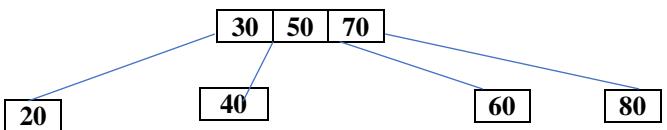


Βήμα 5

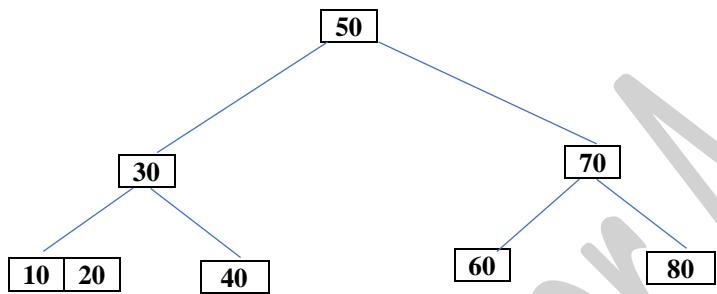


Βήμα 6

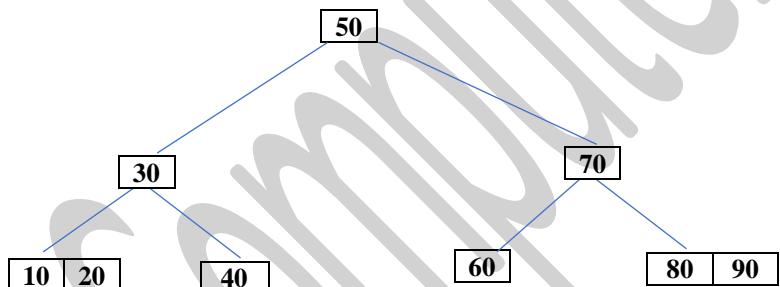




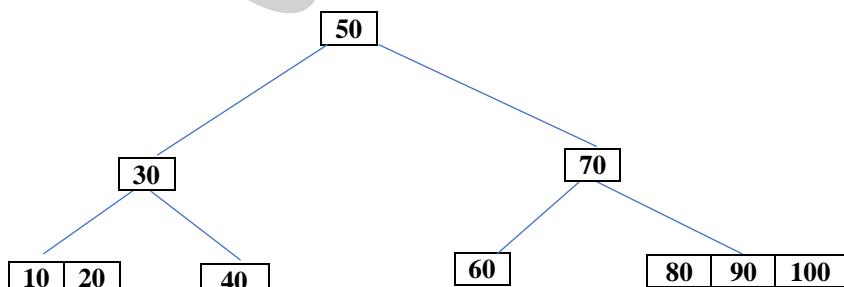
Βήμα 7

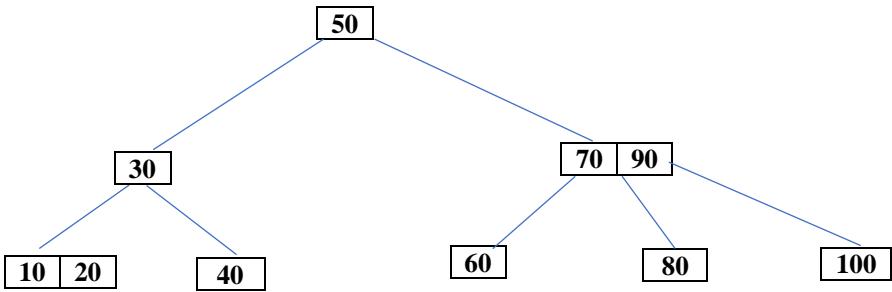


Βήμα 8

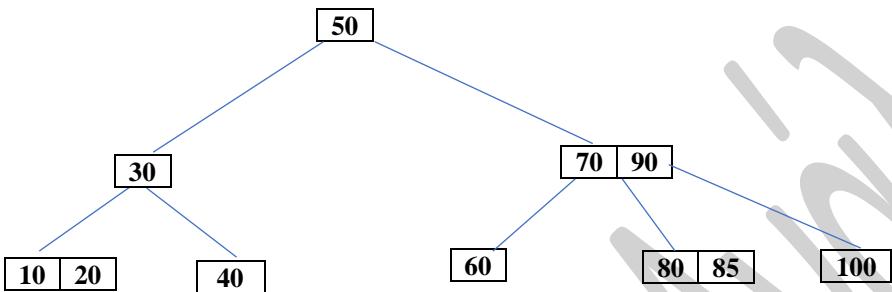


Βήμα 9

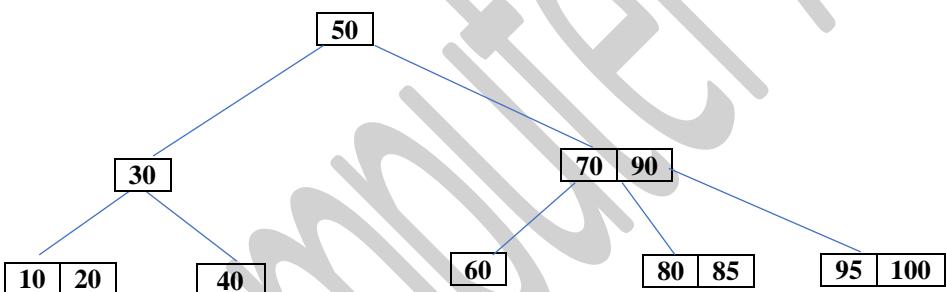




Βήμα 10

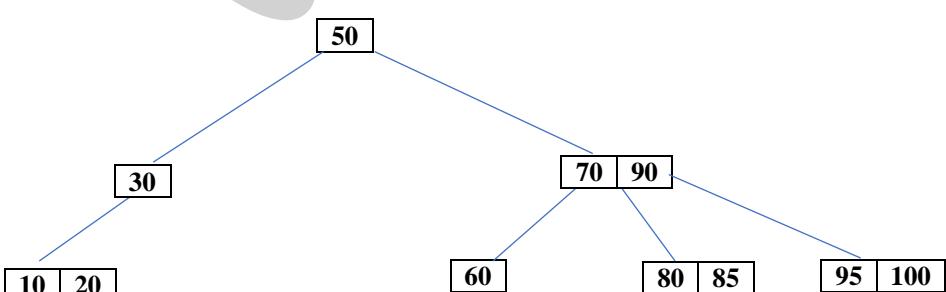


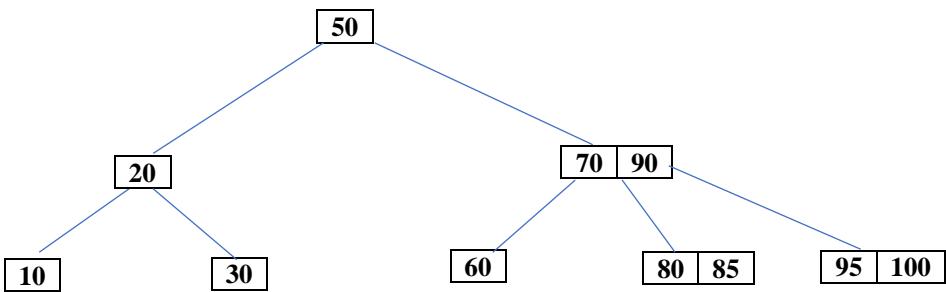
Βήμα 11



Βήμα 11

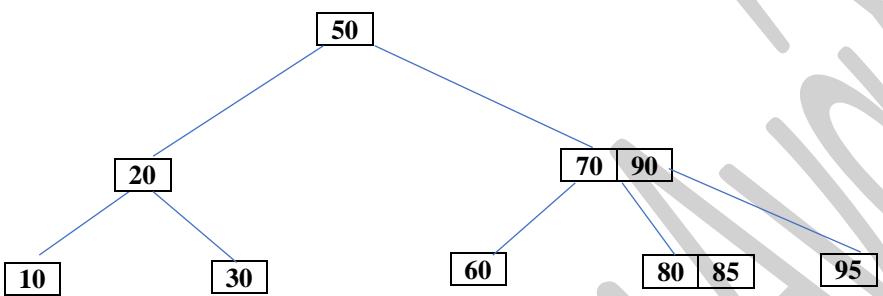
Διαγραφή 40





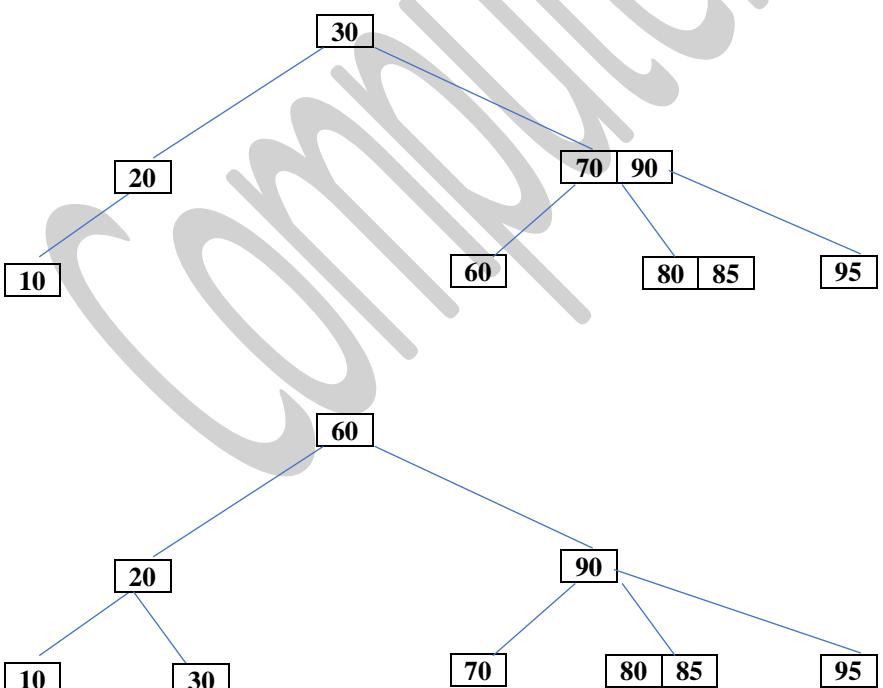
Βήμα 12

Διαγραφή 100



Βήμα 13

Διαγραφή 50



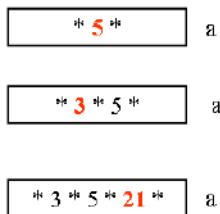
19.2 Παράδειγμα 2 Κατασκευής B-Tree τάξης 4

Κάθε κόμβος εκτός της ρίζας έχει τουλάχιστον $\text{ceil}(4/2)=2$ παιδιά (και τουλάχιστον 1 κλειδί αντίστοιχα) και το πολύ 4 παιδιά (και το πολύ 3 κλειδιά αντίστοιχα). Τα κλειδιά είναι ο αριθμός των παιδιών-1. Γενικά σε ένα B-Tree το πλήθος των παιδιών είναι από το ακέραιο πηλίκο του $B/2$ έως B

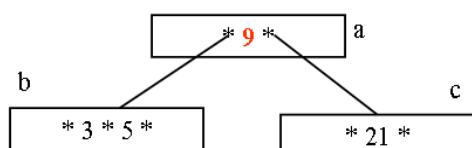
Παράδειγμα

- Insert: 5, 3, 21, 9, 1, 13, 2, 7, 10, 12, 4, 8
- Delete: 2, 21, 10, 3, 4

Insert 5, 3, 21

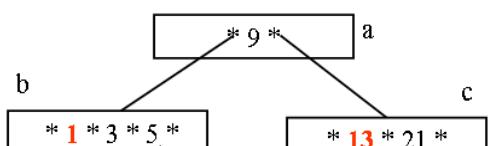


Insert 9



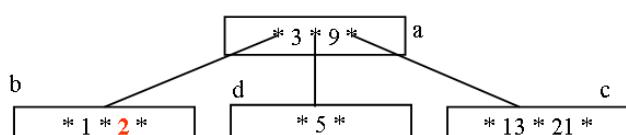
Node a splits creating 2 children: b and c

Insert 1, 13

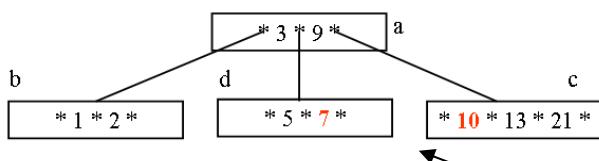


Insert 2

Το 2 πρέπει να τοποθετηθεί μεταξύ 2 και 3. Επειδή δεν χωρά στον κόμβο, ο κόμβος διασπάται

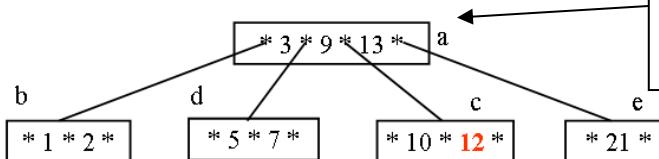


Insert 7, 10



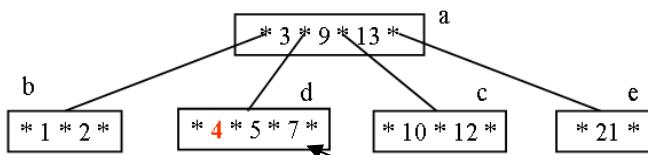
Το 12 πρέπει να τοποθετηθεί μεταξύ 10 και 13. Επειδή δεν χωρά στον κόμβο, ο κόμβος διασπάται

Insert 12



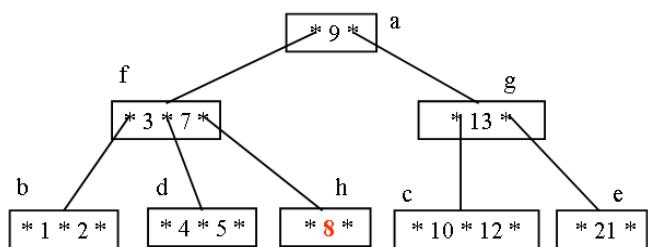
Η ρίζα έχει 3 κλειδιά (διότι έχει 4 παιδιά). Επίσης γεμίζουμε τα φύλλα με κλειδιά από αριστερά προς τα δεξιά

Insert 4



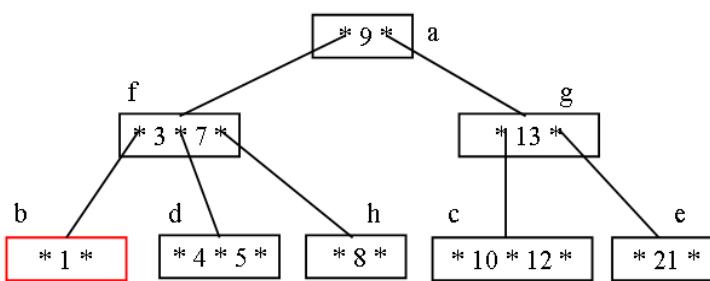
Το 8 πρέπει να τοποθετηθεί μετά το 7. Επειδή δεν χωρά στον κόμβο, ο κόμβος διασπάται

Insert 8



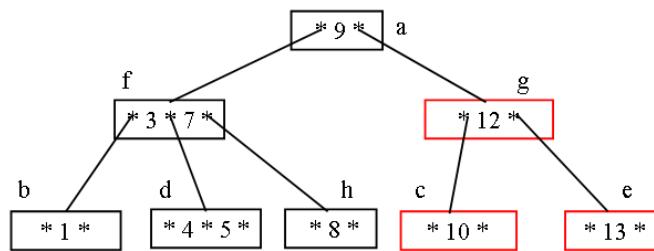
Στη Διαγραφή προσπαθούμε να διατηρήσουμε αμετάβλητη τη δομή του δέντρου (αν αυτό είναι δυνατό να γίνει).

Delete 2



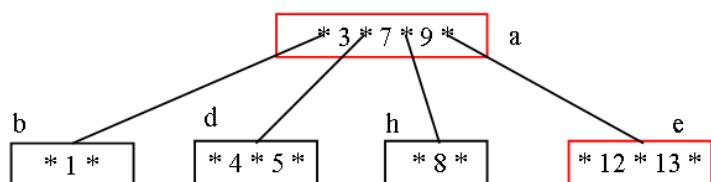
Στη διαγραφή του 21 ο αριστερός του γείτονας είναι πλούσιος και δανείζει κλειδιά

Delete 21

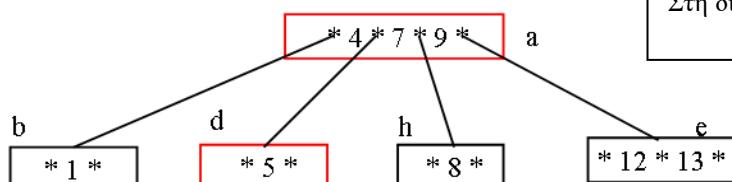


Στη διαγραφή του 10 ο αριστερός του γείτονας είναι φτωχός και γίνεται σύμπτυξη κόμβων

Delete 10

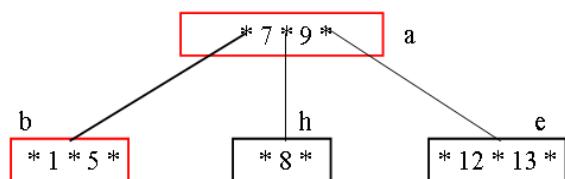


Delete 3



Στη διαγραφή του 3 υπάρχει παιδί που δανείζει κλειδί

Delete 4

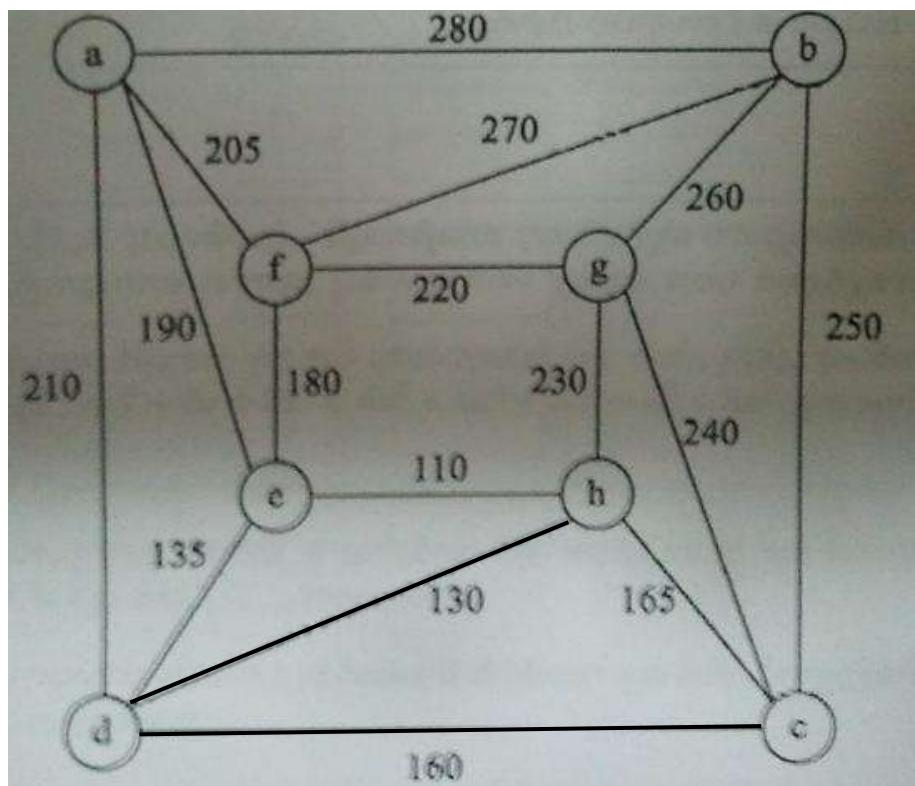


Στη διαγραφή του 4 δεν υπάρχει αριστερό ή δεξιό παιδί που να μπορεί να δανείσει κλειδί και γιαυτό γίνεται σύμπτυξη κόμβων

20 ΑΛΓΟΡΙΘΜΟΣ KRUSKAL

20.1 Παράδειγμα 1

Να βρείτε το Ελάχιστο Γεννητικό Δέντρο (ΕΓΔ) του παρακάτω γραφήματος με τη μέθοδο Kruskal. Το ΕΓΔ να δοθεί σαν μια ακολουθία πλευρών του γραφήματος με τη σειρά κατά την οποία προστίθενται οι πλευρές στο ΕΓΔ.

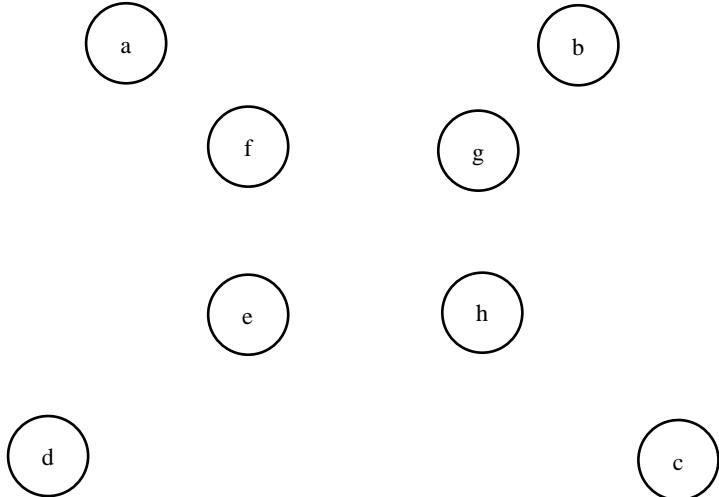


Απάντηση

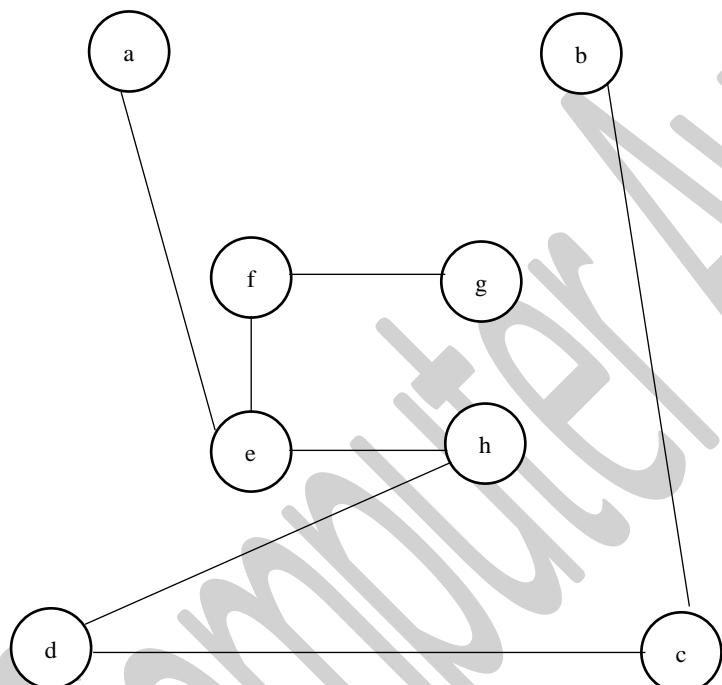
Βήμα 1 Γράφουμε όλες τις ακμές του γράφου ταξινομημένες σε αύξουσα σειρά. Επειδή το γράφημα είναι μη διευθυνόμενο ο τρόπος γράφης των ακμών δεν έχει σημασία

1. (e, h) → 110
2. (d, h) → 130
3. (d, e) → 135
4. (d, c) → 160
5. (h, c) → 165
6. (e, f) → 180
7. (a, e) → 190
8. (a, f) → 205
9. (a, d) → 210
10. (f, g) → 220
11. (g, h) → 230
12. (g, c) → 240
13. (b, c) → 250
14. (g, b) → 260
15. (f, b) → 270
16. (a, b) → 280

Βίμα 2 – Σχεδιάζουμε το γράφημα μόνο με τις κορυφές του



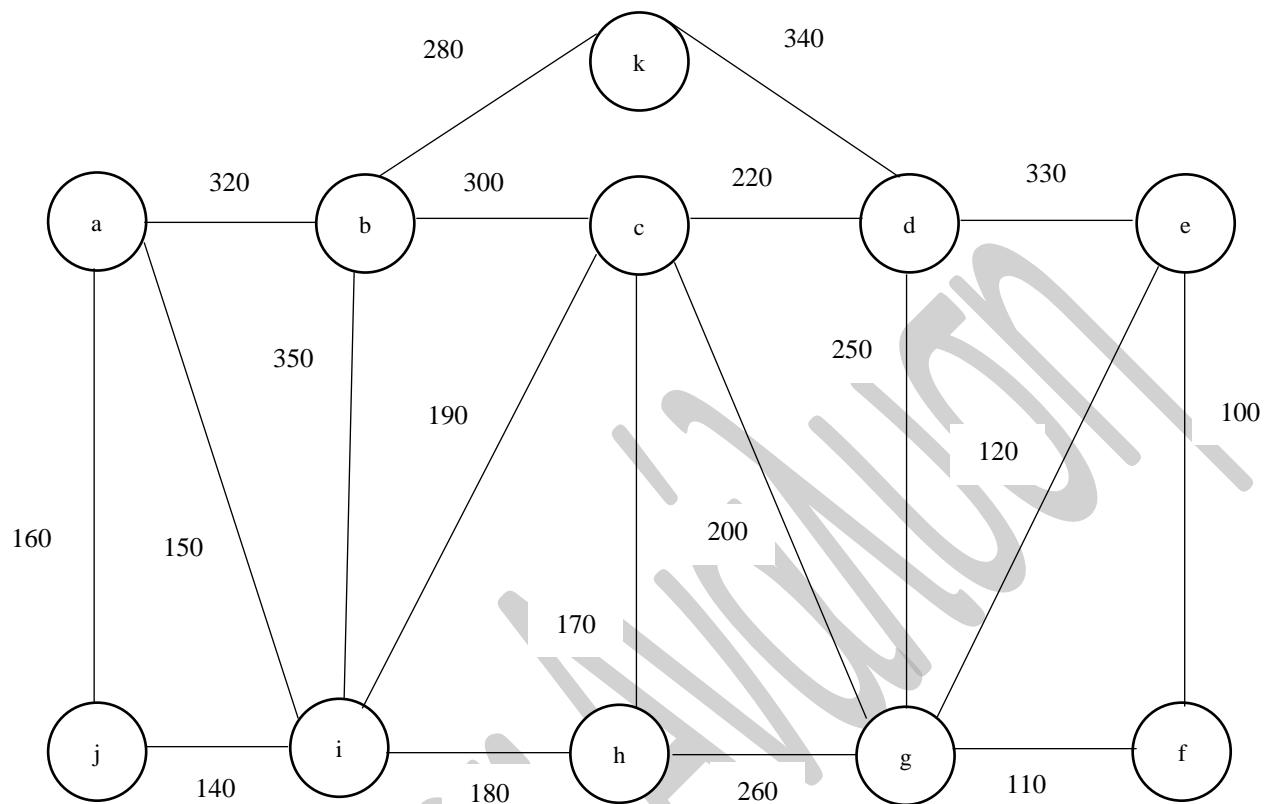
Βίμα 3 – Τοποθετούμε στο γράφο όλες τις ακμές σε αύξουσα σειρά ΔΙΑΓΡΑΦΟΝΤΑΣ ΑΥΤΕΣ ΠΟΥ ΚΛΕΙΝΟΥΝ ΚΥΚΛΟ



Η απάντηση είναι $\text{ΕΓΔ} = \{(e, h), (d, h), (d, c), (e, f), (a, e), (f, g), (b, c)\}$

20.2 Παράδειγμα 2

Να βρείτε το Ελάχιστο Γεννητικό Δέντρο (ΕΓΔ) του παρακάτω γραφήματος με τη μέθοδο Kruskal. Το ΕΓΔ να δοθεί σαν μια ακολουθία πλευρών του γραφήματος με τη σειρά κατά την οποία προστίθενται οι πλευρές στο ΕΓΔ.



Απάντηση

Kruskal

Βήμα 1

1. $(e, f) \rightarrow 100$
2. $(g, f) \rightarrow 110$
3. $(g, e) \rightarrow 120$
4. $(i, j) \rightarrow 140$
5. $(a, i) \rightarrow 150$
6. $(a, j) \rightarrow 160$
7. $(c, h) \rightarrow 170$
8. $(i, h) \rightarrow 180$
9. $(i, c) \rightarrow 190$
10. $(c, g) \rightarrow 200$
11. $(c, d) \rightarrow 220$
12. $(d, g) \rightarrow 250$
13. $(h, g) \rightarrow 260$
14. $(b, k) \rightarrow 280$

15. (b, c)→300

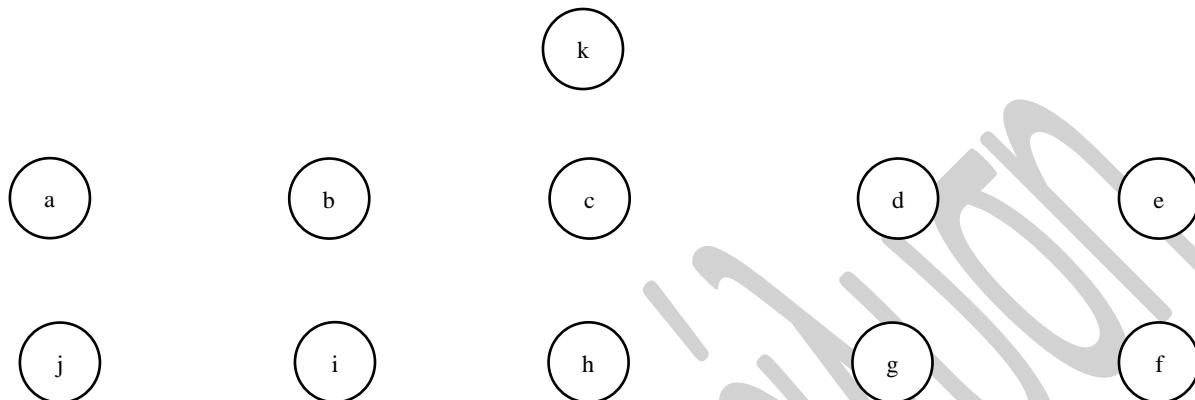
16. (a, b)→320

17. (d, e)→330

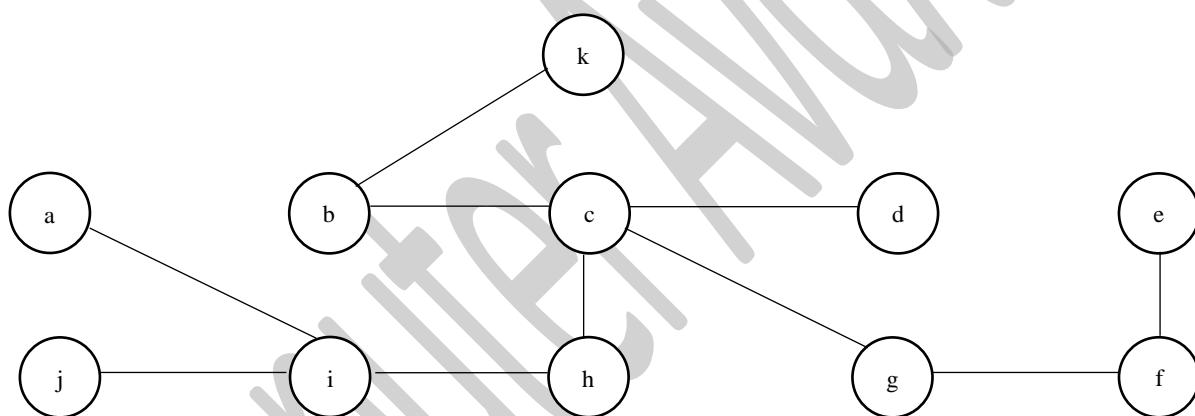
18. (k, d)→340

19. (b, i)→350

Βήμα 2



Βήμα 3



Οι ακόλουθες ακμές διαγράφονται διότι δημιουργούν κύκλο

- (g, e)
- (a, j)
- (i, c)
- (d, g)
- (h, g)
- (a, b)
- (d, e)
- (k, d)
- (b, i)

Η απάντηση είναι: ΕΓΔ= {(e, f), (g, f), (i, j), (a, i), (c, h), (i, h), (c, g), (c, d), (b, k), (b, c)}

21 ΕΡΩΤΗΣΕΙΣ ΘΕΩΡΙΑΣ

21.1 SOS Αρχικοποίηση Πίνακα σε χρόνο O(1) – Σεπτέμβριος 2018

Σας δίνεται ένας πίνακας $A[0..N-1]$. Να δώσετε ένα τρόπο να γίνει αρχικοποίηση του πίνακα με μηδενικά σε $O(1)$. Που έχει εφαρμογή μια τέτοια μέθοδος;

Εναλλακτική Διατύπωση: Δίνεται πίνακας $A[0..N-1]$ ο οποίος πρέπει να αρχικοποιηθεί με την τιμή 0 σε κάθε θέση του. Να δώσετε ένα τρόπο για να αποφευχθεί η αρχικοποίηση.

Απάντηση

Πρόβλημα: Δίνεται array $A [0..N-1]$, το οποίο πρέπει να αρχικοποιηθεί με την τιμή 0, σε κάθε θέση του. Να δώσετε ένα τρόπο για να αποφευχθεί η αρχικοποίηση.

Απάντηση:

Χρησιμοποιούμε επιπλέον πίνακες AUX και S , και οι δυο μεγέθους N , χωρίς να χρειάζονται αρχικοποίηση. Ο S υλοποιεί μια στοίβα η οποία κρατά όλες τις θέσεις του A που έχουν έγκυρες τιμές. Οι θέσεις του πίνακα AUX χρησιμοποιούνται για τυχαία προσπέλαση της στοίβας με τρόπο που θα φανεί παρακάτω. Επίσης χρησιμοποιείται η μεταβλητή TOP που δείχνει στην κορυφή της στοίβας. Αρχικά $TOP=-1$, συνθήκη ισοδύναμη με την αρχικοποίηση του A με μηδενικά.

Κατά τη διάρκεια της εκτέλεσης, διατηρούμε την εξής συνθήκη:

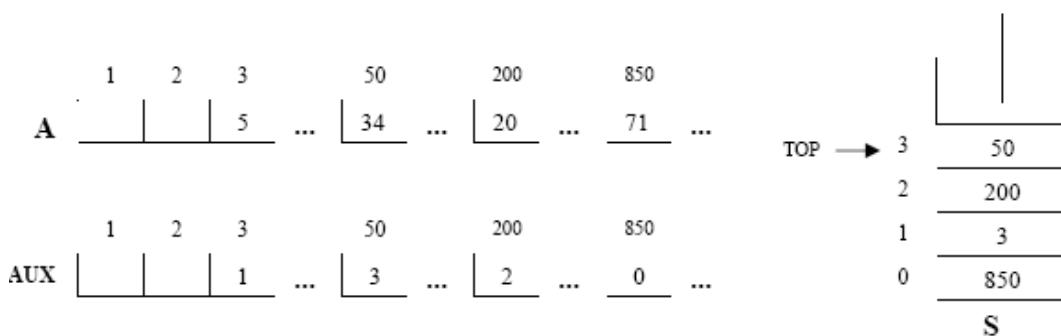
$$A[i] = s, \text{ αν και μόνο αν } 0 \leq AUX[i] \leq TOP \text{ && } S[AUX[i]] = i \quad (1)$$

Οπότε, όταν χρειαστεί να γράψουμε ή να διαβάσουμε μια τιμή από το $A[i]$, ελέγχουμε την τήρηση της παραπάνω συνθήκης. Αν επαληθεύεται, μπορούμε να συνεχίσουμε να κάνουμε την πράξη. Διαφορετικά το i , πρέπει να εισαχθεί στη στοίβα και να ενημερωθεί γι' αυτό και ο AUX , δηλαδή

```
TOP = TOP+1;
S[TOP] = i;
AUX[i] = TOP;
A[i] = s;
```

Έστω ότι ο A δεν έχει καμία τιμή και ότι για πρώτη φορά αποφασίζουμε να γράψουμε στη θέση $A[850]$. Τότε αυξάνουμε το TOP , λαμβάνοντας $TOP=0$ και θέτουμε $S[0]=850$, δηλαδή ενημερώνουμε τη στοίβα ότι ο A έχει έγκυρη τιμή στη θέση $A[850]$.

Το παρακάτω σχήμα δίνει τα περιεχόμενα των A , AUX , S , μετά την ακολουθία ενθέσεων $A[850]=71$, $A[3]=5$, $A[200]=20$, $A[50]=34$



Παρατηρήσεις:

Μόνο ο έλεγχος $0 \leq \text{AUX}[i] \leq \text{TOP}$ δεν θα αρκούσε γιατί ο AUX είναι μη αρχικοποιημένος. Συνεπώς θα μπορούσε, για παράδειγμα, για κάποιο i , να ισχύει $0 \leq \text{AUX}[i] \leq \text{TOP}$, αλλά ο AUX να περιέχει σκουπίδια στη συγκεκριμένη θέση μνήμης

και η επαλήθευση της ανισότητας να είναι γι' αυτό το λόγο ψευδής. Αυτή η περίπτωση μπορεί να απεικονιστεί στο παραπάνω σχήμα αν επιχειρήσει κάποιος να προσπελάσει τη διεύθυνση $A[910]$ και έστω $\text{AUX}[910]=1$. Επειδή $S[1]=3$ σημαίνει ότι η θέση $A[910]$ δεν έχει αρχικοποιηθεί και άρα περιέχει μη έγκυρα δεδομένα. Το επόμενο βήμα θα ήταν να γινόταν εισαγωγή του 910 στη στοίβα και να τεθεί $A[910]=0$ (τιμή αρχικοποίησης)

Η στοίβα δεν μπορεί να χρησιμοποιηθεί από μόνη της γιατί θέλουμε να επαληθεύουμε αν κάποια τιμή του i , έχει περαστεί μέσα στη στοίβα χωρίς μεγάλη επιβάρυνση. Για να πετύχουμε $O(1)$ επιβάρυνση ανά βήμα προσπέλασης χρειαζόμαστε και τη βοήθεια του AUX. (Πόση θα ήταν η επιβάρυνση αν θα χρησιμοποιούσαμε μόνο τη στοίβα;)

Εφαρμογές:

Η παραπάνω τεχνική έχει εφαρμογή στους μεταφραστές όπου όταν χρειάζεται η αρχικοποίηση ενός τεράστιου πίνακα, αυτή αποφεύγεται και ο πίνακας απλώς παίρνει τιμές, όταν αυτό είναι αναγκαίο από το πρόγραμμα που τον χρησιμοποιεί. Μια δεύτερη εφαρμογή (βλ. Mehlhorn p.289) είναι η υλοποίηση boolean πινάκων που χρησιμοποιούνται για την αναπαράσταση ενός συνόλου. Έστω για παράδειγμα ένα σύνολο $S \subseteq U$, $U=\{1, 2, \dots, N\}$. Για την αναπαράσταση του S χρησιμοποιείται πίνακας BV, τέτοιος ώστε, $BV[i] = 1$ ανν $i \in S$ και $BV[i] = 0$, $i \notin S$. Προφανώς για τη σωστή αναπαράσταση του συνόλου, όταν αυτό υπόκειται στις πράξεις Insert, Delete, χρειάζεται αρχικοποίηση του BV με την τιμή 0. Δεν μπορούμε να αποδεχτούμε την υπόθεση ότι δεν αρχικοποιούμε το A, και όποτε χρειαστεί πηγαίνουμε στις αντίστοιχες θέσεις και τις θέτουμε σε true. Η έλλειψη του true δεν σημαίνει false. Αυτό γιατί μια πράξη Access μπορεί προσπελάζοντας μια θέση του A να συναντήσει την τιμή true, η οποία όμως αντιστοιχεί σε σκουπίδια που υπήρχαν στην αντίστοιχη θέση μνήμης και όχι σε πραγματική τιμή και άρα να οδηγηθούμε σε λάθος.

21.2 Τι είναι το Λεξικό Δεδομένων και ποιες πράξεις υποστηρίζει;

Απάντηση

- Έστω σύνολο $S \subseteq U$ (σύμπαν), $x \in S, \inf(x)$
- Το λεξικό είναι δομή δεδομένων που υποστηρίζει τις πράξεις:

Access(x): if $x \in S$ true return($\inf(x)$) else false Στατικές Δομές

Delete: $S = S - \{x\}$

Insert: $S = S \cup \{x\}$

Δυναμικές Δομές

21.3 Ποιοι οι τρόποι αναζήτησης στο Λεξικό Δεδομένων;

Απάντηση

- Η γραμμική αναζήτηση η οποία εφαρμόζεται σε ΜΗ ταξινομημένο πίνακα. Το $next \leftarrow next + 1$ και αρχικά $next \leftarrow left$
- Η δυαδική (ή δυική) αναζήτηση (binary search) που εφαρμόζεται σε ταξινομημένο πίνακα. Το $next \leftarrow \lfloor (left + right) / 2 \rfloor$
- Η αναζήτηση παρεμβολής (interpolation search) που εφαρμόζεται σε ταξινομημένο πίνακα. Το $next$ δίνεται από τον τύπο:

$$next \leftarrow \left\lfloor (high - low) \cdot \frac{key - table[low]}{table[high] - table[low]} \right\rfloor + low$$

21.4 Ποια η διαφορά του μοντέλου RAM από το μοντέλο PM. Σε ποια βασική υπόθεση στηρίζεται η ανάλυση πολυπλοκότητας των αλγορίθμων που μελετάμε;

Απάντηση

Η διαφορά του μοντέλου RAM από το μοντέλο PM είναι ότι **το μοντέλο PM δεν επιτρέπει την προσπέλαση με υπολογισμό διεύθυνσης**. Προσπέλαση σε μια συγκεκριμένη διεύθυνση μπορεί να γίνει μόνο αν υπάρχει δείκτης σε αυτή τη διεύθυνση. Το μοντέλο PM είναι ασθενέστερο της RAM. Η βασική υπόθεση στην οποία στηρίζεται η ανάλυση πολυπλοκότητας των αλγορίθμων που μελετάμε είναι ότι δουλεύουμε **σε RAM μοναδιαίου κόστους** δηλ. το περιεχόμενο κάθε μεταβλητής χωρά σε μια θέση μνήμης και κάθε πράξη χρειάζεται σταθερό χρόνο.

21.5 Ποιες δομές ονομάζονται στατικές και ποιες δυναμικές; Ποιες ονομάζονται εφήμερες και ποιες διαχρονικές (πλήρως και μερικώς διαχρονικές); Ποιες δομές ονομάζονται εκτενείς και ποιες συνοπτικές;

Απάντηση

Οι στατικές δομές δεδομένων επιτρέπουν μόνο την πράξη Access(x) σε κάποιο Λεξικό, ενώ οι Δυναμικές δομές δεδομένων επιτρέπουν επιπλέον και τις πράξεις Insert(x) και Delete(x) με τις οποίες τροποποιούν το λεξικό,

■ Έστω σύμπαν U πηγή ενδιαφέροντος

■ Σύνολο $S \subseteq U, x \in S$ και $\inf(x)$

■ Λεξικό (Dictionary) υποστηρίζει πράξεις:

■ Access(x): if $x \in S$ true{return $\inf(x)$ } else false

Στατικές Δομές

■ Insert(x): $S \rightarrow S \cup \{x\}$

Δυναμικές Δομές

■ Delete(x): $S \rightarrow S - \{x\}$

- 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

21.6 Τι ονομάζεται Ουρά Προτεραιότητας, ποιες πράξεις υποστηρίζει και ποιες οι εφαρμογές της; (Ιούνιος 2014)
Απάντηση

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**

Μιλώντας πιο τυπικά η **Ουρά Προτεραιότητας** είναι:

Είναι ένας Αφηρημένος Τύπος Δεδομένων που:

- ◉ Διατηρεί ένα **σύνολο στοιχείων S**, όπου κάθε στοιχείο έχει μια **συγχετισμένη 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(Q₁, Q₂)**: Ένωση των ουρών Q₁ και Q₂
 - **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

21.7 Ποια η διαφορά των διαχρονικών από τις εφήμερες δομές;

Απάντηση

■ **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.

21.8 Ποια η διαφορά της μερικής από την πλήρη διαχρονικότητα;

Απάντηση

■ **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.

21.9 Ποια **παράμετρο** εισάγουμε για να υποστηρίξουμε τη διαχρονικότητα;

Απάντηση

■ **Για να υποστηρίξουμε τη διαχρονικότητα της δομής εισάγουμε την ακόλουθη παράμετρο :**

■ **Αριθμός εκδοχής:**

- Η αρχική δομή αυτοτελεί την εκδοχή 0.
- Η i-ο στή πράξη τροποποίησης δημιουργεί την εκδοχή i.

■ **Παράμετροι απόδοσης:**

- n = αριθμός στοιχείων στην τρέχουσα εκδοχή
- m = συνολικός αριθμός τροποποήσεων

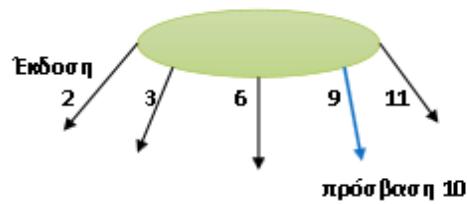
21.10 Με ποιες μεθόδους υλοποιείται η **μερική διαχρονικότητα**;

Απάντηση

A Μέθοδος

■ Μέθοδος παχιού κόμβου (Fat Node)

- Κάθε κόμβος αποθηκεύει αυθαίρετο αριθμό από τιμές (ετικέτες έκδοσης)
- Έχει ετικέτα που υποδεικνύει την έκδοση που δημιουργήθηκε ο διος
- 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 \leq .
 - Ο($\log n$) χρόνος για την εκδόσεις αν οι τιμές εκδόσεων είναι οργανωμένες σε δυαδικό δέντρο)
 - Χωρική Επιβάρυνση: O(1) ανά έκδοση



B Μέθοδος

■ 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
 - Άλλιως 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)

19 Μαΐου 2015

21.11 Με ποιες μεθόδους υλοποιείται η πλήρη διαχρονικότητα;

Απάντηση

■ Χρήση μεθόδου fat node

- Χρονικό κόστος:
 - $O(\log m)$ επιβάρυνση για πρόσθιαση
 - 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: $O(1)$

21.12 Ποιες οι εφαρμογές της ουράς προτεραιότητας; (Σεπτέμβριος 2016)

Απάντηση

◎ 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

21.13 Τι είναι η Διωνυμική Ουρά Προτεραιότητας και τι το Διωνυμικό Δέντρο;

Απάντηση

Ορισμός Διωνυμική Ουράς Προτεραιότητας

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

Ορισμός Διωνυμικού Δέντρου

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

21.14 Ποιες πράξεις υποστηρίζονται από τις ισότητας και τι κάνει η καθεμία; Ποιες είναι οι πολυπλοκότητες των πράξεων αυτών στις Διωνυμικές Σωρούς μεγέθους n ; (Σεπτέμβριος 2016 και Φεβρουάριος 2019)

Απάντηση

Οι πράξεις που υποστηρίζονται από τις Ουρές Προτεραιότητας είναι οι ακόλουθες:

- **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(Q₁, Q₂)**: 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 .

Οι πράξεις ενός Διωνυμικού Σωρού είναι οι ακόλουθες:

1. **MakeQueue()**: Δημιουργεί έναν κενό σωρό σε $\Theta(1)$ 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(Q₁, Q₂)**: 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

Οι πράξεις FindMin(Q) και Meld(Q₁, Q₂) πραγματοποιούνται σε λογαριθμικό χρόνο

21.15 Τι ονομάζεται Διωνυμικός Σωρός; Ποιες οι πράξεις ενός Διωνυμικού Σωρού (Σεπτέμβριος 2016)

Απάντηση

- 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 παδιά 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 των 2 που αντιστοιχεί σε 1 φτιάχνουμε τα αντίστοιχα Διωνυμικά Δέντρα
 - Τοποθετούμε τα Διωνυμικά Δέντρα από αριστερά προς δεξιά κατά αύξουσα τάξη

Οι πράξεις ενός Διωνυμικού Σωρού είναι οι ακόλουθες:

1. **MakeQueue()**: Δημιουργεί έναν κενό σωρό σε $\Theta(1)$ 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(Q₁, Q₂)**: Για να ενώσουμε 2 συρές (2 διωνυμικούς σωρούς) εκτελούμε 2 στάδια
 - Συγχωνεύουμε τις λίστες ριζών, προσέχοντας κάθε λίστα των συρών να έχουν μοναδικό βαθμό
 - Μετά τη συγχώνευση οι βαθμοί στη λίστα που προκύπτει εμφανίζονται σε ανέξουσα σειρά

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

21.16 Ποιες οι Βασικές πράξεις σε δέντρα Αναζήτησης

Απάντηση

Η δομή που θα υλοποιεί το πρόβλημα του λεξικού είναι ένα φυλλοπροσανατολισμένο δέντρο εύρεσης στο οποίο οι τιμές του συνόλου S θα αποθηκεύονται στα φύλλα του. Πιο συγκεκριμένα αυτές θα υλοποιούνται ως εξής:

a)

search(x)

```
{
    v ← ρίζα
    while (v ≠ φύλλο)
    {
        εντόπισε το υποδέντρο i για το οποίο ισχύει ότι  $H_{i-1}(v) \leq x < H_i(v)$ 
        v ← i-οστό παιδί του v
    }
    if πληροφορία(v)=x then βρέθηκε
    else δεν υπάρχει
}
```

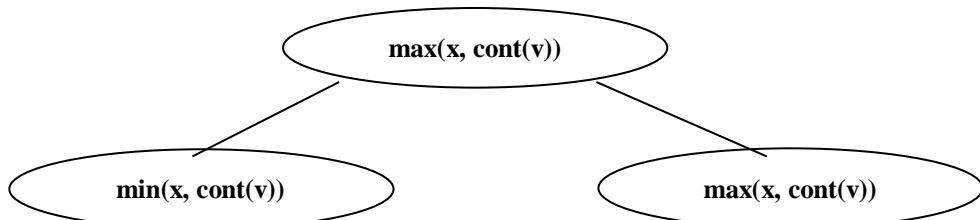
β)

insert(x)

```
{
    v ← το φύλλο στο οποίο τερμάτισε η search(x)
```

if πληροφορία(v)=x then μην κάνεις τίποτα

else αντικατέστησε το v με το υποδέντρο



γ)

delete(x)

{

v ← το φύλλο στο οποίο τερμάτισε η search(x)

if πληροφορία(v) ≠ x then μην κάνεις τίποτα

else

{

u ← ο κοντινότερος πρόγονος του v με βαθμό ≥ 2

σβήσε την i-οστή πλευρά του u και ένα από τα υποδέντρα Hi-1(u), Hi(u)

if degree(u)=1 then

αντικατέστησε το δείκτη από τον πατέρα του u προς το u με το δείκτη στο γιό του u

}

}

Οι 3 προηγούμενες λειτουργίες υλοποιούνται όπως αναφέραμε σε φυλλοπροσανατολισμένο δέντρο εύρεσης και απαιτούν χρόνο $O(\log|S|)$ όπου $|S|$ το πλήθος των φύλλων του δέντρου.

21.17 Τι γνωρίζετε για το **πρόβλημα Union Find**;

Απάντηση

Το πρόβλημα του Union – Find συνίσταται στον χειρισμό διαμερίσεων πάνω στα στοιχεία ενός σύμπαντος. Συγκεκριμένα έστω το σύμπαν $U = \{0, 1, \dots, N - 1\}$, τότε ορίζουμε έναν ΑΤΔ που υποστηρίζει τις εξής πράξεις:

Find (x): Επιστρέφει το όνομα του συνόλου στο οποίο ανήκει το x1.

Union (A, B, C): Ενώνει τα σύνολα με ονόματα A και B κάτω από το κοινό όνομα C.

21.18 Τι γνωρίζετε για τη μέθοδο **weighted union rule**;

Απάντηση

Η μέθοδος weighted union rule αποφεύγει τα μεγάλα βάθη των δέντρων με το να γίνεται πάντοτε κατά την πράξη Union(A,B,C) και την ένωση των δέντρων των δύο ομάδων, η ρίζα του μικρότερου δέντρου γιός του μεγαλύτερου, δηλαδή C=A ή C=B ανάλογα με το ποια ομάδα είναι η μεγαλύτερη.

21.19 Πόσο κοστίζει μια ακολουθία από n-1 unions και m Find όταν χρησιμοποιείται μόνο η τεχνική weighted union rule;

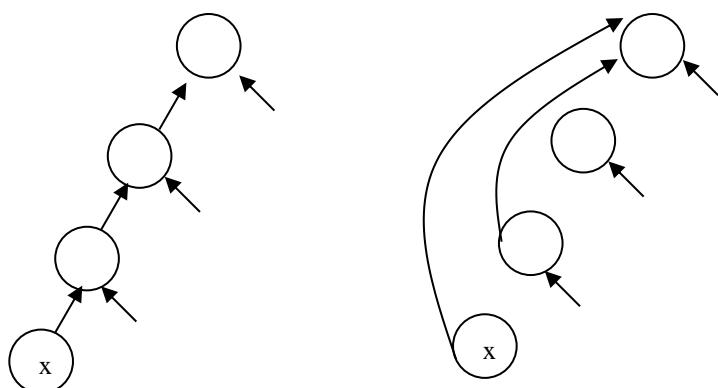
Απάντηση

Μια ακολουθία από n-1 unions και m Finds στοιχίζει $O(n+m \cdot \log n)$ αν χρησιμοποιείται μόνο η τεχνική weighted union rule

21.20 Τι γνωρίζετε για τη μέθοδο **path compression**;

Απάντηση

Η μέθοδος path compression κατά την εκτέλεση της πράξης Find(x), καθώς κινούμαστε στο μονοπάτι από τον κόμβο του x προς τα ρίζα, τροποποιεί όλος τους κόμβους ώστε να έχουν ως πατέρα τη ρίζα.



21.21 Πόσο κοστίζει μια ακολουθία από $n-1$ union και m Find όταν χρησιμοποιούνται και οι δύο τεχνικές;

Απάντηση

Μια ακολουθία από $n-1$ unions και m Find με $m \geq n$ στοιχίζει $O(m \cdot a(m,n))$ αν χρησιμοποιούνται και οι δύο τεχνικές δηλ. και η weighted union rule και η path compression. Η $a(m,n)$ είναι παραλλαγή της συνάρτησης του Ackermann για την οποία ισχύει ότι:

$$a(m, n) = \min\{z \geq 1 : A(z, 4 \left\lceil \frac{m}{n} \right\rceil) > \log n\}$$

21.22 Ποιες περιπτώσεις ως προς την πολυπλοκότητα μελετάμε; (Θέμα 3 Ιούνιος 2015)

Απάντηση

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]

- **Ανάλυση μέσης περίπτωσης (average case analysis)** → μία πιθανοτική κατανομή τίθεται στις πράξεις ενός αφηρημένου τύπου δεδομένων και το μέσο κόστος των πράξεων υπολογίζεται
- **Ανάλυση Χειρότερης Περίπτωσης (worst case analysis)** → υπολογίζονται πολυπλοκότητες χειρότερης περίπτωσης για κάθε πράξη
- **Ανάλυση Επιμερισμένης Πολυπλοκότητας** → υπολογίζεται το συνολικό κόστος μίας ακολουθίας πράξεων και επιμερίζεται το κόστος σε καθεμία πράξη

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.

21.23 Δώστε τον ορισμό της επιμερισμένης πολυπλοκότητας με μια μικρή περιγραφή. Γιατί είναι διαφορετική από την πολυπλοκότητα μέσης περίπτωσης;

Απάντηση

Δομές Δεδομένων –Computer Ανάλυση

Η επιμερισμένη ανάλυση αποτελεί ένα ισχυρό εργαλείο για την ανάλυση δομών δεδομένων και αλγορίθμων. Η **επιμερισμένη ανάλυση** (amortized analysis) χρησιμοποιείται για να δείξει ότι το μέσο κόστος μίας πράξης είναι μικρό, όταν χωρίς υπολογίζεται σαν ο μέσος όρος μίας ακολουθίας πράξεων, παρότι μπορεί να υπάρχουν επιμέρους πράξεις με μεγάλο κόστος. Οι υλοποιήσεις με καλές επιμερισμένες πολυπλοκότητες συχνά είναι απλούστερες και αποδοτικότερες σε σχέση με τις υλοποιήσεις πολυπλοκοτήτων χειρότερης περίπτωσης. Προσχή, η επιμερισμένη ανάλυση αποτελεί ένα εργαλείο προσδιορισμού της απόδοσης μιας δομής ή ενός αλγόριθμου και όχι σχεδιαστικό εργαλείο. Βεβαίως, ενίστε η ανάδραση από αυτή την ανάλυση μπορεί να δώσει κατευθύνσεις για καλύτερο σχεδιασμό.

Η ανάλυση μέσης περίπτωσης και η πιθανοτική ανάλυση δεν έχουν σχέση με την επιμερισμένη ανάλυση. Στην ανάλυση μέσης περίπτωσης, υπολογίζουμε το μέσο όρο σε όλες τις δυνατές εισόδους ενώ στην πιθανοτική υπολογίζουμε το μέσο όρο σε όλες τις τυχαίες επιλογές που κάνουμε. Στην επιμερισμένη ανάλυση υπολογίζουμε το μέσο όρο μίας ακολουθίας πράξεων. Η επιμερισμένη ανάλυση υποθέτει ακολουθία πράξεων χειρότερης περίπτωσης και δεν χρησιμοποιεί τυχαιότητα.

Η τεχνική αυτή ανάλυσης απαιτεί τον καθορισμό των διαφορετικών ακολουθιών πράξεων που μπορεί να συμβούν. Αυτή είναι η συνηθισμένη περίπτωση σε δομές δεδομένων, όπου η κατάσταση της δομής παραμένει αμετάβλητη μεταξύ δύο διαδοχικών πράξεων σε μία τέτοια ακολουθία. Η βασική ιδέα είναι ότι μία ακριβή πράξη θα αλλάξει την κατάσταση, έτοιμη ώστε οι επόμενες πράξεις θα είναι φθηνότερες και, επομένως, μπορούμε να επιμερίσουμε το κόστος.

21.24 Ποιές πράξεις υποστηρίζονται σε σωρούς και ποιοι οι χρόνοι τους; (Ιούνιος 2015)

Απάντηση

Οι πράξεις που υποστηρίζει ένας σωρός είναι

- is_empty () → επιστρέφει ΝΑΙ αν ο σωρός δεν περιέχει κανένα στοιχείο, αλλιώς επιστρέφει ΟΧΙ. Χρόνος O(1)
- push(data) → ένθεση στοιχείου data στην κορυφή του σωρού. Χρόνος O(1)
- pop () → διαγραφή του στοιχείου στην κορυφή του σωρού. Χρόνος O(1)
- top () → επιστροφή του στοιχείου στην κορυφή του σωρού χωρίς να το διαγράφει. Χρόνος O(1)

21.25 Ποιες πράξεις υποστηρίζονται σε ουρές και ποιοι οι χρόνοι τους; (Ιούνιος 2014)

Απάντηση

Οι πράξεις που υποστηρίζει μια ουρά είναι

- is_empty () → επιστρέφει ΝΑΙ αν η ουρά δεν περιέχει κανένα στοιχείο, αλλιώς επιστρέφει ΟΧΙ. Χρόνος O(1)
- push(data) → ένθεση στοιχείου data στο τέλος της ουράς. Χρόνος O(1)
- pop () → διαγραφή του στοιχείου στην αρχή της ουράς. Χρόνος O(1)
- first () → επιστροφή του στοιχείου στην αρχή της ουράς χωρίς να το αφαιρεί. Χρόνος O(1)

21.26 Ποια τα χαρακτηριστικά του IST δέντρου;

Απάντηση

To IST δέντρο έχει τις παρακάτω ιδιότητες:

- i) Για την αποθήκευση του S , $|S| = n$, απαιτεί $O(n)$ χώρο.
- ii) Η επιμερισμένη πολυπλοκότητα ενθύσεων/αποσβέσεων είναι $O(\log n)$ και η μέση επιμερισμένη πολυπλοκότητα είναι $O(\log \log n)$.
- iv) Ο χειρότερος χρόνος αναζήτησης είναι $O((\log n)^2)$.
- v) Η δομή υποστηρίζει σεκουακή προσπέλαση των στοιχείων σε $O(n)$ χρόνο και τις πράξεις Predecessor(x) (εύρεση επόμενου στοιχείου από το x), Successor(x) (εύρεση προηγούμενου στοιχείου από το x) και Min (εύρεση ελαχίστου στοιχείου της δομής) σε $O(1)$.

21.27 Να δώσετε σε ψευδοκώδικα όλες τις πράξεις στο Hashing με αλυσίδες;

Απάντηση

Οποιοδήποτε λεξικό έχει 3 βασικές μεθόδους:

Αρχικοποίηση

Δημιουργία πίνακα table με M λίστες όσο και το μέγεθος του hash πίνακα

Insert(K)

```
index=h(K)
εισαγωγή κλειδιού K στον table[index]
```

Find(K)

```
index=h(K)
διάσχιση λίστας που βρίσκεται στη θέση table[index] και αναζήτηση ταιριάσματος. Αν βρεθεί ταίριασμα τότε εκτυπώνεται το στοιχείο που εντοπίστηκε, αλλιώς επιστρέφεται error. Το κόστος για μια αναζήτηση σε ένα hash πίνακα είναι  $O(T_{hash} + T_{eq})$  όπου  $T_{hash}$  είναι ο χρόνος δεικτοδότησης ενός στοιχείου και  $T_{eq}$  είναι ο χρόνος σύγκρισης δύο στοιχείων στον hash πίνακα
```

Remove(K)

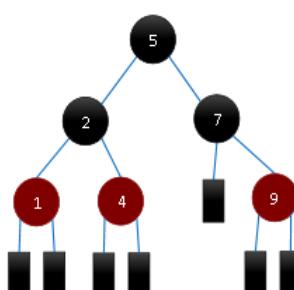
```
index=h(K)
διάσχιση λίστας που βρίσκεται στη θέση table[index] και αναζήτηση ταιριάσματος. Αν βρεθεί ταίριασμα τότε γίνεται διαγραφή, αλλιώς επιστρέφεται error
```

21.28 Να δώστε τον ορισμό του Red-Black Tree;

Απάντηση

Ένα πλήρες δυαδικό δέντρο είναι red-black δέντρο όταν κάθε κόμβος είναι είτε κόκκινος είτε μαύρος και τακανοποιούνται 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



21.29 Θέματα Θεωρίας Ατυπης 2017

Σωστό / Λάθος

- 1) Η δυαδική αναζήτηση απαιτεί γραμμικό χρόνο στην χειρότερη περίπτωση.
- 2) Το insertion sort έχει χειρότερο χρόνο εκτέλεσης $O(n \log n)$.
- 3) Μία στοίβα είναι FIFO δομή και μία ουρά είναι LIFO δομή.
- 4) Οι αλγόριθμοι Quicksort και Mergesort χρησιμοποιούν την στρατηγική "Διαιρεί και Βασίλει".
- 5) Ένα δυαδικό δέντρο ύψους h έχει $O(2^h)$ κόμβους
- 6) Η αποτελεσματικότητα του Quicksort βελτιώνεται πάντα όταν επιλεγεί το μεγαλύτερο στοιχείο ως pivot αντί του μικρότερου.
- 7) Ο χειρότερος χρόνος εκτέλεσης αναζήτησης σε ένα hash πίνακα είναι $O(n)$.
- 8) Η πολυπλοκότητα εύρεσης του median στοιχείου σε μια λίστα είναι $O(\log n)$
- 9) Ο Quicksort έχει αναμενόμενο χρόνο εκτέλεσης $O(n \log n)$ ενώ ο Insertionsort έχει $O(n^2)$ όπου η πολυπλοκότητα του είναι κάτιον της τελευταίας στοιχείου του πίνακα.
- 10) Ο αλγόριθμος Radixsort έχει κόστος εκτέλεσης $O(a(n+c))$ όπου a είναι οι επαναληψεις, n ο αριθμός στοιχείων του πίνακα και c η βάση.
- 11) Για μεγάλη είσοδο, ο Mergesort θα τρέχει πάντα γρηγορότερα από τον Insertion sort (μεταξύ των δύο).
- 12) Οι κόμβοι μιας διασυνδεδεμένης λίστας συνήθως αποθηκεύονται σε συνεχόμενες θέσεις μνήμης.

Απάντηση

- 1) **Λάθος** → Η πολυπλοκότητα χειρότερης (και μέσης περίπτωσης) για τη Δυαδική Αναζήτηση είναι $O(\log n)$.
- 2) **Λάθος** → Η πολυπλοκότητα χειρότερης (και μέσης περίπτωσης) για το Insertionsort είναι $O(n^2)$
- 3) **Λάθος** → Το αντίθετο. Η στοίβα είναι LIFO και η ουρά FIFO
- 4) **Σωστό**
- 5) **Σωστό** → Ένα δυαδικό δέντρο έχει 2^h κόμβους όπου h το ύψος του
- 6) **Λάθος** → Η αποτελεσματικότητα του Quicksort δεν εξαρτάται από το pivot στοιχείο. Μπορούμε να επιλέξουμε ως pivot είτε το πρώτο στοιχείο του πίνακα είτε το μεσαίο στοιχείο του πίνακα
- 7) **Σωστό** → Το κόστος χειρότερης περίπτωσης σε ένα hash πίνακα είναι $O(n)$ διότι όταν κάνουμε αναζήτηση στοιχείου σε hash πίνακα στη χειρότερη περίπτωση θα εξετάσουμε όλο τον πίνακα για να εντοπίζουμε το στοιχείο γιατί τα στοιχεία τοποθετούνται τυχαία βάσει μιας hash function. Η πολυπλοκότητα μέσης και κατανεμημένης περίπτωσης (amortized) είναι $O(1)$
- 8) **Λάθος** → Για να βρούμε το median στοιχείο σε ένα μην ταξινομημένο πίνακα μπορούμε να χρησιμοποιήσουμε ένα σωρό ελαχίστων (min-heap) όπου το μικρότερο στοιχείο του σωρού είναι στην κορυφή του σωρού και απαιτεί χρόνο $O(n \log n)$. Ο χρόνος μπορεί να μειωθεί σε $O(n)$ αν εφαρμόσουμε αλγόριθμο selection sort για να εντοπίσουμε το k-οστό μικρότερο στοιχείο του πίνακα.
- 9) **Λάθος**. → Ο Quicksort είναι καλύτερος από τον Insertionsort μόνο στη μέση περίπτωση. Και οι δύο αλγόριθμοι έχουν την ίδια πολυπλοκότητα χειρότερης περίπτωσης με $O(n^2)$
- 10) **Σωστό** → Γενικά η πολυπλοκότητα του Radix Sort είναι $\Theta(n \log n)$
- 11) **Λάθος** → Ο Mergesort τρέχει γρηγορότερα από τον Insertionsort ανεξάρτητα από το μέγεθος εισόδου
- 12) **Λάθος** → Οι κόμβοι μιας διασυνδεδεμένης λίστας αποθηκεύονται σε τυχαίες θέσεις μνήμης όχι σε συνεχόμενες

22 ΘΕΜΑΤΑ ΜΕ ΣΤΟΙΒΑ ΚΑΙ ΟΥΠΑ ΜΕ ΚΩΔΙΚΑ

Να περιγραφούν οι δομές της στοίβας, ουράς και λίστας και να γραφούν οι χαρακτηριστικές συναρτήσεις γιαντές

Απάντηση

Στοίβα ονομάζεται η δομή δεδομένων στην οποία τα δεδομένα που βρίσκονται στην κορυφή της στοίβας λαμβάνονται πρώτα, ενώ αυτά που βρίσκονται στο βάθος της στοίβας λαμβάνονται τελευταία. Αυτή η μέθοδος επεξεργασίας ονομάζεται LIFO(Last-In- First-Out)

Οι κύριες λειτουργίες σε μία στοίβα είναι:

- η ώθηση (push) στοιχείου στην κορυφή της στοίβας, και
- η απώθηση (pop) στοιχείου από τη στοίβα. (σ60)

Μια στοίβα μπορεί να υλοποιηθεί πολύ εύκολα με τη βοήθεια ενός μονοδιάστατου πίνακα. Μια βοηθητική μεταβλητή (με όνομα συνήθως top) χρησιμοποιείται για να δείχνει το στοιχείο που τοποθετήθηκε τελευταίο στην κορυφή της στοίβας. Η διαδικασία της ώθησης πρέπει οπωσδήποτε να ελέγχει, αν η στοίβα είναι γεμάτη, οπότε λέγεται ότι συμβαίνει υπερχείλιση (overflow) της στοίβας ενώ η διαδικασία απώθησης ελέγχει, αν υπάρχει ένα τουλάχιστον στοιχείο στη στοίβα, δηλαδή ελέγχει αν γίνεται υποχείλιση (underflow) της στοίβας.

22.1 Ολοκληρωμένο πρόγραμμα διαχείρισης στοίβας

*int *stack;*

int top=-1;

int i, n;

void push(*int* x)

{

if (top==n-1)

printf("Stack full!\n");

else

{

top++;

stack[top]=x;

}

}

void pop()

{

if (top>=0)

top--;

else

printf("Stack is empty!\n");

}

void print()

{

int i;

```

for (i=top; i>=0; i--)
    printf("%d\n", stack[i]);

printf("\n");
}

void main()
{
    printf("Give stack size:\n");
    scanf("%d", &n);

    stack=(int*)malloc(n*sizeof(int));

    printf("Filling Stack\n");
    srand(time(NULL));

    for(i=0; i<n; i++)
    {
        push(rand()/RAND_MAX);
        print();
    }

    printf("Emptying Stack:\n");
    for (i=0; i<n; i++)
    {
        pop();
        print();
    }
}

```

Ουρά ονομάζεται η δομή δεδομένων στην οποία τα δεδομένα που τοποθετήθηκαν πρώτα στην ουρά θα επεξεργαστούν πρώτα ενώ τα νέα δεδομένα που μπαίνουν στην ουρά τοποθετούνται στο τέλος της. Η μέθοδος αυτή επεξεργασίας ονομάζεται FIFO : First-In-First-Out

Οι κύριες λειτουργίες που εκτελούνται σε μία ουρά είναι :

- η εισαγωγή (enqueue) στοιχείου στο πίσω άκρο της ουράς, και
- η εξαγωγή (dequeue) στοιχείου από το εμπρός άκρο της ουράς. (σ61)

Στην περίπτωση της ουράς απαιτούνται δύο δείκτες: ο εμπρός (front) και ο πίσω (rear) δείκτης, που μας δίνουν τη θέση του στοιχείου που σε πρώτη ευκαιρία θα εξαχθεί και τη θέση του στοιχείου που μόλις εισήλθε.

Πρέπει να ελέγχεται αν υπάρχει ελεύθερος χώρος στον πίνακα για την εισαγωγή και αν υπάρχει ένα τουλάχιστον στοιχείο για την εξαγωγή. (σ61)

22.2 Ολοκληρωμένο πρόγραμμα διαχείρισης ουράς

```

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int *queue; //Δυναμική δήλωση ουράς
int front=-1,rear=-1; //Ο δείκτης front δείχνει πάντα στην αρχή της ουράς ενώ ο δείκτης rear δείχνει πάντα στο τέλος της ουράς.
int i,n;

void enqueue(int x)
{
    if(rear==n-1)
        printf("Queue full!\n");
    else
    {
        rear++;
        queue[rear]=x;
        if(front==-1)
            front++;
    }
}

void dequeue()
{
    if(front>rear)
    {
        printf("Queue empty!\n");
        front=rear=-1;
    }
    else
        front++;
}

void print()
{
    printf("\nQueue:");
    for(i=front;i<=rear && i>=0;i++)
        printf("%d\t",queue[i]);
    printf("\n");
}

void main()
{
    printf("Dose to megethos tis ouras:\n");
    scanf("%d",&n);
}

```

```

queue=(int*)malloc(n*sizeof(int));

strands(time(NULL)); //Αρχικοποίηση της γεννήτριας τυχαίων αριθμών με τον τρέχοντα χρόνο.

printf("Γέμισμα Ουράς:\n");
for(i=1;i<=n;i++)
{
    enqueue(rand()/RAND_MAX);
    print();
}

printf("Άδειασμα ουράς:\n");
for(int j=1;j<=n;j++)
{
    dequeue();
    print();
}

dequeue();
print();
}

```

22.3 Ολοκληρωμένο πρόγραμμα διαχείρισης απλά συνδεδεμένης λίστας (simple-linked list)

```

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

struct node
{
    int value;
    node *next;
};

```

node *listbase;//Η μεταβλητή listbase χρησιμοποιείται για να δείχνει την αρχή της λίστας.

```

int list_count()
{
    node *v=listbase;
    int count=0;

    if(v==NULL)
        return 0;

```

```

while(v!=NULL)
{
    count++;
    v=v->next;
}

return count;
}

```

void print_list()//Συνάρτηση εκτύπωσης της λίστας.

```
{
node *v=listbase;
```

```

if(v==NULL)
{
    printf("Empty list!\n");
    return;
}

```

```

printf("List:\n");
while(v!=NULL)
{
    printf(" %d\t",v->value);
    v=v->next;
}

```

void push_infront()//Συνάρτηση προσθήκης νέου κόμβου στην αρχή της λίστας.

```
{
node *w;
w=(node*)malloc(sizeof(node));

printf("Dose timi sto neo komvo:\n");
scanf(" %d", &w->value);
w->next= listbase;
listbase=w;
print_list();
}
```

void push_atend()//Συνάρτηση προσθήκης νέου κόμβου στο τέλος της λίστας.

```
{
node *w, *v=listbase;
```

```
w=(node*)malloc(sizeof(node));
```

```

printf("Dose timi sto neo komvo:\n");
scanf("%d", &w->value);

if(v==NULL)
{
    listbase=w;
    w->next=NULL;
    return;
}

while(v->next!=NULL)
{
    v=v->next;

    v->next=w;
    w->next=NULL;
    print_list();
}
}

void push_inside()//Συνάρτηση προσθήκης νέου κόμβου σε τυχαία θέση εντός της λίστας συμπεριλαμβανομένων της αρχικής και της τελικής θέσης.
{
    node *w,*v=listbase;
    int total, i=1,pos;

    w=(node*)malloc(sizeof(node));

    printf("Dose timi sto neo komvo:\n");
    scanf("%d", &w->value);

    total=list_count();

    if(total==0)
    {
        printf("Prepei na iparxei toulaxiston enas komvos\n");
        return;
    }

    do
    {
        printf("Dose tin thesi pou tha mpei o neos komvos apo 1 mexri %d:\n",total);
        scanf("%d", &pos);
    }
    while (pos<1 || pos>total);
}

```

```

if(pos==1)
{
    w->next=listbase;
    listbase=w;
    print_list();
}

else
{
    while(v->next!=NULL && i<pos-1)
    {
        v=v->next;
        i++;
    }

    w->next=v->next;
    v->next=w;
    print_list();
}
}

```

void delete_node()//Συνάρτηση διαγραφής κόμβου από τη λίστα.

```

{
    node *w,*v=listbase;
    int found=0,val;

    if(v==NULL)
    {
        printf("Empty list!\n");
        return;
    }

    printf("Dose tin timi tou komvou pou theleis na diagrapseis:\n");
    scanf("%d", &val);

    if(v->value==val)
    {
        listbase=v->next;
        printf("O komvos me timi %d diagrafike.\n", val);
        free(v);
        print_list();
        return;
    }

    w=v->next;
}

```

```

while(w!=NULL && found==0)
{
    if(w->value!=val)
    {
        v=v->next;
        w=w->next;
    }
    else
    {
        v->next=w->next;
        found=1;
        printf("O komvos me timi %d diagrafike.\n",val);
        free(w);
        print_list();
    }
}

if(found==0)
    printf("O komvos autos den iparxei!\n");
}

void search_node()//Συνάρτηση αναζήτησης και τροποποίησης κόμβου στη λίστα.
{
    node *v=listbase;
    int found=0,val,i=1;
    char ans[5];

    if (v==NULL)
    {
        printf("Empty list!\n");
        return;
    }

    printf("Dose tin timi tou komvou pou theleis na vreis:\n");
    scanf("%"d", &val);

    if(v->value==val)
    {
        printf("O komvos me timi %d vrisketai stin proti thesi.\n",val);
        found=1;
        printf("Theleis na alaxeis tin timi tou komvou? y/n\n");
        scanf("%"s", &ans);
        if (strcmp(ans, "y")==0)
        {
            printf("Dose tin nea timi tou komvou:\n");
        }
    }
}

```

```

scanf("%d", &val);
    v->value=val;
    print_list();
}

return;
}

while(v!=NULL && found==0)
{
    if(v->value!=val)
    {
        i++;
        v=v->next;
    }
    else
    {
        found=1;
        printf("O komvos me timi %d vrethike stin thesi: %d\n",val,i);
        printf("Theleis na alaxeis tin timi tou komvou? y/n\n");
        scanf("%s", &ans);
        if (strcmp(ans, "y")==0)
        {
            printf("Dose tin nea timi tou komvou:\n");
            scanf("%d",&val);
            v->value=val;
            print_list();
        }
        return;
    }
}

if(found==0)
    printf("O komvos autos den vrethike!\n");
}

```

void sort_node()//Συνάρτηση ταξινόμησης κόμβων στη λίστα.

```

{
    node *w,*v;
    int temp, i, k,total=list_count();

    for(k=1; k<total; k++)
    {
        v=listbase;
        w=v->next;
        for(i=1;i<=total-k; i++)

```

```

}

if(v->value>w->value)
{
    temp=v->value;
    v->value=w->value;
    w->value=temp;
}
v=v->next;
w=w->next;
}
}

print_list();
}

```

```
void main()
```

```

{
int ch;
char ans[5];

do
{
    printf("1.Eisagogi stoixeiou stin arxi tis listas.\n");
    printf("2.Eisagogi stoixeiou sto telos tis listas.\n");
    printf("3.Eisagogi stoixeiou se tixaia thesi tis listas.\n");
    printf("4.Ektiposi listas.\n");
    printf("5.Diagrafi komvou.\n");
    printf("6.Euresi-Tropopoaksi komvou.\n");
    printf("7.Taxinomisi listas.\n");
    printf("8.Termatismos programatos.\n");

    printf("Choose:\n");
    scanf("%d", &ch);

    switch (ch)
    {
        case 1: push_infront();
                    break;

        case 2: push_atend();
                    break;

        case 3: push_inside();
                    break;
    }
}

```

```
case 4: print_list();
          break;

case 5: delete_node();
          break;

case 6: search_node();
          break;

case 7: sort_node();
          break;

case 8: return;
}

default: printf("Wrong choice");
}

printf("\n Do you want to continue? y/n\n");
scanf("%s", ans);
system("cls");
}

while(strcmp(ans, "y")==0);
}
```

ΘΕΜΑ 4

Η προδιάταξη: 39-1-39-11-5-39-3-5-36-45

Computer Analytics

24 ΘΕΜΑΤΑ ΦΕΒΡΟΥΑΡΙΟΣ 2019

ΘΕΜΑ 4 (2,5 μονάδες)

Να σχεδιαστεί βήμα-βήμα το (a,b) -δένδρο που προκύπτει από την εισαγωγή σε ένα άδειο δένδρο με τη σειρά των εξής στοιχείων {18, 12, 19, 27, 31, 1, 29, 30, 4, 3, 34} αν το $a=2$ και το $b=4$. Στην συνέχεια διαγράψτε βήμα-βήμα τα στοιχεία {12, 19}.

Computer Analysis

ΕΡΩΤΗΜΑ 2 [1.5M]

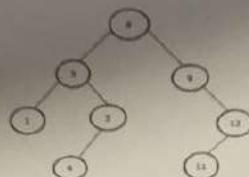
Επιλέξτε την εφαρμογή του αλγόριθμου Μέγεθος Sort (αλγόριθμης με συγχώνευση) στον πίνακα Άνωτρο 20. Η διάταξη είναι: 12, 8, 7, 4, 11, 2, 6, 21.

ΕΡΩΤΗΜΑ 3 [2M]

Περιγράψτε το Binary Interpolation Search. Δώστε της πολυπλοκότητας μέσης και χρόνους πελάτευσης. Παραγράψτε αναλυτικά πώς θα γίνει ο χειρότερος χρόνος ψαζμάτως ισός με $O(\log n)$.

ΕΡΩΤΗΜΑ 4 [1M]

Στο παρακάτω διαδικτύο δένδρο, να εισάγετε τα στοιχεία 7, 13, 15 και στη συνέχεια να διαγράψετε το στοιχείο 3. Να δείξετε σε κάθε στάδιο την κατάσταση του δένδρου, χρησιμοποιώντας σχήματα.

**ΕΡΩΤΗΜΑ 5 [1M]**

Έστω η συνάρτηση κατακέρματος $\text{hash}(\text{key}) = \text{key} \bmod m$. Τοποθετήστε τα στοιχεία 52, 12, 71, 56, 5, 10, 19, 90 σε έναν πίνακα κατακέρματος Α με αλυσίδες οι οποίες υλοποιούνται ως linked lists. Διαπιπτώστε σε ψευδοκώδικα τον αλγόριθμο εύρεσης ενός κλειδιού key.

ΕΡΩΤΗΜΑ 6 [1.5M]

Εφαρμόστε το weighted union rule για τα παρακάτω Unions από 1 έως 7. Οι αριθμοί δηλώνουν τον αριθμό των στοιχείων (weight) κάθε συνόλου που λαμβάνει μέρος στο Union. Τα κεφαλαία γράμματα 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,E,F)
7. U(C,F,G)

ΕΡΩΤΗΜΑ 7 [1.5M]

Να σχεδιαστεί βήμα-βήμα το (a,b)-δέντρο που προκύπτει από την εισαγωγή σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων {10, 1, 5, 6, 20, 19, 16, 13, 12, 25} αν το $a=2$ και το $b=4$. Στην συνέχεια διαγράψτε βήμα-βήμα τα στοιχεία {5, 25}.

ΕΡΩΤΗΜΑ 1 [1.5M]

Ο παρακάτω ψευδοκώδικας αναποτοχεί στον αναδρομικό αλγόριθμο γρήγορης ταξινόμησης (quick sort) για την πίνακα A μεγέθους n=right-left+1. Αρχικά left=0 και right=n-1. Συμπληρώστε τα κομμάτια κώδικα που λείπουν.

```
void QuickSort(int list[], int left, int right)
{
    int pivot, leftArrow, rightArrow;
    leftArrow = left;
    rightArrow = right;
    pivot = list[(left + right) / 2];
    do
    {
        while (list[rightArrow] > pivot)
            .....
        while (....)
            .....
        if (leftArrow <= rightArrow)
        {
            Swap_Data(list[leftArrow], list[rightArrow]);
            .....
        }
        .....
    } while (rightArrow >= leftArrow);
    if (left < rightArrow)
        .....
    if (leftArrow < right)
        .....
}
```

ΕΡΩΤΗΜΑ 5 – ΘΕΩΡΙΑ ΓΙΑ ΕΥΡΕΣΗ ΕΝΟΣ ΚΛΕΙΔΙΟΥ KEY

Find(K)

index=h(K)

διάσχιση λίστας που βρίσκεται στη θέση table[index] και αναζήτηση ταιριάσματος. Αν βρεθεί ταίριασμα τότε εκτυπώνεται το στοιχείο που εντοπίστηκε, αλλιώς επιστρέφεται error

ΕΡΩΤΗΜΑ 1

do

{

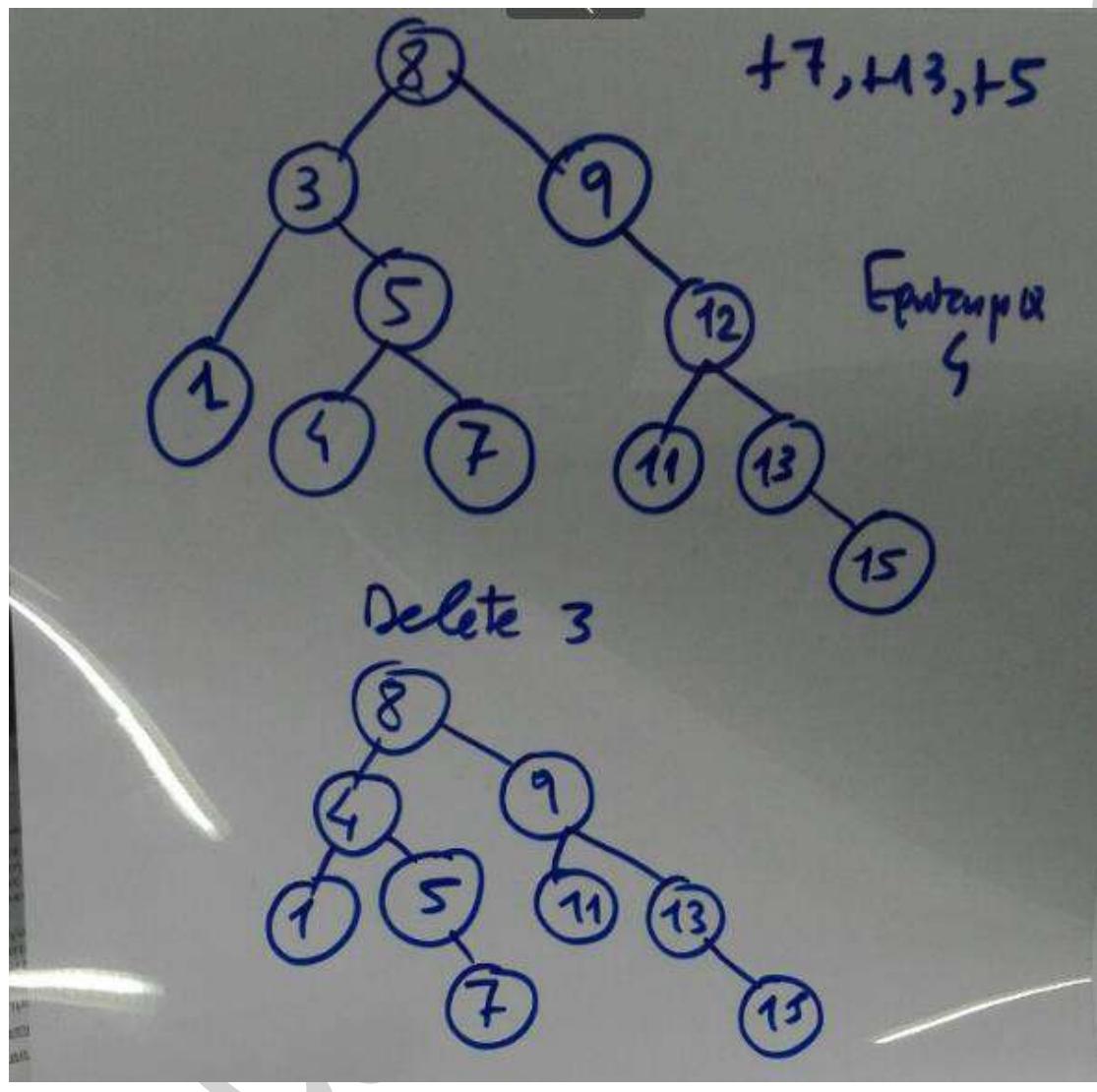
```
while (list[rightArrow]>pivot)
    rightArrow--;
while (list[leftArrow] < pivot
    ++leftArrow;
if (leftArrow<=rightArrow)
{
    Swap_Data(list[leftArrow],list[rightArrow]);
    leftArrow++;
}
```

```

        rightArrow --;
    }
} while (rightArrow >= leftArrow);

if (left < rightArrow)
    quickSort(list, leftArrow, pivot - 1);
if (leftArrow < right)
    quickSort(arr, pivot, rightArrow);
}

```



ΘΕΜΑΤΑ ΣΕΠΤΕΜΒΡΙΟΣ 2019

ΕΡΩΤΗΜΑ 1 [2M]

Ταξινομήστε την ακολουθία:

ΔΟΜΕΣ ΔΕΔΟΜΕΝΩΝ ΜΑΘΗΜΑ 2019/2020

με τη μέθοδο ταξινόμησης External Sorting με Replacement Selection, δείχνοντας όλα τα ενδιάμεσα βήματα του αλγόριθμου. Υποθέστε ότι η κύρια μνήμη έχει μέγεθος $M=3$, και ότι έχετε 6 ταινίες στις διάθεσή σας ($p=3$). (Υπόδειξη: Προτεραιότητα ταξινόμησης A...Ω01...9)

ΕΡΩΤΗΜΑ 2 [1.5M]

Να σχεδιαστεί βήμα-βήμα το δυαδικό δέντρο αναζήτησης που προκύπτει από την εισαγωγή σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων {10, 1, 5, 6, 20, 19, 16, 13, 12}. Για την ίδια ακολουθία αριθμών να κατασκευαστεί το δυαδικό δέντρο ώστε μετά από κάθε εισαγωγή το προκύπτον δέντρο να είναι AVL. Συγκρίνετε το μέσο πλήθος συγκρίσεων για επιτυχή αναζήτηση. Ποιο από τα δύο δέντρα είναι καλύτερο και γιατί.

ΕΡΩΤΗΜΑ 3 [1.5M]

Δίνονται οι ακόλουθες διαπεράσεις ενός δυαδικού δένδρου με στοιχεία χαρακτήρες του Αγγλικού αλφαριθμού:

ΜΕΤΑΔΙΑΤΑΞΗ: IOBCLMFGEDA

ΣΥΜΜΕΤΡΙΚΗ ΔΙΑΤΑΞΗ: IBOACDLFMEG

Ανασχηματίστε τη μορφή του συγκεκριμένου δυαδικού δένδρου.

Ερώτημα 4 [1M]

Δίνονται δύο ταξινομημένοι πίνακες έστω Α και Β, που έχουν n και m στοιχεία αντίστοιχα. Εάν γνωρίζουμε ότι τα στοιχεία σε κάθε πίνακα είναι ταξινομημένα, να δώσετε έναν αλγόριθμο που να δίνει στην έξοδο τα στοιχεία του Α και τα στοιχεία του Β (άρα συνολικά $n+m$ στοιχεία) ταξινομημένα. Πόσες συγκρίσεις θα κάνει ο αλγόριθμος σας στη χειρότερη περίπτωση;

Ερώτημα 5 [1M]

Έστω μη ταξινομημένος πίνακας Α, μεγέθους n . Να δώσετε έναν αλγόριθμο που παράγει ως έξοδο τα k μεγαλύτερα και τα k μικρότερα του median του Α, όπου k μια σταθερά. Ο αλγόριθμος που θα δώσετε θα πρέπει να τρέχει σε χρόνο $O(n)$ στη χειρότερη περίπτωση. Θεωρήστε ότι έχετε στη διάθεσή σας τον γραμμικό Median αλγόριθμο Select.

Καλή Επιτυχία!!!

Τα θέματα επιστρέφονται μαζί με το γραπτό σας.

ΕΡΩΤΗΜΑ 6 [1.5M]

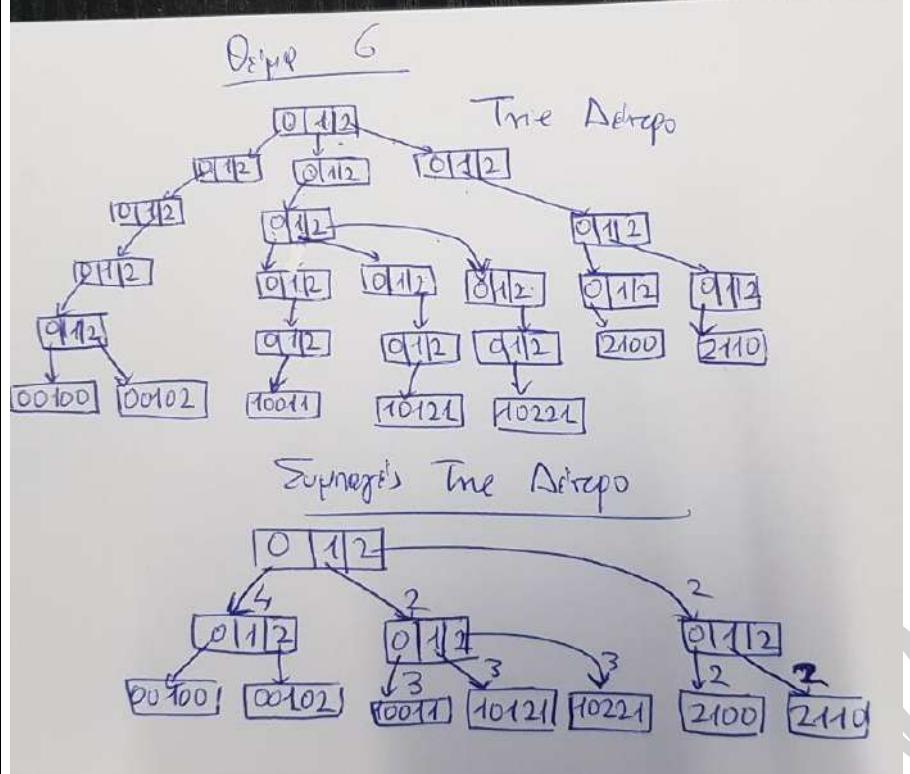
Αποθηκεύστε σε δομή Trie χτισμένη στο τριαδικό αλφάριθμο $\Sigma = \{0, 1, 2\}$ τα στοιχεία {00100, 00102, 10011, 10121, 10221, 2100, 2110}. Προτείνετε την αποθήκευση που καταλαμβάνει το μικρότερο χώρο.

ΕΡΩΤΗΜΑ 7 [1.5M]

Να σχεδιαστεί βήμα-βήμα το (a, b) -δέντρο που προκύπτει από την εισαγωγή σε ένα άδειο δέντρο με τη σειρά των εξής στοιχείων {8, 2, 9, 6, 21, 19, 12, 13, 11, 24} αν το $a=2$ και το $b=4$. Το πρόβλημα βάζεται στη σποιγεία {8, 2, 6}.

Απαντήσεις

Ερώτημα 6

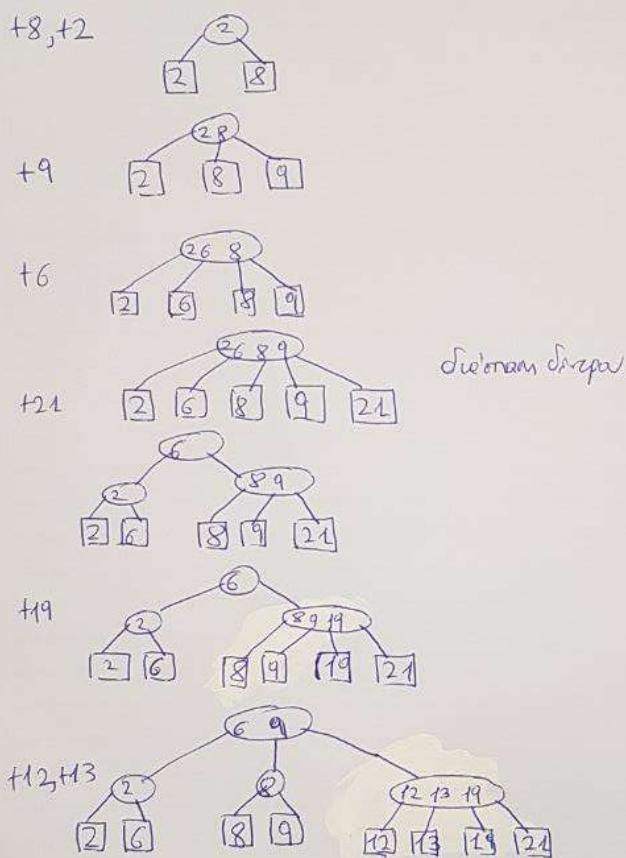


Ερώτημα 7

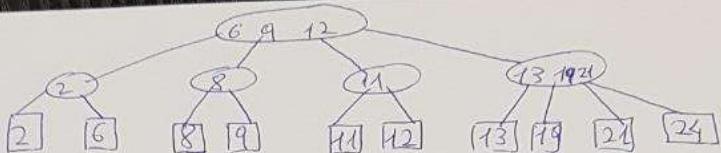
8, 2, 9, 6, 21, 19, 12, 13, 11, 24

$$\begin{array}{l} a=2 \\ b=4 \end{array}$$

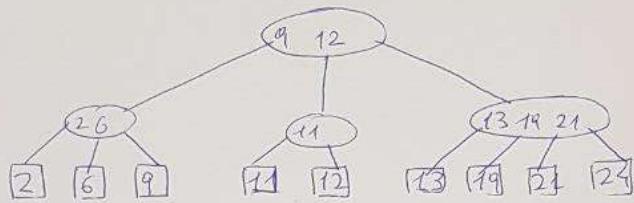
826



+1



-8

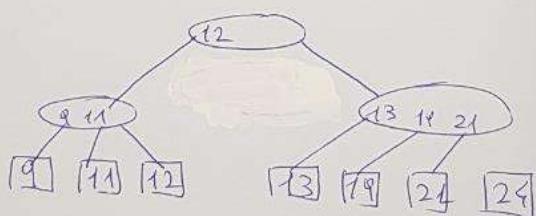


-2

το προγραμμα λέπει ότι το 2 και

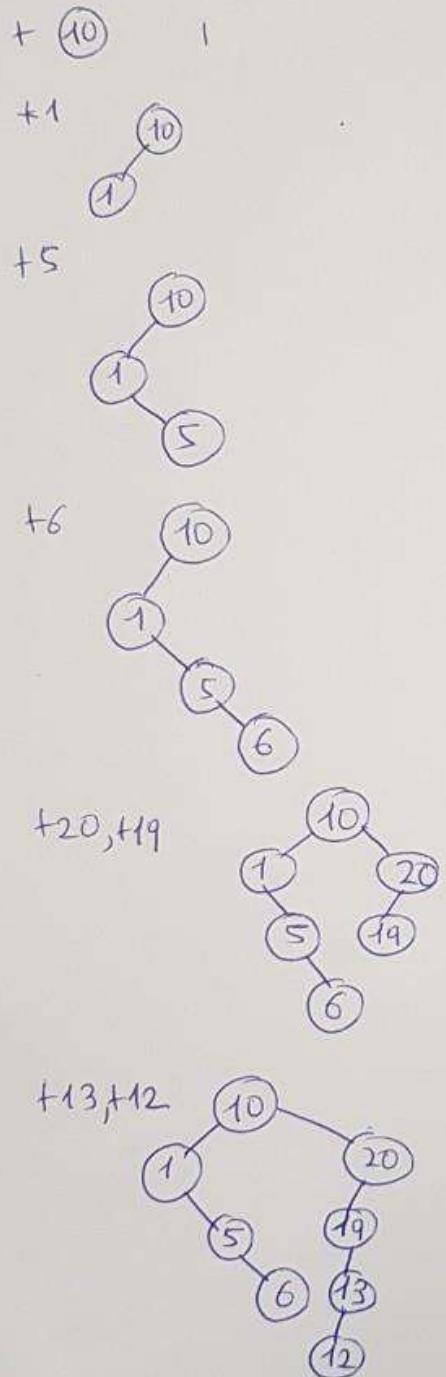


-6



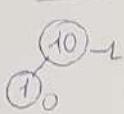
Ερώτημα 2

Διαδικτύο Δίνηση

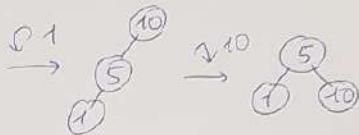
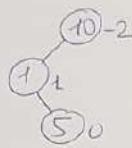


AVL

+10,+1



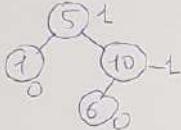
+5



$\leftarrow 1$

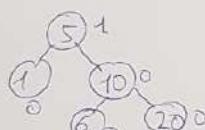
$\nearrow 10$

+6

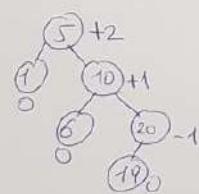


(2) με

+20

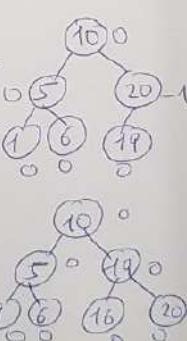


+19

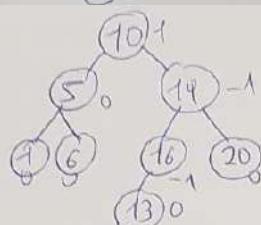


$\leftarrow 5$

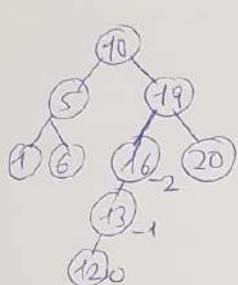
$\nearrow 20$



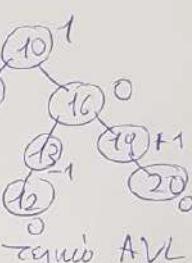
+13



+12



$\leftarrow 16$



$\nearrow 16$

$\nearrow 1$

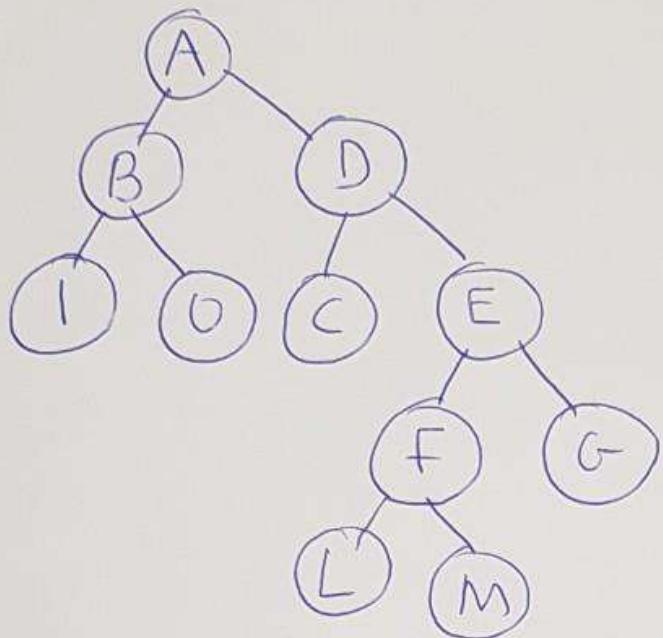
τερματίζεται AVL

Σε επιτάχιη ανθεκτικότητα νομίζουμε ότι
AVL δύναται να χρειαστεί O(log n) επιτάχια
ανθεκτικότητα σε έναν ή περισσότερους σειρές

σειρές

Ερώτημα 3

Ερώτημα 3



To δίκτυο αυτό μπορεί να ταιριάζει σε 2 διαφορετικές περιστάσεις

4.1.1 Θέμα 2 Σεπτέμβριος 2018

ΘΕΜΑ 2 (1,25 μονάδες)

Έστω μη ταξινομημένος πίνακας A , μεγέθους n . Να δώσετε έναν αλγόριθμο που παράγει ως έξοδο τα k μεγαλύτερα και τα k μικρότερα στοιχεία του A , όπου k μια σταθερά. Ο αλγόριθμος που θα δώσετε θα πρέπει να τρέχει σε χρόνο $O(n)$, στη χειρότερη περίπτωση. Θεωρείστε ότι έχετε στη διάθεσή σας τον γραμμικό Median αλγόριθμο Select.

4.1.2 Θέμα 6 Σεπτέμβριος 2012

Η τάξη ενός κλειδιού k σε ένα σύνολο S είναι ο αριθμός κλειδιών που είναι μικρότερα ή ίσα από το k (συμπεριλαμβανομένου του k). Θέλουμε να αποθηκεύσουμε το S σε ένα δυαδικό δέντρο αναζήτησης ώστε να μπορούμε να καθορίσουμε την τάξη ενός οποιουδήποτε κλειδιού σε αυτό, σε χρόνο $O(h)$ όπου h είναι το ύψος του δέντρου (Δεν υπάρχουν διπλές τιμές στο δέντρο). Δεν επιτρέπονται περιστροφές ή άλλους είδους ανακατασκευές του δέντρου. Τι πληροφορία πρέπει να αποθηκεύσουμε σε κάθε κόμβο; Περιγράψτε αλγόριθμο που εισάγει νέο κλειδί k σε χρόνο $O(h)$ όπου h είναι το ύψος του δέντρου. Εάν το k υπάρχει ήδη στο δέντρο δεν γίνεται καμία αλλαγή, διαφορετικά γίνονται οι απαραίτητες αλλαγές στις πληροφορίες που είναι αποθηκευμένες στο δέντρο ώστε ο αλγόριθμος του δεύτερου ερωτήματος να δουλεύει ακόμα.

Απάντηση

Σε κάθε κόμβο θα αποθηκεύσουμε την τιμή ενός στοιχείου του συνόλου S μαζί με ένα αριθμό (κλειδί) που θα είναι το πλήθος των προηγούμενων κόμβων του κόμβου αυτού (τάξη κλειδιού).

Ο αλγόριθμος που βρίσκει την τάξη ενός κλειδιού είναι ο εξής:

- Ξεκίνα από τη ρίζα του δέντρου
- Κάνε αναζήτηση στο δέντρο μέχρι να εντοπίσεις τον κόμβο με το επιθυμητό κλειδί
- Εμφάνισε την τάξη του κλειδιού από τον $2^{\text{ο}}$ πεδίο του κόμβου
- Το κόστος αυτού του αλγορίθμου είναι $O(d)$ όπου d το βάθος του δέντρου

Ο αλγόριθμος που εισάγει ένα κλειδί είναι ο εξής:

- Εκτελούμε τον προηγούμενο αλγόριθμο για να εντοπίσουμε τη θέση του δέντρου στην οποία θα εισαχθεί η νέα τιμή
- Αν η τιμή υπάρχει ήδη στο δέντρο τότε ο αλγόριθμος τερματίζει αλλιώς εισάγουμε το νέο κόμβο στη θέση αυτή
- Στη συνέχεια επαναπολογίσουμε την τάξη κάθε κόμβου του δέντρου. Ξεκινάμε από κάθε κόμβο και υπολογίζουμε το πλήθος όλων των κόμβων που είναι \leq από αυτόν. Η διαδικασία σταματά όταν φτάσουμε στον κόμβο αυτό (δεν εξετάζουμε ποτέ τους μεγαλύτερους κόμβους). Το κόστος αυτής της διαδικασίας είναι $O(h)$ όπου h το ύψος του δέντρου
- Το κόστος αυτού του αλγορίθμου είναι $O(d)$ για να εντοπίσουμε τη θέση που θα εισαχθεί ο νέος κόμβος και $O(1)$ για να γίνει η εισαγωγή του κόμβου

4.1.3 Θέμα 7 Σεπτέμβριος 2012

Απάντηση

Ένα B-δέντρο είναι μια δομή δεδομένων που αποθηκεύει τα δεδομένα ταξινομημένα και επιτρέπει αναζήτησεις, ακολουθιακή πρόσβαση, προσθήκες και διαγραφές σε λογαριθμικό χρόνο. Το B-δέντρο αποτελεί μια γενίκευση ενός δυαδικού δέντρου αναζήτησης στο ότι ένας κόμβος μπορεί να έχει περισσότερα από δύο παιδιά. Σε αντίθεση με τα δέντρα δυαδικής αναζήτησης, το B-δέντρο έχει βελτιστοποιηθεί για συστήματα που διαβάζουν και γράφουν μεγάλα μπλοκ δεδομένων και χρησιμοποιούνται συνήθως σε βάσεις δεδομένων και συστήματα αρχείων.

Ανάλυση B-Δέντρου

- Ο μέγιστος αριθμός στοιχείων σε ένα of items in a B-tree τάξης m και ύψους h :

$$\begin{aligned} \text{root} &= m - 1 \\ \text{level 1} &= m(m - 1) \\ \text{level 2} &= m^2(m - 1) \\ &\dots \\ \text{level } h &= m^h(m - 1) \end{aligned}$$

- Άρα ο συνολικός αριθμός στοιχείων είναι: $(1 + m + m^2 + m^3 + \dots + m^h)(m - 1) = [(m^{h+1} - 1)/(m - 1)](m - 1) = m^{h+1} - 1$

Ένα B-δέντρο τάξης m έχει τα εξής χαρακτηριστικά:

1. Ο αριθμός των κλειδιών σε κάθε εσωτερικό κόμβο είναι κατά ένα λιγότερος από τον αριθμό των παιδιών
2. Όλα τα φύλλα είναι στο ίδιο επίπεδο
3. Όλοι οι εσωτερικοί κόμβοι (με εξαίρεση τη ρίζα) έχουν τουλάχιστον $\lceil m / 2 \rceil$ παιδιά
4. Η ρίζα τουλάχιστον 2 παιδιά αν είναι εσωτερικός κόμβος και δεν έχει παιδιά αν είναι φύλλο
5. Κάθε φύλλο περιέχει όχι περισσότερα από $m - 1$ κλειδιά

β) Αναζήτηση

Search(T, k) //Αναζήτηση σε ένα B-Tree T για το κλειδί k . Αν το k βρεθεί τότε επιστρέφεται ένας δείκτης προς τον κόμβο που περιέχει k αλλιώς επιστρέφεται NULL

```
{
    current_node = root (T)
    while current_node! =NULL
        Ψάξε στα κλειδιά του current_node μέχρις ότου
            1. εντοπιστεί το  $k$  είτε
            2. εντοπιστεί το 1o κλειδί ( $k_i$ ) που είναι μεγαλύτερο από το  $k$  είτε
            3. φτάσουμε στο τελευταίο κλειδί του current_node
        περίπτωση 1:
            επιστρέφεται ο δείκτης στον current_node
        περίπτωση 2:
            current_node = i-οστό παιδί του current_node
```

περίπτωση 3:

current_node = τελευταίο child of current_node

end while**return** NULL

}

4.1.4 Θέμα 8 Σεπτέμβριος 2012

Η μέση ασυμπτωτική απόδοση ενός λεξικού για εισαγωγή, διαγραφή, αναζήτηση είναι $O(\log 2N)$ σε ένα ζυγισμένο δυαδικό δέντρο όπως τα AVL ή τα Red-Black Tree. Σε ένα B-δέντρο απόδοση για τις πράξεις αυτές είναι $O(\log_B N)$ όπου B η τάξη του δέντρου

i) Οταν $B=2$ τότε τα B-δέντρα έχουν την ίδια απόδοση ε τα δυαδικά; Υπάρχουν πλεονεκτήματα ή μειονεκτήματα συγκριτικά;

ii) Το ύψος B-δέντρων είναι ανάλογο του $\log_B N$. Για δεδομένο ένα B-δέντρο μεγαλύτερης τάξης είναι κοντύτερο από ένα δέντρο μικρότερης τάξης. Γιατί να μην διαλέγουμε ολοένα και μεγαλύτερη παράμετρο B αφού θα μειώνεται ο ασυμπτωτικός χρόνος των πράξεων

iii) τα B-δέντρα βρίσκουν εφαρμογή όταν οι κόμβοι αποθηκεύονται στο δίσκο και όχι στην κύρια μνήμη-γιατί;

Απάντηση

i) Οταν $B=2$ τα B-δέντρα έχουν την ίδια απόδοση με τα δυαδικά. Το πλεονέκτημα με τα B-δέντρα είναι ότι τα B-δέντρα διαφέρουν από τα δυαδικά δέντρα στο ότι τα κλειδιά και οι δείκτες είναι συγκεντρωμένοι στη μνήμη και έτσι έχουν καλύτερη συμπεριφορά στη cache τόσο στο δίσκο όσο και στη μνήμη.

ii) Όσο μικρότερο είναι το ύψος του δέντρου τόσο μεγαλύτερο θα είναι το μέγεθος κάθε κόμβου. Έτσι είναι πιθανό να μην μπορούμε να διαβάσουμε ένα κόμβο σε μια λειτουργία ανάγνωσης από το δίσκο και να χρειαστούν περισσότερες λειτουργίες γιαυτό. Επειδή όμως η πρόσβαση στο δίσκο είναι χρονοβόρα όσες περισσότερες αναγνώσεις γίνονται τόσο περισσότερο χρόνο θα καθυστερούμε μέχρι να διαβαστούν όλες οι τιμές του δέντρου από το δίσκο

iii) Σε αντίθεση με τα δέντρα δυαδικής αναζήτησης, τα B-δέντρα έχει βελτιστοποιηθεί για συστήματα που διαβάζουν και γράφουν μεγάλα μπλοκ δεδομένων και χρησιμοποιούνται συνήθως σε βάσεις δεδομένων και συστήματα αρχείων. Αυτά τα μπλοκ δεδομένων δεν χωρούν στην κύρια μνήμη παρά μόνο στο δίσκο και γιαυτό τα B-δέντρα χρησιμοποιούνται όταν οι κόμβοι τους αποθηκεύονται στο δίσκο.

4.1.5 Θέμα 2 Ιούνιος 2013

Έστω ότι μας δίνεται ένα σύνολο S από integers. Προτείνετε μια δομή που να υλοποιεί το πρόβλημα του λεξικού για το σύνολο S (δηλ. insert(x), delete(x), search(x) σε χρόνο $O(\log |S|)$) και επιπρόσθετα να υποστηρίζει την πράξη count(x,x') που θα μας επιστρέφει πόσα στοιχεία μέσα στο S είναι μεγαλύτερα ή ίσα του x και μικρότερα ή ίσα του x'. Η πράξη count θα πρέπει να έχει πολυπλοκότητα $O(\log |S|)$

Απάντηση

Η δομή που θα υλοποιεί το πρόβλημα του λεξικού είναι ένα φυλλοπροσανατολισμένο δέντρο εύρεσης στο οποίο οι τιμές του συνόλου S θα αποθηκεύονται στα φύλλα του. Πιο συγκεκριμένα αυτές θα υλοποιούνται ως εξής:

a)

search(x)

{

```
v ← ρίζα
while (v ≠ φύλλο)
{
```

εντόπισε το υποδέντρο i για το οποίο ισχύει ότι $H_{i-1}(v) \leq x < H_i(v)$

v ← i-οστό παιδί του v

```

    }
    if πληροφορία(v)=x then βρέθηκε
    else δεν υπάρχει
}

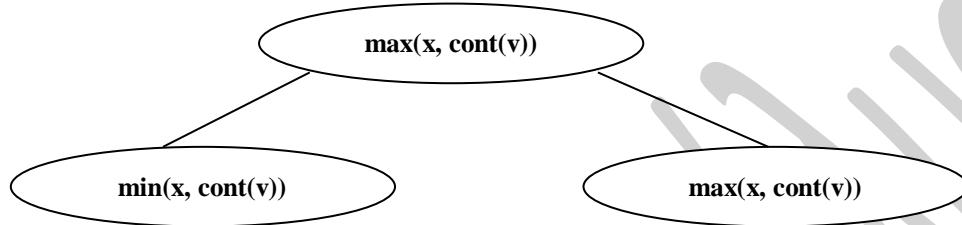
```

β)

```

insert(x)
{
v←το φύλλο στο οποίο τερμάτισε η search(x)
if πληροφορία(v)=x then μην κάνεις τίποτα
else αντικατέστησε το v με το υποδέντρο

```



γ)

```

delete(x)
{
v←το φύλλο στο οποίο τερμάτισε η search(x)

```

if πληροφορία(v)≠x then μην κάνεις τίποτα

else

```

{
    u←ο κοντινότερος πρόγονος του v με βαθμό ≥2
    σβήσε την i-οστή πλευρά του u και ένα από τα υποδέντρα Hi-1(u), Hi(u)

```

if degree(u)=1 then

αντικατέστησε το δείκτη από τον πατέρα του u προς το u με το δείκτη στο γιό του u

}

}

Οι 3 προηγούμενες λειτουργίες υλοποιούνται όπως αναφέραμε σε φυλλοπροσανατολισμένο δέντρο εύρεσης και απαιτούν χρόνο $O(\log |S|)$ όπου $|S|$ το πλήθος των φύλλων του δέντρου.

δ)

```

count(x, x')
{
v←ρίζα του δέντρου
while (v≠φύλλο)
{
    if (πληροφορία(v)≥x) then
    {
        a=a+1
    }
}

```

$a=a+1$

$v \leftarrow \text{ρίζα του υποδέντρου } H_{i+1}(v)$

```

        }
    else
    {

        b=b+1
        v←ρίζα του υποδέντρου Hi-1(v)
    }

}

return a+b
}

```

Η πράξη αυτή όπως και οι προηγούμενες αρχίζει από τη ρίζα του δέντρου και τερματίζεται σε φύλλο άρα ο χρόνος χειρότερης περίπτωσης είναι και πάλι $O(\log |S|)$ όπου $|S|$ το πλήθος των φύλλων του δέντρου

4.1.6 Θέμα 2 Σεπτέμβριος 2007

Ταξινόμηση επαναλαμβανόμενων τιμών: Υποθέστε πως δίνετε ένα σύνολο ν αριθμών με πολλές επαναλήψεις, έτσι ώστε να υπάρχουν μόνο d διαφορετικές τιμές ($d < n$). Χρησιμοποιώντας μόνο συγκρίσεις, δείξτε πως γίνεται να ταξινομηθούν οι αριθμοί με πολυπλοκότητα χειρότερης περίπτωσης $O(n \log d)$

Απάντηση

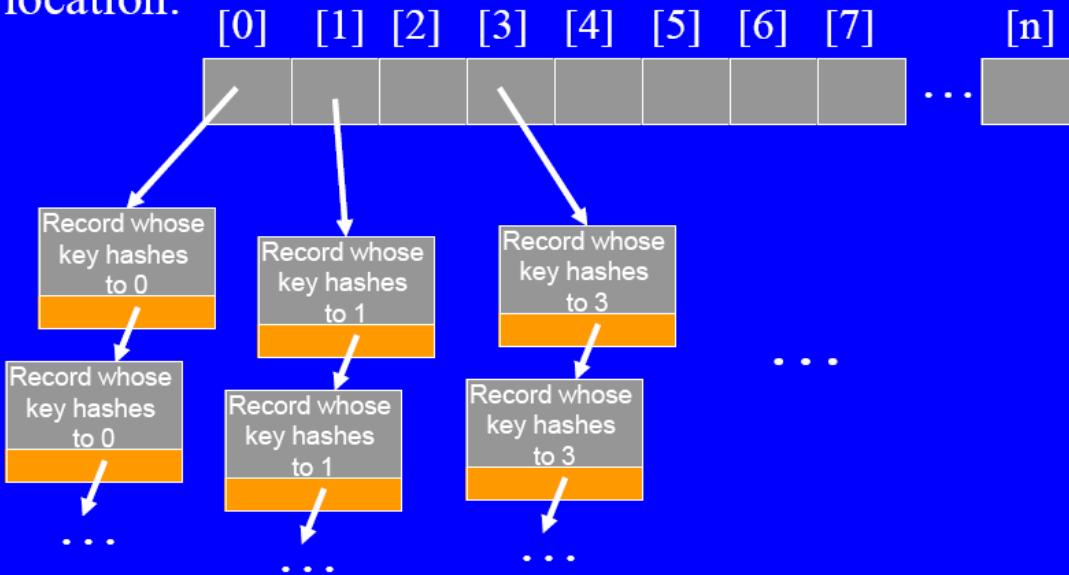
Οι τιμές θα τοποθετούνται σε ένα AVL δέντρο το οποίο είναι δέντρο αναζήτησης πράγμα που σημαίνει ότι οι τιμές στα φύλλα του θα είναι ταξινομημένες,. Επειδή όμως ένα δέντρο περιέχει μοναδικές τιμές και οι τιμές που θέλουμε να ταξινομήσουμε δεν είναι μοναδικές τότε κάνουμε το εξής:

1. Αρχικοποιούμε ένα μετρητή στο μηδέν
2. Τοποθετούμε την πρώτη τιμή σε ένα AVL δέντρο
3. Για κάθε επόμενη τιμή κάνουμε το εξής:
4. Αυξάνουμε το μετρητή κατά 1
5. Κάνουμε αναζήτηση στο AVL δέντρο για το αν υπάρχει αυτή η τιμή. Αν υπάρχει (αυτό είναι αναμενόμενο αφού πολλές από τις τιμές του αρχικού συνόλου είναι ίδιες) τότε δεν ξαναεισάγεται στο δέντρο (διότι οι τιμές του δέντρου είναι μοναδικές). Αν δεν υπάρχει τότε μπαίνει κανονικά στο δέντρο.
6. Αν ο μετρητής γίνει ίσος με το n τερμάτισε αλλιώς πήγανε στο βήμα 3

Το κόστος αναζήτησης στο AVL δέντρο είναι $O(\log_2 d)$ και ομοίως το κόστος εισαγωγής (ένθεσης) τιμής στο δέντρο είναι $O(\log_2 d)$ όπου d είναι το πλήθος των διαφορετικών τιμών αφού μόνο αυτές θα τοποθετηθούν στο AVL δέντρο. Η διαδικασία επαναλαμβάνεται η φορές όσες και οι τιμές του αρχικού συνόλου. Άρα το συνολικό κόστος της μεθόδου είναι $O(n \log_2 d)$

Chained Hashing

- In chained hashing, each location in the hash table contains a list of records whose keys map to that location:



Time Analysis of Hashing

- Worst case: every key gets hashed to same array index! $O(n)$ search!!

Στατικός κατακερματισμός

- Στο **στατικό κατακερματισμό** (static hashing) δεσμεύεται εξαρχής ένας αριθμός από κάδους (buckets).
- Η συνάρτηση κατακερματισμού $h(K)$ είναι μία **συνάρτηση** από το σύνολο των τιμών του κλειδιού διάταξης K στο σύνολο των διευθύνσεων των κάδων B .
- Η συνάρτηση κατακερματισμού χρησιμοποιείται για την αναζήτηση, εισαγωγή αλλά και τη διαγραφή εγγραφών.
- Εγγραφές με διαφορετικές τιμές του κλειδιού διάταξης μπορεί να τοποθετηθούν στον ίδιο κάδο. Επομένως, όλος ο κάδος πρέπει να **σαρωθεί σειριακά** για την εύρεση μιας εγγραφής.

Παραδείγματα Κατακερματισμού

- Υπάρχουν 10 κάδοι (buckets), 4 εγγραφές / κάδο
- Έστω, η δυαδική αναπαράσταση του i-οστού χαρακτήρα είναι ο ακέραιος j. Η συνάρτηση κατακερματισμού επιστρέφει το άθροισμα των δυαδικών αναπαραστάσεων των χαρακτήρων modulo 10

Παράδειγμα:

- $h(\text{Perryridge}) = 5$
- $h(\text{Round Hill}) = 3$

Παραδείγματα Κατακερματισμού

bucket 0

--	--	--

bucket 1

--	--	--

bucket 2

--	--	--

bucket 3

A-217	Brighton	750
A-305	Round Hill	350

bucket 4

A-222	Redwood	700

bucket 5

A-102	Perry ridge	400
A-201	Perry ridge	900
A-218	Perry ridge	700

bucket 6

--	--	--

bucket 7

A-215	Mianus	700

bucket 8

A-101	Downtown	500
A-110	Downtown	600

bucket 9

--	--	--

Συναρτήσεις κατακερματισμού

- Η χειρότερη συνάρτηση κατακερματισμού αντιστοιχεί όλες τις τιμές του κλειδιού διάταξης στον ίδιο κάδο.
- Μια ιδανική συνάρτηση κατακερματισμού είναι **ομοιόμορφη**, δηλαδή σε κάθε κάδο ανατίθεται ο ίδιος αριθμός από τιμές του κλειδιού διάταξης από το σύνολο όλων των πιθανών τιμών.
- Η ιδανική συνάρτηση κατακερματισμού είναι **τυχαία**, έτσι ώστε κάθε κάδος να έχει τον ίδιο αριθμό από εγγραφές ανεξάρτητα από την **πραγματική κατανομή** των τιμών του κλειδιού διάταξης στο αρχείο.
- Οι συνηθισμένες συναρτήσεις κατακερματισμού υπολογίζουν την εσωτερική δυαδική αναπαράσταση του κλειδιού διάταξης.

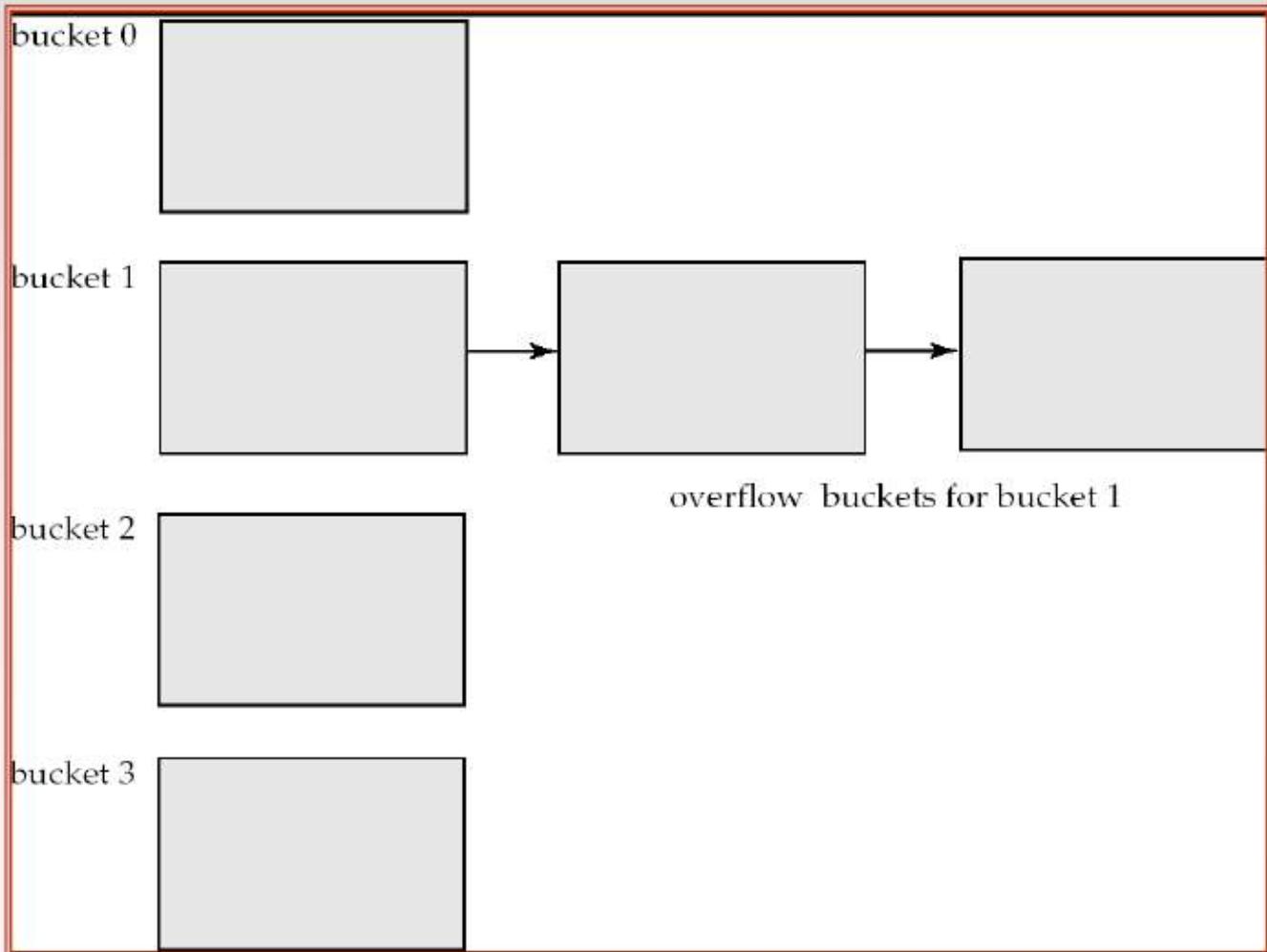
Χειρισμός των Bucket Overflows

- **Σύγκρουση** (collision) συμβαίνει όταν μια νέα εγγραφή κατακερματίζεται σε έναν ήδη γεμάτο κάδο λόγω:
 - ανεπάρκειας των κάδων (λίγοι σε σχέση με τον όγκο της ΒΔ ή
 - ασυμμετρίας στην κατανομή των εγγραφών.

□ Επίλυση συγκρούσεων:

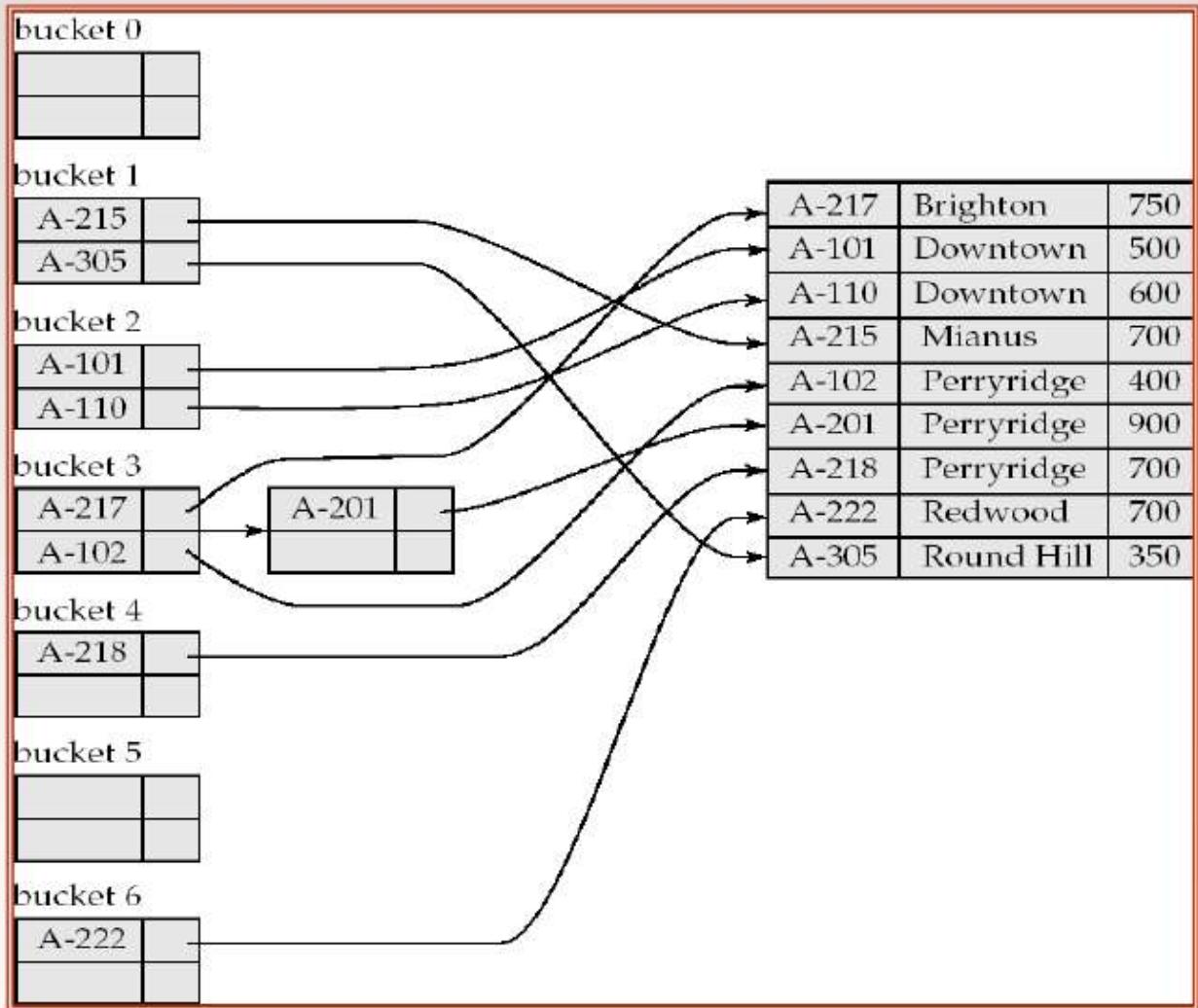
- **Ανοιχτή Διεύθυνσιοδότηση** (open addressing): τοποθέτηση στην επόμενη κενή θέση
- **Αλυσιδωτή Σύνδεση** (chaining): συνδεδεμένη λίστα με κάδους υπερχείλισης (overflow buckets)
- **Πολλαπλός Κατακερματισμός** (multiple hashing): εφαρμογή δεύτερης συνάρτησης κατακ/μού

Χειρισμός των Bucket Overflows



Hash Ευρετήρια

- Το Hashing χρησιμοποιείται όχι μόνο για οργάνωση αρχείων, αλλά και ως δόμησης αποδοτικών ευρετηρίων.



Μειονεκτήματα στατικού κατακ/μού

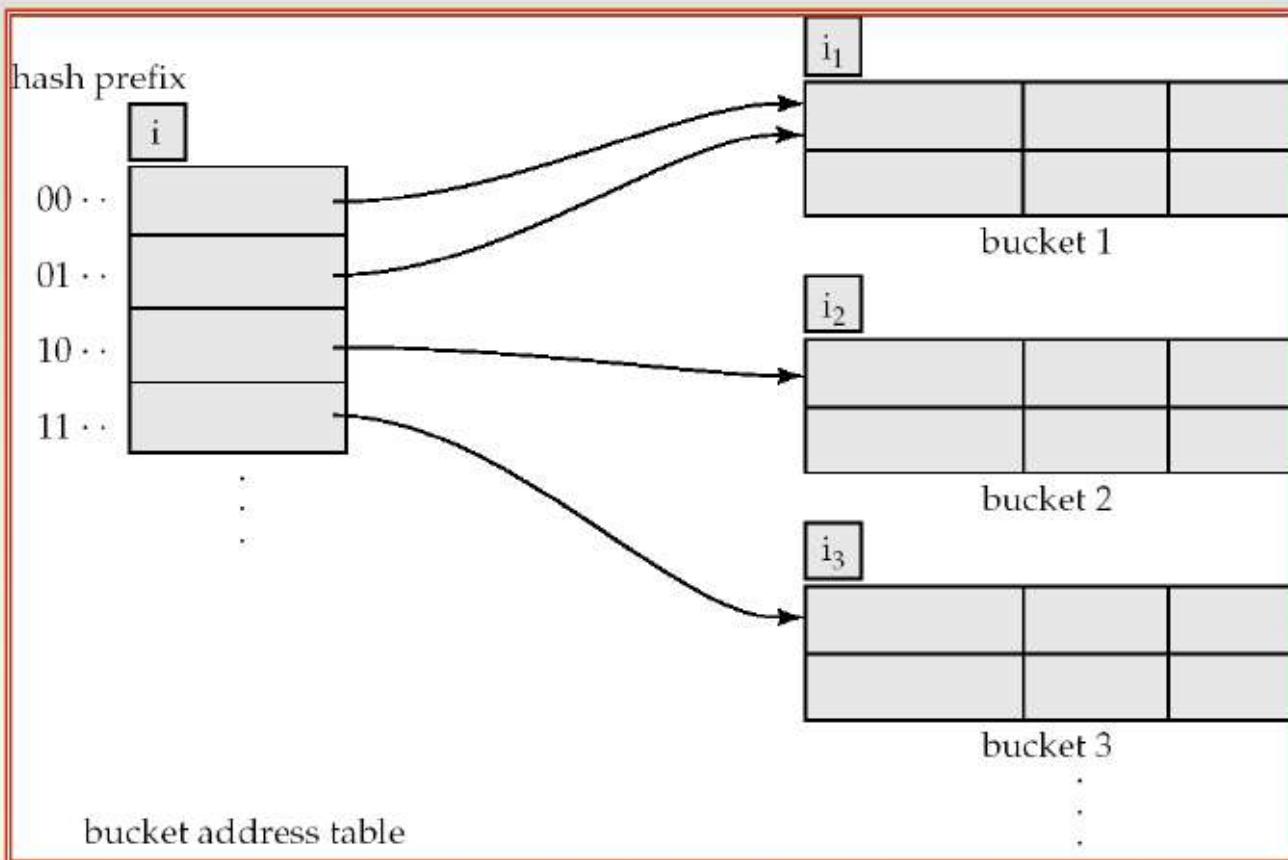
- Το σύνολο των διευθύνσεων κάδων είναι προκαθορισμένο.
 - Οι βάσεις δεδομένων μεγαλώνουν με το χρόνο. Αν ο αριθμός κάδων είναι πολύ μικρός, η απόδοση θα μειωθεί λόγω των αλυσίδων υπερχείλισης.
 - Έστω και εάν είναι προβλέψιμο το μελλοντικό μέγεθος των αρχείων και προσδιοριστεί ανάλογα ο αριθμός των κάδων, στην αρχή θα σπαταλάται άσκοπα σημαντικό ποσό χώρου.
 - Όταν συρρικνώνεται η βάση δεδομένων, πάλι υπάρχει σπατάλη χώρου.
 - Μια λύση είναι η **περιοδική αναδιοργάνωση** του αρχείου με μια νέα συνάρτηση κατακερματισμού, αλλά αυτή η διαδικασία είναι πολύ ακριβή (σε χώρο και χρόνο).
- Αυτά τα προβλήματα αποφεύγονται με τεχνικές που επιτρέπουν να τροποποιείται δυναμικά ο αριθμός των κάδων
 - **Επεκτάσιμος κατακερματισμός** (extensible hashing).

Δυναμικό Hashing

- Βολεύει για ΒΔ το μέγεθος των οποίων ανξομειώνεται
- Επιτρέπει στην hash συνάρτηση να τροποποιείται δυναμικά
- Μια μορφή δυναμικού hashing είναι το **Extendable hashing**
 - Η συνάρτηση Hash παράγει τιμές σε ένα μεγάλο εύρος — τυπικά ακεραίους των b -bits, όπου $b = 32$.

- Ανά πάσα στιγμή μπορούμε να χρησιμοποιήσουμε το πρόθεμα (prefix) της συνάρτησης hash για να δεικτοδοτήσουμε τον πίνακα των διευθύνσεων των κάδων.
- Έστω ότι το μήκος του προθέματος είναι i bits, $0 \leq i \leq 32$.
- Bucket address table size = 2^i . Αρχικά $i = 0$
- Το i αυξομειώνεται καθώς και η ΒΔ αυξομειώνεται.
- Πολλαπλές καταχωρήσεις του bucket address table μπορεί να δείχνουν σε ένα bucket.
- Άρα, ο ακριβής αριθμός των buckets $< 2^i$
- Ο αριθμός των buckets επίσης αλλάζει δυναμικά εξαιτίας της συγχώνευσης ή της διάσπασης τους.

Γενική Extendable Hash Δομή



Ταξινόμηση Δέντρων

Γενικά Δέντρα-έχουν οποιοδήποτε αριθμό κόμβων/παιδιών

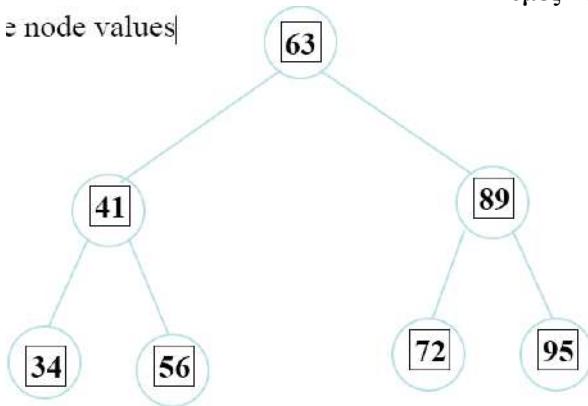
Δυαδικά Δέντρα-κάθε κόμβος έχει max 2 παιδιά

Σωροί-κάθε πατρικός κόμβος έχει τιμή μεγαλύτερη και από 2 παιδιά του

Δυαδικά Δέντρα Αναζήτησης (Binary Search Trees). Κάθε στοιχείο έχει μοναδική τιμή. Τα κλειδιά του αριστερού υποδέντρου είναι $<$ της τιμής στη ρίζα και τα κλειδιά του δεξιού υποδέντρου είναι $>$ της τιμής στη ρίζα. Όλες οι πράξεις στα δυαδικά δέντρα αναζήτησης έχουν πολυπλοκότητα χειρότερης περίπτωσης $O(\log n)$ για αναζητήσεις, εισαγωγές και διαγραφές

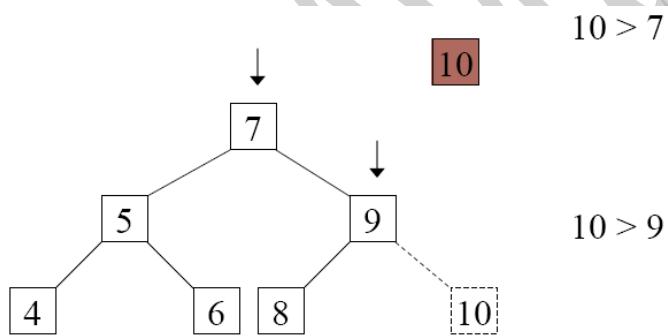
Παράδειγμα Δυαδικού Δέντρου Αναζήτησης

⇒ node values

**Ψυδοκώδικας για Αναζήτηση Κλειδιού σε Δυαδικό Δέντρα Αναζήτησης (BST)**

method search(key)

- Αν το δέντρο είναι κενό τότε επέστρεψε NULL
- αλλιώς αν το στοιχείο του κόμβου είναι ίσο με το στοιχείο αναζήτησης τότε επέστρεψε τον αριθμό του κόμβου (το στοιχείο εντοπίστηκε)
- αλλιώς αν το στοιχείο του κόμβου είναι μεγαλύτερο από το στοιχείο αναζήτησης τότε η αναζήτηση γίνεται στο αριστερό υποδέντρο
- αλλιώς αν το στοιχείο του κόμβου είναι μικρότερο από το στοιχείο αναζήτησης τότε η αναζήτηση γίνεται στο δεξιό υποδέντρο

**Ψυδοκώδικας για Εισαγωγή νέου κλειδιού σε Δυαδικό Δέντρο Αναζήτησης BST**

method insert(key)

- Αν το δέντρο είναι κενό τότε δημιουργησε μια ρίζα με το νέο κλειδί
- αλλιώς
 - σύγκρινε το νέο κλειδί με το κλειδί του κορυφαίου κόμβου
 - αν **κλειδί=κλειδί κόμβου** τότε αντικατέστησε τον κόμβο με το νέο κλειδί
 - αλλιώς αν **νέο κλειδί > κλειδί κόμβου** σύγκρινε το νέο κλειδί με το δεξιό υποδέντρο. Αν το δεξιό υποδέντρο είναι κενό δημιουργησε ένα φύλλο για το νέο κλειδί αλλιώς πρόσθεσε το νέο κλειδί στο δεξιό υποδέντρο
 - αλλιώς αν **νέο κλειδί < κλειδί κόμβου** σύγκρινε το νέο κλειδί με το αριστερό υποδέντρο. Αν το αριστερό υποδέντρο είναι κενό δημιουργησε ένα φύλλο για το νέο κλειδί αλλιώς πρόσθεσε το νέο κλειδί στο αριστερό υποδέντρο

Ψυδοκώδικας για Διαγραφή κλειδιού από Δυαδικό Δέντρο Αναζήτησης BST

- Αν το δέντρο είναι κενό τότε επέστρεψε false
- εφάρμοσε τον αλγόριθμο binary search για εντοπισμό του κλειδιού διαγραφής

- αν το κλειδί διαγραφής δεν εντοπιστεί τότε επέστρεψε false
- αν το κλειδί διαγραφής εντοπιστεί τότε
 - Περίπτωση 1: αν ο κόμβος έχει 2 κενά υποδέντρα αντικατέστησε το σύνδεσμο του πατέρα με null
 - Περίπτωση 2: αν ο κόμβος έχει αριστερό και δεξιό υποδέντρο αντικατέστησε την τιμή του κόμβου με τη μεγαλύτερη τιμή του αριστερού υποδέντρου και διαγράφει το μέγιστο κόμβο του αριστερού υποδέντρου
 -

method remove (key)

I if the tree is empty return false

II Attempt to locate the node containing the target using the binary search algorithm

if the target is not found return false

else the target is found, so remove its node:

Case 1: if the node has 2 empty subtrees

replace the link in the parent with null

Case 2: if the node has a left and a right subtree

- replace the node's value with the max value in the

left subtree

- delete the max node in the left subtree

Computer Ανάλυση